system manual

- system architecture
- internal operations
- maintenance

digital
Order additional copies as directed on the Software Information page at the back of this document.

digital equipment corporation • maynard, massachusetts
The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1972, 1973, 1974 Digital Equipment Corporation

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

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:

<table>
<thead>
<tr>
<th>CDP</th>
<th>DIGITAL</th>
<th>INDAC</th>
<th>PS/8</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTER LAB</td>
<td>DNC</td>
<td>KA10</td>
<td>QUICKPOINT</td>
</tr>
<tr>
<td>COMSYST</td>
<td>EDGRIN</td>
<td>LAB-8</td>
<td>RAD-8</td>
</tr>
<tr>
<td>COMTEX</td>
<td>EDUSYSTEM</td>
<td>LAB-8/e</td>
<td>RSTS</td>
</tr>
<tr>
<td>DDT</td>
<td>FLIP CHIP</td>
<td>LAB-K</td>
<td>RSX</td>
</tr>
<tr>
<td>DEC</td>
<td>FOCAL</td>
<td>OMNIBUS</td>
<td>RTM</td>
</tr>
<tr>
<td>DECCOMM</td>
<td>GLC-8</td>
<td>OS/8</td>
<td>RT-11</td>
</tr>
<tr>
<td>DECTAPE</td>
<td>IDAC</td>
<td>PDP</td>
<td>SABR</td>
</tr>
<tr>
<td>DIBOL</td>
<td>IDACS</td>
<td>PHA</td>
<td>TYPESET 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UNIBUS</td>
</tr>
</tbody>
</table>
PREFACE

This manual was written for customer systems programmers, DEC Software Specialists, and internal maintenance programmers. Readers must be familiar with the DOS User's Manual, DEC-15-ODUMA-B-D. In addition, chapter 8 requires familiarity with the BOSS Reference Manual, DEC-15-OBUMA-A-D.

Technical changes reflected in this revision are indicated by a bar (||) in the appropriate page margin.
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOS OPERATION</td>
<td>1-1</td>
</tr>
<tr>
<td>2</td>
<td>THE RESIDENT MONITOR</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1</td>
<td>INTRODUCTION</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2</td>
<td>THE CAL HANDLER</td>
<td>2-2</td>
</tr>
<tr>
<td>2.3</td>
<td>IOPS ERROR HANDLER, AND THE EXPANDED ERROR PROCESSOR</td>
<td>2-2</td>
</tr>
<tr>
<td>2.3.1</td>
<td>.MED</td>
<td>2-2</td>
</tr>
<tr>
<td>2.3.2</td>
<td>The Expanded Error Processor</td>
<td>2-6</td>
</tr>
<tr>
<td>2.4</td>
<td>THE SYSTEM BOOTSTRAP</td>
<td>2-7</td>
</tr>
<tr>
<td>2.5</td>
<td>SYSTEM I/O INITIALIZATION</td>
<td>2-8</td>
</tr>
<tr>
<td>2.6</td>
<td>RESIDENT MONITOR TIMING FEATURES</td>
<td>2-8</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Clock Operation</td>
<td>2-14</td>
</tr>
<tr>
<td>2.6.2</td>
<td>.TIMER</td>
<td>2-14</td>
</tr>
<tr>
<td>2.7</td>
<td>THE RESIDENT MONITOR PATCH AREA</td>
<td>2-16</td>
</tr>
<tr>
<td>2.8</td>
<td>CONTROL CHARACTERS</td>
<td>2-16</td>
</tr>
<tr>
<td>2.9</td>
<td>TASK CONTROL BLOCKS (for UC15 system - RKØ5 based or RP15/RPØ2 based with UC15 option)</td>
<td>2-17</td>
</tr>
<tr>
<td>3</td>
<td>THE NONRESIDENT MONITOR</td>
<td>3-1</td>
</tr>
<tr>
<td>3.1</td>
<td>INTRODUCTION</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2</td>
<td>COMMANDS TO THE NONRESIDENT MONITOR</td>
<td>3-4</td>
</tr>
<tr>
<td>3.3</td>
<td>CONSIDERATIONS FOR ADDITIONS TO THE NONRESIDENT MONITOR</td>
<td>3-4</td>
</tr>
<tr>
<td>3.4</td>
<td>OFILE</td>
<td>3-8</td>
</tr>
<tr>
<td>4</td>
<td>THE SYSTEM LOADER AND THE LINKING LOADER</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1</td>
<td>MANUAL BOOTSTRAP LOADS AND RESTARTS</td>
<td>4-13</td>
</tr>
<tr>
<td>4.2</td>
<td>LOADING SYSTEM PROGRAMS</td>
<td>4-13</td>
</tr>
<tr>
<td>4.3</td>
<td>TABLES AND INFORMATION BLOCKS USED AND BUILT BY LOADERS</td>
<td>4-15</td>
</tr>
<tr>
<td>4.4</td>
<td>.DAT SLOT MANIPULATION BY THE SYSTEM LOADER</td>
<td>4-15</td>
</tr>
<tr>
<td>4.5</td>
<td>BUFFER ALLOCATION BY THE SYSTEM LOADER</td>
<td>4-20</td>
</tr>
</tbody>
</table>

v
CHAPTER 7  WRITING NEW I/O DEVICE HANDLERS                 Page

7.1  I/O DEVICE HANDLERS, AN INTRODUCTION                     7-1
    7.1.1 Setting Up the Skip Chain and API                  7-1
          (Hardware) Channel Registers                      7-5
    7.1.2 Handling the Interrupt                            7-6

7.2  API SOFTWARE LEVEL HANDLERS, AN                        7-7
      INTRODUCTION                                          7-7
    7.2.1 Setting Up API Software Level Channel              7-7
           Registers                                         7-8
    7.2.2 Queueing                                          7-8

7.3  WRITING SPECIAL I/O DEVICE HANDLERS                    7-10
    7.3.1 Discussion of Example A by Parts                  7-13
    7.3.2 Example A, Skeleton I/O Device Handler            7-14
    7.3.3 Example B, Special Device Handler for             7-16
           AF0IB A/D Converter

CHAPTER 8  BOSS-15                                         8-1

8.1  PROCEDURE FILES                                       8-16
    8.1.1 Procedure File Format                             8-16
    8.1.2 Direct Substitution                               8-19
    8.1.3 Example of Procedure File                         8-19

8.2  BOSS-15 ACCOUNTING                                     8-20

8.3  BPRE                                                   8-21

APPENDIX A  DECTAPE "A" HANDLER (DTA.)                     A-1
APPENDIX B  DISK "A" HANDLERS                               B-1
APPENDIX C  PROCEDURE FILES                               C-1

Index                                                   X-1

TABLES

2-1  RF Platter-Block Number Correspondence                2-8
3-1  Effects and Exits for Nonresident Monitor Commands   3-5
4-1  Tables and Blocks Used by the Loaders                4-16
4-2  .SCOM Registers Used by the System Loader            4-17
4-3  Use of .SCOM+4 by the System Loader                   4-19
5-1  .SCOM Registers                                      5-2
5-2  Overlay Table                                        5-15
5-3  Mass Storage Busy Table Entry                         5-15
5-4  Reserved Address Locations                           5-15
<table>
<thead>
<tr>
<th>Number</th>
<th>FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Resident Monitor CAL Handler</td>
<td>2-3</td>
</tr>
<tr>
<td>2-2</td>
<td>Expanded Error Processor and Monitor Error DIY Route .MED</td>
<td>2-4</td>
</tr>
<tr>
<td>2-3</td>
<td>Resident Monitor Subroutines</td>
<td>2-5</td>
</tr>
<tr>
<td>2-4</td>
<td>.OVRLA, .EXIT and CTRL Q</td>
<td>2-9</td>
</tr>
<tr>
<td>2-5</td>
<td>Resident Monitor Initialization</td>
<td>2-10</td>
</tr>
<tr>
<td>2-6</td>
<td>The Resident Monitor Clock Routine for non-UC15 Systems</td>
<td>2-11</td>
</tr>
<tr>
<td>2-7</td>
<td>The Resident Monitor Clock Routine for UC15 Systems (RK5 based or RF15/RP2 based with UC15 option)</td>
<td>2-12</td>
</tr>
<tr>
<td>3-1</td>
<td>Nonresident Monitor Initialization</td>
<td>3-2</td>
</tr>
<tr>
<td>3-2</td>
<td>QFILE, and Implementation of GET and PUT Logic</td>
<td>3-10</td>
</tr>
<tr>
<td>4-1</td>
<td>System Loader Initialization</td>
<td>4-2</td>
</tr>
<tr>
<td>4-2</td>
<td>The System Loader</td>
<td>4-7</td>
</tr>
<tr>
<td>4-3</td>
<td>The Linking Loader</td>
<td>4-10</td>
</tr>
<tr>
<td>4-4</td>
<td>Bootstrap Load</td>
<td>4-14</td>
</tr>
<tr>
<td>4-5</td>
<td>Standard Interface Load</td>
<td>4-14</td>
</tr>
<tr>
<td>4-6</td>
<td>System Program Load</td>
<td>4-14</td>
</tr>
<tr>
<td>4-7</td>
<td>Linking Loader</td>
<td>4-14</td>
</tr>
<tr>
<td>4-8</td>
<td>Execute</td>
<td>4-14</td>
</tr>
<tr>
<td>5-1</td>
<td>SYSBLK and COMBLK</td>
<td>5-7</td>
</tr>
<tr>
<td>5-2</td>
<td>SGNBLK for RP2 and RF15 Systems</td>
<td>5-10</td>
</tr>
<tr>
<td>5-3</td>
<td>SGNBLK for RK5 Based System</td>
<td>5-11</td>
</tr>
<tr>
<td>6-1</td>
<td>DECTape Directory</td>
<td>6-2</td>
</tr>
<tr>
<td>6-2</td>
<td>DECTape File Bit Map Blocks</td>
<td>6-3</td>
</tr>
<tr>
<td>6-3</td>
<td>Block Format, File-Structured Mode</td>
<td>6-6</td>
</tr>
<tr>
<td>6-4a</td>
<td>Format of the File Directory Data Block</td>
<td>6-8</td>
</tr>
<tr>
<td>6-4b</td>
<td>Format of File-Structured Tape</td>
<td>6-8</td>
</tr>
<tr>
<td>6-5</td>
<td>User File Header Label Format</td>
<td>6-9</td>
</tr>
<tr>
<td>6-6</td>
<td>Master File Directory</td>
<td>6-13</td>
</tr>
<tr>
<td>6-7</td>
<td>User File Directory</td>
<td>6-13</td>
</tr>
<tr>
<td>6-8</td>
<td>Retrieval Information Block</td>
<td>6-15</td>
</tr>
<tr>
<td>6-9</td>
<td>Disk Buffer</td>
<td>6-15</td>
</tr>
<tr>
<td>7-1</td>
<td>CAL Entry to Device Handler (for non-UNIBUS devices only)</td>
<td>7-2</td>
</tr>
<tr>
<td>7-2</td>
<td>PI and API Entries to Device Handlers</td>
<td>7-3</td>
</tr>
<tr>
<td>7-3</td>
<td>Structure of API Software Level Handler</td>
<td>7-9</td>
</tr>
<tr>
<td>8-1</td>
<td>BOSS/DOS Intersection</td>
<td>8-2</td>
</tr>
<tr>
<td>8-2</td>
<td>Connections between DOS Resident Monitor and BOSS Resident Monitor</td>
<td>8-3</td>
</tr>
<tr>
<td>8-3</td>
<td>Points within Resident Monitor Which Transfer Control to Resident BOSS-15</td>
<td>8-4</td>
</tr>
<tr>
<td>8-4</td>
<td>Nonresident BOSS</td>
<td>8-7</td>
</tr>
<tr>
<td>8-5</td>
<td>B.PRE</td>
<td>8-22</td>
</tr>
</tbody>
</table>
CHAPTER 1
DOS OPERATION

The System Manager must use DOSSAV in order to load DOS-15 for the
first time. The DOS System Generator manual, DEC-USGNA-A-D, describes
DOSSAV operation in its appendix. After successful DOSSAV operation,
the System Manager should load the Bootstrap into the highest bank of
core memory; the bootstrap informs the DOS-15 monitor how many banks
of core memory can be used. Before bootstrapping UC15, RK05 based
systems, PIREX must first be loaded and running in the PDP-11 local
loads the Resident Monitor and the System Loader, which in turn load
the Nonresident Monitor. In order to ensure a working system, the
System Manager should place the DOS-15 Checkout Package tape (for
RF15, DEC-15-ORFCA-A-PA; for RP02, DEC-15-ORFCA-A-PA; and for RK05,
Operating instructions for the Checkout Package, and the tape itself,
are distributed as part of the DOS-15 system.

Once the system has been checked out, the System Manager should use
DOSGEN, the DOS System Generator program, to tailor the system to
his needs. As mentioned in the System Generator manual, a complete
tailoring of the system may also involve use of PATCH, PIP and
UPDATE.

Commands to the Nonresident Monitor allow temporary modification of
the system, in order to suit the needs of a particular program. The
Nonresident Monitor modifies the system by changing information in
the .SCOM Table. The System Loader examines the .SCOM Table, along
with three disk-resident information blocks, SYSBLK, COMBLK and SGNBLK,
and carries out all operations necessary to fulfill the operator's
commands. The System Loader "builds" the Resident Monitor by relocat-
ing and linking those routines indicated by the .SCOM table as needed
by the next core load. The Resident Monitor then retains general
control over the system.
CHAPTER 2

THE RESIDENT MONITOR

2.1 INTRODUCTION

The Resident Monitor gets its name because it seems resident to the user. Strictly speaking, however, the only part of the system that is always resident is the Bootstrap. There are two parts of the system that are refreshed only after manual Bootstrap loads and restarts: .SCOM and the Resident Monitor Patch Area. Every time an operator or program changes certain key system parameters, the system will build a new Resident Monitor from blocks stored on the system device.

The Resident Monitor is the interface between the operator and the active devices on one hand, and the program which is running (the Nonresident Monitor), on the other. The Resident Monitor always contains the following routines and tables:

Chapter 5

- .DAT
- .UFDT
- .SCOM

The CAL Handler, which routes all System and I/O
Macro calls
The Startup routine, called after using the Bootstrap
.MED, the Monitor's standard error routine
The Expanded Error Processor, for more flexibility
with error messages
Handlers for the following error conditions:
  Nonexistent Memory
  Memory Protect
  Interrupt-Memory Parity
  Power-Fail
  Software API not set up

The Monitor's TRAN routine (different from I/O .TRAN's)
A clock handler
A poller for UNIBUS device error messages (for
  UC15 systems, RK05 based or RF15/RP02 based
  with UC15 option only)
The .GTBUF and GVBUF processor
The CTRL Q processor
The .USER processor
The .OVRLA processor
TGA

This Chapter

- The Resident Monitor's Patch Area
- Task Control Blocks (for UC15 systems, RK05 based or
  RF15/RP02 based, with UC15 option)

In addition, the user can request the system to retain certain other routines in a resident Monitor status:

- The CTRL X Feature, including a driver for the VT-15
- The Paper Tape or Card Reader Handler for Batch
- The Resident Batch Code

BOSS-15 also has resident routines, which are covered in Chapter 8.

2-1
2.2 THE CAL HANDLER

The CAL instruction transfers control to register 21, bank $0, and loads register $0 with the address of the next instruction after the CAL. All DOS I/O and system macros take the form of a CAL instruction (possibly with some code in the low-order bits), and the next sequential register contains a dispatch code. Some macros require more information in succeeding registers. Figure 2-1, Resident Monitor CAL Handler, illustrates the operation of that portion of the Resident Monitor. The CAL Handler does only minimal error checking -- for a legal function code and for a legal .DAT slot. Aside from that and ensuring the clock is turned on, the CAL Handler is only a dispatcher to other routines.

2.3 IOPS ERROR HANDLER AND THE EXPANDED ERROR PROCESSOR

2.3.1 .MED

There are two error processors in the Resident Monitor: .MED and the Expanded Error Processor. Figure 2-2 illustrates those routines. Figure 2-3 shows two subroutines used by the error routines. .MED (location 3, bank $0) processes IOPS errors from all device handlers except the disk handlers, CDB, MTF, TTA, and LPA. Calls to .MED should take the following form, if not IOPS 4:

LAC INFO /ARGUMENT OF ERROR
DAC* (.MED /ADDRESS OF CAL IS ALREADY IN .MED,
/IF DESIRED
LAW N /N IS ERROR CODE $0$<N<$777. AC MUST BE NEGATIVE.
JMP* (.MED+1

IOPS 4 messages may take the following form:

LAC (4 /AC MUST BE POSITIVE
JMS* (.MED

.MED+1 contains a JMP to the Monitor Error Diagnostic Routine. The above calls to .MED will cause the following printouts:

IOPSN (contents of .MED)
IOPS4

2-2
Figure 2-1
Resident Monitor CAL Handler
Figure 2-2
Expanded Error Processor and
Monitor Error Diagnostic Routine
.MED

Enter from .SCOM+37
SETTLE
Wait for TTY
Error
Y
N
IOPS
Give error message
Put recovery PC in .MED
LOC+2
Y
N
Print the message

Enter from .MED
SETTLE
Wait for TTY
IOPS
Give error message
Y
IOPS 4
N
Resident Monitor Initialization
Loop

NOTE: The Nonresident Monitor HALT and DUMP commands will change this loop to the appropriate action. BOS and Batch- ing Mode abort the $JOB.

Await a character from the keyboard

Y
CTRL
Q

N
CTRL
QAREA adequate

Y
Echo Command
Resident Monitor Initialization
Dispatch to appropriate address

1. Echo Command
2. Restore API, if required
3. Restore PI

Return via .MED

(Wait for a Control Char)
1. Store error number
2. Set up to turn nulls into spaces, if LINK is set
3. Turn off PI
4. Wait 110 ms for the teleprinter to die down
5. Type Carriage RETURN, Line Feed

RETURN

IOPS

Print "IOPS" and error number, zero suppressed

IOPS 4

N: Print a space, followed by the octal contents of .MED, followed by another space

IOPS 20 or 72

N: Print contents of .SCOM+32 (disk block number)

RETURN

Figure 2-3
Resident Monitor Subroutines
2.3.2 The Expanded Error Processor

The disk handlers (except the Bootstrap), CDB., MTF. , TTA. , and LPA. use the Expanded Error Processor. Each error message is "potentially" recoverable by typing CTRL R. That is, the Resident Monitor always returns control to the caller upon a CTRL R. It is up to the caller to respond accordingly. All handlers supplied with the system simply repeat the error message if the error is unrecoverable.

The Expanded Error Processor gives the capability of printing additional information after the standard IOPS message. As with .MED, the AC must contain the error number (\(0 \leq \text{number} \leq 777\)) in bits 9-17. Control must be passed, however, via JMS* (.S COM+37, not JMP* (.MED+1).

The following information pertains to the message: LOC+2 must contain the two's complement of the number of message words to be typed after the standard "IOPSNN nnnnnn" message. If the number is zero or positive, no message will be printed. If the LINK is set, nulls will be printed as spaces. If the LINK is zero, nulls will be ignored. If the AC is positive on calling the expanded error facility, only the special message will be printed. The "IOPS" part will be omitted. The message itself must be packed in .SIXBT.

The following are examples of use of the Expanded Error Processor:

Example a:

```
UNREC LAC STATUS
DAC* (.MED /STATUS REGISTER B
/DAL ADDRESS IS NOW OVERWRITTEN
/BY CONTENTS OF STATUS REGISTER
SIZL /IGNORE NULLS
ERRNUM /ERRNUM = 777
JMS* (.S COM+37
/JMP UNREC /THIS IS AN UNRECOVERABLE ERROR.
/JMP .-1 WILL NOT DO -- EXPANDED
/ERROR PROCESSOR CHANGES THE
/CONTENTS OF .MED.
LAW -6 /5 DATA WORDS FOLLOW
.SIXBIT 'DKA'
/DEVICE NAME
40 /NULL, NULL, SPACE
.SIXBIT 'FIL'
/FILE NAME (2 WORDS)
.SIXBIT 'E'
40 /NULL, NULL, SPACE
.SIXBIT 'SRC'
/EXTENSION
```

The printout from that code will be as follows:

```
IOPS777 nnnnnn DKA FILE SRC
```

where nnnnnn is the contents of .MED, and equals the Status Register B, and ERRNUM is 777.
Example b:

PARITY LAW 61
STL /TURNS NULLS INTO SPACES
JMS* (.SCOM+37
JMP RETRY /THIS IS A RECOVERABLE ERROR
LAW -1
.SIXBT 'DTA'

The printout from that code will be as follows:

IOP61 nnnnnn DTA

where nnnnn is the contents of .MED, the address of the last CAL, deposited by the CAL Handler.

2.4 THE SYSTEM BOOTSTRAP

The System Bootstrap is nothing more than a disk driver. It may load the System Loader and Resident Monitor from Hardware Readin or manual restart. All other Bootstrap operations result from the use of the Monitor TRAN routine. The Monitor TRAN routine sets up the Bootstrap to read or write any block or set of contiguous blocks from the disk to or from any location in core. Before calling the Bootstrap, the Monitor TRAN does a .WAIT to all .DAT slots in the Mass Storage Busy Table, clears all flags, turns off the VT if it were on, and allows the clock to tick positive, so that it will keep time but not interrupt. After the Bootstrap has finished, it calls the Monitor Initialization Routine, which updates the clock and turns on the VT, if necessary.

The Monitor TRAN Routine requires the following parameter table:

<table>
<thead>
<tr>
<th>PARADD</th>
<th>LOC+0 BLKNUM</th>
<th>/FIRST BLOCK NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC+1</td>
<td>FIRSTA-1</td>
<td>/FIRST ADDRESS OF BUFFER, MINUS ONE</td>
</tr>
<tr>
<td>LOC+2</td>
<td>-SIZE</td>
<td>/# OF WORDS TO BE TRANSFERRED IN 2'S COM</td>
</tr>
<tr>
<td>LOC+3</td>
<td>START</td>
<td>/STARTING ADDRESS AFTER DISK I/O</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/COMPLETION</td>
</tr>
</tbody>
</table>

The following code illustrates the use of the Monitor TRAN:

UNIT=10000001 /MONITOR TRAN WILL USE UNIT ONE1
.
.
.SCOM=100
.
.
LAC (PARADD /MONITOR TRAN REQUIRES ADDRESS OF
XOR UNIT /PARAMETER TABLE IN BITS 3-17 AND
STL /UNIT NUMBER IN BITS 0-2 OF AC
JMP* (.SCOM+55 /NONZERO LINK GIVES TRAN OUT
    /.SCOM+55 IS USER ENTRY POINT FOR
    /MONITOR TRAN

See also paragraph 5.7.

1DECdisk TRANs ignore unit number, use block number.
.OVRLA, .EXIT, and manual Q dumps all use the Monitor TRAN routine. Figure 2-4, .OVRLA, .EXIT and CTRL Q, illustrates their operation, and also the Monitor TRAN.

For the RF DECdisk, the user can reference a specific platter just by identifying the block number he wants. That is, the block numbers do not automatically go to zero at the beginning of every platter. The block numbers and platter relationships are shown below:

Table 2-1
RF Platter-Block Number Correspondence

<table>
<thead>
<tr>
<th>Platter Number</th>
<th>Block Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0-1777</td>
</tr>
<tr>
<td>1</td>
<td>2000-3777</td>
</tr>
<tr>
<td>2</td>
<td>4000-5777</td>
</tr>
<tr>
<td>3</td>
<td>6000-7777</td>
</tr>
<tr>
<td>4</td>
<td>10000-11777</td>
</tr>
<tr>
<td>5</td>
<td>12000-13777</td>
</tr>
<tr>
<td>6</td>
<td>14000-15777</td>
</tr>
<tr>
<td>7</td>
<td>16000-17777</td>
</tr>
</tbody>
</table>

(All numbers are in octal)

2.5 SYSTEM I/O INITIALIZATION

There are two routines that do DOS I/O initialization: the startup routine after Bootstrap manual loads and restarts, and the startup routine performed after Monitor TRAN's and after a CTRL C, F, T or S for an error. The startup routine after Bootstrap loads is described in Figure 4-1, The System Loader Interface Routine. Figure 2-5, Resident Monitor Initialization, describes the other routine.

2.6 RESIDENT MONITOR TIMING FEATURES

Figure 2-6, The Resident Monitor Clock Routine, describes the Resident Monitor's time functions. There are three places in DOS which start or try to update the clock -- (1) the first-time initialization after manual Bootstrap loads and restarts, (2) the Resident Monitor Initialization, and (3) the CAL Handler. The following .SCOM registers contain timing information:
Figure 2-4
.OVRLA, .EXIT and CTRL Q

2-9
1. Set up clock so that it keeps running, but does not interrupt (ticks positive)
2. Clear all flags
3. Turn off PI and API
4. Restore cell 4 to transfer to Error Diagnostic Routine
5. Set up proper addressing (Bank or Page), according to .SCOM+4, bit 7

N
VT ON and also CTRL X

Y
Do CTRL X restart

1. Update the clock, and allow it to interrupt
2. Clear TTY Busy Switch (Clear all flags ensures no I/O to TTY)
3. Turn API on or off, depending on contents of register 6 (The System Loader loads register 6 according to .SCOM+4, bit 0)
4. Turn on PI

Exit to Proper location

Figure 2-5
Resident Monitor Initialization
Entry from PI or API

Allow clock to tick positive, so it will not interrupt for an hour

N
 timers

N
 .TIMER

in effect

Y
 Increment the interval once

Y
 Interval done

N

Set up the exit from this routine to go to the .TIMER address in .SCom+61, as if it were a JMS instruction. Set high-order bits of return address with interrupt information

increment .SCom+51

increment .SCom+56

TIMEOUT

in effect

Y
 second up

Y

1. increment .SCom+50
2. format in hhmms
3. increment .SCom+34

subtract one from register 7, the clock register

Y

register seven negative

Y

restore pre-interrupt conditions

exit

Note: The Clock Routine will use PI if API is busy, or down.

Figure 2-6

The Resident Monitor Clock Routine for non-UC15 Systems

2-11
Note: The Clock Routine will use PI if API is busy, or down.

Figure 2-7
The Resident Monitor Clock Routine
for UC15 systems (RK05 based or
RF15/REG2 based with UC15 option)
Note: The Clock Routine will use PI if API is busy, or down.

Figure 2-7 (Cont'd.)

The Resident Monitor Clock Routine
for UC15 systems (RA5 based or
RF15/RF92 with UC15 option)
.SCOM+5$  Time of day, in hhmmss (six bits each)
.SCOM+51  Elapsed time, in ticks
.SCOM+56  Time limit, in seconds (zero, if no limit)
.SCOM+6$  Time left for .TIMER interrupt (zero, if
          .TIMER not in effect)
.SCOM+61  Address of .TIMER user interrupt routine
.SCOM+73  Number of ticks left in the next second
.SCOM+74  Line frequency, in ticks per second

2.6.1 Clock Operation

The Nonresident Monitor's TIME command changes or .senses .SCOM+5$.
.SCOM+51 is not used by any system program. The clock handler simply
increments it upon each clock tick. User programs may deposit a known
quantity into .SCOM+51, in order to time events. The Non-resident
Monitor deposits the argument for a TIMES command into .SCOM+56. If
.SCOM+56 is nonzero, the Resident Monitor will issue an ISZ .SCOM+56
command each second, until it reaches zero. At such a time, the Resi-
dent Monitor will perform a .EXIT. MICLOG, LOGIN, and LOGOUT clear
.SCOM+56.

2.6.2 .TIMER

.TIMER allows users to schedule routines for a specified time from
"now". These routines may return to the interrupted code, if the
programmer desires. .TIMER users should take care that the time-
dependent code follows certain rules:

a. When a programmer does not wish to reset the .TIMER mechan-
ism, but wishes to return to the interrupted program, his
code should look like this:

```
C  $0  /C+1 REACHED VIA JMS
DAC  SAVEAC  /MUST NOT USE NON-REENTRANT CODE
.    .  /POSSIBLY USED BY THE INTERRUPTED
.    .  /PROGRAM. (INCLUDES THE CAL IN-
.    .  /STRUCTION)
LAC  C  /RESTORE THE LINK
RAL
LAC  SAVEAC  /RESTORE THE AC
XIT  JMP*  C
```

2-14
b. When the programmer does wish to reset the .TIMER mechanism, and return to the interrupted code, his routine should look like this:

```
.SCOM=100
CLON=700044
CLOF=700044
INTRVL=-100
/TICKS
.
.
Ø
DAC  SAVEAC
.
.
LAC  ADDRES
DAC* (.SCOM+61
CLOF
/TURN THE CLOCK OFF TO ENSURE NO
/REENTRANCE BEFORE .TIMER RESET AND
/RETURN
LAC  INTRVL
/DISOPED INTERVAL IN TWO'S COMPLEMENT
DAC* (.SCOM+60
LAC  C
/RESTORE THE LINK
RAL
LAC  SAVEAC
/RESTORE THE AC
CLON
/TURN THE CLOCK BACK ON (AFTER NEXT
/INSTRUCTION)
JMP*  C
```

c. When a programmer does not wish to return to the interrupted program, he need not save the AC, and he may use the CAL instruction. He should beware of using I/O buffers that may still be modified by a handler's interrupt section. In many cases, a .INIT to an active .DAT slot will terminate I/O. Teleprinter I/O should be terminated by the following:

```
XCT* (.SCOM+35
```

The user should program a delay of at least 110 milliseconds after such an instruction before he attempts teleprinter I/O.

Note: The interrupt routine will run at the level of the interrupted code, with the same addressing mode and memory protect status. Thus, no debreak and restore is required.
2.7 THE RESIDENT MONITOR PATCH AREA

There are two types of patch area:

1. That allocated by using PATCH
2. That allocated when answering the Patch Area question in system generation

Patch area one is the place for permanent changes to the Resident Monitor. It is always refreshed when the System Loader comes into core. Patch area two is only refreshed on manual Bootstrap loads and restarts. The second area would be appropriate for communication between successive programs loaded by the System Loader. This area should be used because the System Loader refreshes all of core, except the Bootstrap, .SCOM, the CTRL X buffer, and the patch area two.

The combined size is limited by the current assembly at $3\text{8}\text{8}\text{8}$ for RP$\text{8}2$ and RF$\text{5}$ systems, and for RK$\text{0}$ system. Both areas can be initialized, using PATCH. The important dividing line between area one and area two is register 1$\text{0}1$ (.SCOM+1) of RESMON. The way to allocate more space in part one is to increase the value of register 1$\text{0}1$. The way to change the area in part two is to use DOSGEN. The second part will start at the address in register 1$\text{0}1$. The upper bound of the second area will be the sum of the contents of register 1$\text{0}1$, and the number specified to DOSGEN.

2.8 CONTROL CHARACTERS

CTRL C, P, R, S, and T are all special characters that interrupt the current program and transfer control. The Resident Monitor ignores CTRL R except after IOPS 4 and any call to the Expanded Error Processor. CTRL S always transfers control to the address in .SCOM+6. In the case of core-image system programs and EXECUTE, a CTRL S will transfer to register zero, and result in an IOPS 3. The Linking Loader places the starting address of the first load module into .SCOM+6.

A .INIT macro to the teleprinter handler will change the address of either CTRL C, P or T. The Resident Monitor is always initialized to
perform a .EXIT after CTRL C, and ignore CTRL P and T. DDT uses
CTRL T, and CTRL P is ordinarily used by programs for restarts.
MACRO-15 expands .INIT to change the CTRL P address. If the programmer
expands .INIT without the aid of the assembler, a 10 in bits zero and
one of LOC+2 will change the address of CTRL T. A 01 in those bits
will change the address of CTRL C. It should be obvious that special
care should be taken with CTRL C. In addition, modifications to the
CTRL T address should not be made when debugging with DDT. There are
cases, however, when such modifications are desirable. In particular,
all zeroes in LOC+2 (2-17) will cause the teleprinter handler to
ignore CTRL C, P, or T. This address might be used when sensitive
code is being executed, as in DOSGEN. The following .INIT expansion
will cause the Resident Monitor to ignore CTRL C:

CAL-2&777
I
200000

2.9 TASK CONTROL BLOCKS (only for UC15 system - RK85 based or
RF15/RP82 based with UC15 option)

In the UNICHLANNEL-15 system communication between the PDP-15 and the
PDP-11 is through blocks of information called Task Control Blocks.
These blocks are resident in the common/shared memory space (memory
that can be addressed both by the PDP-15 and the PDP-11). The TCB
contains all the information necessary (like the addressed task
code, the method of indicating the completion of a request, memory
address, word count, operation etc.) for the PIREX system to process
that request (refer to UC15 Software Manual, DEC-15-XUCMA-A-D for more
details).

Handlers for the devices on the UNIBUS communicate with the driver
tasks running under PIREX through TCB's. In order to permit these
handlers to be loaded anywhere in core (not restricting them to the
common/shared memory), these TCB's are part of the Resident Monitor.
.SCOM+180 points to a table in the Resident Monitor which contains
the start address of the various TCB's present in the system as
indicated below:

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE (octal words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRTCBS</td>
<td>81</td>
</tr>
<tr>
<td>LPTCBF</td>
<td>117</td>
</tr>
<tr>
<td>CDTCBF</td>
<td>65</td>
</tr>
<tr>
<td>PLTCBF</td>
<td>117</td>
</tr>
<tr>
<td>S1TCB</td>
<td>24</td>
</tr>
<tr>
<td>S2TCBF</td>
<td>120</td>
</tr>
<tr>
<td>S3TCBF</td>
<td>170</td>
</tr>
</tbody>
</table>
The following are available for use when the handlers are in operation.

RKTCB - TCB for the RKØ5 disk cartridge handler
LPTCBF - TCB and buffer space for the LP11/LS11 line printer handler
CDTCBF - TCB and buffer space for the CR11 card reader handler
PLTCBF - TCB and buffer space for the XY11 plotter handler

The following are available for new devices or for other purposes as desired by the user:

S1TCB - Spare TCB space
S2TCBF, S3TCBF - Spare TCB and buffer space

The TCB and buffer space starts below this table and .SCOM+1 points to the end of the TCB and buffer space. Users can add entries to this table for TCB's or TCB and buffers by suitable updating .SCOM+1 and the table.
CHAPTER 3

THE NONRESIDENT MONITOR

3.1 INTRODUCTION

The System Loader brings the Nonresident Monitor into core after a hardware readin, a manual restart, a CTRL C, or a .EXIT. The RCOM Table, SGNBLK, SYSBLK and COMBLK are always coresident with the Nonresident Monitor. This gives the Nonresident Monitor access to all important system parameters.

The Nonresident Monitor announces its presence by typing DOS-15 Vnn on the teleprinter. It remains in core until the operator requests another system program, or until the operator's command implies a refreshed configuration of the Resident Monitor is necessary.

The Nonresident Monitor's actions are limited to (1) decoding commands, (2) manipulating or examining bits and registers in .SCOM, .DAT, .UPDT, SYSBLK, COMBLK, and SGNBLK, and (3) calling the System Loader, when necessary. The Nonresident Monitor has only one entry, which starts an initialization section. Figure 3-1, Nonresident Monitor Initialization, describes that logic. Every time the System Loader brings in the Nonresident Monitor, it passes control to the initialization section. After initialization, and after all commands that do not require the System Loader, the Nonresident Monitor types a $ and awaits an input line, terminated by a Carriage RETURN or an ALT MODE. It then examines the first six characters (or those up to the first blank) and tries to find an entry in the Nonresident Monitor's Command Table. If a match is found, control passes to the appropriate routine, and thence to the next command or the System Loader. If the typed command does not correspond to an entry in the command table, the Nonresident Monitor temporarily assumes the operator wishes a new core-image system program and checks COMBLK for a corresponding entry. If there is no corresponding entry in COMBLK, the Nonresident Monitor will type an error message and await the next command. If COMBLK contains a matching entry, the Nonresident Monitor composes a .OVRLA and passes control to the System Loader via that .OVRLA.

3-1
START

1. Bank bit initialize pointers to SYSBLK, COMBLK and SGNBLK
2. Determine the number of positive .DAT slots
3. Save the contents of .DAT-12, in case the user desires LP ON
   (restore before leaving Nonresident Monitor)
4. Save contents of .SCOM+7 -- Nonresident Monitor will use
   .SCOM+7 for address of LPA or TTA.
5. Change all .UPDT entries that equal BNK or PAG to SYS
6. Compute addresses of .DAT-2, +1, +5 and +6
7. Compute address of beginning of I/O Device Table in SGNBLK

(Y) Returning from a Nonresident Monitor .EXIT

(Y) KEEP ON

(Y) Restore .UPDT and .DAT to SGEN values

(Y) Initialize .DAT-2 and .DAT-3

(Y) Returning from a Nonresident Monitor .EXIT

(Y) In BOSS Mode

(Y) Type out Nonresident Monitor's name

(Y) Has a date been entered

Request a date

Figure 3-1
Nonresident Monitor Initialization
3-2
Clear bit 1 of .SCOM+42 (Nonresident Monitor .EXIT flag)

Y
BOSS Mode

N
Type "S"

N
Need to load BOSS

Y
Load BOSS

Read command string

(Continue to Command Decoder)

Figure 3-1 (Cont.)
Nonresident Monitor Initialization
3.2 COMMANDS TO THE NONRESIDENT MONITOR

This paragraph discusses legal commands listed in the Nonresident Monitor's Command Table. Table 3-1, Effects and Exits for Nonresident Monitor Commands, describes all commands that do not request a new program.

There are five entries in the Command Table that load relocatable system programs. They are DDT, EXECUTE, GLOAD and LOAD. The Nonresident Monitor treats these commands separately, because SYSBLK does not list them. All information necessary for loading these programs resides in the Nonresident Monitor itself.

3.3 CONSIDERATIONS FOR ADDITIONS TO THE NONRESIDENT MONITOR

Programmers should not attempt to add commands to the Nonresident Monitor unless they have access to a copy of the source code. The source code may be purchased from Digital Equipment Corporation, 146 Main Street, Maynard, Massachusetts, under one of the order numbers listed in the footnote. They should then use the EDITOR program to put in the indicated changes, and reassemble.

New additions to the Nonresident Monitor require the following actions:

1. Update the Nonresident Monitor's Command Table.
   The Command Table is in two parts:
   a) The .SIXBT names of the commands
   b) The corresponding transfer vector

2. Write the code for the command.

3. Consider the kind of exit the command will take:
   a) Commands that end with a request for a new command should end with JMP KLCOM
   b) Commands that re-configure the Nonresident Monitor should end with JMP NRME1.
### Table 3-1

**Effects and Exits**  
for Nonresident Monitor Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MODIFIER</th>
<th>ACTION TAKEN</th>
<th>EXIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>ON</td>
<td>Set bit G of .SCOM+4.</td>
<td>EXIT</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>Clear bit G of .SCOM+4.</td>
<td>EXIT</td>
</tr>
<tr>
<td>ASSGN</td>
<td>handler</td>
<td>Check whether handler is available.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If yes, load .DAT slot with proper handler code. (The proper loader will load the handler, and insert its starting address into the .DAT slot.)</td>
<td>Command</td>
</tr>
<tr>
<td></td>
<td>UIC</td>
<td>Load proper slot via a .USER</td>
<td>Command</td>
</tr>
<tr>
<td>BANK</td>
<td>ON</td>
<td>Set bit 11 of .SCOM+4.</td>
<td>Command</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>Clear bit 11 of .SCOM+4.</td>
<td>Command</td>
</tr>
<tr>
<td>BATCH</td>
<td>PR</td>
<td>Set bit G and clear bit 2 in location 1777 of the Bootstrap's bank. If bit 2 of .SCOM+33 is set (i.e., if VT is ON) and bit 17 of .SCOM+33 is set (i.e., CTRL X is set for VT), set bit 1 of .SCOM+33 in order to tell the Resident Monitor Initialization to start up CTRL X.</td>
<td>EXIT</td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>Set bits G and 2 of location 1777 of the Bootstrap's bank, and set bit 1 of .SCOM+33 as with BATCH PR.</td>
<td>EXIT</td>
</tr>
<tr>
<td>BUFFS</td>
<td>number</td>
<td>Put number indicated into .SCOM+26, and set Nonresident Monitor Initialization to leave .SCOM+26 alone.</td>
<td>Command</td>
</tr>
<tr>
<td>CHANNEL</td>
<td>7</td>
<td>Clear bit 13 of .SCOM+4.</td>
<td>Command</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Set bit 13 of .SCOM+4</td>
<td>Command</td>
</tr>
<tr>
<td>DATE</td>
<td>date</td>
<td>Enter date into .SCOM+47.</td>
<td>Command</td>
</tr>
<tr>
<td></td>
<td>no date</td>
<td>Print date from .SCOM+47.</td>
<td>Command</td>
</tr>
</tbody>
</table>

---

1This table assumes error-free input
Table 3-1 (cont.)

Effects and Exits
for Nonresident Monitor Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MODIFIER</th>
<th>ACTION TAKEN</th>
<th>EXIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td></td>
<td>See Section 3.4.</td>
<td></td>
</tr>
<tr>
<td>GETP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HALF</td>
<td>ON</td>
<td>Set bit 0 of .SCOM+33. Clear bits 0 and 1 of .SCOM+33.</td>
<td>.EXIT</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td></td>
<td>.EXIT</td>
</tr>
<tr>
<td>HALT</td>
<td></td>
<td>If not in BOSS-15 mode, put a HLT instruction (instead of a JMP) into the</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>exit from non-IOPS 4 errors to .MED. If in BOSS mode, do nothing.</td>
<td></td>
</tr>
<tr>
<td>INSTRUCT</td>
<td>none</td>
<td>Print INSALL SRC ) By loading Print INSERR SRC ) INSTRC BIN</td>
<td>.EXIT Command</td>
</tr>
<tr>
<td></td>
<td>ERRORS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEEP</td>
<td>ON</td>
<td>Set bit 16 of .SCOM+42. Clear bit 16 of .SCOM+42. Initialize to SGEN default</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>values all entries in .DAT and .UPPT, except change SCR default values to</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>current UIC.</td>
<td></td>
</tr>
<tr>
<td>LOG</td>
<td></td>
<td>Output five spaces after Carriage RETURNS. After ALT MODE, go to next</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>command.</td>
<td>(after ALT MODE)</td>
</tr>
<tr>
<td>LOGIN</td>
<td>uic</td>
<td>Make specified UIC current (.SCOM+41). *Then set up .UPTD entries; set .DAT</td>
<td>.EXIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>entries and system parameters (.SCOM+4, 20, 26, and 33) to SYSGEN default</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>values; clear .SCOM+42 and 56.</td>
<td></td>
</tr>
<tr>
<td>LOGOUT</td>
<td></td>
<td>Set current UIC to SCR. Then same as LOGIN (above) from *.</td>
<td>.EXIT</td>
</tr>
<tr>
<td>LOGW</td>
<td></td>
<td>For BOSS-15, print message. In all cases, after a Carriage RETURN, output</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>five spaces. After ALT MODE, type four bells '+P, and await CTRL P. After</td>
<td>(after ALT MODE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTRL P, go to next command.</td>
<td></td>
</tr>
</tbody>
</table>
Table 3-1 (cont.)
Effects and Exits
for Nonresident Monitor Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MODIFIER</th>
<th>ACTION TAKEN</th>
<th>EXIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>ON</td>
<td>Set bit 3 of .SCOM+42. Clear bit 3 of .SCOM+42.</td>
<td>.EXIT .EXIT</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MICLOG</td>
<td>mic</td>
<td>Check mic with SGNBLK. If correct set bit 0 of .SCOM+42 and make 'SYS' the</td>
<td>.EXIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>current UIC. Then same as LOGIN (above) from * (except .SCOM+42 not cleared).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If incorrect, ignore command.</td>
<td></td>
</tr>
<tr>
<td>PAGE</td>
<td>ON</td>
<td>Clear bit 11 of .SCOM+4.</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>Set bit 11 of .SCOM+4.</td>
<td></td>
</tr>
<tr>
<td>PROTECT</td>
<td>n</td>
<td>If n is between 0 and 7, inclusive, enter it into .SCOM+54.</td>
<td>Next Command</td>
</tr>
<tr>
<td>PUT</td>
<td></td>
<td>See Section 3.4.</td>
<td></td>
</tr>
<tr>
<td>QDUMP</td>
<td></td>
<td>Enter MANSAV, the address of the manual CTRL Q, into the exit from non-IOPS 4</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>errors to .MED.</td>
<td></td>
</tr>
<tr>
<td>REQUEST</td>
<td>none</td>
<td>Print the current assignments for .DAT and .UFDT.</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td>USER</td>
<td>Print the current assignments for all positive .DAT and .UFDT slots.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>prog</td>
<td>Print required .DAT and .UFDT slots, and the assignments and use for each.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SCOM</td>
<td></td>
<td>Print the information for the current system.</td>
<td>Next Command</td>
</tr>
<tr>
<td>TIME</td>
<td>time</td>
<td>Enter time into .SCOM+50.</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>Print time from .SCOM+50.</td>
<td></td>
</tr>
<tr>
<td>VT</td>
<td>ON</td>
<td>Set bit 2 of .SCOM+33. Clear bits 1, 2, and 17 of .SCOM+33.</td>
<td>.EXIT</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>Execute STDP.</td>
<td></td>
</tr>
<tr>
<td>X4K</td>
<td>ON</td>
<td>Enter 404040 into .SCOM+20.</td>
<td>Next Command</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>Deposit zero into .SCOM+20.</td>
<td></td>
</tr>
<tr>
<td>33TTY</td>
<td>ON</td>
<td>Clear bit 2 of .SCOM+4.</td>
<td>.EXIT</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>Set bit 2 of .SCOM+4.</td>
<td></td>
</tr>
<tr>
<td>LA38</td>
<td>ON</td>
<td>Set bit 2 of .SCOM+20 and clear bit 2 in .SCOM+4.</td>
<td>.EXIT</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td>Clear bit 2 of .SCOM+20 and set bit 2 in .SCOM+4.</td>
<td></td>
</tr>
</tbody>
</table>

3-7
4. After assembly, the programmer must call PATCH, in order to make his relocatable binary program absolute. Commands to PATCH should be as follows:

> DOS15

> READR 16077 DOSNRM BIN

16077 indicates the highest location the new monitor can occupy. (SYSBLK begins at 16100.) DOSNRM BIN happens to be the file name used by program development. The programmer may, of course, substitute his own file name. More information may be found in the PATCH manual -- DEC-15-UPATA-A-D.

3.4 QFILE

QFILE is a system program that allows users to (1) store core images in named files, and (2) retrieve such core images for examination via DUMP (or possibly for a slow, core-swapping capability). QFILE implements the following Resident Monitor system macros and Nonresident Monitor commands:

.GET, GET, GETP, GETS, GETT, .PUT and PUT

Users can not obtain QFILE by typing its name to the Nonresident Monitor. The Resident Monitor will load QFILE as part of its response to the commands and macros listed above.

PUT creates a file that contains the data in the CTRL QAREA; .PUT creates a file from the current core image. GET, GETP, GETS, GETT and .GET all overlay core with the contents of the QAREA or file. (The different commands specify different startup locations.) In addition to the above capabilities, the Resident Monitor provides the capability of overlaying core with the contents of the CTRL Q area. The following instructions show how to use that routine:

UNITNO=4ggggg
.SCOM=1gg

LAC START /STARTING ADDRESS AFTER THE CTRL Q

XOR UNITNO /GET UNIT NUMBER IN HIGH-ORDER THREE BITS
JMP* (.SCOM+64) /ADDRESS OF CTRL Q GET ROUTINE

3-8
Figure 3-2, QFILE, and Implementation of GET and PUT Logic, shows
the information flow associated with QFILE. QFILE uses the follow-
ing registers:

.SCOM+7,10 & 11 .SIXBT Filename and Extension
.SCOM+65

Command parameters, packed as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>Device unit number</td>
</tr>
<tr>
<td>8</td>
<td>NRM PUT, when set</td>
</tr>
<tr>
<td>9</td>
<td>PUT logic, when set</td>
</tr>
<tr>
<td>15-17</td>
<td>Function Code</td>
</tr>
</tbody>
</table>

.SCOM+66-71 .DAT-14

CTRL Q Area parameters

File must be on the device assigned
to this .DAT slot.

NOTE

All GET and .GET operations change all
of core, except registers 0 through 4
of bank zero.
Note: This chart assumes error free input.

Figure 3-2
QFILE, and Implementation of GET and PUT Logic
CHAPTER 4

THE SYSTEM LOADER AND THE LINKING LOADER

The System Loader is the third major part of the DOS-15 Monitor. The other two are the Resident and Nonresident parts. The Resident and Nonresident Monitors communicate with the System Loader by manipulating certain .SCOM registers. When commands to either part imply a new configuration is needed, that part sets up the appropriate .SCOM registers, and passes control to the System Bootstrap via the Monitor TRAN routine. The Bootstrap then loads the System Loader into high core, and gives it control.

The System Loader examines the .SCOM registers, and loads a fresh copy of the Resident Monitor, including any features that the user wishes to be resident, such as the CTRL X feature. It will also load the desired system program and all handlers required by the new configuration. In addition, it will allocate all required buffers. The Nonresident Monitor is treated like any other core-image system program.

The System Loader never loads user programs. It only loads core-image system programs, the INSTRUCT command processing program, the Linking Loader and Execute. The latter two load user programs.

The System Loader uses two device handlers to interface with the disk: the System Bootstrap, and the System Loader Disk Handler (DKL./DPL./RKL.). .XXL. arrives in core along with SYSBLK, COMBLK and SGNBLK, as well as the loader itself. The Bootstrap loads core-image programs only. The .XXL. takes care of relocatable programs and any handlers loaded by the System Loader. Those include all handlers for core-image system programs, the Linking Loader's own handlers, and any needed by the Execute file. The Linking Loader loads some handlers needed by user programs it links.

There are two parts to the System Loader: the System Loader Interface, and the System Loader Proper (.SYSLD). Figure 4-1 describes the System Loader Interface. Figure 4-2 describes the System Loader Proper, and Figure 4-3 describes the Linking Loader.
Bootstrap Loads

Turn on the clock
Initialize
- .SCOM+0 First free register below the Bootstrap
- .SCOM+4 SGEN default
- .SCOM+20 Bit zero set, if extra 4K; rest zero
- .SCOM+33 VT & HALF, as per SGEN
- .SCOM+74 Line frequency
-Move to highest bank

Normal Initialization

1. Zero .SCOM+36, to indicate no entries in the Mass Storage Busy Table
2. Move Resident Monitor into lower core
3. Set up: Jump to Skip Chain
   CAL* error
   Legal CAL jump
4. Turn API on or off, depending on bit φ of .SCOM+4 (set=on)
5. Bank bit initialize Resident Monitor to talk to the Bootstrap, and load .SYSLD into the proper bank upon a subsequent .EXIT or .OVRLA.
6. Initialize the Bootstrap with the proper IOPS 4 address for disk not ready
7. Calculate the Skip Chain from SGNBLK
8. Set all API channel registers to point to IOPS 3 (with the exception of the clock interrupt) and all software levels to point to IOPS 3φ
9. Put transfer vector to .DAT slots into .SCOM+23
10. Put number of positive .DAT slots into .SCOM+24
11. Put pointer to .UFDT+φ into .SCOM+25

Was last program

Nonresident Monitor

N

Y

.data image of .DAT
and .UFDT in from
block 37 of the sys-
tem device, unit φ

.data image of .DAT
and .UFDT out to
block 37 of the sys-
tem device, unit φ

1. Zero .DAT-7 (i.e., not yet set up)
2. Set up .DAT-2 and .DAT-3 for TTA.
3. Update .SCOM+1 and +2 to point just above the Skip Chain, .DAT and .UFDT

Figure 4-1
System Loader Initialization

4-2
1. Put number of system device's "A" handler (DRA. or DPA. or RKA.) into .SCOM+57
2. Set up tabbing for current teleprinter
3. Set .SCOM+2Ø to initial state (as in first time initialization)
4. Set up for CTRL Ø -- ignore Ø-dumps if RF system and QAREA too small, or nonexistent
5. Set up for IOPS errors upon the following interrupts:
   Nonexistent Memory (IOPS 31)
   Memory Protect Violation (IOPS32)
   Memory Parity Error (IOPS33)
   Power Fail Not Set Up (IOPS34)

   N
   Bit Ø
   of the
   Bootstrap
   =1
   Y -- Non-BOSS Batch

   Set up for the
   proper input
   device (CD or PR)

   N
   Loading
   Nonresident
   Monitor

   Y
   Has
   a $JOB card
   been seen

   Y
   N
   Set switch to ig-
   nore input until $JOB

Next Page
Figure 4-1 (Cont.)
System Loader Initialization
4-3
1. Set up CTRL C to clear the Batch Switch (bit 1 of 17777 of the Bootstrap)
2. Set up CTRL T to abort current job, and start the Batch Monitor looking for the next $JOB line
3. Relocate proper batch handler (PR or CD) to low core
4. Put handler entry point into .DAT-2
5. Set IOPS errors to abort job — effectively a CTRL T
6. Set up all batch device .DAT slots to refer to the handler currently in core. That is, only one batch input device is allowed at any one time
7. Clear $JOB read switch (bit 1 of Bootstrap 17777)
8. Perform .INIT to .DAT-2
From Preceding Page

- **Loading a relocatable program**
  - Y
  - N
    - loading EXECUTE
      - Y
        - 1. Allocate the number of buffers indicated by .SCOM+26
        - 2. Set up File Buffers Transfer Vector Table pointer, in .SCOM+30
        - N
          - Loading INSTR
            - Y
              - 1. Store one of the following codes into .SCOM+6:
                - LOAD 100000
                - GLOAD 300000
                - DDT 400000
                - DDTNS 500000
              - N
                - Set .SCOM+5 to 14
            - N
              - N

- B

1. Allocate number of buffers indicated by .SCOM+26
2. Set up File Buffers Transfer Vector Table in .SCOM+30
3. Set .SCOM+6 = 0
4. Put 13 into .SCOM+5

(Loading a Core-Image Program)
1. Find entry in SYSBLK and COMBLK
2. Build Overlay Table from information in COMBLK, and set .SCOM+31 to first word in the table
3. Store the number of overlays in the overlay processor of the Resident Monitor

Will EXECUTE use system device

- N
  - Y
    - Tell .SYSLD by setting .SCOM+11 to XCS (avoids two handlers in core for same device)

Next Page

Figure 4-1 (Cont.)
System Loader Initialization

4-5
1. Store the list of active .DAT slots derived from COMBLK in the System Loader command area, just below the Bootstrap, and delimit the list with a zero.
2. If the Nonresident Monitor was not the last program, restore .SCOM+26 to default.
3. Allocate space for, and set up .SCOM+30 to point to the File Buffers Transfer Vector Table.

From Preceding Page

Loading PIP or is PIP among the overlays
N
Y
Build the Device Table

1. Zero .SCOM+6
2. Put 1 into .SCOM+5

Y
Loading Nonresident Monitor
N
Set bit 3 of .SCOM+4
Clear bit 3 of .SCOM+4

1. Move the RCOM Table to position below the Bootstrap
2. Build the IOC Table

Figure 4-1 (Cont.)
System Loader Initialization
Entry from initialization

1. Set up for Page or Bank Mode
2. Set up .DAT-7 for the System Loader disk handler (DKA., RKA. or DPA.)
3. Clear free core, and initialize bank bits in pointers to the Bootstrap
4. Make a Mass Storage Busy Table consisting of one entry

Loading EXECUTE

Y

Loading INSTRC

N

Set up to read INSTRC BIN

Y

Set up CTRL P address

N

+SCom+11

=XBET 'XCS'

N

1. Change XCS to XCT
2. Allow reading of EXECUTE file by the System Loader Handler

Clear memory bank pointers of banks that do not exist

N

Loading a core-image program

Y

Load handlers into extra 4k, if it exists

N

Under BOSS-15 control

Y

Put System Device's code into .DAT+@, to allow subsequent insertion into .DAT-7

Next Page
Figure 4-2
The System Loader
Figure 4-2 (Cont.)

The System Loader

1. Set up Mass Storage Busy
   Table Entries for all
   active .DAT slots
2. Set .SCOMM to first free
   location in core—often
   becomes first location of
   EXECUTE

Note: Subroutine IOPROS accepts
.DAT slots as input. If the in-
dicated .DAT slot contains zero,
the slot is unassigned, and IOPROS
returns. If not zero, IOPROS checks
whether the desired handler has al-
ready been loaded. If the handler
is in core, IOPROS loads the .DAT
slot with the handler's starting ad-
dress and returns. If the handler has
not been loaded, the handler code is made
an unresolved .GLOBAL, to be satisfied
by the loop that follows immediately.
From Preceding Page

Load and relocate EXECUTE or the Linking Loader, and place starting address into .SCOM+5

\[ Y \]
Loading EXECUTE

\[ N \]

1. Set up Mass Storage Busy table with one entry per active .DAT slot
2. Move the IOC table from the System Loader's area (just beneath the Bootstrap) to the Linking Loader's area

Set .SCOM+2 and +3 to delimit free core

BOSCK1

Exit to address in .SCOM+5

\[ B \]

1. Allocate all necessary buffers
2. If the system has an extra 4K, put the first free address beneath the handlers into .SCOM+28
3. Update first free location in core shown in .SCOM+2 -- .OVRLA updates the first free address beneath the Bootstrap, .SCOM+3

BOSCK1

Exit via .OVRLA

Note: Subroutine BOSCK1 does the following, if loading a program under BOSS-15:
(1) .USER to .UFDT-7, (2) .SEEK to .DAT-7 for PRCFIL PRC.

Figure 4-2 (Cont.)
The System Loader

4-9
1. Clear all of core above the loader, including the extra 4K, if present, and excluding the Bootstrap
2. Initialize the Load Table with the first free address in every bank or page
3. Indicate all core below the address in .SCOM+2 as not free
4. Compute transfer vectors to .DAT-1, -3, -4, -5, and -7, and a pointer to .UFDT-1
5. Save the contents of .UFDT-1

N

DDT

Y

Load DDT and set the symbol flag, if not DDTNS

Type appropriate name, and await command string

CTRL P

Y

or no back arrow

1. Check for P, G and C switches
2. Translate all file names after left arrow into .SIXBT, pad with blanks, and store in symbol table
3. After ALT MODE, load to end-of-file each file on .DAT-4, and put starting address of the first file (i.e., not DDT) into .SCOM+6

Load from Paper Tape

N

Y

After every end of tape, type ↑P and await CTRL P -- continue until number of tapes equals the number of commas, plus one

Next Page

Figure 4-3
The Linking Loader

4-10
NOTE:
During the library searches diagrammed on this page, the Linking Loader tests for more unresolved .GLOBL's after each resolution. Whenever there are no more unresolved .GLOBL's, the Linking Loader halts its library searches, and goes directly to the COMMON area allocation code (next to the last box on this page). Thus, the libraries are never searched more than is necessary.
Figure 4-3 (Cont.)

The Linking Loader
4.1 MANUAL BOOTSTRAP LOADS AND RESTARTS

Manual Bootstrap loads and restarts bring blocks 0-36 of the system device into the lowest bank. These blocks contain the Resident Monitor, the System Loader Interface Routine, and SYSBLK, COMBLK and SGNBLK. Figure 4-4 illustrates the core load after manual Bootstrap loads and restarts. The Interface sets up .SCOM+Ø, 4, 2Ø, 27, 33, 54 and 74 from SGNBLK values determined at system generation time, and then transfers the whole core image of the Interface to the Bootstrap's bank. (DOS requires 16K, because this bank must be different from bank Ø.) At all other times, the Bootstrap loads the System Loader into its own bank. This preserves the image of .SCOM, part two of the Resident Monitor patch area, and the CTRL X buffer. For UC15 systems (RK05 based and RP15/RP02 based with UC15 option) this has no effect on the core layout in the PDP-11 local memory. PIREX is reinitialized meaning all permanent tasks are put in a 'WAIT' state while temporary tasks are put in a 'EXIT' state and all pending PDP-15 requests are flushed (refer to UC15 Software Manual, DEC-15-XUCMA-A-D for more information).

4.2 LOADING SYSTEM PROGRAMS

The System Loader Interface Routine gets control in the highest bank, either by a transfer from the lowest bank, or by load from the Bootstrap. After setting up for the System Loader Proper (.SYSLD), according to the program to be loaded and the settings of certain SCOM registers, the Interface Routine brings it in as a complete overlay. Figure 4-5 illustrates the core configuration of the Interface when it is in the highest bank. (The addresses provided are for a 16K system.) The System Loader loads handlers from the lowest part of free core up, with the exception that the extra 4K is filled first, if it exists. Core image system programs are usually loaded just beneath the Bootstrap (always in the highest bank). Such core images must be wholly within the top bank of core, and above register 17 of that bank. Figure 4-6 illustrates the core maps for system programs.

Whenever the Linking Loader is loaded (LOAD, GLOAD, DDT, and DDTNS), the System Loader loads all handlers for .DAT slots -1, -4, and -5, and then loads the Linking Loader itself. (DDT is loaded by the Linking loader.) Wherever INSTRC ($INSTRUCT command processing program) is loaded, the handler assigned to .DAT slot-12 is also loaded. Figure 4-7 illustrates the core maps for the Linking Loader and INSTRC.
Figure 4-4
Bootstrap Load

Figure 4-5
Standard Interface Load

Figure 4-6
System Program Load

* All system programs except MACCI, which is always loaded in Bank 1 regardless of the size (16K, 24K, 32K) of the system.
For EXECUTE, the System Loader loads EXECUTE's handler, and reads the EXECUTE file, in order to determine the active .DAT slots. The System Loader then loads all the handlers required, and sets up the .DAT slots. Figure 4-8 illustrates core maps for EXECUTE.

BOSS-15 Mode operation requires the system "A" handler be assigned to .DAT-7. This requires a sleight of hand on the part of the System Loader, which needs the "L" handler on .DAT-7. It therefore loads the "A" handler as if it were assigned to .DAT+∅, and transfers the set up .DAT slot ∅ contents to .DAT-7 before transferring control to the program being loaded. .DAT+∅ is then restored to its original status.

4.3 TABLES AND INFORMATION BLOCKS USED AND BUILT BY LOADERS

The System Loader uses SYSLK, COMBLK, SGNBLK, block 37 of the system device, .SCOM, the RCOM Table, the IOC Table, the Device Table, the Mass Storage Busy Table, the File Buffers Transfer Vector Table, the Overlay Table, .DAT, .UFDT and three bits in the Bootstrap. Tables 4-1, 4-2, and 4-3 describe how the Loaders use these blocks and tables.

4.4 .DAT SLOT MANIPULATION BY THE SYSTEM LOADER

The System Loader maintains the .DAT slot device handler assignments as they were the last time the Nonresident Monitor was in core. The Loader saves the .DAT and .UFDT on the system device whenever the Nonresident Monitor was the last program in core. Thereafter, the Loader refreshes .DAT and .UFDT from the image on the disk. If KEEP is off, the Nonresident Monitor's initialization routine restores the .DAT and .UFDT to default values.

When loading core-image system programs, the System Loader determines the active .DAT slots by examining COMBLK. When loading EXECUT, the System Loader sets up .DAT-4, and any active slots indicated by the Execute file itself. When loading the Linking Loader, the System Loader sets up .DAT-1, -4, and -5 and also .DAT-12, if loading INSTRC. The Linking Loader will set up other active .DAT slots according to the .IODEV commands in the assembly of the program units being loaded.

Both the System Loader and the Linking Loader set up .DAT slots in this manner: (In the following procedure, "loader" refers to either one.)
<table>
<thead>
<tr>
<th>NAME</th>
<th>USE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSBLK</td>
<td>The System Loader obtains Monitor .TRAN parameters from SYSBLK when it builds</td>
<td>165000 of .SYSLD's bank</td>
</tr>
<tr>
<td>COMBLK</td>
<td>Indicates number of buffers required, the active .DAT slots, and the names</td>
<td>171000 down, in .SYSLD's bank</td>
</tr>
<tr>
<td>SGNBLK</td>
<td>Default settings for .SCOM registers, number of words per buffer, size of Resident Monitor's patch area (part two), Skip Chain, .DAT and .UPDT default contents, and handler information.</td>
<td>161000 of .SYSLD's bank</td>
</tr>
<tr>
<td>Block 37 of the System Device</td>
<td>Image of .DAT and .UPDT, when last program was loaded (excluding the Nonresident Monitor).</td>
<td></td>
</tr>
<tr>
<td>.SCOM Table</td>
<td>See Table 4-II.</td>
<td>1000 of 1st bank</td>
</tr>
<tr>
<td>RCOM Table</td>
<td>Moved for use by the Nonresident Monitor.</td>
<td>175000 of the highest bank</td>
</tr>
<tr>
<td>IOC Table</td>
<td>Built by Interface Routine for .SYSLD itself.</td>
<td>Just beneath the System Loader</td>
</tr>
<tr>
<td>Device Table</td>
<td>Built by Interface Routine if loading PIP, or if PIP is among the overlays listed in COMBLK</td>
<td>Just above .SCOM+1</td>
</tr>
<tr>
<td>Mass Storage Busy Table</td>
<td>Built by the System Loader itself.</td>
<td>Pointed to by .SCOM+62</td>
</tr>
<tr>
<td>File Buffers Transfer Vector Table</td>
<td>Allocated by the Interface Routine, and initialized by it for non-core Image programs. System Loader proper initializes for core-image programs.</td>
<td>Pointed to by .SCOM+30</td>
</tr>
<tr>
<td>Overlay Table</td>
<td>Built by the Interface Routine</td>
<td>Pointed to by .SCOM+31</td>
</tr>
<tr>
<td>.DAT and .UPDT</td>
<td>Image stored and restored from block 37 of the System Device. The System Loader loads all handlers for core-image programs and EXECUTE Files, and sets up the appropriate .DAT slots. The System Loader also loads handlers assigned to .DAT-1, -4, and -5 when loading the Linking Loader, and .DAT-7 and +6 for BOSS-15.</td>
<td>Pointed to by .SCOM+23 and .SCOM+25</td>
</tr>
<tr>
<td>BOOTSTRAP</td>
<td>Bits $</td>
<td>, 1, and 2 of location 17777 in the Bootstrap's bank used for Batch (non-BOSS) information.</td>
</tr>
<tr>
<td>.SCOM+</td>
<td>Description of Use by the System Loader</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Set in first-time initialization routine. Used to locate the System Loader Command Area, which is just below the Bootstrap.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>System Loader Interface routine updates this indication of the first free register above the Resident Monitor each time it moves a piece down to low core.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>The Interface and .SYSLD itself continually update this indication of the first free location as they move code and build tables.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Updated as with .SCOM+2. Last free location in core.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>First Time Initialization routine sets this register according to a SGNBLK parameter. Refer to Table 4-III.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Interface Routine stores code of program to be loaded into .SCOM+5. .SYSLD uses .SCOM+5 for starting address when loading EXECUT or LOAD. The .OVRLA routine loads .SCOM+5 with starting address of the Monitor Recovery Routine. The Bootstrap transfers to the address in .SCOM+5 after all its operations.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Interface Routine stores codes for DDT, DDTNS, LOAD and GLOAD into .SCOM+6. For other programs, the Interface Routine zeroes .SCOM+6.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.SYSLD saves contents of .DAT-1 in .SCOM+7, when loading the Linking Loader. When loading EXECUT, .SCOM+7 contains the first three characters of the Execute file's name. Contains .DAT-12 when loading Nonresident Monitor.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>.SYSLD saves contents of .DAT-4 in .COM+10, when loading the Linking Loader. When loading EXECUT, .SCOM+10 contains the second three characters of the Execute file's name.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>.SYSLD saves contents of .DAT-5 in .SCOM+11, when loading the Linking Loader. When loading EXECUT, .SCOM+11 contains the extension of the Execute file's name. (The Interface routine sets .SCOM+11 to XCS, telling .SYSLD that EXECUT will be using the system device. .SYSLD then restores .SCOM+11 to XCT.)</td>
<td></td>
</tr>
<tr>
<td>12-15</td>
<td>The Interface routine initializes these transfer vectors for API software levels to point to SERR, an error routine that will produce an IOPS30.</td>
<td></td>
</tr>
<tr>
<td>16, 17</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>.SCOM+</td>
<td>Description of Use by the System Loader</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Bit zero set in first time initialization, if system contains an extra 4K. If the system does contain an extra 4K, the System Loader will load handlers in that page -- from the bottom up -- when loading a core-image program. Whenever there is an extra 4K, the System Loader will update bits 3-17 with the address of the first free cell in the extra 4K. If bit 2 is set, change Resident Monitor so that it will tab for a KSR33, and send filler characters when outputting carriage returns.</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>The Interface Routine refreshes this pointer to .DAT.</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>The Interface Routine refreshes this indication of the number of positive .DAT slots.</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>The Interface Routine refreshes this pointer to .UFDT+Ø.</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>When the Nonresident Monitor was the last program, the System Loader allocates the number of buffers indicated by the contents of .SCOM+26. If the Nonresident Monitor was not the last program, the System Loader restores .SCOM+26 to the default value if program to be loaded is core image. Otherwise, untouched.</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>The first time initialization routine sets this indication of the number of words per file buffer.</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>The Initialization Routine loads this pointer to the File Buffer Transfer Vector Table.</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>When loading a core-image program, the Interface Routine loads .SCOM+31 with the pointer to the Overlay Table, or with zero, if there is none.</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>See Interface Routine table, to determine how that routine reacts to the bits in .SCOM+33.</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>34, 35</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>System Loader loads with the number of active .DAT slots assigned to the system device.</td>
<td></td>
</tr>
<tr>
<td>37-42</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>43, 44</td>
<td>Contains name of the program to be loaded.</td>
<td></td>
</tr>
<tr>
<td>45-56</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>System Loader loads with the number of entries in the Mass Storage Busy Table.</td>
<td></td>
</tr>
<tr>
<td>60, 61</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>System Loader loads with the address of the first entry in the Mass Storage Busy Table.</td>
<td></td>
</tr>
<tr>
<td>63-</td>
<td>Unaffected.</td>
<td></td>
</tr>
<tr>
<td>Bit</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>If set, place &quot;API ON&quot; constant into $000000. If clear, place &quot;API OFF&quot; constant in same register.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>If set, change the Resident Monitor so it will tab with the KSR 35/37 tabbing mechanism.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Loader will set this bit, if loading the Nonresident Monitor; clear it otherwise.</td>
<td></td>
</tr>
<tr>
<td>4-6</td>
<td>Ignored.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Loader sets this bit if bit 11 is cleared, and loading the Linking Loader or Execute. Otherwise clear.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Sets or clears, after comparing current core size (known by location of Bootstrap, and status of bit $0, .SCOM+2$) with SGNBLK parameter. Also, modifies Resident Monitor to give IOPS77 after attempts to use CTRL Q.</td>
<td></td>
</tr>
<tr>
<td>9, 10</td>
<td>Ignored</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Indicates whether to clear or set bit 7, when loading Linking Loader or Execute.</td>
<td></td>
</tr>
<tr>
<td>12-17</td>
<td>Ignored</td>
<td></td>
</tr>
</tbody>
</table>
1. Each .DAT slot will contain a handler number -- either the system default, or one inserted via an ASSIGN command to the Nonresident Monitor. This handler number is the relative location of the handler name in the IOC Table, which the Interface Routine builds. (The IOC Table contains handler names in Radix 50.)

2. For each active .DAT slot, the loader uses the handler number in that slot to find the name in the IOC table, and converts the name to .SIXBT.

3. If the handler is already in core, the loader simply inserts the starting address of the handler into the .DAT slot.

4. If the handler is not yet in core, the loader does a .SEEK to <IOS> UIC for the handler, reads it into core, relocates it, and places the starting address of the handler into the .DAT slot.

The System Loader always sets up .DAT-2 and -3. (It reserves .DAT-7 for its own use.) When not in non-BOSS Batch Mode, -2 is assigned to TTA. In non-BOSS Batch Mode, the batch input device goes to -2. If loading the Nonresident Monitor and bit three of .SCOM+42 is set, the System Loader will set up .DAT-12 for the LPA, if it is in the system, or else for TTA. If in BOSS mode, the Nonresident Monitor assigns LPA to .DAT+6, and the System Loader assigns .DAT-7 to the system device "A" handler. The System Loader then ensures that both handlers are in core. The Resident BOSS set up routine subsequently routes all .DAT slots connected to TTA. to Resident BOSS.

4.5 BUFFER ALLOCATION BY THE SYSTEM LOADER

The System Loader allocates space for buffers equal to the contents of .SCOM+26 times the contents of .SCOM+27. The first time initialization routine sets .SCOM+27 to the standard number of locations per buffer. Before the Nonresident Monitor does an .OVRLA to a software system program, it checks whether a BUFFS command has been issued. If so, it leaves .SCOM+26 as is. If not, it uses the default number of buffers for that program, as shown in SYSBLK.
CHAPTER 5

SYSTEM INFORMATION BLOCKS AND TABLES

5.1 CORE-RESIDENT NON-REFRESHED REGISTERS

The .SCOM table, the Bootstrap and the resident Patch Area are the only registers not refreshed by the System Loader. Table 5-1 describes the .SCOM Table.

5.2 DISK-RESIDENT UNCHANGING BLOCKS: SYSBLK, COMBLK AND SGNBLK

SYSBLK, COMBLK and SGNBLK occupy blocks 34, 35, and 36 (octal) on the system device (unit zero). SYSBLK and COMBLK (blocks 34 and 35) contain the parameters for loading all core image system programs. SGNBLK contains all the other information needed to run DOS. All three arrive in core along with the Resident Monitor and the System Loader Interface, and start at location 16100 of the highest bank. The Nonresident Monitor and System Loader use them, and DOSGEN and PATCH modify them, when necessary.

5.2.1 SYSBLK

SYSBLK contains the parameters required for implementation of .OVRLA to any system program, or any of the system program overlays.

The order of entries in SYSBLK is unimportant, except for the first three permanent entries: RESMON, SYSLD, and QAREA. The first word of SYSBLK contains the block address (the unrellocated address) of the first free word after itself. Figure 5-1 describes SYSBLK.

5.2.2 COMBLK

COMBLK contains information the System Loader and the Nonresident Monitor need to remember about the current core-image system programs. The last location in COMBLK (that is, location 377 of block 35) contains the block address of the first entry in COMBLK. The remainder of COMBLK consists of variable-length entries associated with the system programs. The Nonresident Monitor searches COMBLK when it finds no match for a typed command in its own Command Table. Figure 5-1 illustrates the organization of COMBLK. The System Generator adds 5-1
<table>
<thead>
<tr>
<th>REGISTER</th>
<th>BIT</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td></td>
<td>First register below the Bootstrap (set by the System Loader Interface)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>First register above the Resident Monitor (set by the System Loader Interface)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Lowest free register available for storage (set by the System Loader or the Linking Loader)</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Highest free register available for storage (set by the System Loader, the Linking Loader or DDT)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Initialized from SGNBLK values by the &quot;first time&quot; section of the System Loader Interface Routine, and by the LOGIN, LOGOUT and MICLOG logic of the Nonresident Monitor; modified by the Nonresident Monitor, unless otherwise indicated.</td>
</tr>
<tr>
<td>$ = 1</td>
<td></td>
<td>API is available.</td>
</tr>
<tr>
<td>1 = 1</td>
<td></td>
<td>EAE is available (always set)</td>
</tr>
<tr>
<td>2 = 1</td>
<td></td>
<td>Teleprinter is Model 35 or 37</td>
</tr>
<tr>
<td>3 = 1</td>
<td></td>
<td>Nonresident Monitor is in core</td>
</tr>
<tr>
<td>4,5</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>6 = 1</td>
<td></td>
<td>9-Channel Magnetic Tape System</td>
</tr>
<tr>
<td>7 = 1</td>
<td></td>
<td>Page Mode Operation</td>
</tr>
<tr>
<td>8 = 1</td>
<td></td>
<td>QAREA inadequate for current core size (set by the System Loader Interface Routine)</td>
</tr>
<tr>
<td>9 = 1</td>
<td></td>
<td>DOS disk file structure (always set)</td>
</tr>
<tr>
<td>1$ = 1</td>
<td></td>
<td>RB$9 disk is system device.</td>
</tr>
<tr>
<td>11 = 1</td>
<td></td>
<td>Bank Mode System</td>
</tr>
<tr>
<td>12,13</td>
<td></td>
<td>Line Printer Line Size:</td>
</tr>
<tr>
<td>$$</td>
<td></td>
<td>No Line Printer</td>
</tr>
<tr>
<td>$1</td>
<td></td>
<td>8$ Characters</td>
</tr>
<tr>
<td>1$</td>
<td></td>
<td>12$ Characters</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>132 Characters</td>
</tr>
<tr>
<td>14 = 1</td>
<td></td>
<td>Background/Foreground System (always clear)</td>
</tr>
<tr>
<td>15-17</td>
<td></td>
<td>Drum size (ignored -- DOS does not support drum)</td>
</tr>
<tr>
<td>REGISTER</td>
<td>BIT</td>
<td>MEANING</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Core Image System Program Starting address.</td>
</tr>
<tr>
<td>6</td>
<td>0 = 1</td>
<td>DDT in core.</td>
</tr>
<tr>
<td></td>
<td>1 = 1</td>
<td>GLOAD</td>
</tr>
<tr>
<td></td>
<td>2 = 1</td>
<td>DDTNS</td>
</tr>
<tr>
<td>7-11</td>
<td></td>
<td>User program starting address. When using the Linking Loader, .SCOM+7, 10 and 11 contain the handler numbers for handlers needed by the Linking Loader in .DAT-1,-4, and -5 respectively. When the Linking Loader passes control to DDT, .SCOM+10 contains the size of the Busy Table (for later clearing by DDT) and .SCOM+11 has the starting address of the symbol table. When using EXECUTE, 7-11 contain the .SIXBT representation of the name and extension of the Execute File. When using QFILE (for implementation of .GET, .PUT and the Nonresident Monitor GET and PUT commands), 7-11 contain the .SIXBT representation of the name and extension of the core image file.</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>API Level 4 service routine entry point</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>API Level 5</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>API Level 6</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>API Level 7</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Program Counter on Keyboard Interrupts.</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>AC on Keyboard Interrupts.</td>
</tr>
<tr>
<td>20</td>
<td>0 = 1</td>
<td>20K or 28K system.</td>
</tr>
<tr>
<td></td>
<td>1 = 1</td>
<td>UC15 system - RK5/5 based or RF15/RP2 based</td>
</tr>
<tr>
<td></td>
<td>2 = 1</td>
<td>30 CPS LA30 console device.</td>
</tr>
<tr>
<td>3-17</td>
<td></td>
<td>First free address in top page.</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>Magtape Status Register.</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>Reserved for Magtape Handler.</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>Pointer to .DAT+0.</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>Number of positive .DAT slots.</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>Pointer to .UPDT+0.</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>Number of buffers.</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>Number of words per buffer.</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>Pointer to Buffer Transfer Vector Table.</td>
</tr>
<tr>
<td>REGISTERS</td>
<td>BIT</td>
<td>MEANING</td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>Pointer to first entry in the Overlay Table (zero, if none).</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>Bad block number on IOPS 2Ø and 72.</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>CTRL X status register.</td>
</tr>
<tr>
<td></td>
<td>Ø = 1</td>
<td>HALF ON</td>
</tr>
<tr>
<td></td>
<td>1 = 1</td>
<td>Display Buffer already set up.</td>
</tr>
<tr>
<td></td>
<td>2 = 1</td>
<td>VT ON</td>
</tr>
</tbody>
</table>
|           | 17 = 1 | If VT ON, display mode is on.
<p>| 34        |     | If in BOSS mode, elapsed time in seconds. |
| 35        |     | Instruction to clear TT Busy Switch. |
| 36        |     | Number of Entries in the Mass Storage Busy Table. |
| 37        |     | Entry point for Expanded Error Processor. |
| 4Ø        |     | JMP to Expanded Error Processor. |
| 41        |     | The logged-in UIC. |
| 42        |     | Bit Register. |
|           | Ø = 1 | MICLOG successful. |
|           | 1 = 1 | .EXIT from Nonresident Monitor. |
|           | 2 = 1 | .OVRLA from Nonresident Monitor. |
|           | 3 = 1 | LP ON -- LPA to .DAT-12 when loading Nonresident Monitor. |
|           | 4 = 1 | Dump core on calls to .MED (except IOPS 4). |
|           | 5 = 1 | Halt on calls to .MED (except IOPS 4). |
| 6-13      |     | Unused. |
| 14        |     | Set up .DAT+6 (use by Batch mode) |
| 15        |     | Load System Device Handler into .DAT-7. |
| 16        |     | KEEP ON. |
| 17        |     | Batch Mode. |
| 43,44     |     | .SIXBT Representation of the name of the core image system program to be loaded (if any). |
| 45,46     |     | .SIXBT Representation of the name of the Nonresident Monitor |</p>
<table>
<thead>
<tr>
<th>REGISTERS</th>
<th>BIT</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td></td>
<td>Date [mm (bits 0-5), dd (6-11), yy (12-17, module 1970 decimal)]</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>Time [hh (bits 0-5), mm (6-11), ss (12-17)]</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td>Elapsed time, in ticks.</td>
</tr>
<tr>
<td>52</td>
<td></td>
<td>BOSS Bit Register</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>BOSS15 Mode.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Control Card Read by user, 5/7 ASCII image saved in first block of NRBoss.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Resident BOSS reached &quot;EOF&quot; on run time file (RTF).</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>User exceeded time estimate.</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>I/O CAL to go to TTY (.DAT-3 and positive .DAT slots).</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Terminal IOPS error by user.</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>QDUMP to be given to user on IOPS errors.</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Operator abort (Control T).</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Job active.</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Exit from BOSS15 Mode.</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>User tried to do a .PUT. Core will be dumped and a listing given on LP.</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>User tried to do a .GET.</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Not defined.</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>Not defined.</td>
</tr>
<tr>
<td></td>
<td>14-16</td>
<td>.SYSLD error number.</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Job abort.</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td>Reserved for CTRL X code.</td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>Default Protection Code.</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>Entry to Monitor TRAN routine.</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>Two's complement of time limit, in seconds (zero, if no limit).</td>
</tr>
<tr>
<td>57</td>
<td></td>
<td>System Device Code, for use by the Linking Loader.</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>Number of ticks until clock interrupt specified in last .TIMER (zero, if .TIMER not in use).</td>
</tr>
<tr>
<td>REGISTER</td>
<td>BIT</td>
<td>MEANING</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>61</td>
<td></td>
<td>.TIMER address.</td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>Address of the first word in the Mass Storage Busy Table.</td>
</tr>
<tr>
<td>63</td>
<td></td>
<td>Number of words per Mass Storage Busy Table Entry.</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>JMP to CTRL Q GET routine.</td>
</tr>
<tr>
<td>65</td>
<td></td>
<td>QFILE Communication Register.</td>
</tr>
<tr>
<td>66</td>
<td></td>
<td>First Block of the CTRL Q Area.</td>
</tr>
<tr>
<td>67</td>
<td></td>
<td>Starting Address minus one of the CTRL Q Area.</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>Two's complement of number of word in Qdump</td>
</tr>
<tr>
<td>71</td>
<td></td>
<td>Starting Address after DUMP or GET.</td>
</tr>
<tr>
<td>72</td>
<td></td>
<td>Starting Address after CTRL Q.</td>
</tr>
<tr>
<td>73</td>
<td></td>
<td>Two's complement of the number of ticks left in the next second.</td>
</tr>
<tr>
<td>74</td>
<td></td>
<td>Two's complement of the line frequency.</td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>Number of RTF Lines (for BOSS Mode).</td>
</tr>
<tr>
<td>76*</td>
<td>0=1</td>
<td>SPOOLER ENABLED</td>
</tr>
<tr>
<td></td>
<td>1=1</td>
<td>SPOOLER RUNNING</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>MAC11 Communication Bit</td>
</tr>
<tr>
<td>5-17</td>
<td></td>
<td>SPOOLER area disk start block number</td>
</tr>
<tr>
<td>77*</td>
<td>6-17</td>
<td>SPOOLER area size (in blocks)</td>
</tr>
<tr>
<td>100*</td>
<td>6-17</td>
<td>pointer to TCB and Buffer Table</td>
</tr>
<tr>
<td>101-105</td>
<td></td>
<td>unused</td>
</tr>
</tbody>
</table>

*For RK05 system only. Unused for RP02 and RF15 systems.
<table>
<thead>
<tr>
<th>Word #</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g$</td>
<td>0000nn</td>
<td>Pointer to first free word after SYSBLK (There is one set of seven words/core image program.)</td>
</tr>
<tr>
<td>7N+1</td>
<td>.SIXBT</td>
<td>Name of System Program or overlay</td>
</tr>
<tr>
<td>7N+2</td>
<td>.SIXBT</td>
<td>Number of first block on system device occupied by this program or overlay.</td>
</tr>
<tr>
<td>7N+3</td>
<td>nnnnnn</td>
<td>Number of blocks occupied by this program or overlay.</td>
</tr>
<tr>
<td>7N+4</td>
<td>0000nn</td>
<td>Thirteen-bit first address for this program or overlay</td>
</tr>
<tr>
<td>7N+5</td>
<td>address</td>
<td>Program size</td>
</tr>
<tr>
<td>7N+6</td>
<td>nnnnnn</td>
<td>Thirteen-bit starting address for this program or overlay</td>
</tr>
<tr>
<td>7N+7</td>
<td>address</td>
<td></td>
</tr>
</tbody>
</table>

(free area)

<table>
<thead>
<tr>
<th>C</th>
<th>COMBLK</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>(000100</td>
<td>Number of words in this entry (in this case, 10)</td>
</tr>
<tr>
<td>501</td>
<td>.SIXBT</td>
<td>Name of this system program (left-justified and zero-filled)</td>
</tr>
<tr>
<td>502</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>503</td>
<td>r</td>
<td>Name of an overlay (left-justified and zero-filled) -- overlays are optional</td>
</tr>
<tr>
<td>504</td>
<td>.SIXBT</td>
<td>Number of buffers required by this system program (Bits 0-6 = 0 means the end of any overlay names. This is why program and overlay names must be left-justified.)</td>
</tr>
<tr>
<td>505</td>
<td>o</td>
<td>Number of buffers required by this system program (Bits 0-6 = 0 means the end of any overlay names. This is why program and overlay names must be left-justified.)</td>
</tr>
<tr>
<td>506</td>
<td>.DAT&amp;777</td>
<td>Active .DAT slot</td>
</tr>
<tr>
<td>507</td>
<td>.DAT&amp;777</td>
<td>Active .DAT slot (Note: 777777 for a .DAT slot means all positive .DAT slots.)</td>
</tr>
<tr>
<td>510</td>
<td>(000005</td>
<td>Number of words for this entry (in this case, 5)</td>
</tr>
<tr>
<td>511</td>
<td>p</td>
<td>Name of this system program</td>
</tr>
<tr>
<td>512</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>513</td>
<td>o</td>
<td>Number of buffers required by this program (Note that this program has no overlays.)</td>
</tr>
<tr>
<td>514</td>
<td>2</td>
<td>.DAT slot for this program</td>
</tr>
<tr>
<td>777</td>
<td>000500</td>
<td>Pointer to first word in COMBLK (equals count from first word in SYSBLK). The two contiguous blocks on the system device that hold SYSBLK and COMBLK are treated by the system as one large block. In this case, COMBLK happens to start at location 500 of the two blocks combined.</td>
</tr>
</tbody>
</table>

Figure 5-1

SYSBLK and COMBLK

5-7
names of core-image system programs by making them the new first entry. In this way, SYSBLK and COMBLK build toward the center.

5.2.3 SGNBLK

SGNBLK (block 36 on the system device) contains all the system parameters not directly associated with core-image system programs. The bulk of SGNBLK is concerned with I/O (.DAT slots, .UFDT slots, Skip Chain Order, Handlers, and skip IOT codes and mnemonics). The first few registers hold such important system information as the system device, .SCOM+4 contents, and so on. The very first word in SGNBLK points to the block address of the first free word after SGNBLK. The next entry is an offset word indicating the total length (including itself) of the miscellaneous system parameter table to follow. This table includes the size of the .DAT and the size of the skip chain. The end of the handler and skip IOT table is the first free entry of the block.

The .DAT slot table corresponds to the legal range of .DAT slots, with the maximum negative set to 15, and the maximum positive set to a number not to exceed 77. The .DAT slots are in the form in which they appear when the Nonresident Monitor is in core. That is, the unit number is in bits 0-2, and the number of the handler right-justified in bits 3-17. The handler number for the first handler in the Device Handler-Skip IOT Table is zero, for the pseudo-handler NON, TTA. is one, and so on. The constant $00000$ indicates a fixed or illegal .DAT slot (such as -2, -3, and 0). DOSGEN will not modify such slots.

The .UFDT Table is in one-to-one correspondence with the .DAT slot Table. An entry of .SIXBT 'UIC' indicates that the logged in UIC is to be substituted for the name UIC in the table. An entry of .SIXBT 'SYS' indicates BK or PAG is to be substituted, in accordance with the current addressing mode. Otherwise, the contents of each location will be the .SIXBT representation of the corresponding .UFDT slot.

The Skip Chain Table lists the system skip IOT's in order. A negative skip (one that skips on "off", not "on") is represented in one's complement. Not all skips in the handler Skip IOT Table (described below) need to be included in the Skip Chain Table.
The Device Handler/Skip IOT Table contains all the handler names and skip IOT numbers and mnemonics for each I/O device identified to the system. Every such device has an entry in the table. A handler name must be exactly three characters in length, with the last character not an octal digit. The device code for a device is exactly two characters. The first two characters of each handler name for a device must be the device code. This fact is essential for understanding the format of a device entry, since the device code is never stored as such in an entry, but is inferred from the device handler name. The typical entry for a device is the following:

1. The first words of an entry contain the handler names for a device in .SIXBT. Each handler name is different, and the end of the list of handlers is determined by a word with zeros in bits 0-5 (the first character position).

2. The word that terminated the list of handler names contains the number of skip IOT's for the device. For each skip IOT, there are three words in the table: two for the skip mnemonic and one for the actual code.

The next device entry follows the last skip for the previous device. Handlers may be entered without any skips, but no devices may be entered without at least one handler name. Figure 5-2 illustrates the organization of SGNBLK. Appendix D of SGEN-DOS Utility Programs, DEC-15-USGNA-A-D, lists SGNBLK, SYSBLK and COMBLK, as they are supplied by Digital Equipment Corporation.

5.3 DISK-RESIDENT CHANGING BLOCKS

The System Loader uses block 37 of the system device to store an image of .DAT and .UPDT. Other disk-resident changing blocks are the storage Allocation Table and the Bad Allocation Table. These tables are described in Chapter 6.

5.4 TEMPORARY TABLES BUILT FROM DISK-RESIDENT TABLES

5.4.1 The Overlay Table

The System Loader builds the Overlay Table from the entries in SYSBLK referenced by a core-image system program's entry in COMBLK. That is, the Overlay Table contains an entry for the system program itself, and one for each of its overlays. Figure 5-3 illustrates the format of an entry in the Overlay Table. The first entry in the Overlay Table is
<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td>Pointer to first free entry in SGNBLK</td>
</tr>
<tr>
<td>1</td>
<td>$\emptyset\emptyset\emptyset\emptyset$</td>
<td>Number of miscellaneous parameters</td>
</tr>
<tr>
<td>2</td>
<td>$\emptyset\emptyset\emptyset\emptyset$</td>
<td>Size of .DAT plus size of .UFDT = (number of positive .DAT slots + 16)$\emptyset$*2. (Initial value is 2$\emptyset$, positive .DAT slots.)</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset\emptyset\emptyset\emptyset$</td>
<td>Number of skips in Skip Chain</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset\emptyset\emptyset\emptyset$</td>
<td>System device code in .SIXBT</td>
</tr>
<tr>
<td>5</td>
<td>nnnnnn</td>
<td>Original contents of .SCOM+4</td>
</tr>
<tr>
<td>6</td>
<td>nnnnnn</td>
<td>Original contents of .SCOM+2$\emptyset$</td>
</tr>
<tr>
<td>7</td>
<td>nnnnnn</td>
<td>Number of words per buffer (.SCOM+27)</td>
</tr>
<tr>
<td>8</td>
<td>nnnnnn</td>
<td>Default number of buffers (.SCOM+26)</td>
</tr>
<tr>
<td>9</td>
<td>.SIXBT</td>
<td>Monitor Identification Code</td>
</tr>
<tr>
<td>10</td>
<td>nnnnnn</td>
<td>Information on VT and CTRL X (.SCOM+33)</td>
</tr>
<tr>
<td>11</td>
<td>.SIXBT</td>
<td>Default files protection code (.SCOM+54)</td>
</tr>
<tr>
<td>12</td>
<td>nnnnnn</td>
<td>Size of the Resident Monitor Patch Area</td>
</tr>
<tr>
<td>13</td>
<td>7777$\emptyset$</td>
<td>Minus the number of clock ticks in a second (-74 for 6$\emptyset$ hz, -62 for 5$\emptyset$ hz.)</td>
</tr>
<tr>
<td>14</td>
<td>$\emptyset\emptyset\emptyset\emptyset$</td>
<td>Device assignments for the .DAT (made by handler numbers). (Termination at 53 assumes 2$\emptyset$ positive slots.)</td>
</tr>
<tr>
<td>15</td>
<td>.SIXBT</td>
<td>UIC assignments for the .UPDT. (Termination at 111 assumes 2$\emptyset$ positive slots.)</td>
</tr>
<tr>
<td>16</td>
<td>nnnnnn</td>
<td>Skip Chain Table (Negative skips in one's complement.) (Termination at 137 assumes 26$\emptyset$ skips in chain.)</td>
</tr>
<tr>
<td>17</td>
<td>.SIXBT</td>
<td>The last part of the SGNBLK is the Device Handler-Skip IOT Table. Each entry starts with the .SIXBT representations of all handlers for a particular device. (First two characters equal device code, for all handlers.) Zeros in the first six bits of a word indicate the end of the handler names, and says that the rest of the word contains the number of skips for this entry's device. The skip IOT's follow immediately. As above, one's complement skips indicate negative skips. Note, however, the confusing fact that a one's complement of a skip IOT is a positive number. Thus, 7$\emptyset$nnn complemented is $\emptyset$7nnnn.</td>
</tr>
<tr>
<td>312</td>
<td>.SIXBT</td>
<td>SGNBLK ends at 312, in the DOS-15 RP82 and RP15 system distributed by Digital Equipment Corporation.</td>
</tr>
</tbody>
</table>

Figure 5-2

SGNBLK for RP82 and RP15 Systems

5-10
<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(\emptyset)nnn</td>
<td>Pointer to first free entry in SGNBLK</td>
</tr>
<tr>
<td>1</td>
<td>(\emptyset)nnn15</td>
<td>Number of miscellaneous parameters</td>
</tr>
<tr>
<td>2</td>
<td>(\emptyset)nnn</td>
<td>Size of .DAT plus size of .UFDT = (number of positive .DAT slots + 16) * 2. (Initial value is (2\emptyset) positive .DAT slots.)</td>
</tr>
<tr>
<td>3</td>
<td>(\emptyset)nnn</td>
<td>Number of skips in Skip Chain</td>
</tr>
<tr>
<td>4</td>
<td>2213(\emptyset)</td>
<td>System device code</td>
</tr>
<tr>
<td>5</td>
<td>nnnn(\emptyset)</td>
<td>Original contents of .SCOM+4</td>
</tr>
<tr>
<td>6</td>
<td>nnnn(\emptyset)</td>
<td>Original contents of .SCOM+2(\emptyset)</td>
</tr>
<tr>
<td>7</td>
<td>nnnn(\emptyset)</td>
<td>Number of words per buffer (.SCOM+27)</td>
</tr>
<tr>
<td>10</td>
<td>nnnn(\emptyset)</td>
<td>Default number of buffers (.SCOM+26)</td>
</tr>
<tr>
<td>11</td>
<td>(\emptyset)SIXBT</td>
<td>Monitor Identification Code</td>
</tr>
<tr>
<td>12</td>
<td>nnnn(\emptyset)</td>
<td>Information on VT and CTRL X (.SCOM+33)</td>
</tr>
<tr>
<td>13</td>
<td>(\emptyset)(\emptyset)(\emptyset)(\emptyset)nnn</td>
<td>Default files protection code (.SCOM+54)</td>
</tr>
<tr>
<td>14</td>
<td>(\emptyset)(\emptyset)nnn</td>
<td>Size of the Resident Monitor Patch Area</td>
</tr>
<tr>
<td>15</td>
<td>7777(\emptyset)</td>
<td>Minus the number of clock ticks in a second (-74 for 60 Hz, -62 for 5(\emptyset) Hz)</td>
</tr>
<tr>
<td>16</td>
<td>(\emptyset)nnn</td>
<td>Spooler area last block number.</td>
</tr>
<tr>
<td>17</td>
<td>(\emptyset)(\emptyset)nnn</td>
<td>Spooler area size.</td>
</tr>
<tr>
<td>2(\emptyset)</td>
<td>(\emptyset)(\emptyset)nnn</td>
<td>Device assignments for the .DAT (made by handler numbers). (Termination at 55 assumes (2\emptyset) positive slots.)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>55</td>
<td>(\emptyset)nnn</td>
<td>...</td>
</tr>
<tr>
<td>56</td>
<td>(\emptyset)SIXBT</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>113</td>
<td>(\emptyset)SIXBT</td>
<td>...</td>
</tr>
<tr>
<td>114</td>
<td>nnnnn</td>
<td>Skip Chain Table (Negative skips in one's complement.) (Termination at 145 assumes (32\emptyset) skips in chain.)</td>
</tr>
<tr>
<td>145</td>
<td>nnnnn</td>
<td>...</td>
</tr>
<tr>
<td>146</td>
<td>(\emptyset)SIXBT</td>
<td>The last part of the SGNBLK is the Device Handler-Skip IOT Table. Each entry starts with the (\emptyset)SIXBT representations of all handlers for a particular device. (First two characters equal device code, for all handlers.) Zeroes in the first six bits of a word indicates the end of the handler names, and says that the rest of the word contains the number of skips for this entry's device. The skip IOT's follow immediately. As above, one's complement skips indicate negative skips. Note, however, the confusing fact that a one's complement of a skip IOT is a positive number. Thus, 7(\emptyset)nnnn complemented is (87)nnnn.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>344</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

SGNBLK ends at 344, in the DOS-15 RK\(\emptyset\)5 system distributed by Digital Equipment Corporation.

Figure 5-3

SGNBLK for RK\(\emptyset\)5 Based System
pointed to by .SCOM+31. .SCOM+31 will contain zero, if there are no entries in the Overlay Table. This will occur during Linking Loader or EXECUTE loads.

.OVRLA is the only Monitor function that looks at the Overlay Table. If the .OVRLA processor finds a match to the .OVRLA argument in the Overlay Table, it uses the parameters listed in the table to bring it in via a Monitor TRAN. Note that this bypasses the System Loader, and does not change the handler load. Thus, the overlay must use only those .DAT slots required by the original program, the one listed in COMBLK.

If the .OVRLA processor does not find a match in the Overlay Table, it calls in the System Loader, which searches COMBLK for the requested program. This type of overlay request does not require that .DAT slot assignments be the same. On the other hand, the System Loader refreshes all of core except .SCOM, etc. Thus, communication between overlays is more difficult. The resident patch area, however, can be used for this purpose.

5.4.2 The Device Table

The Device Table is built by the System Loader interface whenever PIP is being loaded, or when PIP is listed in COMBLK among the overlays for a program. It is located just above the register pointed to by .SCOM+1, and has an entry for each positive .DAT slot. If a slot has an assigned device, the low-order twelve bits of the corresponding entry in the Device Table will contain the device's code, in .SIXBT. Bit 3 is set when the slot is busy. If no device is assigned to a slot, the corresponding entry in the Device Table will contain zero.

5.4.3 The Input/Output Communication (IOC) Table

The System Loader Interface builds the IOC Table and locates it just below the first register of the System Loader. It contains an entry for each handler in the system, in the order that they appear in SGNBLK. The entries themselves contain the handler name in Radix 50. The System Loader and the Linking Loader use the handler number supplied by the Nonresident Monitor to index down the IOC Table. They use the contents of the entry for a .SEEK to the IOS UIC.
5.4.4 The Device Assignment Table (.DAT)

The Device Assignment Table makes the association between logical and physical devices. The Monitor knows its location by the contents of .SCOM+23, which points to the entry zero in the Table. Specific slots are found by indexing on the contents of .SCOM+23. The number of negative slots is fixed at 15. The number of positive slots is specified by .SCOM+24, and may be any positive number less than 100. It is specified at system generation time.

The Nonresident Monitor places the handler number in the low order bits and the unit number in the high order bits. It derives the handler number from SGNBLK. As mentioned above, the System Loader and the Linking Loader subsequently use the IOC Table to determine the handler name. After either loader has loaded and relocated a handler, it places the handler's starting address in all .DAT slots that reference that handler. The unit number remains in the high-order three bits. Slots with no handler (NON) contain zero. Active .DAT slots are designated by COMBLK, for core-image system programs, and by .IODEV pseudo-ops for the Linking Loader and EXECUTE.

5.4.5 The User File Directory Table (.UFDT)

.UFDT+0 is offset from .DAT+0 (pointed to by .SCOM+23) by the sum of the positive and negative .DAT slots. Each .DAT slot has a corresponding .UFDT slot. UIC's in the .UFDT are packed in .SIXBT. The address of .UFDT+0 is stored in .SCOM+25.

5.4.6 The Skip Chain

Register 1 of Bank 0 contains a jump to the beginning of the Skip Chain. The Skip Chain is defined during System Generation, is located in SGNBLK, and is rebuilt every time the System Loader is called in. The System Generator Manual (DEC-15-USGNA-A-D) describes considerations for constructing the Skip Chain.

5.5 TEMPORARY TABLES BUILT FROM SCRATCH

5.5.1 File Buffer Transfer Vector Table

The System Loader allocates space for the buffer pool, and creates the File Buffer Transfer Vector Table. .SCOM+30 points to the first entry.
in the table, and the number of entries is specified by .SCOM+26. Each entry in the table contains the address of a buffer, or its one’s complement. Negative addresses indicate a busy buffer. Since references to buffers must be indirect anyway, buffers are allocated without regard to bank boundaries.

5.5.2 The RCOM Table

The Nonresident Monitor requires certain information about the Resident Monitor that does not warrant reserving additional .SCOM registers. The System Loader therefore puts this information into the RCOM table, whenever it is loading the Nonresident Monitor. The RCOM Table starts at register 17598 of the highest bank. QFILE uses the RCOM Table when processing a GET command.

5.5.3 The Mass Storage Busy Table

Entries in this table are allocated by the System Loader or the Linking Loader. The Mass Storage Busy Table is pointed to by .SCOM+62. .SCOM+63 contains the number of words per entry in the table, and .SCOM+36 contains the current number of entries. Generally speaking, there are as many entries in the Busy Table as there are active .DAT slots, although the disk handlers are the only ones that currently refer to the Busy Table.

The .INIT command to a disk handler establishes a Busy Table entry. The .CLOSE command (or the Rewind .VTAPE command) deletes the corresponding entry. Figure 5-4 illustrates a typical Busy Table Entry.

The first word of an active entry in the Busy Table contains the .DAT slot in bits 9-17. The disk handlers save information about the UFQ current for this .DAT slot in the Mass Storage Busy Table. They save information about the file current to the .DAT slot (if any) in the buffer pointed to by Word 1 of the Busy Table Entry. More information on the disk handlers and file structure is contained in Chapter 6.

5.6 RESERVED WORD LOCATIONS

Word locations 0 through 77 are dedicated systems locations and cannot be employed by the user. The contents of these locations are described in Table 5-4.
### Table 5-2
Overlay Table

<table>
<thead>
<tr>
<th>Word #</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>N,N+1</td>
<td>.SIXBT name of Overlay</td>
</tr>
<tr>
<td>N+2</td>
<td>First block number</td>
</tr>
<tr>
<td>N+3</td>
<td>First address, minus 1</td>
</tr>
<tr>
<td>N+4</td>
<td>Size, in two's complement</td>
</tr>
<tr>
<td>N+5</td>
<td>Fifteen-bit starting address</td>
</tr>
</tbody>
</table>

### Table 5-3
Mass Storage Busy Table Entry

<table>
<thead>
<tr>
<th>Word #</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Device Type_{5-2}, Unit Number_{3-5}, Write Check_{6-17}</td>
</tr>
<tr>
<td>N+1</td>
<td>Buffer Address, or $\emptyset$, if none allocated</td>
</tr>
<tr>
<td>N+2</td>
<td>Three-character UIC</td>
</tr>
<tr>
<td>N+3</td>
<td>First UFD block for this UIC</td>
</tr>
<tr>
<td>N+4</td>
<td>UFD Entry size for files in this UFD</td>
</tr>
</tbody>
</table>

### Table 5-4
Reserved Address Locations

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>Stores the contents of the extended PC, link, extend mode status, and memory protect status during a program interrupt</td>
</tr>
<tr>
<td>1</td>
<td>EEM (for PDP-9 compatibility)</td>
</tr>
<tr>
<td>2</td>
<td>JMP to Skip Chain</td>
</tr>
<tr>
<td>3</td>
<td>.MED, entry to Monitor Error Diagnostic routine</td>
</tr>
<tr>
<td>4</td>
<td>JMP to error handler</td>
</tr>
<tr>
<td>5</td>
<td>Stores system type (Bank or Page) indicator during Teletype interrupts</td>
</tr>
<tr>
<td>6</td>
<td>Used for API ON/OFF indicator</td>
</tr>
<tr>
<td>7</td>
<td>Stores real time clock count</td>
</tr>
<tr>
<td>10-17</td>
<td>Autoindex registers</td>
</tr>
<tr>
<td>20</td>
<td>Stores the contents of the extended PC, link, mode status, and memory protect status on a CAL instruction.</td>
</tr>
<tr>
<td>21</td>
<td>JMP to CAL handler</td>
</tr>
<tr>
<td>22-37</td>
<td>Seven pairs of word counter-current address registers for use with 3-cycle I/O device data channels.</td>
</tr>
<tr>
<td>40-77</td>
<td>Store unique entry instructions for each of $32_{10}$ automatic priority interrupt channels</td>
</tr>
</tbody>
</table>
5.7 BOOTSTRAP NON-BOSS BATCH BITS.

The high-order three bits of word 17777 in the Bootstrap are reserved for the Monitor, and have the following meanings:

Bit Ø  
1 = In non-BOSS Batch Mode
Ø = Not in non-BOSS Batch

Bit 1  
1 = $JOB ASCII line or card just read by batch device
Ø = Last line or card not $JOB

Bit 2  
1 = Batch device is card reader
2 = Batch device is paper tape reader
CHAPTER 6

FILE STRUCTURES

6.1 DECTAPE FILE ORGANIZATION

DECTape can be treated either as a directoried or non-directoried
device.

6.1.1 Non-Directoried DECTape

A DECTape is said to be non-directoried when it is treated as magnetic
tape by issuing the .MTAPE commands: REWIND or BACKSPACE, followed by
.READ or .WRITE. No directory of identifying information of any kind
is recorded on the tape. A block of data (255_10 word maximum), exactly
as presented by the user program, is transferred into the handler lef-
er and recorded at each .WRITE command. A .CLOSE terminates record-
ing with a software end-of-file record consisting of two words: 001004..776773

Because braking on DECTape allows for tape roll, staggered recording
of blocks is employed in DOS to avoid constant turnaround or time-
consuming back and forth motion of physically sequential block record-
ing. When recorded as a non-directoried DECTape, block 0 is the
first block recorded in the forward direction. Thereafter, every fir-
"n block is recorded until the end of the tape is reached, at which time
recording, also staggered, begins in the reverse direction. Five
passes over the tape are required to record all 1110_8 blocks.

6.1.2 Directoried DECTape

Just as a REWIND or BACKSPACE command declares a DECTape to be non-
directoried, a .SEEK or .ENTER implies that a DECTape is to be con-
sidered directoried. A directory listing of any such DECTape is
available via the (L)ist command in PIP. A fresh directory may be
recorded via the N or S switch in PIP.

The directory of all DECTapes except system tapes occupies all 400_8
words of block 100_8. It is divided into two sections: (1) a 40_8 word
Directory Bit Map and (2) a 340_8 word Directory Entry Section.
The Directory Bit Map defines block availability. One bit is allocated for each DECTape block (1100_8 bits = 40_8 words). When set to 1, the bit indicates that the DECTape block is occupied and may not be used to record new information.

The Directory Entry Section provides for a maximum of 56_10 files on a DECTape. Each file on the DECTape has a four-word entry. Each entry includes the three-word file name and extension, a pointer to the first DECTape block of the file, and a file active or present bit. Figure 6-1 illustrates the DECTape directory.

![Diagram of DECTape directory]

Figure 6-1

DECTape Directory

Additional file information is stored in blocks 71 through 77 of every directoryed DECTape. These are the File Bit Map Blocks. For each file in the directory, a 40_8-word File Bit Map is reserved in block 71 through 77. The bit maps are contiguous, and the Nth file uses the
Nth bit map. Each block is divided into eight File Bit Map Blocks. A File Bit Map specifies the blocks occupied by that particular file and provides a rapid, convenient method to perform DECTape storage retrieval for deleted or replaced files. Note that a file is never deleted until the new one of the same name is completely recorded on the .CLOSE of the new file. When a fresh directory is written on DECTape, blocks 71 through 100 are always indicated in the Directory Bit Map as occupied. Figure 6-2 illustrates DECTape file bit maps.

| Block 71x | Bit Map for File 0 |
| Block 72x | Bit Map for File 7 |
| Block 77x | Bit Map for File 15 |

Figure 6-2

DECTape File Bit Map Blocks

Staggered recording (at least every fifth block) is used on directoryed DECTapes, where the first block to be recorded is determined by examination of the Directory Bit Map for a free block. The first block is always recorded in the forward direction; thereafter, free blocks are chosen which are at least five beyond the last one recorded. The last word of each data block recorded contains a data link or pointer to the next block in the file. When turnaround is necessary, recording proceeds in the same manner in the opposite direction. When reading, turnaround is determined by examining the data link. If reading has been in the forward direction, and the data link is smaller than the last block read, turnaround is required. If reverse, a block number greater than the last block read implies turnaround.

A software end-of-file record (001005, 776773) terminates every file. The data link of the final block is 777777.

6-3
Data organization for each I/O medium is a function of the data modes. On directoried DECtape there are two forms in which data is recorded: (1) packed lines - IOPS ASCII, IOPS Binary, Image Alphanumeric, and Image Binary, and (2) dump mode data - Dump Mode.

In IOPS or Image Modes, each line (including header) is packed into the DECTape buffer. In IOPS Binary, a 2's complement checksum is computed and stored in the second word of the header. When a .WRITE which will exceed the remaining buffer capacity is encountered, the buffer is output, after which the new record is placed in the empty buffer. No record may exceed $255_{10}$ words, including header, because of the data link and even word requirement of the header word pair count. An end-of-file is recorded on a .CLOSE. It is packed in the same manner as any other line.

In Dump Mode, the word count is always taken from the I/O macro. If a word count is specified which is greater than $255_{10}$ (note that space for the data link must be allowed for again), the DECTape handler will transfer $255_{10}$ word increments into the DECTape buffer and from there to DECTape. If some number of words less than $255_{10}$ remain as the final element of the Dump Mode .WRITE, they will be stored in the DECTape buffer, which will then be filled on the next .WRITE, or with an EOF if the next command is .CLOSE. DECTape storage is thus optimized in Dump Mode since data is stored back-to-back. See Appendix A.

6.2 MAGNETIC TAPE

DOS provides for industry-compatible magnetic tape as either a directoried or non-directoried medium. The magnetic tape handlers communicate with a single TC-59D Tape Control Unit (TCU). Up to eight magnetic tape transports may be associated with one TCU; these may include any combination of transports TU-10A or B and TU-30A or B.

There are a number of major differences between magnetic tape and DECTape or Disk; these differences affect the operation of the device handlers. Magnetic tape is well suited for handling data records of variable length. Such records, however, must be treated in serial fashion. The physical position of any record may be defined only in relation to the preceding record. Three techniques available in I/O operations to block-addressable devices are not honored by the magnetic tape handlers:

6-4
6.2.1 Non-directoried Data Recording (MTF)

MTF is intended to satisfy the requirements of the FORTRAN programmer while still providing the assembly language programmer maximum freedom on the design of his tape format. MTF writes out a record to the tape each time the main program issues a .WRITE. The length of the record is always two times the word pair count in the header word pair. FLOPS records are always as long as the buffer size returned on a .INIT (up to 25610 words). MTF returns a standard buffer size of 3778, after a .INIT. The FORTRAN user may dynamically change this size, however, via the following instructions.

Example:

(FORTRAN STATEMENTS)                         (MACRO STATEMENTS)
.
.

.. TITLE SETMTB
.. GLOBL .DA, MTBSIZ, SETMTB

SETMTB g

CALL SETMTB (IBFSIZE)

JMS* .DA
JMP START

IBFSIZE g

START LAC* BUFSIZE (any buffer size)
DAC* MTBSIZ
JMP* SETMTB
.. END

6.2.2 Directoried Data Recording (MTA., MTC.)

The programmer can make the fullest possible use of those features peculiar to magnetic tape by using MTF. On the other hand, MTF does not offer the powerful file-manipulation facilities available in the system. Directoried I/O allows device independence, and extensive use of the storage medium with a minimum of effort.

MTA. and MTC. do not support non-directoried data recording.
Every block recorded by MTA. (with the exception of end-of-file markers, which are hardware-recorded) includes a two-word Block Control Pair and not more than $255_{10}$ words of data. The data will contain the records from one or more .WRITE's.

The Block Control Pair serves three functions: it specifies the character of the block (label, data, etc.), provides a word count for the block, and gives an 18-bit block checksum. The Block Control Pair has the following format:

**Word 1:**

Bits 0 through 5: Block Identifier (BI). This 6-bit byte specifies the block type. Values of BI may range from 0 to $77_{8}$. Current legal values of BI, for all user files, are as follows:

<table>
<thead>
<tr>
<th>BI Value</th>
<th>Block Type Specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>User-File Header Label</td>
</tr>
<tr>
<td>10</td>
<td>User-File Trailer Label</td>
</tr>
<tr>
<td>20</td>
<td>User-File Data Block</td>
</tr>
</tbody>
</table>

Bits 6 through 17: Block Word Count (BWC). This 12-bit byte holds the 2's complement of the total number of words in the block (including the Block Control Pair). Legal values of BWC range from $-3$ to $-40_{10}$.

**Word 2:**

Bits 0 through 16: Block Checksum. The Block Checksum is the full-word, unsigned, 2's complement sum of all the data words in the block and word 1 of the Block Control Pair.

![Figure 6-3](image)

**Figure 6-3**

Block Format, File-Structured Mode
One of the main file functions of MTA. and MTC. is that of identifying and locating referenced files. This is carried out by two means: first, names of files recorded are stored in a file directory at the beginning of the tape; and second, file names are contained in the file's header and trailer labels.

6.2.2.1 Magnetic Tape File Directory

The directory, a single-block file (and the only unlabeled file on any file-structured tape), consists of the first recorded data block on the tape. It is a $257_{10}$ word block with the following characteristics:

a. Block Control Pair (words 1 and 2)

Word 1

Block Identifier = $74_8$ = File Directory Data Block

Block Word Count = $-401_8 = 7377_8$.

Word 2:

Block Checksum: As described above.

b. Active File Count (Word 3, Bits 9 through 17) 9-bit one's complement count of the active file names present in the File Name Entry Section (described below).

c. Total File Count (Word 3, Bits 0 through 8) 9-bit one's complement count of all files recorded on the tape, including both active and inactive files, but not the file directory block.

d. File Accessibility Map (Words 4 through 17): The File Accessibility Map is an array of $252_{10}$ contiguous bits beginning at bit 0 of word 4 and ending as bit 17 of word 17. Each of the bits in the Accessibility Map refers to a single file recorded on tape. The bits are assigned relative to the zeroth file recorded; that is, bit 0 of word 4 refers to the first file recorded; bit 1, word 4, to the second file recorded; bit 0, word 6, to the $37_{10}$th file recorded; and so on, for a possible total of $252_{10}$ files physically present.

A file is only accessible for reading if its bit in the Accessibility Map is set to one. A file is made inaccessible for reading (corresponding bit = 0) by a .DELETE of the file, by a .CLOSE (output) of another file of the same name, or by a .CLEAR. A file is made accessible for reading (corresponding bit = 1) by a .CLOSE (output) of that file. Operations other than those specified above have no effect on the File Accessibility Map.
Figure 6-4a. Format of the File Directory Data Block, showing relationship of active and inactive files to file name entries and to Accessibility Map.

Figure 6-4b. Format of file-structured tape, showing directory block and data files.
e. File Name Entry Section (Words 18 through 257): The File Name Entry Section, beginning at word 18 of the directory block, includes successive 3-word file name entries for a possible maximum of 80 entries. Each accessible file on the tape has an entry in this section. Entries consist of the current name and extension of the referenced file in .SIXBT (left-adjusted and, if necessary, zero-filled).

The position of a file name entry relative to the beginning of the section reflects the position of its accessibility bit in the map. That bit, in turn, defines the position of the referenced file on tape with respect to other (active or inactive) files physically present. Only active file names appear in the entry section, and accessibility bits for all inactive files on the tape are always set to zero; accessibility bits for all active files are set to one.

To locate a file on the tape having a name that occupies the second entry group in the File Name Entry Section, the handler must (a) scan the Accessibility Map for the second appearance of a 1-bit, then (b) determine that bit's location relative to the start of the map. That location specifies the position of the referenced file relative to the beginning of the tape. The interaction of the File Name Entry Section and the Accessibility Map are shown in Figure 6-4.

6.2.2.2 User-File Labels

Associated with each file on tape is one label, the header label. It precedes the first data block of the file. Each label is 2710 words in length. Label format is shown in Figure 6-5.

![Figure 6-5](image)

User File Header Label Format
6.2.2.3 File-Names in Labels

The handler will supply the contents of the file-name fields (Word 3) in labels. These are used only for control purposes during the execution of .SEEK's. The name consists simply of the two's complement of the position of the recorded file's bit in the Accessibility Map; the "name" of the first file on tape is 777777, that of the third file is 777775, and so on. A unique name is thus provided for each file physically present on the tape. Since there may be a maximum of 252 files present, legal file-name values lie in the range 777777 to 777404.

6.2.3 Continuous Operation

Under certain circumstances, it is possible to perform successive I/O transfers without incurring the shut-down delay that normally takes place between blocks. The handler stacks transfer requests, and thus ensures continued tape motion, under the following conditions:

a. The I/O request must be received by the CAL handler before a previously-initiated I/O transfer has been completed.

b. The unit number must be identical to that of the previously initiated I/O transfer.

c. The I/O request must be one of those listed below to ensure successful completion. The handler in processing requests in continuous mode depends on receiving control at the CAL level in order to respond to I/O errors. The functions for which continuous operation is attempted include only the following:

   1. .MTAPE
   2. .READ
   3. .WRITE
   4. .TRAN

d. With .MTAPE, more than one logical record may be in a physical block, so tape motion may stop if fewer successive .READ's or .WRITE's are issued than there are records in a block.

e. The previously-requested transfer must be completed without error. In general, successive error-free READ's (WRITE's) to the same transport will achieve non-stop operation. The following examples illustrate this principle.

Example 1: Successful Continued Operation

SLOT = 1
INPUT = 0
BLOKNO = 0
READ1 .TRAN SLOT, INPUT, BLOKNO, BUFF1, 257
READ2 .TRAN SLOT, INPUT, BLOKNO, BUFF2, 257
RETURN JMP READ1
The program segment in Example 1 will most probably keep the referenced transport (.DAT slot 1) up to speed. The probability decreases as more time elapses between READ1 and READ2, and between READ2 and RETURN. Each .TRAN request causes an implicit .WAIT until its operation is completed.

Example 2: Unsuccessful Continued Operation

SLOT = 1
INPUT = 0
BLOKNO = 0
READ .TRAN SLOT, INPUT, BLOKNO, BUFF, 257
STOP .WAIT SLOT
RETURN JMP READ

The program segment in Example 2 will not keep the tape moving because the .WAIT at location STOP prevents control from returning to location READ until the transfer first initiated at READ has been completed.

Example 3: Unsuccessful Continued Operation

SLOT1 = 1
SLOT2 = 2
INPUT = 0
BLOKNO = 0
READ1 .TRAN SLOT1, INPUT, BLOKNO, BUFF1, 257
READ2 .TRAN SLOT2, INPUT, BLOKNO, BUFF2, 257
RETURN JMP READ1

This program segment will not provide non-stop operation because of the differing unit specification at READ1 and READ2.

6.2.4 Storage Retrieval on File-Structured Magnetic Tape

The use of a file accessibility map as well as block identifiers in Magtape file directories makes it almost impossible to retrieve the area of a deleted file from a magnetic tape. The execution of the deletion command (i.e., .DELETE) removes the name of the object file from the file directory, and clears the corresponding bit in the File Accessibility Map.

The only circumstance under which a file area may be easily retrieved is when the deleted file is also the last file physically on the tape. Under these conditions, the handler can retrieve the area occupied by the deleted file when the next .ENTER - .WRITE - .CLOSE sequence is executed. Users may also copy the active files to another device, renew the directory, and recopy the files.

6-11
6.3 DISK FILE STRUCTURE

6.3.1 Introduction

The DOS-15 disk file structure is in some ways analogous to DECTape file structure. Ordinarily, each disk user has a directory which points to named files, just as each DECTape has a directory. The DECTape has only one directory, but the disk has as many directories as users have cared to establish. A single user's disk directory might correspond to a single DECTape directory. A single disk file's size is also limited only by the available space, as is true with DECTape. Although DECTape directories may reference a maximum of \( 56_{10} \) files, the number of files associated with any one directory on the disk is limited only by the available disk space.

The DECTape directory is in a known location -- at block 100. Since the disk may have a variable number of directories, the Monitor must know how to find each user's directory. It therefore maintains a Master File Directory (MFD) at a known location\(^1\), and the Master File Directory points to each User File Directory (UFD). DOS-15 allows only those users who know the Master Identification Code to have access to any protected UFD's within the MFD. Figure 6-6 illustrates the MFD. Appendix B is a flowchart of the Disk "A" Handlers.

6.3.2 User Identification Codes (UIC)

The Monitor finds User File Directories by seeking associated User Identification Codes (UIC's), which are all listed in the Master File Directory. The UIC is a three-character code that is necessary for all non-.TRAN I/O to the disk. .TRAN macros use no directory references. A programmer may operate under as many UIC's as he wishes, provided all are unique and none is reserved\(^2\). He may establish a new User File Directory by (1) logging in his new UIC to the Monitor via the LOGIN command, (2) calling PIP, and (3) issuing an N DK command. This establishes a new User File Directory, or refreshes (wipes clean) an old directory under that UIC. (.ENTER will also create a new MFD entry and/or a UFD, if none exists.) Figure 6-7, User File Directory, illustrates the organization of a UFD.

---

\(^1\)On the RF and RK disk, the first block of the MFD is 1777 octal.
On the RP disk, the first block of the MFD is 47040 octal.

\(^2\)The following are reserved UIC's: @@, ???, PAG, BNK, SYS, IOS, CTP.
<table>
<thead>
<tr>
<th>Word #</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>777777</td>
<td>Dummy UIC used by system.</td>
</tr>
<tr>
<td>1</td>
<td>nnnnnn</td>
<td>Bad Allocation Table's first block number, or 777777, if there is none.</td>
</tr>
<tr>
<td>2</td>
<td>nnnnnn</td>
<td>SYSBLK's first block number, or -1, if there is none.</td>
</tr>
<tr>
<td>3</td>
<td>4@-2+blknum</td>
<td>MFD entry size in bits @-2, plus the block number of the first submap.</td>
</tr>
<tr>
<td>4N</td>
<td>.SIXBT</td>
<td>UIC for this UFD.</td>
</tr>
<tr>
<td>4N+1</td>
<td>nnnnnn</td>
<td>Block number for the first block of this UFD or 777777, if no UFD exists (as after PIP's NDK).</td>
</tr>
<tr>
<td>4N+2</td>
<td>P@+M</td>
<td>Protection code in bit @, plus the UFD entry size for each file.</td>
</tr>
<tr>
<td>4N+3</td>
<td>spare</td>
<td>Unused at this writing.</td>
</tr>
<tr>
<td>376</td>
<td>nnnnnn</td>
<td>Pointer to previous MFD block, or 777777 if none.</td>
</tr>
<tr>
<td>377</td>
<td>nnnnnn</td>
<td>Pointer to next MFD block, or 777777 if none.</td>
</tr>
</tbody>
</table>

**Figure 6-6**

Master File Directory

<table>
<thead>
<tr>
<th>Word #</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8N</td>
<td>.SIXBT</td>
<td>Name of this file and its extension.</td>
</tr>
<tr>
<td>8N+1</td>
<td>.SIXBT</td>
<td>Truncation code in bit @, plus the number of the first block of the file.</td>
</tr>
<tr>
<td>8N+2</td>
<td>.SIXBT</td>
<td>Number of blocks in this file.</td>
</tr>
<tr>
<td>8N+3</td>
<td>T@+blknum</td>
<td>Pointer to the first block of the Retrieval Information Block.</td>
</tr>
<tr>
<td>8N+4</td>
<td>nnnnnn</td>
<td>Protection code in bits @-1, plus the first word in ribptr used by the RIB-- if the last block of the file has room for the RIB, the handlers will put it there, and load word 8N+6 accordingly.</td>
</tr>
<tr>
<td>8N+5</td>
<td>ribptr</td>
<td>Date of file's creation -mmddyy (yy modulo 7@).</td>
</tr>
<tr>
<td>8N+6</td>
<td>P@-1+ribwr</td>
<td>Pointer to previous block, or 777777 if none.</td>
</tr>
<tr>
<td>8N+7</td>
<td>crdate</td>
<td>Pointer to next UFD block, or 777777 if none.</td>
</tr>
</tbody>
</table>

**Figure 6-7**

User File Directory

6-13
6.3.3 Organization of Specific Files on Disk

The Disk Handlers write out files in almost the same way that a DECTape handler does. Disk file blocks, however, have a forward and backward link. (Non-dump records are therefore limited to lengths of $254_{10}$ words.) Further, upon receipt of a .CLOSE I/O macro, the disk handlers fill out a Retrieval Information Block (RIB). The RIB performs the same functions as the file bitmap on DECTape, and also associates the logical sequence of blocks in the file with the physical locations of the blocks on the disk. The disk handler uses the RIB to implement .RTRAN commands and to delete files. Figure 6-8, The Retrieval Information Block, illustrates a RIB.

After a user has created a disk file he can access logical records sequentially via .READ commands, just as with DECTape files. He can also access physical blocks of that file by referencing relative block numbers in the .RTRAN command. (The .RTRAN commands require the file be opened with the .RAND command.)

6.3.4 Buffers

The handlers break buffers from the pool into three parts: (1) File Information (about $40_8$ words)*, (2) the Block List -- addresses of pre-allocated blocks (between 4 and $253_8$ addresses, inclusive), and (3) data buffer ($256_8$ words). Figure 6-9, Disk Buffer, illustrates the breakdown of disk buffers.

6.3.4.1 Commands That Obtain And/or Return Buffers

The following commands obtain buffers from the pool, and return them immediately after execution:

- .DELETE
- .RENAME
- .CLEAR

The following commands obtain a buffer from the pool and do not return it until a subsequent .CLOSE is performed:

- .FSTAT
- .ENTER
- .SEEK
- .RAND

*This number is determined by assembly parameters.
<table>
<thead>
<tr>
<th>Word #</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>nnnnnn</td>
<td>Total number of blocks described by this physical block.</td>
</tr>
<tr>
<td>1</td>
<td>nnnnnn</td>
<td>First data block pointer.</td>
</tr>
<tr>
<td>2</td>
<td>nnnnnn</td>
<td>Second data block pointer.</td>
</tr>
<tr>
<td>3</td>
<td>nnnnnn</td>
<td>Third data block pointer.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>376</td>
<td>nnnnnn</td>
<td>Pointer to previous RIB block or -1 if no previous RIB block.</td>
</tr>
<tr>
<td>377</td>
<td>nnnnnn</td>
<td>Pointer to next RIB block or -1 if no next RIB block.</td>
</tr>
</tbody>
</table>

* Zeroth word of the RIB may not be zeroth word of physical block. This occurs whenever the entire RIB will fit in the last data block of the file.

Figure 6-8
Retrieval Information Block

\[40^8 \text{ Words}^*\]

- File Information becomes 'Current Set' when file active (see 6.3.4.2).
- Addresses of Preallocated Blocks (Block List or Temp List or TLIST)
- Data Buffer

\[400^8 \text{ Words}\]

*This is not a fixed number. It is different for FP, RK and RF.

Figure 6-9
Disk Buffer

6-15
The following commands return a buffer to the pool, if any was allocated.

.INIT
.CLOSE
.MTAPE (rewind)

6.3.4.2 The Current Set

The handlers retain information about the last file and .DAT slot processed in an internal storage area. This area is called the "Current Set", and is swapped back to the file's buffer whenever a command to a different file is used. Thus,

.WRITE to .DAT slot A
.WRITE to .DAT slot B

will swap the Current Set, but...

.WRITE to .DAT slot A
.TRAN to .DAT slot A
.WRITE to .DAT slot A

will not swap the Current Set.

6.3.5 Pre-allocation

The handlers pre-allocate blocks on the disk upon all .ENTER commands, and whenever sufficient .WRITE commands have been issued to use up the pre-allocated blocks. The number of pre-allocated blocks will be the minimum of the number of free blocks on the device and the number of address slots available in the Temp List (block list).

When the handlers pre-allocate blocks, they fill out the bit maps, and immediately fill out the RIB and write it out in one of the pre-allocated blocks.

Upon a .CLOSE command, the handlers give back unused blocks, and rewrite the RIB.
The number of blocks in the Block List depends on the size of the buffer, which is determined at system generation by setting the buffer size. The larger the Block List, the faster will be output. Smaller Block Lists may give more efficient allocation of core and disk space. Smaller buffers save core. Further, the number of pre-allocated blocks may affect concurrently opened files on a disk that is tight for space. Thus, if the Block List is sixty entries long, and there are forty blocks left on the disk, a .ENTER to .DAT slot will pre-allocate all forty, leaving none for any subsequent .ENTER's to different .DAT slots.

IOPS 70 will occur when there are less than four free blocks on the disk when a handler tries to pre-allocate blocks.

6.3.6 Storage Allocation Tables (SAT's)\(^1\)

The disk handlers use a Storage Allocation Table, in order to distinguish between allocated and free blocks. If more than one physical block is required, the individual blocks are called Submaps.

Unlike DECTape, the Storage Allocation Table is never held in core. When the handlers wish to preallocate some blocks, they read in the required Submap, and write out the updated one.

Storage Allocation blocks use the following format:

<table>
<thead>
<tr>
<th>WORD 0</th>
<th>Total blocks on the disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD 1</td>
<td>Number of blocks described by this Submap</td>
</tr>
<tr>
<td>WORD 2</td>
<td>Number of blocks occupied in this Submap</td>
</tr>
<tr>
<td>WORD 3</td>
<td>First word of the bit map (eighteen blocks per word)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>WORD 376</td>
<td>Pointer to previous Submap (or 777777)</td>
</tr>
<tr>
<td>WORD 377</td>
<td>Pointer to next Submap (or 777777)</td>
</tr>
</tbody>
</table>

The bit maps refer to blocks in numerical order. Thus, bit 0 of word three of a Submap will refer to block N, bit 1 will refer to block N+1, and so on. The block is free if the corresponding bit equals 0. Starting and ending block numbers for all Submaps are retained in the handlers. Bit 0 of word three in the first submap, refers to block zero.

\(^1\)The first SAT block is located at 1776\(_b\) for the RF and RK system and 764\(_b\) for the RP system.
6.3.7 Bad Allocation Tables (BAT's)

Occasionally, a particular block on the disk will not record data correctly. In such instances, the handlers should be prevented from using the bad blocks. Accordingly, PIP maintains a Bad Allocation Table. Whenever a user updates that table, PIP will set the appropriate bit in the Storage Allocation Table. The block is thus made unavailable. Refer to PIP manual (DEC-15-UPIPA-A-D) for more information.
CHAPTER 7

WRITING NEW I/O DEVICE HANDLERS

This chapter contains information essential for writing new I/O device handlers to work in DOS.

7.1 I/O DEVICE HANDLERS, AN INTRODUCTION

All communications between user programs and I/O device handlers are made via CAL instructions followed by an argument list. The CAL Handler in the Monitor (Figure 2-1) performs preliminary setups, checks the CAL calling sequence, and transfers control via a JMP* instruction to the entry point of the device handler. When the control transfer occurs (see Figures 7-1 and 7-2), the AC contains the address of the CAL in bits 3 through 17 and bits 0, 1, and 2 indicate the status of the Link, Bank/Page mode, and Memory Protect, respectively, at the time of the CAL. Note that the contents of the AC at the time of the CAL is not preserved when control is returned to the user.

On machines that have an API, the execution of a CAL instruction automatically raises the priority to the highest software level (level 4). Control passes to the handler while it is still at level 4, allowing the handler to complete its non-reentrant procedures before debreaking (DBK) from level 4. This permits the handler to receive reentrant calls from software levels higher than the priority of the program that contained this call. Device handlers which do not contain reentrant procedures (including all handlers supplied with DOS) may avoid system failure caused by inadvertent reentries by remaining at level 4 until control is returned to the user.

If the non-reentrant method is used, the debreak and restore (DBR) instruction should be executed just prior to the JMP* which returns control to the user, allowing debreak from level 4 and restoring the conditions of the Link, Bank/Page mode, and Memory Protect. Any IOT's issued at the CAL level (level 4 if API present, mainstream if no API) should be executed immediately before the

```
DBR
JMP*
```
Figure 7-1
CAL Entry to Device Handler

\[\text{LOC}+8\] \text{CAL ARG.}
\[\text{LOC}+1\] \text{CODE}
\[\cdot\]
\[\text{LOC}+N\] \text{NXT INST}

For non-unibus devices both these branches would be replaced by a single initiate function routine.
Figure 7-2
PI and API Entries to Device Handlers

On a PI or API interrupt, the device handler is entered in Bank or Page mode, depending on the setting of bit 11 in .SCOM+4. If = 1, Bank mode; if = 0, Page mode.
exit sequence in order to ensure that the exit takes place before the interrupt from the issued IOT occurs.

The CAL instruction must not be used at any level (API or PIC) that might interrupt a CAL. A CAL at such a level will destroy the content of location 00020 for the previous CAL.

Care must also be taken when executing CALS at level 4. For example, a routine that is CALed out of level 4 must know that if a debreak (DBR or DBK) is issued, control will return to the calling program (which had been at level 4) at a level lower than level 4.

7.1.1 Setting Up the Skip Chain and API (Hardware) Channel Registers

When the Monitor is loaded, the Program Interrupt (PI) Skip Chain and the Automatic Priority Interrupt (API) channels are set up to handle the TTY keyboard and printer and clock interrupts only. The Skip Chain contains the other skip IOT instructions, but indirect jumps to an error routine result if a skip occurs, as follows:

```
SKPDTA    /Skip if DECTape flag.
SKP
JMP* INT1  /INT1 contains error address.
SKFLPT
SKP
JMP* INT2  /INT2 contains error address.
SKPTTI
SKP
JMP TELINT /To teleprinter interrupt handler.
```

All unused API channels, memory protect, memory parity, and powerfail, also contain JMP's to the error address.

When a device handler is called for the first time in a core load, it must call a Monitor routine (.SETUP) to set up its skip(s) in the Skip Chain, or its API channel, prior to performing any I/O functions.
The calling sequence is as follows:

**CAL N**  
/N = API channel register 40 through 77 (see User's Handbook Vol. 1, for standard channel assignments),
/O if device not connected to API.
16  
/.SETUP function code.
**SKP IOT**  
/Skip IOT for this device.
**DEVINT**  
/Address of interrupt handler.
(normal return)

### 7.1.2 Handling the Interrupt

**DEVINT** exists in the device handler in the following format to allow for either API or PI interrupts. The following is for UNIBUS devices only:

```
ONLY1    LAC  (NOP) /
DACPIC    DAC  DEVIION /
DACPIC    DAC  DEVIOF /
DACPIC    DAC  IGNRPI /
JMP      COMMON /

DEVIN T  JMP  DEVPI C /
DEVIN T  JMP  DEVI ON /
DEVIN T  LAC*  (Ø) /
DEVIN T  DAC  DEVOUT /
DEVIN T  JMP  COMMON /

COMMON   CAPI- /
DEVION  ION  /

DEVIOF  IOF  /

LAC  (TCB /
SIOA  /GET ADDRESS OF TCB IN AC
JMP  .-1 /
LIO R  /YES. LOAD REGISTER IN INTERRUPT LINK

/DISSMISS ROUTINE /

LAC  DEVAC /
DVS WCH  ION  /
D BR  /
JMP*  DEVO UT /
```

If the Index, Autoincrement, or EAE registers are used by the I/O device handler, it is necessary to save and restore them.

7-5
The following is for non-UNIBUS devices:

```
ONLY1  LAC   (NOP) /LEAVE PI ALONE. WHEN API IS RUNNING
DAC    DEVION /THESE REGISTERS
DAC    DEVIOP /ARE AVAILABLE
DAC    IGNRPI /THIS IS ONCE ONLY CODE
JMP    COMMON

DEVPIC  DAC    DEVAC /SAVE AC
LAC*   (Ø)    /SAVE PC, LINK, ADDRESSING MODE AND
                /MEMORY PROTECT
DAC    DEVOUT
JMP    COMMON

DEVINT  JMP    DEVPIC /PI ENTRY
DAC    DEVAC /API ENTRY; SAVE AC
LAC    DEVINT
DAC    DEVOUT /SAVE PC, LINK, ADDRESSING MODE AND
                /API IS OPERATING, SO LEAVE PI ALONE.
                /PI INTERRUPTS ARE NOT POSSIBLE, BE-
                CAUSE SETUP EFFECTIVELY NOP'S PI
                /SKIPS.
IGNRPI  JMP    ONLY1

COMMON  DEVCF /CLEAR DEVICE DONE FLAG.
DEVION  ION   /PI ALLOWS INTERRUPTS; API DOES A NOP.

DEVIOF  IOF   /API DOES NOP; PI TURNS IO OFF TO ENSURE
                /NON-REENTRANCE AFTER ISSUING IOT'S.
DEVIOT

/DISMISS ROUTINE

LAC    DEVAC /RESTORE AC.
DVUSWCH ION   /ION OR NOP.
DBR     .    /DEBREAK AND RESTORE CONDITIONS
JMP*    DEVOUT /OF LINK, ADDRESSING MODE AND MEMORY
                /PROTECT.
```

If the Index, Autoincrement, or EAE registers are used by the I/O device handler, it is necessary to save and restore them.
.SETUP allows either API or PI, but not both for a single device. The System Generator Manual gives the method for incorporating new handlers and associated Skip Chain entries into the Monitor.

7.2 API SOFTWARE LEVEL HANDLERS, AN INTRODUCTION

The information presented in the following paragraphs assumes that the reader is familiar with the system input/output considerations described in the PDF-15 User’s Handbook Vol. 1.

7.2.1 Setting Up API Software Level Channel Registers

When the Monitor is loaded, the API software-level channel registers (40 through 43) are initialized to

```
JMS* .SCOM+12 /LEVEL 4
JMS* .SCOM+13 /LEVEL 5
JMS* .SCOM+14 /LEVEL 6
JMS* .SCOM+15 /LEVEL 7
```

where .SCOM is equal to absolute location 000100 and .SCOM+12 through .SCOM+15 (000112 through 000115) each contains the address of an error routine.

Therefore, prior to requesting any interrupt at these software priority levels, the user must modify the contents of the .SCOM registers so that they point to the entry point of the user’s software level handlers.

Example:

```
.SCOM=100
LAC (LV5INT / set level 5 entry.
DAC* (.SCOM+13

LV5INT exists in the user’s area in the following format:

```
LV5INT 0 /PC, LINK, BANK/PAGE MODE, MEM.PROT.
DAC SAV4AC /SAVE AC
/SAVE INDEX, AUTOINCREMENT AND EAE REGISTERS, IF LEVEL 5
/Routines USE THEM AND LOWER LEVEL ROUTINES ALSO USE THEM.
/SAVE MQ AND STEP COUNTER, IF SYSTEM HAS EAE AND IT IS USED
/AT DIFFERENT LEVELS.
```

/RESTORE SAVED REGISTERS.
DBR /DEBREAK FROM LEVEL 5 AND RESTORE
JMP* LV5INT /L, BANK/PAGE MODE, MEM. PROT.
7.2.2 Queueing

High priority/high data rate/short access routines cannot perform complex calculations based on unusual conditions without holding off further data input. To perform the calculations, the high priority program segment must initiate a lower priority (interruptable) segment to perform the calculations. Since many data handling routines would generally be requesting calculations, there will exist a queue of calculation jobs waiting to run at the software level. Each data handling routine must add its job request to the appropriate queue (taking care to raise the API priority level as high as the highest level that manipulates the queue before adding the request) and issue an interrupt request (ISA) at the corresponding software priority level. The general flow chart, Figure 7-3, depicts the structure of a software handler involved with queued requests.

Care must be taken about which routines are called when a software level request is honored; that is, if a called routine is "open" (started but not completed) at a lower level, it must be reentrant, or errors will result.

NOTE

The DOS hardware I/O device handlers do not contain reentrant procedures and must not be re-entered from higher levels.

Resident handlers for Power Fail, Memory Parity, nonexistent memory violation, and Memory Protect violation have been incorporated into the system and effect an IOPS error message if the condition is detected. The user can, via a .SETUP, tie his own handler to these skip IOT or API channel registers.
Figure 7-3
Structure of API Software Level Handler
7.3 WRITING SPECIAL I/O DEVICE HANDLERS

This section contains information prepared specifically to aid those users who plan to write their own special I/O device handlers for DOS.

DOS is designed to enable users to incorporate their own device handlers. Precautions should be taken when writing the handler however, to ensure compatibility with the Monitor.

Here is a summary of handler operation. The handler is entered via a JMP* from the Monitor as a result of a CAL instruction. The contents of the AC contain the address of the CAL in bits 3 through 17. Bit 0 contains the Link, bit 1 contains the Bank/Page Mode status, and bit 2 contains the Memory Protect status. The previous contents of the AC and Link are lost.

In order to show the steps required in writing an I/O device handler, a complete handler (Example B) was developed with the aid of a skeleton handler (Example A). In addition, Appendices A and B are complete flowcharts of the DTA and the A version of the disk handlers. The skeleton handler is a non-reentrant type (discussed briefly at the beginning of this chapter) and uses the Debreak and Restore Instruction (DBR) to leave the handler at software priority level 4 or at a hardware level for interrupt servicing (if API), and restore the status of the Link, Bank/Page Mode, and Memory Protect. Example A is referenced by part numbers to illustrate the development of Example B, a finished Analog-to-Digital Converter (ADC) I/O Handler. The ADC handler shown in Example B was written for the Type AP81B Analog to Digital Converter. This handler is used to read data from the ADC and store it in the user's I/O buffer.

The reader, while looking at the skeleton of a specialized handler as shown in Example A, should make the following decisions about his own handler. (The decisions made in this case are in reference to developing the ADC handler):

a. Services that are required of the handler (flags, receiving or sending of data, etc.) - By looking at the ADC IOT's shown in the Reference Manual, it can be seen that there are three IOT instructions to be implemented. These instructions are: Skip if Converter Flag Set, Select and Convert, and Read Converter Buffer.
The only service the ADC handler performs is that of receiving data and storing it in user specified areas. This handler will have a standard 256-word buffer.

b. **Data Modes used** (for example, TOPS ASCII, etc.) - Since there is only one format of input from the Type AF01B ADC, mode specification is unnecessary in Example C.

c. **Which I/O macros are needed** for the handler's specific use; that is, .INIT, .CLOSE, .READ, etc. For an ADC, the user would be concerned with four of the macros.

1. **.INIT** would be used to set up the associated API channel register or the interrupt skip IOT sequence in the Program Interrupt Skip Chain. This is done by a CAL.(N) as shown in Part III of Example A, where (N) is the channel address.

2. **.READ** is used to transfer data from the ADC. When the .READ macro is issued, the ADC handler will initiate reading of the specified number of data words and then return control to the user. The analog input data received is in its raw form. It is up to the programmer to convert the data to a usable format.

3. **.WAIT** detects the availability of the user's buffer area and ensures that the I/O transfer is completed. It would be used to ensure a complete transfer before processing the requested data.

4. **.WAITR** detects the availability of the user's buffer area as in (3) above. If the buffer is not available, control is returned to a user specified address, which allows other processing to continue.

d. **Implementation of the API or PIC interrupt service routine** - Example A shows an API or PIC interrupt service routine that handles interrupts, processes the data and initiates new data requests to fully satisfy the .READ macro request. Note that the routines in Example A will operate with or without API. Example B uses the routines exactly as they are shown in Example A.

During the actual writing of Example B, consideration was given to the implementation of the I/O macros in the new handler in one of the following ways:

1. Execute the function in a manner appropriate to the given device as discussed in(c). .INIT, .READ, .WAIT, and .WAITR were implemented into the ADC handler (Example B) under the subroutine names ADINIT, ADREAD, ADWAIT (.WAIT and .WAITR).

Wait for completion of previous I/O. (Example B shows the setting of the ADUND switch in the ADREAD subroutine to indicate I/O underway.)
(2) Ignore the function if meaningless to the device. See Example B (.FSTAT results in JMP ADIGN2) in the dispatch table DSPCH. For ignored macros, the return address must be incremented in some cases, depending upon the number of arguments following the CAL (see Chapter 3).

(3) Issue an error message in the case where it is not possible to perform the I/O function — (An example would be trying to execute a .ENTER on the paper tape reader.) In Example B, the handler jumps to DVERR6 which returns to the Monitor with a standard error code in the AC.

e. Special considerations for UNIBUS device handlers

When new handlers are written for devices on the UNIBUS in a UC15 system (Rk based or RF/RP based UC15 option) the following has to be considered.

Since communication between the device handler on the PDP-15 and the driver task running under PIREX on the PDP-11 is through Task Control Blocks (TCB), space in the Common Memory (memory that can be addressed by the PDP-15 and the PDP-11) must be provided. The system as supplied by DEC has space reserved in the Resident Monitor for 3 user defined devices/programs/tasks, (refer to Section 2.9 for more information). This TCB must be properly setup (refer to the UC15 Software Manual, DEC-15-XUCMA-A-D for more information) before the handler calls PIREX to initiate the operation.

Driver tasks (TTT)\textsuperscript{1} running under PIREX report errors by setting the appropriate code (XX) in the device error status table in PIREX (refer to UC15 Software Manual, DEC-15-XUCMA-A-D for more information). DOS-15 system prints out this error message, which appears as follows:

\texttt{IOPSUC TTT XX}

Users have to decipher this message. An example of this is,

\texttt{IOPSUC LPU 4}

which reports that the LP11/LS11 line printer is not ready. There is no error message type out from the handler. This method of error handling is incorporated to permit error report during operation of these devices/tasks etc., under PIREX when their corresponding handlers are not present in core on the PDP-15 (e.g., during Spooling).

\textsuperscript{1}Each task running under PIREX has a 3 character code assigned to it which is present in the PIREX error table at assembly time.
After the handler has been written and assembled, the Monitor must then be modified to recognize the new handler. This is accomplished by the use of the System Generator Program (DOSGEN) described in the DEC-15-USGNA-A-D manual.

When the system generation is complete, the PIP program (refer to DEC-15-UPIPA-A-D) must be used to add the new handler to the IOS UPD. At this time, the user is ready to use his specialized device handler in the DOS-15 system.

7.3.1 Discussion of Example A by Parts

Part 1 Stores CAL pointer and argument pointer, and picks up function code from argument string.

Part 2 By getting proper function code in Part 1 and adding a JMP DSPCH, the CAL function is dispatched to the proper routine.

Part 3 This is the .SETUP CAL used to set up the PI skip chain or the API channel register.

Part 4 Shows the API and PI handlers. It is suggested these be used as shown.

Part 5 This area reserved for processing interrupt and performing any additional I/O.

Part 6 Interrupt dismiss routine.

Part 7 Increments argument pointer in bypassing arguments of ignored macro CAL's.
7.3.2 Example A, Skeleton I/O Device Handler

ENTRY ROUTINE

GLOBAL DEV,

MED=3

DEV,

DAC DVCALP
DAC DVARGP
LAC DVARGP
AND 777777
S2 DVARGP
TAB (JMP OSPCH)
DAC OSPCH

DISPATCH WITH

MODIFIED JUMP

1 = INIT
2 = ISTATUS, DELETE, REMAIN
3 = SEEK
4 = ENTER
5 = CLEAR
6 = CLOSE
7 = TAPE
12 = READ
11 = WRITE
12 = WAIT
13 = TRAN

ILLEGAL FUNCTIONS IN ABOVE TABLE CODE AS

JMP OVERR6

FUNCTION CODE ERROR

OVERR6 LA= 6
JMP (MED+1)
ERROR CODE 6
TO MONITOR

DATA MODE ERROR

OVERR7 LA= 7
JMP (MED+1)
ERROR CODE 7
TO MONITOR

DEVICE NOT READY

OVERR4 LAC (RETURN)
RETURN (ADDRESS TO HANDLE)
TO RETURN TO WHEN NOT READY
CONDITION HAS BEEN REMOVED

DAC (MED)
LAC (4)
JMP (MED+1)
ERROR CODE 4
TO MONITOR

I/O UNDERWAY LOOP

DBR DVCALP
BREAK FROM LEVEL 4
LOOP

NORMAL RETURN FROM CAL

DBR DVARGP
BREAK FROM LEVEL 4
RETURN AFTER CAL AND
ARGUMENT STRING

THE DUNIT ROUTINE MUST INCLUDE
A SETUP CALLING SEQUENCE FOR
EACH FLAG CONNECTED TO API
AND/OR PI AT SGEN TIME;
THE SETUP CALLING SEQUENCE IS

OVINIT CAL N

N = API CHANNEL REGISTER
(47 -77), N = 0 IF NOT CONNECTED
TO API

16

SKIP TO TEST THE FLAG

THIS SPACE MAY BE USED FOR I/O SUBROUTINES

INTERRUPT HANDLER FOR API OR PI
ONLY

LAC (NOP)
DAC DEVION
DAC DEVIOF
DAC DEVSWCH
DAC IGNRP
JMP COMMON

DAC DEVIAC
LAC (*
DAC DEVOU
JMP COMMON

JMP DEVPIC
DAC DEVIAC
LAC DEVIAT
DAC DEVIAC
JMP COMMON

IGNRP JMP ONLY1
COMMON DEVF
DEVION 10V

THIS AREA DEVOTED TO PROCESSING INTERRUPT AND
PERFORMING ANY ADDITIONAL I/O DESIRED.

DEVIOF I0F
DEVINT

DISABLE PI OR NOP
DISMISSAL BEFORE INTERRUPT
FROM THIS I/O OCCURS

INTERRUPT HANDLER DISMISS ROUTE

OVISM LAC DEVIAC
OVSWCH 10M
JBR
JMPDev OUT

RESTORE AC
ION OR NOP
BREAK AND RESTORE
/LINK, BANK/PAGE MODE, MEMORY
/PROTECT

IF THE HANDLER USES THE AUTO_INCREMENT, INDEX
FOR EAE REGISTERS, THEIR CONTENTS
SHOULD BE SAVED AND RESTORED, FUNCTIONS
POSSIBLY IGNORED SHOULD CONTAIN
PROPER INDEXING TO BYPASS
/CAL ARGUMENT STRING
/CODE TO BYPASS IGNORED FUNCTIONS

OVIGNZ 152
JMP DVARUP

BYPASS FILL POINTER

7-15
7.3.3 Example B, Special Device Handler for AF01B A/D Converter

```
PAGE 1

1 /ADC HANDLE
2 / SKIP IF CONVERSION FLAG IS SET
3 701321 A ADSC#701321 /SELECT AND CONVERT (ADC FLAG IS CLEARED
4 701324 A ADSC=701324 /AND A CONVERSION IS INITIALIZED)
5 / READ CONVERTER BUFFER INTO AC AND CLEAR FLAG
6 701412 A ADRB#701412
7 /,GLRL ADC,
8 440203A IDX=15?
9 00073 A ;MED=3 /MED (MONITOR ERROR DIAGNOSTIC)
10 / ADC, DAC, ADCA, DAC
11 ADC, DAC /DCALLP /SAVE CAL POINTER
12 DAC, ADAP /AND ARGUMENT POINTER
13 ADAP /POINTS TO FUNCTION CODE
14 /GET CODE
15 ADAP /POINTS TO CAL + 2
16 TAD /JMP DSPCH
17 /DSPCH /DISPATCH WITH
18 40727 A DSPCH XX /MODIFIED JUMP
19 70240 A XX /ADDINIT /1#,INIT
20 /2#,FSTAT,DELETE,RENAME
21 600227 R JMP ADAP2 /3#,SEEK
22 600224 R JMP ADAP2 /4#,ENTER
23 600223 R JMP ADAP2 /5#,CLEARN
24 600222 R JMP ADAP2 /6#,CLOSE
25 600221 R JMP ADAP2 /7#,HTRAP
26 600220 R JMP ADAP2 /8#,READ
27 600219 R JMP ADAP2 /9#,WRITE
28 600218 R JMP ADAP2 /10#,WAIT
29 600217 R JMP ADAP2 /11#,TRA
30 600216 R JMP ADAP2 /12#,EJECT
31 / ILLEGAL FUNCTIONS IN ABOVE TABLE Coded AS
32 / JMP ADAP2
33 / EJECT
```
;2023 R 760026 A
;2024 R 62159 R
;2025 R 760027 A
;2026 R 62155 R
;2027 R 440151 R

;2030 R 278156 R
;2032 R 440151 R
;2034 R 620257 A
;2036 R 760151 A
;2038 R 760151 A

;2042 R 144247 R
;2044 R 627275 R

/ FUNCTION CODE ERROR
/ ADERR6 LAW 6 / ERROR CODE 6
/ DATA MODE ERROR
/ ADERR7 LAW 7 / ERROR CODE 7
/ THE ADINT ROUTINE MUST INCLUDE A SETUP
/ FOR EACH FLAG ASSOCIATED WITH THE DEVICE
/ ADINIT IDX ADARGP / IXK TO RETURN BUFF SIZE
/ DEC LAC (256 / STANDARD BUFFER SIZE (ORIGINAL)
/ OCT IDX ADARGP
/ ADCHM CAL 57 / #API CHANNEL
/ ADCKSM 16 / SETUP TOPS FUNCTION CODE
/ ADCPF ADSP / ASC SKIP ICT
/ ADLBWP ANCLI / ADDR. OF INTERRUPT
/ ADUND LAC .+2 / SET-UP ONCE ONLY
/ ADWC DAC ADCMOD / SKIP SETUP CODE IF MORE
/ ADWPT JMP ADSTP / INIT'S ARE DONE
/ STOP ADC ROUTINE CLEANS I/O UNDERWAY SWITCH
/ ADSTOP 2EM ADUND
/ JMP ADIG 1 / RETURN
/ THE PREVIOUS TAGS IN THE CAL AREA ARE USED FOR
/ STORAGE DURING THE ACTUAL READ FUNCTION
/ ADCKSM IS FOR STORAGE THE CHECKSUM
/ ADCPF IS THE CURRENT BUFFER POINTER
/ ADLBWP IS THE LINE BUFFER HEAD/PTR
/ ADUND IS FOR DEVICE UNDERWAY SWITCH
/ ADWC IS USED AS THE COUNTER
/ ADWPT IS USED TO STORE CURRENT WORD COUNT
/ , EJECT
ADWAIT LAC ADUNO
SNA
JMP ADIG.1
/I/O UNDERWAY LOOP
ADGUSY DDR
JMP* ADCALP

ADREAD LAC ADUNO /CHECK TO SEE IF I/O IS UNDERWAY
SRA:CM A /IF NOT SET IT WITH -1
JMP ADIGUSY /IT WAS SET, GO BACK TO CAL
DAC ADUNO /SET IT
LAC* ADCALP /LOOK AT MODE
AND 1704 /BITS 6-8 ONLY
SZA /JOPS BINARY?
JMP ADERR7 /NO, ERROR
LAC* ADCALP /GET LINE BUFFER HEADER POINTER
DAC ADCHP /STORE IT
DAC ADCHR /ALSO STORE IT FOR LATER HEADER
IDX ADCALP /INCREMENT ARG. POINTER
LAC* ADCALP /GET L.EXT.(1)Compat)
DAC ADWC /STORE IT IN WORD COUNTER
DBH ADCPET /ZERO WORD COUNT REG.
DCH ADCWSH /ZERO CHECKSUM REG,
IDX ADCHF /GET PAST HEADER PAIR
IDX ADCHR /NOW POINTING AT BEGINNING OF
BUFFER
ADSC /START UP DEVICE
ADX2 ADCHP /INCR. FOR EXIT
ADIGN DBR /BREAK FROM LEVEL 4
JMP* ADCALP /RETURN AFTER CAL
/INTERUPT HANDLER FOR API OR PI-
/
ONLY1 LAC /NOP
DAC ADIGON
DAC ADCHON
DAC ADCHOT
DAC ADCHOW
DAC ADCHPL
JMP COMMON
ADCPIC DAC ADGCAC /SAVE AC
LAC* (0) /SAVE PC, LINK, EX., MODE
DAC ADCHOT /MEM, PROT,
EJECT
<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Bytes</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>119</td>
<td>0011 R</td>
<td>620116 R</td>
<td>JMP COMMON</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>0011 R</td>
<td>620115 R</td>
<td>ADDCT JMP</td>
<td>/PIC ENTRY</td>
</tr>
<tr>
<td>121</td>
<td>0012 R</td>
<td>040153 R</td>
<td>DAC ADDCIC</td>
<td>/API ENTRY, SAVE AC</td>
</tr>
<tr>
<td>122</td>
<td>0013 R</td>
<td>200111 R</td>
<td>LAC ADDCINT</td>
<td>/SAVE PC, LINK, EX, MODE</td>
</tr>
<tr>
<td>123</td>
<td>0014 R</td>
<td>340152 R</td>
<td>DAC ADDCOUT</td>
<td>/MEM, PROT</td>
</tr>
<tr>
<td>124</td>
<td>0015 R</td>
<td>600077 R</td>
<td>IGNRP1 JMP</td>
<td>ONLY</td>
</tr>
<tr>
<td>125</td>
<td>0016 R</td>
<td>701312 A</td>
<td>COMMON ADDB</td>
<td>/READ CONVERTER BUFFER</td>
</tr>
<tr>
<td>126</td>
<td>0017 R</td>
<td>700042 A</td>
<td>ADDION ION</td>
<td>/ENABLE PIC FOR OTHER DEVICES</td>
</tr>
<tr>
<td>127</td>
<td>0018 R</td>
<td>050035 R</td>
<td>DAC* ADDCRP</td>
<td>/STORE DATA IN USER BUFFER</td>
</tr>
<tr>
<td>128</td>
<td>0019 R</td>
<td>440035 R</td>
<td>IDX ADDCRP</td>
<td>/INC, BUFFER POINTER</td>
</tr>
<tr>
<td>129</td>
<td>0012 R</td>
<td>440041 R</td>
<td>IDX ADDWPC</td>
<td>/INC, WORD PAIR COUNTER</td>
</tr>
<tr>
<td>130</td>
<td>0013 R</td>
<td>340034 R</td>
<td>TAD ADDCKSM</td>
<td>/ADD CHECKSUM</td>
</tr>
<tr>
<td>131</td>
<td>0014 R</td>
<td>240034 R</td>
<td>DAC ADDCKSM</td>
<td>/STORE IT</td>
</tr>
<tr>
<td>132</td>
<td>0015 R</td>
<td>440040 R</td>
<td>ISZ ADWC</td>
<td>/IS ISO COMPLETE</td>
</tr>
<tr>
<td>133</td>
<td>0016 R</td>
<td>600142 R</td>
<td>JMP ADWPC</td>
<td>/NO KEEP GOING</td>
</tr>
<tr>
<td>134</td>
<td>0017 R</td>
<td>200041 R</td>
<td>LAC ADDWPC</td>
<td>/YES, COMPUTE WORD PAIR COUNTER</td>
</tr>
<tr>
<td>135</td>
<td>0018 R</td>
<td>740030 A</td>
<td>IAC ADWPC</td>
<td>/MAY BE CND</td>
</tr>
<tr>
<td>136</td>
<td>0019 R</td>
<td>740022 A</td>
<td>SWHAP TO TOP HALF</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>0012 R</td>
<td>740022 A</td>
<td>RAR MAKE WORDS</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>0013 R</td>
<td>530036 R</td>
<td>AND TO FORMAT ONLY</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>0014 R</td>
<td>530036 R</td>
<td>DAC* ADLHP</td>
<td>STORE IN HEADER #1</td>
</tr>
<tr>
<td>140</td>
<td>0015 R</td>
<td>440036 R</td>
<td>IDX ADLHP</td>
<td>/INC, TO STORE CKSUM</td>
</tr>
<tr>
<td>141</td>
<td>0016 R</td>
<td>340034 R</td>
<td>TAD ADDCKSM</td>
<td>/ADD WORD PAIR COUNTER</td>
</tr>
<tr>
<td>142</td>
<td>0017 R</td>
<td>060036 R</td>
<td>DCM ADDLP</td>
<td>/STORE IN HEADER #2</td>
</tr>
<tr>
<td>143</td>
<td>0018 R</td>
<td>142036 R</td>
<td>O2M ADDMK</td>
<td>/CLEAR DEVICE UNDERWAY</td>
</tr>
<tr>
<td>144</td>
<td>0019 R</td>
<td>600144 R</td>
<td>JMP ADDMK</td>
<td>/EXIT</td>
</tr>
<tr>
<td>145</td>
<td>0012 R</td>
<td>700022 A</td>
<td>ADDCT IOF</td>
<td>/DISABLE PIC OR NOP</td>
</tr>
<tr>
<td>146</td>
<td>0013 R</td>
<td>700134 A</td>
<td>ADISC BEFORE INTERRUPT FROM THIS IOT OCCURS</td>
<td></td>
</tr>
<tr>
<td>147</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>148</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>0014 R</td>
<td>200153 R</td>
<td>ADDISM LAC</td>
<td>/RESTORE AC</td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BOSS enables DOS users with a card reader and a line printer to run jobs sequentially, with a minimum of operator intervention. BOSS supports a subset of the DOS system programs, and adds a line editor, its own resident and nonresident routines (called Resident BOSS and Nonresident BOSS), and the Procedure Files. Paragraph 8.1 describes Procedure Files. Figure 8-1 shows which monitor supports each system program.

The DOS programs run by BOSS are identical to those run by DOS. Exceptions are the Resident and Nonresident Monitors, which are explained later. BOSS expands the information on Control Cards into a series of commands in the format expected by the DOS system programs. Nonresident BOSS does this command expansion, and stores the expanded commands in a disk file, the Run Time File (RTF). Since DOS programs expect to communicate with an operator at a teleprinter, BOSS feeds the expanded commands to the programs via .DAT slots assigned to TTA. In BOSS mode, therefore, BOSS attaches .DAT-2 to the Run Time File, and directs most teleprinter output to the Line Printer. Programs can force I/O to the teleprinter by setting bit 4 in .SCOM+52, and proceeding with macros directed to TTA.

Whenever bit 9 of .SCOM+52 is set, the System Loader Interface attaches the Resident BOSS code to the Resident Monitor. The main purposes of Resident BOSS are to (1) ensure that BOSS will retain control of the teleprinter, (2) feed commands to programs via the Run Time File, (3) properly route internal Monitor commands, such as .EXIT, .GET and .PUT, and (4) direct teleprinter output to the Line Printer. Figure 8-2 illustrates the connections between the DOS Resident Monitor and the BOSS Resident Monitor that accomplish these changes. Figure 8-3, the flowchart for Resident BOSS, further describes Resident BOSS.

Resident BOSS communicates with Nonresident BOSS by TRANing information to and from the first block of Nonresident BOSS. Nonresident BOSS gains control on all error conditions, such as IOPS, operator abort, Time Estimate exceeded, and after a BOSS15 command. Figure 8-4 is a flowchart of Nonresident BOSS.
Figure 8-2
Connections Between DOS Resident Monitor and BOSS Resident Monitor

8-3
Points within the Resident Monitor which transfer control to Resident BOSS - 15

Figure 8-3
Resident BOSS-15

8-4
DAT3.B (.DAT-3 or any .DAT slot assigned to TTA.)

I/O TO TELEPRINTER?

Y

N

COMPOSE A .WRITE TO .DAT+6, USING CALLER'S DATA MODE

1. CLEAR BIT 4 OF .SCOM+52 (ONLY IF - .DAT SLOT)
2. MAKE SURE TELEPRINTER IS NOT BUSY

TRANSFER TO TTA

WRITE?

N

Y

MOVE USER'S BUFFER (MINUS THE HEADER WORD PAIR) TO RESIDENT BOSS (ALWAYS 428 WORDS)

ISSUE .WRITE TO .DAT+6 (THE LINE PRINTER) AND WAIT FOR COMPLETION

RETURN TO USE AFTER CAL

DAT2.B

Y

.READ

N

USING CALLER'S BUFFER ADDRESS AND WORD COUNT, COMPOSE A .READ TO .DAT-7, THE RUN TIME FILE (RTF) ON THE SYSTEM DEVICE

TRANSFER TO TTA

ISSUE .READ TO .DAT-7, AND AVOID COMPLETION

READ THIS LINE BEFORE?

Y

N

HAVE WE REACHED THE END OF FILE OF "RTF"?

N

Y

INCREMENT THE LINE COUNT

SET BIT 2 OF .SCOM+52

B.EXIT

Processing for I/O Macros addressed to .DAT slots -2 or -3, or any slot assigned to TTA.
Points within the Resident Monitor which transfer control to Resident BOS-15.

Figure 8-3 (Cont.)

Resident BOSS-15

8-5
Points within the Resident Monitor which transfer control to Resident BOSS-15

Figure 8-3 (Cont.)
RESIDENT BOSS-15
8-6
Start Address 420

Execute bank bit initialization code

Save value of TIME OUT clock, and disable it

Disable CTRL C and set up CTRL T address; save contents of .SCOM+42, set MICLOG bit and clear user bit

Set .UFDT -15 and -14 to "CTP" and delete all files in the CTP UFD named PRCFIL PRC

Perform .ENTER on .DAT-15 for PRCFIL PRC; set line count = 0

Save current logged-in UIC in D.12; force "CTP" to be logged-in UIC

Save system device code in D.11

First time loaded after a BOSS15 command to DOS

Y

N

CKEOP

Has an abort taken place

N

Y

Set bit 17 in Job Status Word AC_STA

End of Run time file reached

N

Y

Set bit 2 in Job status word Print, "END OF RUN TIME FILE REACHED BY USER" on the LP

Next Page

Figure 8-4
Nonresident BOSS

8-7
Figure 8-4 (Cont.)
Nonresident BOSS

8-8
Figure 8-4 (Cont.)
Nonresident BOSS
Figure 8-4 (Cont.)
Nonresident BOSS

8-10
NEWJOB

Is there an active job

Y

N

ENDJOB

Clear Job Status Word

Make sure the Time Out Clock will be disabled upon exit from NR BOSS

Set bit 8 in .SCOM+52 (Job Active Flag)

Store JOB I.D. into account information buffer

Get a character

Is it the terminator code

Y

N

Pad with nulls until nine characters have been packed

Pack character into account buffer

9 characters reached

Enter date and start time into account information buffer

Start elapsed time clock, by depositing zero into .SCOM+34

Print Job Start Message on LP

SUBSTT

EXIT

Figure 8-4 (Cont.)
Nonresident BOSS
Figure 8-4 (Cont.)
Nonresident BOSS
8-12
ENTADD

Set bit zero of ADDSTA
Set bit one of .SCOM+56

Zero directory entry block

Set up to pack into ADDFIL

Did user supply a file name?

N

Y

Use default file name "TMP"

Pack file name and extension, if any

Was extension in card?

Y

N

Use default extension "ADD"

NXTCRD

CHECK

Exit BOSS mode

N

Y

Zero bits 2-17 of .SCOM+52

NXTCRD

N

Y

Is current control card $JOB

OPENJOB

Reopen Job Procedure file

NEWJOB

ENDJOB

Any thing in "RT"?

Y

N

Job active?

N

Y

Clear Job Active Flag

Enter finish and elapsed time into Account Information Buffer, and .CLOSE .DAT-14

Convert elapsed time to hnmss

Inhibit Card Reader
(Set bit 1 of .SCOM+52)

Set End Job Flag
(bit zero of Job Status word)

EXIT

Next Page
Figure 8-4 (Cont.)
Nonresident BOSS

8-13
FIGURE 8-4 (Cont.)
Nonresident BOSS

8-14
RESTORE .SCOM+42 with user mode bit (bit 17) set; restore user's UIC

WRITE OUT BOSS line in "RTF" (BOSS=15) and count it

STORE line count in bits 3-8 of .SCOM+75

"RTF" longer than 7778 lines

CLOSE "RTF" and ADD files

TRAN out first 4000 words of Nonresident BOSS

CLEAR various bits in .SCOM+52

.EXIT (TO NRM)

WRITE out "DUMP" line to "RTF"

DISABLE the Time Out Clock, so the user gets a complete dump; count two lines

WRITE out "ALL" to "RTF" and enter with an ALT MODE

EXIT

EXBOS1

ISSUE .INIT to "RTF"; zero all BOSS registers and bits

MAKE "SCR" the login UIC

DMPPRC

Figure 8-4 (Cont.)
Nonresident BOSS

8-15
8.1 PROCEDURE FILES

To each BOSS command there corresponds a disk-resident ASCII file, called a Procedure File. The Procedure File contains DOS commands. When DOS executes the commands in the Procedure File, it carries out the function specified by the BOSS control card. The DOS commands in the Procedure Files contain fields (for instance, a file name) that Nonresident BOSS fills in with text strings from the control card. These fields are called, "Variable Fields". Before executing the DOS commands contained in the Procedure File, all the variable fields have to be resolved. This process is very similar to a macro expansion, where (1) DOS is the assembly language, (2) the BOSS command name is the macro name, (3) the contents of the BOSS control card are the macro arguments, and (4) the Procedure File is the macro definition. The expanded DOS commands are put in a Disk File, called the "Run Time File (RTF)". The RTF can contain the expansion of one or more Procedure Files, up to 777 bytes ASCII records.

BOSS expands Procedure Files strictly on a text string, character basis. It has no knowledge of the intrinsic function of each BOSS control card, except for $JOB, $END, $CRT, and $ADD ($END, $CRT, $ADD have no Procedure Files) Appendix C contains a listing of all standard Procedure Files.

8.1.1 Procedure File Format

In order to ensure successful expansion, all Procedure Files must follow a strict format. The first record of the Procedure File must be a control record, with parameter information. The first record may also contain comments, because BOSS interprets only pertinent information, and ignores the rest. The numbers 0, 1, 2, 3, and 4 specify different options. All other characters are ignored. The option digits can appear in any order, and anywhere on the record. The option specified by each number is given below:

- 0 - Expanded Substitution (default, if "3" not given explicitly)

This option specifies that the Procedure File is to be expanded according to the normal rules of substitution, which are given below.
1 - Open Ended File (default, if "2" not given explicitly)

This option instructs the Nonresident BOSS Monitor to leave the RTF open after expanding the current Procedure File. BOSS then searches for the next control card.

2 - Closed End File

This option instructs Non-resident BOSS to close the RTF after expanding the current Procedure File, and to execute the DOS commands in the RTF. Procedure Files corresponding to commands that may possibly be followed by "Data Cards" should be of Type 2.

3 - Direct Substitution

This option indicates the BOSS should not expand the Procedure File according to normal rules. Refer to paragraph 8.1.2 for information on Direct Substitution.

4 - Test Mode

This option indicates that BOSS should echo the Procedure File expansion on the Line Printer. This allows a check on the Procedure File.

The following combinations are illegal:

Ø and 3
1 and 2

If BOSS finds an illegal option combination, it will print,

ILLEGAL PROC FILE

and search for the next control card.

BOSS uses all other records in the Procedure File as macro definition records. Records after the first one are all Macro Definition Records. For each such record, a record will be written in the RTF. Each Macro Definition Record has the same format. Two types of fields are used: K-fields and V-fields. K-fields specify constant character strings that will be written into the RTF exactly as they appear in the Procedure File. V-fields specify variable character strings to be substituted from specified strings on the Control Cards. Each Macro Definition Line of a Procedure File can contain any number of K- and V-fields, in any combination. V-fields are delimited by 0-signs. K-fields are delimited by adjacent V-fields, or the end or beginning
of the record. Since there are only two types of fields, only one need have delimiters. Two adjacent V-fields, however, require two adjacent @-signs.

K-fields

K-fields may be any string of legal IOPS ASCII characters, except the @-sign.

V-fields

A V-field has the following format*

\[ V = @A^n U^n D^n \{ \text{V-field} \} @ \]

The two @-signs delimit the field. The first part of the field (A, D, U or O) is a card-position identifier, and must be present. It identifies the position on the current Control Card of the character string to be substituted in the RTF. The legal combinations are:

- A00, A01, ..., A09
- U00, U01, ..., U09
- D00, D01, ..., D09, D10, ..., D17
- O

With the exception of D10 through D17, each of the above position identifiers corresponds to a unique character string of the Control Card, according to the following scheme:

\[ $CMD;O A00:D00(U00);A01:D01(U01);...;A09:D09(U09) \]

The D10...D17 position identifiers do not correspond to character strings found on the Control Card, but rather to character strings defined by BOSS. Thus,

- D10 - Unused
- D11 - .SIXBT representation of System Device Code ('DK' or 'DP' or 'RK')
- D12 - Current Logged in UIC
- D13 - .SIXBT representation of Carriage RETURN
- D14 - .SIXBT representation of ALT MODE
- D15 - Unused
- D16 - Unused
- D17 - Unused

* Standards for this format description are identical to those specified in Chapter 5 of the DOS-15 User's Manual, DEC-15-ODUMA-B-D.
The parentheses in a V-field must be present. They are used to specify a default string. The default string is used in case the string on the Control Card specified by the position identifier is null. A set of parentheses must be included, even if the default string is null. The default string itself can be a variable, resulting in nested variables. Nesting has a theoretical limit of $2^{17}$ variable fields.

8.1.2 Direct Substitution

When processing a Direct Substitution Procedure File, BOSS places the fields on the Control Card into the RTF just as they stand with only leading spaces ignored. That is, BOSS does not necessarily expect to find file names, and so on, as with normal substitution. Fields on the Control Cards are separated by semi-colons (;), and are processed in a serial manner. The ampersand (&) is used for a special purpose. It causes the current record being composed for the RTF to be terminated with a Carriage RETURN, and written out, and a new record started. This is so that the limit of seventy-five characters per line will not be exceeded.

There are only two legal field types within the Procedure File. They are as follows:

1. A00 through A99
2. D10 through D17 (System Defined)

In making up Direct Substitution Procedure Files, the following rules must be followed:

1. The first line must contain a three (3). This declares the file to be direct substitution.
2. The "A" fields must appear in sequential order, starting at A00. Each "A" field can be used only once within the Procedure File.
3. The "D" fields can only be "D10" through "D17". They can be used any number of times, in any order.
4. Variable expressions must follow the standard V-field format, as in expanded substitution.

8.1.3 Example of Procedure File

The following example shows a typical Direct Substitution Procedure File, the Control Cards used to call it, and the resulting lines produced in the Run Time File.
Procedure File\textsuperscript{1} - Map PRC

3 PROCEDURE FILE TO RUN CHAIN WITH NO OVERLAYS
CHAIN
\texttt{@A00(TMPXCT)@D14()@}
\texttt{@A01(SZ)@D14()@}
\texttt{@A02(FILTMP)@D14()@}
\texttt{@D14()@}

Control Cards as They Appear

\texttt{SMAP TEST1;SZ,VTC/ABC,DEF,NAM1,&NAM2,;}
\texttt{S*G1 NAM3,NAM4,NAM5;/TEST1,SUB1,SUB2,&;}
\texttt{S*G2 SUB3,SUB4,SUB5}

Run Time File Lines

\texttt{CHAIN}
\texttt{TEST1 (ALT MODE)
SZ,VTC/ABC,DEF,NAM1,)
NAM2,NAM3,NAM4,NAM5/ (ALT MODE)
TEST1, SUB1, SUB2,)
SUB3, SUB4, SUB5 (ALT MODE)
(ALT MODE)}

Note: D14=Altmode, <ALTMODE> is an Altmode, and <CR> is a Carriage Return.

8.2 BOSS-15 ACCOUNTING

BOSS has a very simple accounting mechanism. It keeps an account record for each job in a random access file in the CTP UPD. Hence, the file is protected, and can only be accessed after successful execution of a $MIC command.

The name of the accounting file is ACCNTG nnn. (The first has an extension of $G1.) Each file is ten physical blocks long, and contains enough information for 31$G jobs, thirty-one per physical block. When BOSS fills up one file, it increments the extension, and starts a new one. Every time a job ends, BOSS checks whether ACCNTG $G1 exists. If it does not, BOSS creates one. If it does, BOSS checks whether it is full. If not full, BOSS makes a new entry; if full, BOSS

\textsuperscript{1}Direct Substitution File
searches for the first unused extension number. If all extension numbers have been used (up to 999) BOSS prints this message to the operator on the teleprinter:

MAX NUMBER OF ACCOUNTING FILES REACHED
PLEASE PROCESS AND DELETE THEM

Every time the system manager processes an accounting file, he should delete the file.B010

For each completed job, BOSS writes out an eight-word record to the accounting file. The records have the following format:

<table>
<thead>
<tr>
<th>Word #</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Job I.D.,</td>
</tr>
<tr>
<td>2</td>
<td>in</td>
</tr>
<tr>
<td>3</td>
<td>.SIXBT</td>
</tr>
<tr>
<td>4</td>
<td>Date, packed mmdyy</td>
</tr>
<tr>
<td>5</td>
<td>Start Time, in hhmmss</td>
</tr>
<tr>
<td>6</td>
<td>End Time, in hhmmss</td>
</tr>
<tr>
<td>7</td>
<td>Run Time, in hhmmss</td>
</tr>
<tr>
<td>8</td>
<td>Terminal Job Status Word</td>
</tr>
</tbody>
</table>

A word whose contents equal 7777778 immediately follows the last job accounting record in each physical block of the accounting file.

8.3 B.PRE

Figure 8-5 is a flowchart of B.PRE, the J0SS Line Editor.
START

Initialize .DAT slots -14 and -15

Zero FLAG

Read updated file name and extension from .DAT-2

Has file name been entered

Y

Assume updated file name equals the original file name

N

Read file name and extension from .DAT-2

CDREAD

Read a card from .DAT+5

EOF or EDIT card

EDIT

neither

1. set FLAG to 1
2. Issue .ENTER to .DAT-14

Write card image to .DAT-14

Read a card image from .DAT+5

EOF

Y

EOF

N

FLAG = 0

CLOSE .DAT-14

.EXIT

Figure 8-5
B.PRE
9-22
Figure 8-5 (Cont.)
B.PRE
APPENDIX A

DECTAPE "A" HANDLER (DTA.)

The following flow charts describe the operations of the DECTape "A" Handler.
Entry from CAL Handler

1. Save pointer to CAL
2. Save subfunction or data mode

---

Call for .DAT slot B

\( Y \) Call for .DAT slot C

\( N \) Call for .DAT slot A

\( Y \) .DAT slot A inactive

\( N \) .DAT slot B inactive

\( Y \) .DAT slot C inactive

\( N \) Swap descriptor blocks for .DAT slots A and B

---

ERROR IOPS 17

---

IOPS 22 ERROR

---

Swap descriptor blocks for .DAT slots A and C

---

Is there more than Y one output file on the same unit

---

Get function code, and make up dispatch instruction

---

Is there a buffer for this slot

\( A \) WAIT

\( N \) Request a buffer

---

Next Page
From Preceding Page

ERROR
IOPS 55

Was a buffer available

Y

Set up word pointers within the new buffer -- e.g., buffer+377 = link

A

Dispatch to function code

.INIT .OPER .SEEK .ENTER .CLEAR .CLOSE .MTAPE .READ .WRITE .WAIT .TRAN

(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)

B C D E F G H I J K

B .INIT (Function Code #1)

1. Give user standard buffer size (377)
2. Set input or output file indicator
3. Wait for previous I/O to finish for DEC tape

First .INIT for this core load

N

1. Do .SETUP to API and Skip Chain
2. Test buffer size. If not 440ₐ or greater, terminate with an IOPS 7₈

Return to user after CAL
.OPER, Function Code #2

D.ELETE .RENAME .FSTAT

N
Y

Dispatch to requested sub-function and process

Exit to user after CAL

.DSEEK, Function Code #3

Y
N

Is there an active file on this slot

Y
N

Was .INIT for output

Y
N

I/O underway

Y
N

(Loop back to user CAL)

Bring directory into core, if not already in

Y
N

File in directory

Y
N

ERROR IOPS 13

1. Obtain starting block number
2. Read the first file block into core

Return to user after CAL
.ENTER (Dispatch Code 4)

Was this slot INITed for input?

Y

ERROR
IOPS 7

N

I/O Underway

Y

N

Bring Directory into core, if not already in

is file in directory?

N

Set indicator so that file is deleted upon a .CLOSE to this .DAT slot

Search directory bit map (in core) for first free block

Y

Was a block available?

N

ERROR
IOPS 15

Y

Set up to write out this block, when the time comes

Exit to user after CAL

F

 CLEAR
(Dispatch Code 5)

Y

I/O Underway

N

Unclosed file on this slot?

Y

ERROR
IOPS 10

1. Clear out file bit maps
2. Clear directory block with the SYS block bits set in the directory map

Exit to user after CAL

A-5
(Loop back to user CAL)

I/O underway

Input File

Has a .WRITE been executed?

1. Clear bit in bit maps
2. Clear switches

1. Put End-of-File indicator in buffer
2. Write out last block in file

Was there an old file by this name?

The following is done on the in-core bit maps:
1. Zero its bits in the directory bit map
2. Overwrite its file bit map with the new one

Write out updated directory and file bit map

Return the buffer to the system

Return to user after CAL

.CLOSE
(Dispatch Code 6)
(Loop back to user's CAL)

**READ**
(Dispatch Code 10)

- **L/O underway**: Y → H
  - Has EOF been encountered?:
    - Y → Pass 001005,776773 sequence to user's buffer
    - N →
      - 1. Transfer line to user's buffer
      - 2. Set data validity bits
      - Was EOF just read?:
        - Y → Set EOF indicator
        - N →
      - Any more data in buffer?:
        - Y → Read in next block of file
        - N → Exit to user after CAL
Loop back to user CAL

1. Set up block to transfer in or out
2. Set up core address=1
3. Set up word count
4. Start transfer

Exit to user after CAL

INTERRUPT SECTION

Entry from PI or API

Save information to restore later

Stop tape

I/O underway

Switch still on

Read status register "B"

Error

Was this a search

Correct block

Y

N

Right direction

N

Change direction

Set up Current Address & word count for read or write

N

Right direction

N

Change direction

Set up current address and word count for search

N

Clear I/O underway switch

Was block 100 read in

Y

Set directory in core switch

N

N

Y
(ERROR Logic)

Was there a select error

Y

On Search

Y

DOT

Y

Timing error

N

Change direction

N

Timing error

ERROR IOPS 12

Try again

Y

Parity error

N

Y

Tried 0 times

Timing error

Try again

1. Accept data as is
2. Get rest of data

Y

N

Start up DECTape

1. Restore PIC interrupt entry and AC
2. Turn interrupt on, if this was a PIC
3. Debreak and Restore

Exit to interrupted code

1. Read Status Register "A" and save it
2. Clear Status Register "A"
3. Disable interrupts
4. Set return in .MED (register 3, bank β)
5. Clear I/O underway switch and enable CTRL P

.MED

Give IOPS 4

1. Set I/O underway switch
2. Set up Current Address and Word Count for search

M

A-10
APPENDIX B

DISK "A" HANDLERS

The following flow charts describe the operation of the Disk "A" Handlers.
Entry from CAL Handler

1. Save the pointer to the CAL
2. Save the .DAT slot number and subfunction code (bits 5-8 of LOC+∅)

First Call in this core load

Y

Do SETUP for PI and API interrupts

N

N

RK Disk

Y

Set up Standard TCB format in the RKTCB

N

RF disk

N

Determine number of platters

At least one platter

N

IOPS21

Y

Calculate the maximum block number, for use at .CLEAR time

Calculate size of the TEMP list for pre-allocated blocks, and set the BUF.OK switch (SGEN size ok)

Y

.WAIT or .WAITR

Y

WAIT

N

N

Disk I/O Underway

N -- Fall through to "IO.OFF"

Next Page
From Preceding Page

1. Calculate pointers to the arguments of the CAL
2. Save step counter and MQ for EAE

N

.TRAN

Y

Save current set

Y

"Current" slot equal to new slot

N

First call after new core load or a .TRAN

N

Save the current set in its appropriate buffer

Make new .DAT slot the "current" one

FINDBY

Find or set up the Busy Table entry for this .DAT slot

Save status of Write Check

.TRAN

Y

TRAN

N

Next Page
From Preceding Page

Does the current slot have a buffer

Y

N

Y .CLOSE

N

Y .INIT

N

Buffers large enough for file structure

N IOPS 70

Y Request a buffer (.GTBUF)

Y

Buffer available

N IOPS 55

Y

1. Save pointer to buffer, and zero entire buffer
2. Complete the Busy Table entry

1. Get UIC from the Busy Table entry
2. Bring in the Current Set from buffer
3. Set up pointers to: User's Directory Entry, temporary block list, Data Block Words 0,1,2,3,376 & 377

DISPCH Branch on function code

Note: .TRAN, .WAIT and .WAITR have already been intercepted.

INIT OPER SEEK ENTER CLEAR CLOSE MTAPE READ WRITE
INIT

Open output file on this slot

1. Wipe out entry in UFD, and
2. Give back pre-allocated blocks

1. Return any allocated buffer
2. Zero any old busy table entry
3. Make a new entry in the busy table
4. Save Write Check bit in busy table
5. Indicate "current" .DAT slot is zero

Exit to LOC+4

OPER

Any sequential or random I/O to this .DAT slot going on

N

Sub-functions
1, 2, 3, 4, 5

N

Y

Next Page

IOPS 10

IOPS 6

B-5
Calculate RIB block number, and the desired pointer's position within that RIB block

Proper RIB block in core

Depending on the location of the desired block, relative to the RIB block in core, read in the next or preceding RIB block

Proper RIB block now in core

Save pointer to desired data block

Store starting word number and the number of data words desired

Assume transfer starting at word zero, through word 375, and set parameters accordingly

Use word count given in CAL during disk pack input
Set up user's buffer to receive the link words for block

File one block long

N

N - no forward link Rib block contains N - no backward link both link pointers

Is this the last block

Y

Set backward link to adjacent pointer in RIB, and forward link to -1

1. Set backward link to adjacent pointer in RIB block
2. Read in next RIB block and adjust RIB number indicator in the current set
3. Set forward link to first pointer in the new RIB block

N

1. Set forward link to adjacent pointer in RIB
2. Read in previous RIB block and adjust RIB number indicator in the current set
3. Set backward link to last pointer in the new RIB block

Is this the first block

Y

Set forward link to adjacent pointer in RIB, and backward link to -1

RP/RK

Set up driver with the correct block number

RF or RP/RK

RF

Set up driver with the correct word number

Bring in or send out required block or data

Exit to LOC+5
SEEK

Any unclosed file on this .DAT slot

N

Was this file successfully .FSTATed

Y

FINDER

obtain file information

ERROR

IOPS 13, 51 or 71

Found File

File truncated

Y

IOPS 10

N

MIC logged in

Y

Logged in UIC = file's UIC

N

UFDP protected

Y

File protected

Y

IOPS 64

N

Read in first block of file

Exit to LOC+3
ENTER

Has this slot been opened and not closed? Y

N

ENTER

FINDER

Search MFD and UFD
START = Pointer to SAT word 3

(NO entry in MFD for this UIC)

(Entry in MFD, but UFD is empty)

(Entry in MFD, but UFD is not empty)

(UFD exists, but does not contain a file by the given name)

(UFD exists, and contains a file by the given name)

NOMFD

UICEMP

NOTOLD

YESOLD

1. Save number of the last MFD block read
2. Save file name

ENTSET

Preallocate some blocks

Read in last MFD block

Y

Did FINDER find an empty MFD slot?

N

GETNXT

Get next block number

1. Make forward link of last MFD block point to the next block
2. Write out the block
3. Clear the buffer

Set pointer to free slot found by FINDER

Is block with free entry in core?

N

Read it in

1. Set up entry pointers
2. Insert new UIC, entry size and zero protection code (unprotected) into new MFD entry

*MFDSET

That is, has a .RAND, .SEEK or .ENTER been issued without a .CLOSE?
UICOMP

CKDIRP
Is directory protected?

Y 

IOPS63

N

ENTSET
Preallocate some blocks

Read in the MFD block which contains proper entry

MFDSET

GETNXT
Obtain a block for the UFD

Set up entries in the Busy Table, the MFD block and the Current Set

Write out the MFD block and clear the buffer

1. Set up a new UFD block in the buffer, with a back link of -1
2. Set pointer to indicate location to receive the new entry

COMUFD
YESOLD

Is old file truncated

Y

N

WIPOUT
Remove file entry from the UPD

ENTERS

NOTOLD

1. Set "Old file in" switch
2. Set pointers to UPD block number and the first word of the old file's entry (to be used at .CLOSE)

CKDIRP
Check directory protection

Protection Violation

IOPS63

ENTSET
Preallocate some blocks

Did FINDER locate a free entry in UPD

N

Y

Read in first UPD block

SEARCH
Look for a free entry

Found free one

Y

N

GETNXT
Obtain a block #

1. Change forward link of last UPD block (still in core)
2. Write out last UPD block
3. Clear user's buffer

COMUFN

1. Read in UPD block with free entry
2. Load "UPDI" pointer in current set with this block number
COMUPD

1. Save pointer to new UFD entry in "UFD2" of the Current Set
2. Set up pointers to UFD entry slots
3. Store file name and extension in the UFD entry

GETRIB
Obtain a RIB block

1. Store RIB pointer in UFD entry
2. Store protection code & date
3. Insert data block number
4. Write out UFD block with entry
5. Clear buffer

BLDRIB
Set up the RIB

Set "WREXSW" (Write-executed switch)

GETRIB
Obtain a data block

Clear the buffer to zero

Return to user
After CAL

ENTSET

1. Read in the first Submap
2. Make it the "Current Map"
3. Zero indicator of the number of preallocated blocks

LSTFIL
Preallocate some blocks

(Number of blocks pre-allocated is the minimum of number available and the size of the "Temp List")

At least 4 pre-allocated

RETURN

Return any preallocated blocks

IOFS15

B-13
CLEAR

MIC

login

N

IOPS63

1. Clear a buffer
2. Set words 0, 1, 2, 376, and 377 to -1
3. Set bits 0-2 of word 3 to MFD size
4. Set bits 3-17 of word 3 to point to first submenu
5. Write out buffer to block 1777, if RF/RK or 47040 if RP
6. Clear the buffer

(How the handlers write out the bit maps)

Which disk

RP/RK

RF

1. Set backward and forward links to -1
2. Turn on bits that correspond to MFD block and first submenu block

More than four platters

N

Y

Set bit in appropriate bit map for MFD

More submaps

Y

N

1. Set bit in the first bit map that corresponds to second bit map
2. Set forward link to next block
3. Write out the buffer and clear it
4. Set back link to first submenu, and forward link to -1

Write out the buffer

Return to user
From Preceding Page

Did RIB information fit in the last data block

N

Y

Reset RIB block pointer in UFD to last data block

Set "word in RIB" in UFD

Was there an old file with the same name

N

WRTUFD

Write out current UFD block

Y

Is the UFD block with the old file reference in core

Y

1. Write out UFD block currently in core
2. Reset UFD1 to UFD block with old file
3. Read in the UFD block with the old file's entry

SAMUFD

N

1. Reset UFD entry pointer (UFD2)
2. Wipe out the old file's entry

UNUSED

(Give back any unused blocks)

Read in first RIB block used
Save the forward data link

Next Page
CLOSE

Has a write been executed

Set for internal looping (implicit WAIT) until done

Will 2-word EOF record fit in current data block

1. Write out current block
2. Obtain another and
3. Clear the buffer
(Subroutine SETWRD)

1. Write 2-word EOF line in buffer
2. Set forward data link to -1
3. Increment file size

Will RIB fit in last data block

TRAN RIB words into last data block

Write out last data block

1. Read in UFD block for this file
2. Fill in file size and turn off Truncated file bit
3. Save pointer to first RIB block

Next Page
From Preceding Page

Was last data block used for RIB?  

Y  

Reset RIB Block pointer in UF D to last Data Block

INFPROM

Set 'Word-in-RIB' in UF D

(UFD entry is now complete. UF D is still in core.)

Was there an old file with the same name?

N  

Y  

Is UF D block with old file's entry now in core?

Y  

Write out current UF D block

WRUF D

1. Write out UF D currently in core
2. Reset UF D1 to UF D block with the old file's reference
3. Read it in

SAMUF D

1. Set UF D2 to old file's entry slot
2. Wipe out the old entry

(Give back unused blocks.)

Should any blocks be given back from any RIB block used?

N  

Y  

Next Page

NOUNU
From Preceding Page

Any unused blocks in this RIB block

N

Read in next RIB block

THISRIB

1. Find area in this block where blocks should be given back
2. Adjust word $ of this block to reflect only those used
3. Write out the block
4. Fudge subroutine LSTFIL so it appears UNBUSY called

LSTFIL
(Actual transfer is to LSTMOV)

C.INPT

Set Return to LOC+2

1. Perform .GVBUF
2. Zero current set
3. Make "current" .DAT slot zero

Return to LOC+2
READ

Has an EOF been read
Y
N
Is User's buffer size zero
Y
N
SETUP
Check Header word pair
Set up the word pair counters for moving data
Dump Mode
non-Dump Mode
LINFIT
1. Make Word Pair Count negative
2. Zero checksum word in record to be read
3. Clear line error flag

PWORDS
Pass record to user
Y
N
Record too long for buffer
Y
1. Set pointers for a skip over the next record
2. Set "Short Line " Flag
3. Set return in PWORDS to go to ENDIN1

PWORDS
Skip rest of line

ENDIN1
READ-WRITE Common Setup Routine

1. Save pointer to argument data block
2. Set up return address
3. Set pointer to checksum word in data buffer
4. Save checksum word

Was a .SEEK or .ENTER executed

N

1. Set up pointers to "receiver" data buffer
2. Index SETUP return pointer past arguments to Dump Mode exit
3. Save Word Count from CAL

Y

Dump Mode

Index return pointer to Non-Dump Mode

GETWPF
Extract Word Count from line buffer header word pair

0 \leq w.p.c. \leq 177_{10}

RETURN

IOPS23

IOPS11

Return
TRAN

1. Set up return address
2. Clear "Current Slot" number
3. Get Word Count

Word count zero

Y

Return to user

N

1. Set up for input or output
2. Get argument block number
3. Read it or write it

Return to user
(Loop on CALL)

WAIT

I/O underway

RETURN to LOC+N

WAITR

Go to Argument address
1. Initialize the map count number, block count, TLIST count pointers and TLIST count.

2. Read in the current submap.

Start at bit 0 word 3

Read in next submap

Any more sub-

map

Exit

X blocks allocated from this submap

Y End of sub-

map or end

of TENTRY

Check sub-

map not full

Pull the sub-

map

Contaminated

Compute starting location for search

TERROR?
COMMON ROUTINE FOR READING AND WRITING TO AND FROM THE DISK

1. SAVE THE CALLING ADDRESS
2. GET THE ARGUMENTS
3. COMPUTE DISK HARDWARE BLOCK NUMBER
4. SET I/O UNDERWAY FLAG

Y

RK DISK ?

N

START DISK I/O

EXIT TO LOC. IN EXITAD

SETUP TCB AND CALL PIREX TO START DISK DISK I/O

INT ( INTERRUPT HANDLER)

1. SAVE PC AT EXITAD
2. SAVE AC
3. TURN OFF I/O UNDERWAY FLAG
4. LOWER PRIORITY TO LEVEL 4

ERROR ?

Y

N

1. CLEAR DISK FLAG
2. PROCESS ERROR
3. IF NECESSARY, RETRY 10 TIMES

Y

N

WRITE CHECK REQUIRED?

N

CLEAR DISK FLAG

RESET BUFFER COUNTS

RETURN LOC. IN EXITAD

N

SET FOR WRITE CHECK

WRITE CHECK REQUIRED?

Y

EXIT

DISK "A" HANDLERS

B-28
APPENDIX C
PROCEDURE FILE

ASG

1 ASSIGN DEVICE UIC TO ,DAT
A @D02(@n11())@ <SUZ2(@n12())@> @AZZ2(())>

ASM

2 MACRO AND LINE EDITOR
A @D02(@n11())@ <SUZ2(@n12())@> -14/8233(@n11())@ <SUZ3(@n12())@> -15
A,PRE
@AS0(FILMP)@  
@AS1(@AS0(FILMP)@)@  
A @D02(@n11())@ <SUZ2(@n12())@> -11/82331(@n11())@ <SUZ1(@n12())@> -1
A @D012(@n11())@ <SUZ2(@n12())@> -14/SUZ5(@n11())@ <SUZ5(@n12())@> -13
A @D04(@L)@ <SUZ4(&n12())@> -12
**ACRO
@D14@@AZZ2(FILMP)@014(())@  

BNK

2 BANK MODE OPERATION-OK
BANK 0

BUF

1 number of buffers
BUFFS @AZ2(())@

CHN

1 SPECIFY 7 OR 9 TRACK Magnetape
C @A95(())@

CMP

1 SOURCE COMPARE
A @D02(@n11())@ <SUZ2(@n12())@> -15/SUZ1(@n11())@ <SUZ1(@n12())@> -14
SQC01
@D14@@A95(())@A91(())@014(())@  

DIR

1 LIST DIRECTORY
P
1 LP@AZ2(@n11())@ <SUZ2(@n12())@>014(())@  

C-1
KEP

1 RETAIN DEVICE ASSIGNMENTS
KEEP A005()@

LCM

13 SUPPLEMENT TO LIB PROG-UPDATE .LINP
*AA22(CLOSE@O13())@* AAP1(O13())@* AA22()@

LIB

1
A 0001(0011())@ <0002(0012())@> -14
A 0011(0012())@ <0001(0012())@> -15
A 0021(0011())@ <0002(0012())@> -17
A 0023(LP)@ <0023(0012())@> -12
UPDATE
00(LUS)@*|AP3(';';'LINP')@014()@

LINK

13 DIRECT SUB FILE - BUILDS LINKS FOR EXECUTE FILE-USE WITH OVL PROG
*AA22(0014())@0014()@

LOG

2 LOGIN NIC
LOGIN SAP(SAP)@

LST

2 LIST CONTENTS OF FILE ON LINE PRINTER
LP
* U:0003(0011())@ <0023(0012())@> *AAP:FILE(1)@014()@

MAP

13 DIRECT SUB FILE FOR CHAIN OPTION AND RED CODE ONLY
CHAIN
*AA22(TAPVCT)@014()@
*AA22(SF)@014()@
*A002(FILE)@014()@
*014()@

MIC

2 LOGIN MIC
FILE SAD(1)@
MNT

1 MOUNT TAPE ON DRIVE #
LOG MOUNT &M(7)_TAPE* PA0010 ON DRIVE &AZ1(4) - WRITE RAPP(LCK)

MSG

13 MESSAGE TO OPERATOR-DIRECT SUB FILE
LOG &RA011(*)

MSW

13 MESSAGE TO OPERATOR W/ W A I T = D I R E C T SUB
LOG &RA011(*)

NDR

1 CREATE NEW DIRECTORY
DIR
  *&ZA01(8)11(*) <221228012(19)>28014(*)

OVL

13 DIRECT SUB FILE - USE FOR BUILDING OVERLAYS(CHAIN)
CHAIN
  */&AM(1)PROXY?28014(*)
  */&AM(2)8014(*)
  */&AM(3)128014(*)

PAG

2 PAGE MODE OPERATION ON
PAGE ON

PRT

1 SPECIFY PROTECTION CODE
P *AAX(2)

QDP

1 DUMP CORE ON TERMINAL ERRORS NO ARGUMENTS

XCT

2 EXECUTE
  */&AM(1)S12(*) <221228012(19)> -4
  */&AM(2)PROXY?2
INDEX

Accessibility map, 6-9
Additions to Non-resident Monitor, 3-4
Automatic Priority Interrupt (API), 7-1
   hardware, 7-4
   implementation, 7-11
ON/OFF, 4-19
software, 7-7

Bad Allocation Table (BAT), 6-18
Bank/Page mode, 7-1
Batch mode .DAT slot assignments, 4-20
Block checksum, 6-7
Block control pair, 6-6, 6-7
Block list, 6-14
Block word count (BWC), 6-6
BOSS-15, 8-1
   accounting, 8-20
   .DAT slot assignments, 4-20
   line editor (B.PRE), 8-21
Bootstrap, system, 2-1, 2-7, 4-13
Buffer allocation, 4-20, 5-13, 6-14

CAL handler, 2-2, 7-1
Characters, control, 2-16
Clock operation, 2-14
Clock routine, 2-8
COMBLK, 4-13, 5-1
Commands to Non-resident Monitor, 3-4
Control characters, 2-14
Current set, 6-14

Data modes
   Dump, 6-4
   Image, 6-4
   IOPS, 6-4
DDT loading, 4-13
Dectape file organization, 6-1
Device assignment table (.DAT), 5-13
Device table, 5-12
Disk file structure, 6-11
Disk handler, 2-6
Disk resident tables, 5-1, 5-9
Directed data recording, 6-5
Directed Dectape, 6-1
Dump mode, 6-4

Error handler, IOPS, 2-2
Error processor, 2-2, 2-6
EXECUTE, 4-13

File accessibility map, 6-7
File Bit Map, Dectape, 6-2
File buffer transfer vector table, 5-13
File identification and location, 6-7
File information, see Current set
File locating, 6-7

File storage, 3-8
FIOPS, 6-5

Handlers, I/O device, 7-1

Image mode, 6-4
Input/Output (I/O)
   communication table, 5-12
   initialization, 2-8
I/O device handlers, 7-1
   writing special, 7-10
IOPS mode, 6-4
   error handler, 2-2

Linking Loader, 4-13
Link status, 7-1
Loader buffer allocation, 4-20
Loader, system, 4-1, 4-13

Magnetic tape, 6-4
   file directory, 6-7
   handlers, 6-5
   storage retrieval, 6-11
Mass Storage Busy Table, 5-14
Master File Directory (MFD), 6-12
.MEM error processor, 2-2
Memory protect, 7-1
Monitor, resident, 4-13

Non-directoried Dectape, 6-1
Nonresident Monitor, 2-14, 3-1
   additions, 3-4
   commands, 3-4

Operation of DOS, 1-1
Overlay Table, 5-9, 5-15

Patch area, Resident Monitor, 2-16
PATCH, commands to, 3-8
PIC interrupt service routine
   implementation, 7-11
PIP, 6-18
Pre-allocation of blocks, 6-16
Priority, software level, 7-1
Procedure files, BOSS, 8-16
Program control characters, 2-16

Q file, 3-8
Queueing, 7-8

ROM table, 5-14
Reserved word locations, 5-14

X-1
Resident Monitor, 2-1, 4-13
PATCH area, 2-14
Timing features, 2-8
Retrieval Information Block (RIB), 6-14
Run time file (RTP), 8-1, 8-16

.SCOM registers, 5-1 to 5-6
Used by Loaders, 4-17 to 4-19
SGNBLK, 4-13, 5-1, 5-8, 5-10
Skip chain, 5-13
Software level priority, 7-1
Spare TCB's, 2-17
Special I/O device handlers, 7-10
Startup routines, 2-8
Storage, 4-26, 6-11, 6-16
Storage allocation tables (SAT's), 6-1
Submaps, 6-17
SYSBLK, 4-13, 5-1
System
  bootstrap, 2-7
  initialization, 2-8
  Loader, 4-1, 4-13

Tables used by Loaders, 4-16
Task Control Block, 2-17
TCBTAB, 2-17
Temp List (TLIST), see Block list
.TIMER routine, 2-14
Timing features, 2-8
TRAN routine, 2-7

User File Directory Table (.UFDT) 5-12
User file labels, 6-9, 6-10
User identification code (UID), 6-12
HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes newsletters and Software Performance Summaries (SPS) for the various Digital products. Newsletters are published monthly, and contain announcements of new and revised software, programming notes, software problems and solutions, and documentation corrections. Software Performance Summaries are a collection of existing problems and solutions for a given software system, and are published periodically. For information on the distribution of these documents and how to get on the software newsletter mailing list, write to:

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to Digital's software should be reported to a Software Support Specialist. A specialist is located in each Digital Sales Office in the United States. In Europe, software problem reporting centers are in the following cities.

Reading, England
Paris, France
The Hague, Holland
Tel Aviv, Israel

Milan, Italy
Solna, Sweden
Geneva, Switzerland
Munich, West Germany

Software Problem Report (SPR) forms are available from the specialists or from the Software Distribution Centers cited below.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation
Software Distribution Center
146 Main Street
Maynard, Massachusetts 01754

Digital Equipment Corporation
Software Distribution Center
1400 Terra Bella
Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computer Users Society, maintains a user exchange center for user-written programs and technical application information. A catalog of existing programs is available. The society publishes a periodical, DECUSCOPE, and holds technical seminars in the United States, Canada, Europe, and Australia. For information on the society and membership application forms, write to:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS
Digital Equipment, S.A.
81 Route de l'Aire
1211 Geneva 26
Switzerland
NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Please indicate the type of user/reader that you most nearly represent.

☐ Assembly language programmer
☐ Higher-level language programmer
☐ Occasional programmer (experienced)
☐ User with little programming experience
☐ Student programmer
☐ Non-programmer interested in computer concepts and capabilities

Name ____________________________ Date ____________________________

Organization _______________________________________________________
Street ____________________________________________________________
City ____________________________ State __________ Zip Code ____________
or
Country

If you do not require a written reply, please check here. ☐