RT-11
Software Support Manual

DEC-11-ORPGA-B-D

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 © 1973,1974,1975, by 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:

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

UNIBUS
## CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER 1</th>
<th>RT-11 OVERVIEW</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>1.2</td>
<td>SYSTEM CONCEPTS AND TERMINOLOGY</td>
<td>1-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 2</th>
<th>MEMORY LAYOUT</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>FOREGROUND JOB AREA LAYOUT</td>
<td>2-3</td>
</tr>
<tr>
<td>2.2</td>
<td>JOB BOUNDARIES IN F/B</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3</td>
<td>'FLOATING' USR POSITION</td>
<td>2-6</td>
</tr>
<tr>
<td>2.4</td>
<td>MONITOR MEMORY ALLOCATION</td>
<td>2-7</td>
</tr>
<tr>
<td>2.5</td>
<td>MEMORY AREAS OF INTEREST</td>
<td>2-9</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Monitor Fixed Offsets</td>
<td>2-9</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Table Descriptions</td>
<td>2-13</td>
</tr>
<tr>
<td>2.5.2.1</td>
<td>$PNAME (Permanent Name Table)</td>
<td>2-13</td>
</tr>
<tr>
<td>2.5.2.2</td>
<td>$STAT (Device Status Table)</td>
<td>2-13</td>
</tr>
<tr>
<td>2.5.2.3</td>
<td>$ENTRY (Handler Entry Point Table)</td>
<td>2-14</td>
</tr>
<tr>
<td>2.5.2.4</td>
<td>$DVREC (Device Handler Block Table)</td>
<td>2-14</td>
</tr>
<tr>
<td>2.5.2.5</td>
<td>$HSIZE (Handler Size Table)</td>
<td>2-14</td>
</tr>
<tr>
<td>2.5.2.6</td>
<td>$DVSIZ (Device Directory Size Table)</td>
<td>2-15</td>
</tr>
<tr>
<td>2.5.2.7</td>
<td>$UNAM1, $UNAM2 (User Name Tables)</td>
<td>2-15</td>
</tr>
<tr>
<td>2.5.2.8</td>
<td>$OWNER (Device Ownership Table)</td>
<td>2-16</td>
</tr>
<tr>
<td>2.5.2.9</td>
<td>DEVICE Macro</td>
<td>2-16</td>
</tr>
<tr>
<td>2.5.3</td>
<td>F/B Impure Area</td>
<td>2-18</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Low Memory Bitmap (LOMMAP)</td>
<td>2-21</td>
</tr>
<tr>
<td>2.5.4.1</td>
<td>S/J Restrictions</td>
<td>2-22</td>
</tr>
<tr>
<td>2.6</td>
<td>USING AUXILIARY TERMINALS AS THE CONSOLE TERMINAL</td>
<td>2-23</td>
</tr>
<tr>
<td>2.7</td>
<td>MAKING TTY SET OPTIONS PERMANENT IN F/B MONITOR</td>
<td>2-25</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Carriage Width</td>
<td>2-26</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Other Options</td>
<td>2-26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 3</th>
<th>FILE STRUCTURES AND FILE FORMATS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>DEVICE DIRECTORY SEGMENTS</td>
<td>3-1</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Directory Header Format</td>
<td>3-1</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Directory Entry Format</td>
<td>3-2</td>
</tr>
<tr>
<td>3.1.2.1</td>
<td>Status Word</td>
<td>3-3</td>
</tr>
<tr>
<td>3.1.2.2</td>
<td>Name and Extension</td>
<td>3-4</td>
</tr>
<tr>
<td>3.1.2.3</td>
<td>Total File Length</td>
<td>3-4</td>
</tr>
<tr>
<td>3.1.2.4</td>
<td>Job Number and Channel Number</td>
<td>3-4</td>
</tr>
<tr>
<td>3.1.2.5</td>
<td>Date</td>
<td>3-5</td>
</tr>
<tr>
<td>3.1.2.6</td>
<td>Extra Words</td>
<td>3-7</td>
</tr>
<tr>
<td>3.2</td>
<td>SIZE AND NUMBER OF FILES</td>
<td>3-7</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Directory Segment Extensions</td>
<td>3-8</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>3.3 MAGTAPE AND CASSETTE FILE STRUCTURE</td>
<td>3-11</td>
<td></td>
</tr>
<tr>
<td>3.3.1 Magtape File Structure</td>
<td>3-11</td>
<td></td>
</tr>
<tr>
<td>3.3.1.1 Moving MT to Other Industry-Compatible Environments</td>
<td>3-13</td>
<td></td>
</tr>
<tr>
<td>3.3.1.2 Recovering from Bad Tape Errors</td>
<td>3-13</td>
<td></td>
</tr>
<tr>
<td>3.3.2 Cassette File Structure</td>
<td>3-14</td>
<td></td>
</tr>
<tr>
<td>3.3.2.1 File Header</td>
<td>3-15</td>
<td></td>
</tr>
<tr>
<td>3.4 RT-11 FILE FORMATS</td>
<td>3-16</td>
<td></td>
</tr>
<tr>
<td>3.4.1 Object Format (.OBJ)</td>
<td>3-16</td>
<td></td>
</tr>
<tr>
<td>3.4.1.1 Global Symbol Directory</td>
<td>3-20</td>
<td></td>
</tr>
<tr>
<td>3.4.1.2 ENDGDSD Block</td>
<td>3-22</td>
<td></td>
</tr>
<tr>
<td>3.4.1.3 TXT Blocks and RLD Blocks</td>
<td>3-22</td>
<td></td>
</tr>
<tr>
<td>3.4.1.4 ISD Internal Symbol Directory</td>
<td>3-28</td>
<td></td>
</tr>
<tr>
<td>3.4.1.5 ENDMOD Block</td>
<td>3-28</td>
<td></td>
</tr>
<tr>
<td>3.4.1.6 Librarian Object Format</td>
<td>3-28</td>
<td></td>
</tr>
<tr>
<td>3.4.2 Formatted Binary Format (.LDA)</td>
<td>3-30</td>
<td></td>
</tr>
<tr>
<td>3.4.3 Save Image Format (.SAV)</td>
<td>3-31</td>
<td></td>
</tr>
<tr>
<td>3.4.4 Relocatable Format (.REL)</td>
<td>3-32</td>
<td></td>
</tr>
<tr>
<td>3.4.4.1 Non-Overlay Programs</td>
<td>3-33</td>
<td></td>
</tr>
<tr>
<td>3.4.4.2 REL Files With Overlays</td>
<td>3-42</td>
<td></td>
</tr>
</tbody>
</table>

CHAPTER 4
SYSTEM DEVICE

4-1

4.1 DETAILED STRUCTURE OF THE SYSTEM DEVICE 4-1

4.2 CONTENTS OF MONITR.SYS 4-2

4.3 KMON OVERLAYS 4-3

4.4 DETAILED OPERATION OF THE BOOTSTRAP 4-3

4.5 FIXING THE SIZE OF A SYSTEM 4-5

CHAPTER 5
I/O SYSTEM, QUEUES, AND HANDLERS 5-1

5.1 QUEUED I/O IN RT-11 5-1
| 5.1.1 I/O Queue Elements | 5-1 |
| 5.1.2 Completion Queue Elements | 5-5 |
| 5.1.3 Timer Queue Elements | 5-7 |
| 5.2 DEVICE HANDLERS | 5-8 |
| 5.2.1 Device Handler Format | 5-8 |
| 5.2.2 Entry Conditions | 5-9 |
| 5.2.3 Data Transfer | 5-9 |
| 5.2.4 Interrupt Handler | 5-9 |
| 5.3 ADDING A HANDLER TO THE SYSTEM | 5-11 |
| 5.4 WRITING A SYSTEM DEVICE HANDLER | 5-14 |
| 5.4.1 The Device Handler | 5-14 |
| 5.4.2 The Bootstrap | 5-15 |
| 5.4.3 Building the New System | 5-16 |
| 5.5 DEVICES WITH SPECIAL DIRECTORIES | 5-19 |
| 5.5.1 Special Devices | 5-19 |
| 5.5.1.1 Interfacing to Special Device Handlers | 5-19 |
| 5.5.1.2 Programmed Requests to Special Devices | 5-20 |

iv
<table>
<thead>
<tr>
<th>Page</th>
<th>5-21</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6</td>
<td>ADDING A SET OPTION</td>
</tr>
<tr>
<td>5.7</td>
<td>CONVERTING USER-WRITTEN HANDLERS</td>
</tr>
<tr>
<td>6-1</td>
<td>F/B MONITOR DESCRIPTION</td>
</tr>
<tr>
<td>6.1</td>
<td>INTERRUPT MECHANISM AND .INTEN ACTION</td>
</tr>
<tr>
<td>6.2</td>
<td>CONTEXT SWITCH</td>
</tr>
<tr>
<td>6.3</td>
<td>BLOCKING A JOB</td>
</tr>
<tr>
<td>6.4</td>
<td>JOB SCHEDULING AND USE OF .SYNCH REQUEST</td>
</tr>
<tr>
<td>6.5</td>
<td>USR CONTENTION</td>
</tr>
<tr>
<td>6.6</td>
<td>I/O TERMINATION</td>
</tr>
<tr>
<td>7-1</td>
<td>RT-11 BATCH</td>
</tr>
<tr>
<td>7.1</td>
<td>CTL FORMAT</td>
</tr>
<tr>
<td>7.2</td>
<td>BATCH RUN-TIME HANDLER</td>
</tr>
<tr>
<td>7.3</td>
<td>BATCH COMPILER</td>
</tr>
<tr>
<td>7.3.1</td>
<td>BATCH Job Initiation</td>
</tr>
<tr>
<td>7.3.2</td>
<td>BATCH Job Termination</td>
</tr>
<tr>
<td>7.3.3</td>
<td>BATCH Compiler Construction</td>
</tr>
<tr>
<td>7.4</td>
<td>BATCH EXAMPLE</td>
</tr>
<tr>
<td>7.5</td>
<td>CTT TEMPORARY FILES</td>
</tr>
<tr>
<td>A-1</td>
<td>SAMPLE HANDLER LISTINGS</td>
</tr>
<tr>
<td>A.1</td>
<td>RC11/RS64 DEVICE HANDLER</td>
</tr>
<tr>
<td>A.2</td>
<td>RC11/RS64 BOOTSTRAP</td>
</tr>
<tr>
<td>A.3</td>
<td>LP/LS11 DEVICE HANDLER</td>
</tr>
<tr>
<td>A.4</td>
<td>CR11 DEVICE HANDLER</td>
</tr>
<tr>
<td>A.5</td>
<td>TC11 DEVICE HANDLER</td>
</tr>
<tr>
<td>B-1</td>
<td>FOREGROUND TERMINAL HANDLER</td>
</tr>
<tr>
<td>C-1</td>
<td>VERSION 1 EMT SUMMARY</td>
</tr>
<tr>
<td>D-1</td>
<td>FOREGROUND SPOOLER EXAMPLE</td>
</tr>
<tr>
<td>APPENDIX E</td>
<td>S/J AND F/B MONITOR FLOWCHARTS</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>E.1</td>
<td>KMON (KEYBOARD MONITOR) FLOWCHARTS</td>
</tr>
<tr>
<td>E.1.1</td>
<td>KMON Subroutines</td>
</tr>
<tr>
<td>E.1.2</td>
<td>KMON Overlays</td>
</tr>
<tr>
<td>E.2</td>
<td>USR (USER SERVICE ROUTINES) FLOWCHARTS</td>
</tr>
<tr>
<td>E.3</td>
<td>CSI (COMMAND STRING INTERPRETER) FLOWCHARTS</td>
</tr>
<tr>
<td>E.3.1</td>
<td>CSI Subroutines</td>
</tr>
<tr>
<td>E.4</td>
<td>RMON (RESIDENT MONITOR) FLOWCHARTS FOR SINGLE-JOB MONITOR</td>
</tr>
<tr>
<td>E.4.1</td>
<td>EMT Processors</td>
</tr>
<tr>
<td>E.4.2</td>
<td>Clock Interrupt Service</td>
</tr>
<tr>
<td>E.4.3</td>
<td>Console Terminal Interrupt Service</td>
</tr>
<tr>
<td>E.4.4</td>
<td>I/O Routines</td>
</tr>
<tr>
<td>E.5</td>
<td>RMON (RESIDENT MONITOR) FLOWCHARTS FOR FOREGROUND/BACKGROUND MONITOR</td>
</tr>
<tr>
<td>E.5.1</td>
<td>EMT Processors</td>
</tr>
<tr>
<td>E.5.2</td>
<td>Job Arbitration, Error Processing</td>
</tr>
<tr>
<td>E.5.3</td>
<td>Queue Managers (I/O, USR, Completion)</td>
</tr>
<tr>
<td>E.5.4</td>
<td>Clock Interrupt Service</td>
</tr>
<tr>
<td>E.5.5</td>
<td>Console Terminal Interrupt Service</td>
</tr>
<tr>
<td>E.5.6</td>
<td>Resident Device Handlers (TT, Message)</td>
</tr>
</tbody>
</table>

Entry Point Index E-151
FIGURES

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Monitor Memory Layout</td>
<td>2-1</td>
</tr>
<tr>
<td>2-2</td>
<td>Foreground Job Area Layout</td>
<td>2-4</td>
</tr>
<tr>
<td>2-3</td>
<td>Job Limits</td>
<td>2-5</td>
</tr>
<tr>
<td>2-4</td>
<td>Background SYSLOW Examples</td>
<td>2-6</td>
</tr>
<tr>
<td>2-5</td>
<td>Memory Allocation</td>
<td>2-8</td>
</tr>
<tr>
<td>3-1</td>
<td>Directory Entry Format</td>
<td>3-1</td>
</tr>
<tr>
<td>3-2</td>
<td>Directory Segment</td>
<td>3-6</td>
</tr>
<tr>
<td>3-3</td>
<td>Object Module Processing</td>
<td>3-17</td>
</tr>
<tr>
<td>3-4</td>
<td>Formatted Binary Block</td>
<td>3-18</td>
</tr>
<tr>
<td>3-5</td>
<td>GSD Structure</td>
<td>3-20</td>
</tr>
<tr>
<td>3-6</td>
<td>TXT Block Format</td>
<td>3-22</td>
</tr>
<tr>
<td>3-7</td>
<td>RLD Format</td>
<td>3-24</td>
</tr>
<tr>
<td>3-8</td>
<td>Library File Format</td>
<td>3-28</td>
</tr>
<tr>
<td>3-9</td>
<td>Library Header Format</td>
<td>3-29</td>
</tr>
<tr>
<td>3-10</td>
<td>Entry Point Table Format</td>
<td>3-29</td>
</tr>
<tr>
<td>3-11</td>
<td>Library End Trailer</td>
<td>3-30</td>
</tr>
<tr>
<td>3-12</td>
<td>Formatted Binary Format</td>
<td>3-31</td>
</tr>
<tr>
<td>3-13</td>
<td>REL File Without Overlays</td>
<td>3-33</td>
</tr>
<tr>
<td>3-14</td>
<td>REL File With Overlays</td>
<td>3-43</td>
</tr>
<tr>
<td>3-15</td>
<td>Overlay Segment Relocation Block</td>
<td>3-44</td>
</tr>
<tr>
<td>5-1</td>
<td>I/O Queue Element</td>
<td>5-2</td>
</tr>
<tr>
<td>5-2</td>
<td>Completion Queue Element</td>
<td>5-6</td>
</tr>
<tr>
<td>5-3</td>
<td>SYNCH Element</td>
<td>5-7</td>
</tr>
<tr>
<td>5-4</td>
<td>Timer Queue Element</td>
<td>5-7</td>
</tr>
</tbody>
</table>

TABLES

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Fixed Offsets</td>
<td>2-10</td>
</tr>
<tr>
<td>2-2</td>
<td>Impure Area</td>
<td>2-19</td>
</tr>
<tr>
<td>2-3</td>
<td>Bitmap Byte Table</td>
<td>2-21</td>
</tr>
<tr>
<td>2-4</td>
<td>Default Functions for TTY Options</td>
<td>2-25</td>
</tr>
<tr>
<td>2-5</td>
<td>TTCNFG Option Bits</td>
<td>2-27</td>
</tr>
<tr>
<td>3-1</td>
<td>Directory Header Words</td>
<td>3-2</td>
</tr>
<tr>
<td>3-2</td>
<td>File Types</td>
<td>3-3</td>
</tr>
<tr>
<td>3-3</td>
<td>ANSI MT Labels Under RT-11</td>
<td>3-12</td>
</tr>
<tr>
<td>3-4</td>
<td>CT File Header Format</td>
<td>3-16</td>
</tr>
<tr>
<td>7-1</td>
<td>BATCH Compiler Data Base Description</td>
<td>7-7</td>
</tr>
<tr>
<td>C-1</td>
<td>V1 Programmed Requests</td>
<td>C-1</td>
</tr>
</tbody>
</table>
PREFACE

The RT-11 Software Support Manual covers the internal description of the RT-11 software system. Chapter 1 presents an overview of the system and discusses conventions used throughout the manual. Chapters 2 through 6 describe in detail various aspects of the monitor and system structure, including memory layout, monitor tables, file structures, file formats, system device structure, bootstrap operation, I/O queuing system, device handlers and F/B monitor description. Chapter 7 discusses the operation of the BATCH compiler and run-time handler.

The appendixes provide example handler listings, including a foreground terminal handler (Appendix B) and a sample foreground program (Appendix D). Complete flowcharts of both the Single-Job and Foreground/Background Monitors are shown in Appendix E.

The reader should be thoroughly familiar with the RT-11 system. Although the information in this manual is aimed at V02B users, it should be adequate for Version 2 users also; excluding a few minor alterations (to permit the addition of the new V02B devices), the construction of the monitors has changed very little between the two versions. A comprehensive list of differences between the V2 and V02B systems is included in Getting Started with RT-11 (V02B) (DEC-11-ORCPA-E-D).

It is assumed that the user has read the RT-11 System Reference Manual (DEC-11-ORUGA-B-D) or (DEC-11-ORUGA-C-D) and all other documentation included in the RT-11 kit, and is an experienced PDP-11 programmer. It is recommended that RT-11 monitor source listings be available for reference.
CHAPTER 1
RT-11 OVERVIEW

1.1 INTRODUCTION

RT-11 is a single-user programming and operating system designed for the PDP-11 series of computers. It permits the use of a wide range of peripherals and up to 28K of either solid state or core memory (hereafter referred to as memory).

RT-11 provides two operating environments: Single-Job (S/J) operation, and a powerful Foreground/Background (F/B) capability. Either environment is controlled by a single user from the console terminal keyboard by means of the appropriate monitor--S/J or F/B. The monitors are upwards compatible; features that are used only in a F/B environment are treated as no-ops under the S/J Monitor.

A feature common to both operating environments is the inclusion of a full complement of system development and utility programs to aid the programmer in the development of his own applications.

The normal use and operation of the monitors and system programs is discussed in detail in the RT-11 System Reference Manual. Concepts and applications that are specialized and useful to the more experienced programmer are included in this manual.

1.2 SYSTEM CONCEPTS AND TERMINOLOGY

The basic concepts necessary to use RT-11 effectively are defined in the RT-11 System Reference Manual. The user should be familiar with those concepts before proceeding to use this manual.
Abbreviations used throughout this document are:

<table>
<thead>
<tr>
<th>TERM</th>
<th>MEANING</th>
</tr>
</thead>
</table>
| KMON | Keyboard Monitor  
The console terminal interface to RT-11. KMON runs as a background job and allows the user to run programs, assign device names, and generally control the system. |
| USR  | User Service Routines  
The nonresident (swapping) part of RT-11. The USR performs file-oriented operations. |
| CSI  | Command String Interpreter  
The CSI is part of the USR. It accepts a string of characters from memory or from the console and performs specified file operations; or syntactically analyzes a command string and constructs a table from the information supplied. |
| RMON | Resident Monitor  
RT-11 provides a choice of two Resident Monitors: a Single-Job Monitor and a Foreground/Background Monitor. RMON specifically provides the following services:  

- EMT dispatcher  
- Keyboard (console) interrupt service  
- TT: resident device handler (F/B only)  
- Read/Write processor  
- USR swap routines  
- I/O queuing routines  
- System device handler  
- System I/O tables  
- Message handler (F/B only)  
- Job scheduler (F/B only) |
| $CSW  | Channel Status Word  
Each bit in the CSW contains information relevant to the status of a channel; see Chapter 9 (.SAVESTATUS) of the RT-11 System Reference Manual. |
<table>
<thead>
<tr>
<th>TERM</th>
<th>MEANING</th>
</tr>
</thead>
</table>
| JSW  | Job Status Word  
The JSW contains information in bytes 44 and 45 about the job currently in memory. |
| F/B  | The Foreground/Background version of the monitor |
| S/J  | The Single-Job version of the monitor |
| B/G  | The background job |
| F/G  | The foreground job |
| <CR> | Carriage Return |
| <LF> | Line Feed |

Various mnemonic names (e.g., BLIMIT, SYSLOW), referred to from within the text and in diagrams and flowcharts, represent the actual symbolic names as they appear in the monitor source listings.

To avoid confusion, underlining is used in most examples to designate computer printout; square brackets, [ and ], are used to enclose comments. Values for symbolic names used in examples can be found in Chapter 5 of *Getting Started With RT-ll* (V02B) (DEC-ll-ORCPA-E-D).
CHAPTER 2
MEMORY LAYOUT

RT-11 operates properly in any configuration between 8K and 28K (words) of memory (16K to 28K for the F/B Monitor). No user intervention is required when programs are moved to a different size machine; i.e., programs correctly developed in one environment will work in any size environment (providing there is sufficient memory) with no relinking necessary.

Figure 2-1 shows a general diagram of the memory layout in an RT-11 system.

Figure 2-1
Monitor Memory Layout
The memory area diagrammed is arranged as follows:

<table>
<thead>
<tr>
<th>Memory Area</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-477</td>
<td>Reserved for I/O vectors, RT-11 system communication area.</td>
</tr>
<tr>
<td>500-SYSLOW</td>
<td>Space available for user (background) programs. (The high limit of memory for the background is contained in SYSLOW, a location in the monitor data base.)</td>
</tr>
</tbody>
</table>

Space for foreground programs and LOADed handlers is allocated as needed, reducing the amount of space available for a background job.

The areas marked KMON and USR/CSI are the areas that these units normally occupy when they are in memory. The amount of memory that a user program occupies is determined by:

1. The initial size of the program, or
2. The amount of memory the user program requests via a .SETTOP programmed request.

When a user program (background job) is executed (via the KMON commands R, RUN, or GET and START), the top of memory is set to correspond to the size of the program. If the top of user memory never exceeds KMON, both KMON and USR/CSI are resident. If all of memory (up to SYSLOW) is requested (via a .SETTOP), neither the KMON nor the USR is resident and swapping of the USR is required. Programs performing many file-oriented operations gain from having the USR resident, since no time is spent swapping the USR.

The KMON, USR, and RMON modules normally occupy the upper segment of memory. This implies that larger memory configurations automatically have more free memory available.

The area marked DEVICE REGISTERS is the top 4K of memory in any PDP-11 computer. This area is reserved for the status and control registers of peripheral devices.
2.1 FOREGROUND JOB AREA LAYOUT

The foreground job area is located above the KMON/USR, as shown in Figure 2-1, and is allocated by the FRUN command. The actual layout of the job within the foreground area is shown in Figure 2-2. The impure area (described in Section 2.5.3) occupies the lowest 207 words of the job area and contains terminal ring buffers, I/O channels, and other job-specific information.

The foreground stack is located immediately above the impure area with a default size of 128 words; this may be changed using the FRUN /S switch. The program may specify a different location for the stack by using an .ASECT into location 42, in which case the /S switch is ignored and the program itself must allocate stack space. Wherever the stack is located, stack overflow will most probably cause program malfunction before penetrating the task area boundary, since either the program itself or the impure area will be corrupted.

NOTE

Users must not use a relocatable symbol as the contents of location 42 when resetting the initial stack pointer via an .ASECT in a foreground job; such a symbol is not relocated when it occurs in an .ASECT in a foreground job. To set the stack to relative location 1000 in a foreground job, use:

```
.ASECT
.=42
.WORD 1000
```
The space allocated for the foreground program is sufficient to contain the program code itself, as indicated by location 50 (in block 0 of the file); location 50 is set by the Linker and designates the program's high limit. If the foreground job requires working space, this space must either be reserved from within the program (e.g., using .BLKW) or allocated at run-time using the PRUN /N switch. Space allocated with the /N switch is located above the program as shown in Figure 2-2. Location 50 will point to the top of the program area and a .SETTOP will permit access to any working space.

2.2 JOB BOUNDARIES IN F/B

The actual job boundaries are stored (in RMON) in limit tables for both foreground and background jobs. The FLIMIT table contains high and low boundaries for the foreground, and the BLIMIT table contains boundaries for the background. .SETTOPs are permitted for any job up to its high limit. The SYSLOW pointer mentioned earlier is equivalent to the background BLIMIT high pointer entry. This is shown in Figure 2-3.
2.3 'FLOATING' USR POSITION

The RT-11 USR is normally located in the memory area directly below that pointed to by SYSLOW. For the Version 1 monitor, this was directly below the RMON. For the Version 2 and 2B monitors, the USR position varies as handlers, the scroller, and foreground jobs (in F/B) are loaded into memory; the SYSLOW pointer is corrected for each change in memory configuration. In any case, the SYSLOW position is considered the normal USR swapping position.

It is possible, however, to cause the USR to swap into another location in memory. This is done by setting location 46 (in the system communication area) to the address at which the USR is to swap; if the contents of location 46 are nonzero and even, the monitor loads the USR at the new address. Note, however, that if no swapping is required, the USR is not loaded at the address indicated in location 46. Location 46 is cleared by an exit to the Keyboard Monitor (via an .EXIT, .HRESET, .SRESET, or CTRL C).
It is possible to make the USR permanently resident (i.e., non-swapping). Using the SET USR NOSWAP Keyboard Monitor command makes the USR permanently resident at its normal position, that is, below the memory area pointed to by SYSLOW.

2.4 MONITOR MEMORY ALLOCATION

RT-11 uses a dynamic memory allocation scheme to provide memory space for LOAded handlers, foreground jobs (F/B Monitor only) and the display text scroller. Memory is allocated in the region above the KMOn/USR and below RMON. If there is insufficient memory in this region (initially, after the system is bootstrapped, there is none), memory is taken from the background region by "sliding down" the KMOn/USR the required number of words.

When memory allocated in this manner is released, the memory block is returned to a singly-linked free memory list, the list head of which is in RMON. Any contiguous blocks are concatenated into a single larger block. A block found to be contiguous with the KMOn/USR is reclaimed by "sliding up" the KMOn/USR, removing the block from the list.

Memory allocation and release is achieved by calls to the GETBLK and PUTBLK routines located in the KMOn overlays (the GETBLK and PUTBLK routines are flowcharted in Appendix E). The requested number of words is passed to GETBLK in R0, and the address of the block is returned in R4. An extra word of memory is allocated by GETBLK, which then stores the size of the block in that word. R4 points to the first available word in the block (see Figure 2-5a). When releasing memory, R4 must point to the first available word, the same address returned by GETBLK during allocation (as shown in Figure 2-5b). The block will be linked into the free memory list (shown in Figure 2-5c).
a) Allocating a memory block

Call sequence:
R0 = SIZE
JSR PC, GETBLK

 returns with R4 pointing to the allocated block

b) Releasing a memory block

Call sequence:
R4 ← BLOCK
JSR PC, PUTBLK

c) Free memory list

Figure 2-5
Memory Allocation
When a block of memory of sufficient size is not available, GETBLK must create a hole in memory by sliding down the KMON/USR. This is achieved by a call to KUMOVE, a small routine located physically at the front of the KMON. KUMOVE does the actual work of moving the KMON/USR up in memory. For moves downward, an auxiliary subroutine, MOVEDN, located at the top of the USR, is used.

Whenever a request is made for a block of a certain number of words, the memory allocator searches memory for the first highest block that is large enough to satisfy the request (that is, equal to or larger than the requested number). The goal of the memory allocator is to minimize the amount of free (unused) memory in the foreground region, making the maximum amount of memory available to the background. Contiguous blocks of free memory are merged and reclaimed whenever possible. The search time of the singly-linked list is not a factor, since at any time there will be few nodes (free memory areas) in the list, and the allocator minimizes the number.

2.5 MEMORY AREAS OF INTEREST

This section describes memory areas of particular interest and indicates the contents of those locations. The areas covered are:

1. Monitor Fixed Offsets (F/B & S/J)
2. F/B Impure Area
3. Resident Bitmap (F/B & S/J)
4. Tables

2.5.1 Monitor Fixed Offsets

Certain values are maintained at fixed locations from the start of the Resident Monitor in both F/B and S/J; these quantities (listed in Table 2-1) may be accessed by user programs. The technique used to access these offsets is as follows:

\[
\begin{align*}
\text{OFFSET} & = \text{the byte offset to the word desired} \\
\text{RMON} & = 54 \\
\text{MOV} & @\# \text{RMON}, \text{Rn} \quad ; \text{ANY GENERAL REGISTER} \\
\text{MOV} & \text{OFFSET}(\text{Rn}), \text{Rn}
\end{align*}
\]
Rn now contains the desired quantity. If a byte quantity is desired, a better method is:

CLR Rm
MOV @RMON,Rn
BISB OFFSET(Rn),Rm

This ensures that the high-order bits of the register are not set by a MOVB into the register.

<table>
<thead>
<tr>
<th>Offset (from Start of RMON) Octal Decimal</th>
<th>Tag</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>2</td>
<td>Serves as a link to interrupt entry code.</td>
</tr>
<tr>
<td>2</td>
<td>$CSW</td>
<td>160_10</td>
<td>Default I/O channels for the background (16_10 @ 5 words each).</td>
</tr>
<tr>
<td>244 164</td>
<td>$SYSCH</td>
<td>10_10</td>
<td>Internal I/O channel used for system functions.</td>
</tr>
<tr>
<td>256 174</td>
<td>BKEY</td>
<td>2</td>
<td>Segment number of the directory now in memory. 0 implies no directory is there.</td>
</tr>
<tr>
<td>260 176</td>
<td>CHKKEY</td>
<td>2</td>
<td>Device index and unit number of the device whose directory is in memory. Bits 1-5 are the device index, bits 8-10 are the unit number.</td>
</tr>
<tr>
<td>262 178</td>
<td>$DATE</td>
<td>2</td>
<td>Current date value. (The format is shown in Chapter 3, section 3.1.2.5.)</td>
</tr>
<tr>
<td>264 180</td>
<td>DFLG</td>
<td>2</td>
<td>&quot;Directory operation in progress&quot; flag. Used to inhibit ^C from aborting a job until directory operation is finished.</td>
</tr>
<tr>
<td>266 182</td>
<td>$USRRC</td>
<td>2</td>
<td>Normal location of USR.</td>
</tr>
<tr>
<td>270 184</td>
<td>QCOMP</td>
<td>2</td>
<td>Address of I/O completion manager, COMPLT.</td>
</tr>
<tr>
<td>272 186</td>
<td>SPUSR</td>
<td>2</td>
<td>Flag word used by MT/CT. If a USR function performed by MT or CT fails, this word is made non-zero.</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Offset (from Start of RMON) Octal Decimal</th>
<th>Tag</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>274 188</td>
<td>SYUNIT</td>
<td>2</td>
<td>High-order byte contains the unit number of the current system device.</td>
</tr>
<tr>
<td>276 190</td>
<td>SYSVER</td>
<td>1</td>
<td>Monitor version number (2 in Version 2).</td>
</tr>
<tr>
<td>277 191</td>
<td>SYSUPD</td>
<td>1</td>
<td>Version update number.</td>
</tr>
<tr>
<td>300 192</td>
<td>CONFIG</td>
<td>2</td>
<td>System configuration word. A 16-bit series of flags whose meanings are:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 -&gt; S/J Monitor</td>
</tr>
<tr>
<td>1</td>
<td>1 -&gt; F/B Monitor</td>
</tr>
<tr>
<td>2</td>
<td>1 -&gt; VT11 hardware exists</td>
</tr>
<tr>
<td>3</td>
<td>1 -&gt; RT-11 BATCH controls the background</td>
</tr>
<tr>
<td>5</td>
<td>0 -&gt; 60-cycle KW11L clock</td>
</tr>
<tr>
<td>6</td>
<td>1 -&gt; 50-cycle clock</td>
</tr>
<tr>
<td>7</td>
<td>1 -&gt; 11/45 FPP present</td>
</tr>
<tr>
<td>8</td>
<td>0 -&gt; No foreground job present</td>
</tr>
<tr>
<td>9</td>
<td>1 -&gt; Foreground job is in memory</td>
</tr>
<tr>
<td>15</td>
<td>1 -&gt; KW11L clock is present</td>
</tr>
</tbody>
</table>

Any bits not currently assigned are reserved by DIGITAL for future use and should not be used arbitrarily by user programs.

| 302 194 | SCROLL | 2 | Address of the VT11 scroller.                                      |
| 304 196 | TTKS   | 2 | Address of console keyboard status.                                |
| 306 198 | TTKB   | 2 | Address of console keyboard buffer.                                |

(continued on next page)
<table>
<thead>
<tr>
<th>Offset (from Start of RMON)</th>
<th>Tag</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octal</td>
<td>Decimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>200</td>
<td>2</td>
<td>Address of console printer status.</td>
</tr>
<tr>
<td>312</td>
<td>202</td>
<td>2</td>
<td>Address of console printer buffer.</td>
</tr>
<tr>
<td>314</td>
<td>204</td>
<td>2</td>
<td>Largest output file permitted with an indefinite length request (initially defined as -1, which implies that no limit is defined).</td>
</tr>
<tr>
<td>316</td>
<td>206</td>
<td>2</td>
<td>Offset from start of RMON to the dispatch table for EMT's 340-357. (This is used by the BATCH processor.)</td>
</tr>
<tr>
<td>320</td>
<td>208</td>
<td>2</td>
<td>Pointer to the impure area for the current executing job.</td>
</tr>
<tr>
<td>322</td>
<td>210</td>
<td>2</td>
<td>Executing job's number (0 = B/G, 2 = F/G).</td>
</tr>
<tr>
<td>320</td>
<td>208</td>
<td>4</td>
<td>Two words of time of day in the S/J Monitor.</td>
</tr>
<tr>
<td>322</td>
<td>210</td>
<td>10</td>
<td>Start of low memory protection map. (This map protects vectors at locations 0-476.)</td>
</tr>
<tr>
<td>324</td>
<td>212</td>
<td>2</td>
<td>Address of monitor routine to handle .SYNCH request.</td>
</tr>
<tr>
<td>352</td>
<td>234</td>
<td>2</td>
<td>Pointer to current entry point of USR.</td>
</tr>
<tr>
<td>354</td>
<td>236</td>
<td>2</td>
<td>Pointer to VT11 vector. The vector is initially positioned at 320.</td>
</tr>
<tr>
<td>356</td>
<td>238</td>
<td>1</td>
<td>Error count byte (for future use by system programs).</td>
</tr>
<tr>
<td>357</td>
<td>239</td>
<td>5</td>
<td>Reserved by DIGITAL for future use.</td>
</tr>
</tbody>
</table>
2.5.2 Table Descriptions

The monitor device tables discussed in this section include:

$PNAME
$STAT
$ENTRY
$DVREC
$FSIZE
$DVSIZ
$UNAM1,$UNAM2
$OWNER

The size of these tables is fixed and is governed by the $SLOT assignment; the default value is $14_{10}$ entries per table. To alter this, it is necessary to first edit a new value of $SLOT$ into the monitor source program, then reassemble and relink new monitors.

2.5.2.1 $PNAME$ (Permanent Name Table) - $PNAME$ is the central table around which all the others are constructed. There is an entry in $PNAME$ for each device in the system. Each entry consists of a single word that contains the .RAD50 code for the two-character permanent device name for that device; for example the entry for DECtape is $RAD50 /DT/$. The position of devices in this table is non-critical, but their relative position determines the general device index used in various places in the monitor; thus, all other tables must be organized in the same order as $PNAME$ (the index into $PNAME$ serves as the index into all the other tables for the equivalent device).

2.5.2.2 $STAT$ (Device Status Table) - Each device in the system must have a status entry in its corresponding slot in $STAT$. The status word is broken down into two bytes as follows:

Even byte - contains a device identifier. Each unique type of device in the system has an identifying integer. Those defined are:

0 = RK05 Disk
1 = TC11 DECTape
2 = Reserved
3 = Line Printer (LP11, LS11, LV11)
4 = Console Terminal (LT33/35, LA30/36, VT05, VT50)
5,6 = Reserved
7 = PC11 High-speed Reader
10 = PC11 High-speed Punch
11 = Magtape (TM11, TU10)
12 = RP11 Disk
13 = TA11 Cassette
14 = Card Reader (CR11, CM11)
15 = Reserved
16 = RJS03/4 Fixed-head Disks
17 = Reserved
20 = TJU16 Magtape
21 = RP11/RP02/RP03 Disk
22 = RX11/RX01 Floppy Disk

Odd byte - Bit flags with the following meanings:

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1 = File-structured device (disk, DECTape)</td>
</tr>
<tr>
<td></td>
<td>0 = Nonfile-structured device (PC11, LT33, etc.)</td>
</tr>
<tr>
<td>14</td>
<td>1 = Read-only device</td>
</tr>
<tr>
<td>13</td>
<td>1 = Write-only device</td>
</tr>
<tr>
<td>12</td>
<td>1 = Device whose directory is not a standard</td>
</tr>
<tr>
<td></td>
<td>RT-11 directory (MT, CT); the device</td>
</tr>
<tr>
<td></td>
<td>handlers for these devices are expected</td>
</tr>
<tr>
<td></td>
<td>to be able to perform their own directory</td>
</tr>
<tr>
<td></td>
<td>operations</td>
</tr>
<tr>
<td>11-8</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

2.5.2.3 $ENTRY (Handler Entry Point Table) - Whenever a handler is made resident, either by a .FETCH or with the LOAD command, the $ENTRY slot for that device is made to point to the fourth word of the device handler. The entry is zeroed when the handler is .RELEASEd or UNLOADed.

2.5.2.4 $DVREC (Device Handler Block Table) - This table (filled in at system bootstrap time) reflects the absolute block position of each of the device handlers on the system device. Since handlers are treated as files under RT-11, their position on the system device is not necessarily fixed. Thus, each time the system is bootstrapped, the handlers are located and $DVREC is updated with the value of the second block of the handler file. (Because the handlers are linked at 1000, the actual handler code starts in the second block of the file.) A zero entry in the $DVREC table indicates that no handler for the device in that slot was found on the system device.

2.5.2.5 $SHSIZE (Handler Size Table) - This table contains the size, in bytes, of each device handler. The table is set up at assembly time with the correct values and is used when a .FETCH is executed to provide the size of the specified handler. This size is also returned to the user as one of the values returned in a .DSTAT request.
2.5.2.6 $DVSIZ$ (Device Directory Size Table) - Entries in this table are non-zero for file-structured devices only and reflect the number of $256_{10}$-word blocks contained on the device. The current devices and their entries are:

<table>
<thead>
<tr>
<th>Device</th>
<th>Number of 256-Word Blocks</th>
<th>Device</th>
<th>Number of 256-Word Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RK11</td>
<td>$1300_8$</td>
<td>RP02</td>
<td>$116300_8$</td>
</tr>
<tr>
<td>TC11</td>
<td>$1102_8$</td>
<td>RJS03</td>
<td>$2000_8$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RJS04</td>
<td>$4000_8$</td>
</tr>
<tr>
<td>RF11</td>
<td>$2000_8$ (1 platter)</td>
<td>RX01</td>
<td>$752_8$</td>
</tr>
<tr>
<td></td>
<td>$4000_8$ (2 platters)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$6000_8$ (3 platters)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$10000_8$ (4 platters)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The default for RF11 and RJS03/4 is one platter, or $2000_8$ blocks. It is possible to alter the system to indicate the correct number of platters. Instructions are in Appendix A of the RT-11 System Reference Manual.

2.5.2.7 $SUNAM1$, $SUNAM2$ (User Name Tables) - These tables are used in conjunction with ASSIGN keyboard functions. The form of the ASSIGN command is:

```
.ASSIGN pnam:unam<CR>
```

where:

- `pnam` - a system device name/unit number
- `unam` - a user-assigned device name

A typical example is:

```
.ASSIGN DT1:DK
```
The default device name, DK, is now directed to DECTape unit 1. The user-assigned name is stored in an available slot in $UNAM2, while the device's permanent name/unit is stored in the corresponding slot in $UNAM1. The system uses a common device name lookup routine that maps any user-assigned name in the $UNAM2 table into a physical device name to be used in an operation. The total number of ASSIGNs permitted is limited by the value of $SLOT.

The command:

.Assign<CR>

zeros $UNAM2, thus removing all user assignments.

2.5.2.8 $OWNER (Device Ownership Table) - This table is used only under F/B to arbitrate device ownership. The table is $SLOT*2 words in length and is divided into 2-word entries per device. Each 2-word entry is divided into eight 4-bit fields capable of holding a job number. Thus, each device is presumed to have up to eight units, each assigned independently of the others. However, if the device is nonfile-structured, the ownership is assigned to all units.

When a job attempts to access a particular unit of a device, the F/B Monitor checks to be sure the unit being accessed is either public or belongs to the requesting job. If the unit is owned by the other job, a fatal error is generated.

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

2.5.2.9 DEVICE Macro - The DEVICE macro call is used in RMON to allow quick and easy insertion of new devices at assembly time. The form of the macro call is:

    DEVICE NAME,SIZ,STAT,ENTRY
where:

NAME - two characters of the permanent device name

SIZ - the size of the device's directory in 256-word blocks; 0 means nonfile-structured or special

STAT - the sum of all $STAT table entries that apply for this device plus the device id (from section 2.5.2.2):

\[
\begin{align*}
\text{FILST}\$ & = 100000 \quad \text{File-structured device} \\
\text{RONLY}\$ & = 40000 \quad \text{Read-only device} \\
\text{WONLY}\$ & = 20000 \quad \text{Write-only device} \\
\text{SPECL}\$ & = 10000 \quad \text{Non RT-ll file-structured device (including MT and CT)}
\end{align*}
\]

ENTRY - the 2-character device name with SYS appended, if this is a system device.

Thus, a sample call is:

DEVICE TT,0,4

The SIZ entry is 0, since TT is a nonfile-structured device.

The entry for DECTape is:

DEVICE DT,1102,1+FILST\$,DTSYS

The 1+FILST\$ indicates that the device code is 1 and FILST\$ is defined as 100000. The entry for DTSYS is present because DT can be a system device.

In addition to the DEVICE macro, another macro, HSIZE, is defined and sets the handler size for the $HSIZE table. The format of the HSIZE macro call is:

\[
\text{HSIZE HAN,BYT,TYPE}
\]
where:

HAN - the 2-letter device name
BYT - the handler size in bytes
TYPE - SYS if the device can be a system device; blank otherwise

Chapter 5 shows the use of HSIZE in adding a handler to the RT-11 system. The KMON portion of the monitor source listing should be consulted for greater detail.

2.5.3 F/B Impure Area

An impure area is defined here as that area of memory where the monitor stores all job-dependent data. Thus, the impure area contains all information that the monitor requires to effectively run two independent jobs, both of which are memory-resident. This section details the contents and location of each word (byte) in the impure area.

A table that points to the impure area for a particular job is in the F/B monitor's data base. This table is at $IMPUR and currently consists of two words: the first is a pointer to the background's impure area (which is permanently resident in RMON at location BKGD), the second is the foreground's pointer.

Under RT-11, a background job is always running and will be the KMON if no other background job exists. However, the foreground impure area pointer may be 0 if no foreground job is in memory. When an FRUN command is given, a foreground impure area is created for the job and the $IMPUR entry for the foreground pointer is updated to point to the impure area.

Table 2-2 is a detailed breakdown of the contents of the impure area. The offset mentioned is the offset from the start of the impure area itself; thus, the first word in the area has a 0 offset.
<table>
<thead>
<tr>
<th>Offset</th>
<th>Mnemonic</th>
<th>Octal Length (Bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I.JSTA</td>
<td>2</td>
<td>Job status.</td>
</tr>
<tr>
<td>2</td>
<td>I.QHDR</td>
<td>2</td>
<td>I/O Queue Header.</td>
</tr>
<tr>
<td>4</td>
<td>I.CMPE</td>
<td>2</td>
<td>Last entry in completion queue. I/O completion routines are queued for execution. This is the pointer to the last routine to be entered.</td>
</tr>
<tr>
<td>6</td>
<td>I.CMPL</td>
<td>2</td>
<td>Completion queue header.</td>
</tr>
<tr>
<td>10</td>
<td>I.CHWT</td>
<td>2</td>
<td>Pointer to channel during I/O wait. When a job is waiting for I/O, the address of the channel area in use goes here.</td>
</tr>
<tr>
<td>12</td>
<td>I.PCHW</td>
<td>2</td>
<td>Saved channel pointer during execution of a completion routine. The contents of I.PCHW are put in R0 when a completion routine is entered.</td>
</tr>
<tr>
<td>14</td>
<td>I.PERR</td>
<td>2</td>
<td>Error byte 52 and 53 saved during completion routines.</td>
</tr>
<tr>
<td>16</td>
<td>I.PTTI</td>
<td>2</td>
<td>Previous TT input character.</td>
</tr>
<tr>
<td>20</td>
<td>I.TTLC</td>
<td>2</td>
<td>Terminal input ring buffer line count.</td>
</tr>
<tr>
<td>22</td>
<td>I.TID</td>
<td>2</td>
<td>Pointer to job ID area.</td>
</tr>
<tr>
<td>24</td>
<td>I.JNUM</td>
<td>2</td>
<td>Job number of job that owns this impure area.</td>
</tr>
<tr>
<td>26</td>
<td>I.CNUM</td>
<td>2</td>
<td>Number of I/O channels defined. 16,10 is default, .CDFN can be used to define new ones.</td>
</tr>
<tr>
<td>30</td>
<td>I.CSW</td>
<td>2</td>
<td>Pointer to job's channel area.</td>
</tr>
<tr>
<td>32</td>
<td>I.IOCT</td>
<td>2</td>
<td>Count of total I/O operations outstanding.</td>
</tr>
<tr>
<td>34</td>
<td>I.SCTR</td>
<td>2</td>
<td>Suspension count. Zero means the number of .SPNDS = the number of .RSUMS.</td>
</tr>
<tr>
<td>36</td>
<td>I.SPLS</td>
<td>2</td>
<td>Address of the .DEVICE request list.</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Offset</th>
<th>Mnemonic</th>
<th>Octal Length (Bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>I.TRAP</td>
<td>2</td>
<td>Address of user trap routine. Set by .TRPSET.</td>
</tr>
<tr>
<td>42</td>
<td>I.FPP</td>
<td>2</td>
<td>Address of FPP exception routine. Set by .SFPA.</td>
</tr>
<tr>
<td>44</td>
<td>I.SWAP</td>
<td>4</td>
<td>Address and number of extra words to be included in the context switch operation. Set by .CNTXSW request.</td>
</tr>
<tr>
<td>50</td>
<td>I.SP</td>
<td>2</td>
<td>Saved stack pointer. When this job is made inactive, the active value of SP is saved here.</td>
</tr>
<tr>
<td>52</td>
<td>I.BITM</td>
<td>24</td>
<td>Low memory protection bitmap. This map reflects the user's .PROTECT requests.</td>
</tr>
</tbody>
</table>

(76 through 332 concern the console terminal)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Mnemonic</th>
<th>Octal Length (Bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>76</td>
<td>I.IRNG</td>
<td>2</td>
<td>Input ring buffer low limit.</td>
</tr>
<tr>
<td>100</td>
<td>I.IPUT</td>
<td>2</td>
<td>Input &quot;PUT&quot; pointer for interrupts.</td>
</tr>
<tr>
<td>102</td>
<td>I.ICTR</td>
<td>2</td>
<td>Input character counter.</td>
</tr>
<tr>
<td>104</td>
<td>I.IGET</td>
<td>2</td>
<td>Input &quot;GET&quot; pointer for .TTYIN.</td>
</tr>
<tr>
<td>106</td>
<td>I.ITOP</td>
<td>2</td>
<td>Input ring buffer high limit.</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>144</td>
<td>Input ring buffer.</td>
</tr>
<tr>
<td>254</td>
<td>I.OPUT</td>
<td>2</td>
<td>Output &quot;PUT&quot; pointer for interrupts.</td>
</tr>
<tr>
<td>256</td>
<td>I.OCTR</td>
<td>2</td>
<td>Output character counter.</td>
</tr>
<tr>
<td>260</td>
<td>I.OGET</td>
<td>2</td>
<td>Output &quot;GET&quot; pointer for interrupts.</td>
</tr>
<tr>
<td>262</td>
<td>I.OTOP</td>
<td>2</td>
<td>Output ring buffer high limit.</td>
</tr>
<tr>
<td>264</td>
<td></td>
<td>50</td>
<td>Output ring buffer.</td>
</tr>
<tr>
<td>334</td>
<td>I.QUE</td>
<td>20</td>
<td>Initial I/O queue element.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 2-2 (Cont.)
Impure Area

<table>
<thead>
<tr>
<th>Offset</th>
<th>Mnemonic</th>
<th>Octal Length (Bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>354</td>
<td>I.MSG</td>
<td>12</td>
<td>Message channel. Used by .RCVD and .SDAT. This channel is permanently open.</td>
</tr>
<tr>
<td>366</td>
<td></td>
<td>10</td>
<td>Job ID area. Contains (&lt;CR&gt;&lt;LF&gt;)B(&lt;CR&gt;&lt;LF&gt;) or (&lt;CR&gt;&lt;LF&gt;)F(&lt;CR&gt;&lt;LF&gt;) for terminal prompting. Space has been left for up to a 3-character job name.</td>
</tr>
</tbody>
</table>

2.5.4 Low Memory Bitmap (LOWMAP)

RT-ll maintains a bitmap which reflects the protection status of low memory, locations 0-476. This map is required in order to avoid conflicts in the use of the vectors. In F/B, the .PROTECT request allows a program to gain exclusive control of a vector or a set of vectors. When a vector is protected, the bitmap is updated to indicate which words are protected. If a word in low memory is protected, it will not be destroyed when a new background program is run.

The bitmap is a 2010 byte table which starts 326 bytes from the beginning of the Resident Monitor. Table 2-3 lists the offset from RMON and the corresponding locations represented by that byte:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Locations (octal)</th>
<th>Offset</th>
<th>Locations (octal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>326</td>
<td>0-16</td>
<td>340</td>
<td>240-256</td>
</tr>
<tr>
<td>327</td>
<td>20-36</td>
<td>341</td>
<td>250-277</td>
</tr>
<tr>
<td>330</td>
<td>40-56</td>
<td>342</td>
<td>300-316</td>
</tr>
<tr>
<td>331</td>
<td>60-76</td>
<td>343</td>
<td>320-336</td>
</tr>
<tr>
<td>332</td>
<td>100-116</td>
<td>344</td>
<td>340-356</td>
</tr>
<tr>
<td>333</td>
<td>120-136</td>
<td>345</td>
<td>360-376</td>
</tr>
<tr>
<td>334</td>
<td>140-156</td>
<td>346</td>
<td>400-416</td>
</tr>
<tr>
<td>335</td>
<td>160-176</td>
<td>347</td>
<td>420-436</td>
</tr>
<tr>
<td>336</td>
<td>200-216</td>
<td>350</td>
<td>440-456</td>
</tr>
<tr>
<td>337</td>
<td>220-236</td>
<td>351</td>
<td>460-476</td>
</tr>
</tbody>
</table>

2-21
Each byte in the table reflects the status of $16_{10}$ words of memory. The first byte in the table controls locations 0-16, the second byte controls locations 20-36, and so on. The bytes are read from left to right. Thus, if locations 0-3 are protected, the first byte of the table contains:

$$11000000$$

Note that only individual words are protected, not bytes. Thus, protecting location 0 always implies that the word at location 0 is protected, meaning both locations 0 and 1. If locations 24 and 26 are protected, the second byte of the table contains:

$$00110000$$

since the leftmost bit represents location 20 and the rightmost bit represents location 36. To protect locations 300-306, the leftmost 4 bits of byte 342 must be set:

$$11110000$$

resulting in a value of 360 for that byte.

2.5.4.1 S/J Restrictions - The S/J Monitor does not support the .PROTECT request. If users wish to protect vectors, the protection must be done in one of two ways:

1. Manually, with PATCH, or
2. Dynamically (from within the user's program)

To protect locations 300-306 dynamically, the following instructions are used:

```assembly
MOV @54, R0
BISB #+B11110000, 342(R0)
```
Protecting locations with PATCH implies that the vector is permanently protected, even if the system is re-bootstrapped, while the second method provides a temporary measure and does not hold across bootstraps. However, users are cautioned that the second method involves storing directly into the monitor; for this reason it is recommended that S/J users use method 1.

2.6 USING AUXILIARY TERMINALS AS THE CONSOLE TERMINAL

This section describes how RT-ll can be modified to allow a terminal other than the standard console unit 0 to become the console terminal. This procedure is useful in cases where it is desirable to be able to use different console capabilities at different times (for example, at certain times the hard copy output of an LA30 is required, while at other times the speed of a VT05 is desirable). The only information required to make the alteration is:

1) the address of the auxiliary terminal's interrupt vectors, and
2) the I/O page addresses of the keyboard and printer status register and buffer.

RT-ll is designed so that all console references are done indirectly through centralized pointers. Thus, changing several system locations causes all operations to be transferred to a new terminal.

For this example, assume that the new terminal's interrupt vectors are at 300,302 and 304,306 and that its I/O page addresses are:

TKS at 177500
TKB at 177502
TPS at 177504
TPB at 177506

Also assume that the new terminal is a parallel interface so that no fill characters are required.
The bootstrap must also be changed to relocate the new vector locations when the monitor is first loaded into memory. The bootstrap contains a list of items that must be relocated; the list is located at RELST in the bootstrap code. The exact position of RELST varies with each monitor and must be obtained from Chapter 5 of Getting Started With RT-11 (V02B). The patching procedure is:

```
.R PATCH <CR>
```

**PATCH Version number**

**FILE NAME**

- *MONITR.SYS/M<CR>
- *BASE;9<CR>
- *69/ VECTIN<LF>
- 62/ STATIN<LF>
- 64/ VECTOUT<LF>
- 66/ STATOUT<CR>
- *36/ nnnn VECTIN<LF>
- 302/ nnnnn STATIN<LF>
- 304/ nnnnn VECTOUT<LF>
- 306/ nnnnn STATOUT<CR>
- *0,xx304/ 177560 177560<LF>
- 0,xx306/ 177562 177562<LF>
- 0,xx310/ 177564 177564<LF>
- 0,xx312/ 177566 177566<CR>
- *0,xx342\ 0 360<CR>
- *E
```

[The current values for the BASE address and for the input/output vectors and status are in Chapter 5 of Getting Started with RT-11. They must be copied into the new terminal's vectors.]

[nnnn are arbitrary numbers]

[xx = 16 for S/J, 17 for F/B. Modify monitor's central I/O page pointers]

[Protect new vectors]

[Bootstrap must be rewritten. Rebootstrap; system will appear on new terminal.]
It is also possible to write a user program that would perform this procedure dynamically at run-time. Such a program would modify the monitor's protection map and the central I/O page pointers, then set up locations 300-306 and exit. If done dynamically, the monitor file itself is unchanged; thus when the system is bootstrapped, the console terminal reverts to the usual unit.

2.7 MAKING TTY SET OPTIONS PERMANENT IN F/B MONITOR

The F/B Monitor may be configured for different console terminal requirements by use of the TTY options of the SET command. These changes are not permanent and must be made each time the monitor is bootstrapped. By using the patching procedures in this section, the various options required for the installation may be made a permanent part of the F/B Monitor.

Table 2-4 is a description of the TTY options and their default functions in the F/B Monitor as distributed.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAB/NOTAB</td>
<td>NOTAB</td>
<td>Hardware tabs converted to spaces.</td>
</tr>
<tr>
<td>CRLF/NOCRLF</td>
<td>CRLF</td>
<td>&lt;CR&gt;&lt;LF&gt; inserted if WIDTH reached.</td>
</tr>
<tr>
<td>FORM/NOFORM</td>
<td>NOFORM</td>
<td>Form Feed converted to Line Feeds.</td>
</tr>
<tr>
<td>FB/NOFB</td>
<td>FB</td>
<td>CTRL F/CTRL B cause context switch.</td>
</tr>
<tr>
<td>PAGE/NOPAGE</td>
<td>PAGE</td>
<td>CTRL S holds output, CTRL Q continues it.</td>
</tr>
<tr>
<td>SCOPE/NOSCOPE</td>
<td>NOSCOPE</td>
<td>VT05, VT50, VT11 is the console terminal (rubout produces backspace, space, backspace).</td>
</tr>
<tr>
<td>WIDTH</td>
<td>72(10)</td>
<td>Width of carriage.</td>
</tr>
</tbody>
</table>
The three options enabled are PAGE, CRLF, and FB. The carriage width is set to 72(10) characters (110 octal).

To permanently change these options, the words TTCNFG, TTWIDT and LISTFB in the F/B Monitor must be patched. The exact locations of these words and the BASE address are found in Chapter 5 of Getting Started with RT-11 (V02B). The numbers used in the following examples are for illustration purposes only and may not be correct for all systems.

2.7.1 Carriage Width

The carriage width is the line width at which the CTRL option generates a carriage return/line feed. This width is changed by patching the word TTWIDT, which for this example is assumed to be located at 21354.

```
.R PATCH <CR>

PATCH Version number

FILE NAME--
.MONITR.SYS/M<CR>
.BASE;0<CR>
.0,21354\110204<CR>
.E
```

[The /M is necessary; set relocation registers; open with backslash]

In this example, the width is changed from 72_{10} to 132_{10} (204_{8}).

2.7.2 Other Options

Other options are changed by setting or clearing the appropriate bits in TTCNFG. To determine the new value to be inserted in TTCNFG, Table 2-5 is used. For each option, select the permanent value desired. Add together the octal bit patterns for each value selected to determine the new value of TTCNFG.
Table 2-5
TTCNFG Option Bits

<table>
<thead>
<tr>
<th>Option</th>
<th>Bit Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAB</td>
<td>000001</td>
</tr>
<tr>
<td>CRLF</td>
<td>000002</td>
</tr>
<tr>
<td>FORM</td>
<td>000004</td>
</tr>
<tr>
<td>FB</td>
<td>000010</td>
</tr>
<tr>
<td>PAGE</td>
<td>000200</td>
</tr>
<tr>
<td>SCOPE</td>
<td>100000</td>
</tr>
<tr>
<td>Any NO option</td>
<td>000000</td>
</tr>
</tbody>
</table>

For example, the monitor default is PAGE, CRLF and FB. Adding together the bit patterns for PAGE, CRLF and FB produces the octal value 212 (= 200 + 10 + 2).

To change this to SCOPE, PAGE, FB, add together the numbers 100000, 200 and 10 to get 100210, the new value of TTCNFG. Using the location of TTCNFG obtained from Getting Started With RT-11, the procedure is:

```
.R PATCH <CR>

PATCH Version number
FILE NAME=--
.MONITR.SYS/M<CR>
.BASE;R<CR>
T TTCNFG, 212 100210<CR>
.E
```

If the FB option is changed, an additional step is necessary. Bit 15 of LISTFB must be changed to reflect the new FB option. Bit 15 must be 0 if the option is FB and must be 1 if the option is NOFB. For example, to change the monitor default to FORM, TAB, NOFB, the value of TTCNFG is 5 (4 + 1 + 0), and bit 15 of LISTFB must be a 1. The patch procedure is:
PATCH Version number

FILE NAME--
*MONITR.SYS/M<CR>
*BASE;ØR<CR>
*Ø,TTCNFG/ 212 5<CR>
*Ø,LISTFB/ 3316 1Ø3316<CR>
E

[The /M is necessary; set relocation register; change TTCNFG; set bit 15 in LISTFB.]

After making any of these patches, it is necessary to bootstrap the system to load the new version of the monitor.
CHAPTER 3
FILE STRUCTURES AND FILE FORMATS

3.1 DEVICE DIRECTORY SEGMENTS

The device directory begins with physical block 6 of any file-structured device and consists of a series of directory segments that contain the names and lengths of the files on that device. The directory area is variable in length, from 1 to 31 (decimal) directory segments. PIP allows specification of the number of segments when the directory is zeroed. The default value is four directory segments. Each directory segment is made up of two physical blocks; thus, a single directory segment is 512 words in length.

A directory segment has the following format:

<table>
<thead>
<tr>
<th>5 header words</th>
</tr>
</thead>
<tbody>
<tr>
<td>file entries</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
</tbody>
</table>

3.1.1 Directory Header Format

Each directory segment contains a 5-word header block, leaving 507 (decimal) words for directory entries. The contents of the header words are described in Table 3-1.
Table 3-1
Directory Header Words

<table>
<thead>
<tr>
<th>Word</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The number of segments available for entries. This number is specified in PIP when the device is zeroed and must be in the range $1 \leq N \leq 31_{10}$.</td>
</tr>
<tr>
<td>2</td>
<td>Segment number of the next logical directory segment. The directory may, in certain cases, be a linked list. This word is the link word between logically contiguous segments; if equal to 0, there are no more segments in the list. Refer to Section 3.2.1, Directory Segment Extensions, for more details on the link word.</td>
</tr>
<tr>
<td>3</td>
<td>The highest segment currently open (each time a new segment is created, this number is incremented). This word is updated only in the first segment and is unused in any but the first segment.</td>
</tr>
<tr>
<td>4</td>
<td>The number of extra bytes per directory entry. This number can be specified when the device is zeroed with PIP. Currently, RT-11 does not allow direct manipulation of information in the extra bytes.</td>
</tr>
<tr>
<td>5</td>
<td>Block number where files in this segment begin.</td>
</tr>
</tbody>
</table>

3.1.2 Directory Entry Format

The remainder of the segment is filled with directory entries. An entry has the following format:
3.1.2.1 Status Word - The Status Word is broken down into two bytes of data:

Even byte: Reserved for future use.

Odd byte: Indicates the type of entry. Currently RT-11 recognizes the file types listed in Table 3-2:

<table>
<thead>
<tr>
<th>Value</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tentative File, i.e., one that has been .ENTERed but not .CLOSEd. Files of this type are deleted if not eventually .CLOSEd and are listed by PIP as &lt;UNUSED&gt; files.</td>
</tr>
<tr>
<td>2</td>
<td>An empty file. The name, extension, and date fields are not used. PIP lists an empty file as &lt;UNUSED&gt; followed by the length of the unused area.</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 3-2 (Cont.)
#### File Types

<table>
<thead>
<tr>
<th>Value</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A permanent entry. A tentative file that has been .CLOSEd is a permanent file. The name of a permanent file is unique; there can be only one file with a given name and extension. If another exists before the .CLOSE is done, it is deleted by the monitor as part of the .CLOSE operation.</td>
</tr>
<tr>
<td>10</td>
<td>End-of-segment marker. RT-11 uses this to determine when the end of the directory segment has been reached during a directory search.</td>
</tr>
</tbody>
</table>

3.1.2.2 Name and Extension - These three words (in .RAD50) represent the symbolic name and extension assigned to a file.

3.1.2.3 Total File Length - The file length consists of the number of blocks currently a part of the file. Attempts to read or write outside the limits of the file result in an End of File error.

3.1.2.4 Job Number and Channel Number - A tentative file is associated with a job in one of two ways:

1. Under the S/J Monitor, the sixth word of the entry holds the channel number on which the file is open. This enables the monitor to locate the correct tentative entry for the channel when the .CLOSE is given. The channel number is loaded into the even byte of the sixth word.

2. In F/B, the channel number is put into the even byte of the sixth word; in addition, the number of the job that is opening the file is put into the odd byte of the word. This is required to uniquely identify the correct tentative file during the .CLOSE and is necessary because both jobs may have files open on their respective channels; the job number differentiates the tentative files.

**NOTE**

This sixth word is used only when the file is marked as tentative. Once it becomes permanent, the word becomes unused. Its function while permanent is reserved for future use.

3-4
3.1.2.5 Date - When a tentative file is created via .ENTER, the system date word is put into the creation date slot for the file. The date word is in the following format:

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>10</th>
<th>9</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN</td>
<td>MONTH (1-12.)</td>
<td>DAY (1-31.)</td>
<td>YEAR-110(8)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.1.2.6 Extra Words - The number of extra words is determined by the number of extra bytes per entry in the header words. Although PIP provides for allocation and listing of extra words, RT-11 provides no direct facilities for manipulating this extra information. Any user program wishing to access these words must perform its own direct operations on the RT-11 directory.

Figure 3-2 shows a typical RT-11 directory segment:
FOUR SEGMENTS AVAILABLE
NO NEXT SEGMENT
HIGHEST OPEN IS #1
NO EXTRA WORDS/ENTRY
FILES START AT BLOCK 168

PERMANENT ENTRY
RAD5# FOR "MON"
RAD5# FOR "ITR"
RAD5# FOR "SYS"

FILE IS 3410 (428) BLOCKS LONG

NO CREATION DATE

AN EMPTY ENTRY
(The name and extension of an
empty is not important)

6418 (158) BLOCKS LONG

PERMANENT
RAD5# FOR "PIP"

RAD5# FOR "MAC"
FILE IS 910 (118) BLOCKS LONG

NO CREATION DATE

TENTATIVE FILE ON CHANNEL 1
RAD5# FOR "PIP"

RAD5# FOR "MAC"
JOB #, CHANNEL #

EVERY TENTATIVE MUST BE FOLLOWED BY
AN EMPTY ENTRY

FILE IS 52818 (15288) BLOCKS LONG

END OF DIRECTORY SEGMENT

Figure 3-2
Directory Segment
When the tentative file PIP.MAC is .CLOSEd, the permanent file PIP.MAC is deleted.

To find the starting block of a particular file, first find the directory segment containing the entry for the desired file. Then take the starting block number given in the fifth word of that directory segment and add to it the length of each file in the directory before the desired file. For example, in Figure 3-2, the permanent file PIP.MAC will begin at block number 160 (octal).

3.2 SIZE AND NUMBER OF FILES

The number of files that can be stored on an RT-11 device depends on the number of segments in the device's directory and the number of extra words per entry. The maximum number of directory segments on any RT-11 device is $31_{10}$. This theoretically leaves room for a maximum of:

$$31 \times \left[ \frac{512 - 5}{7 + N} \right]$$

directory entries, where $N$ equals the number of extra information words per entry. If $N=0$, this indicates that the maximum is $2232_{10}$ entries.

If files are added sequentially (that is, one immediately after another) without deleting any files, roughly one half the total number of entries will fit on the device before a directory overflow occurs. This results from the way filled directory segments are handled.

When a directory segment becomes full and it is necessary to open a new segment, approximately one half the entries of the filled segment are moved to the newly-opened segment (this process is illustrated in Section 3.2.1); thus, when the final segment is full, all previous segments have approximately one half their total capacity. If this process were not done and a file was deleted from a full segment, the space from the deleted file could not be reclaimed. Every tentative file must be followed by an empty entry (for recovering unused blocks when the file is made permanent). Though only one file is deleted, two entries (tentative and empty) are needed to reclaim the space.
If files are continuously added to a device, the maximum number of entries will be:

\[
(M+1) \left\lfloor \frac{507}{2(7+N)} \right\rfloor
\]

where \( M \) equals the number of segments available on the device and \( N \) equals the number of extra words.

The theoretical total can be realized by compressing the device (using the PIP /S operation) when the directory fills up. PIP packs the directory segments as well as the physical device.

3.2.1 Directory Segment Extensions

RT-11 allows a maximum of 31 (decimal) directory segments. This section covers the processing of a directory segment. For illustrative purposes, the following symbols are used:

\[
\begin{align*}
\text{n} & \quad \text{This represents a directory segment with some} \\
\text{n} & \quad \text{number of directory entries. n is the segment number.}
\end{align*}
\]

\[
\begin{align*}
\text{n} & \quad \text{This represents a segment which is full, i.e., no} \\
\text{n} & \quad \text{more entries will fit in the segment.}
\end{align*}
\]

Systems start out with entries entered into segment 1:

\[
\begin{align*}
\text{1} & \quad \text{This represents a directory segment with some} \\
\text{1} & \quad \text{number of directory entries. 1 is the segment number.}
\end{align*}
\]

As entries are added, segment 1 fills:

\[
\begin{align*}
\text{1} & \quad \text{This represents a segment which is full, i.e., no} \\
\text{1} & \quad \text{more entries will fit in the segment.}
\end{align*}
\]

When this occurs and an attempt is made to add another entry to the directory, the system must open another directory segment. If another segment is available, the following occurs:
1. one half of the entries from the filled segment are put into the next available segment,
2. the shortened segment is re-written to the disk,
3. the directory segment links are set, and
4. the file is entered in the newly created segment.

NOTE
If the last segment becomes full and an attempt is made to enter another file, a fatal error occurs and an error message is generated:

?M-DIR OVFLO?

Thus, in the normal case, the segment appears as:

```
1  
↓

1  
↓
Link
2
```

Before extension

After extension; half the entries are in the new segment, half in the old; segment 1 is linked to segment 2.

If many more files are entered, they fill up the second segment and overflow into the third segment, if it is available:
In this case, the links between the segments are not strictly necessary, as the segments are contiguous. However, the links do become necessary if a large file is deleted from segment 2 and many small files are entered, since it would then be possible to overflow segment 2 again. If this occurred and a fourth segment existed, the directory would appear:

In this case, segment 2 overflows into segment 4 and the links are used to link logical pieces rather than physical pieces.
3.3 MAGTAPE AND CASSETTE FILE STRUCTURE

3.3.1 Magtape File Structure

This section covers the magtape file structure as implemented in RT-11, Version 2B. The structure is slightly different from that of Version 2. However, RT-11 V02B can read magtapes written under Version 2.

RT-11 magtapes use a subset of the VOL1, HDRL, and EOF1 ANSI standard labels. Each magtape file has the format:

```
HDRL*---data---*EOF1*
```

where each asterisk represents a tape mark.

A volume containing a single file has the following format:

```
VOL1 HDRL*---data---*EOF1**
```

A volume containing two files has the following format:

```
VOL1 HDRL*---data---*EOF1*HDRL*---data---*EOF1**
```

A double tape mark following an EOF1 label indicates logical end of tape.

Each label occupies the first 80 bytes of a 256-word physical block, and each byte in the label contains an ASCII character (i.e., if the content of a byte is listed as 'l', the byte contains the ASCII code for 'l'). Table 3-3 shows the contents of the first 80 bytes in the three labels. Note that VOL1, HDRL, and EOF1 each occupy a full 256-word block, of which only the first 80 bytes are meaningful.

The meanings of the table headings are:

- CP - character position in label
- Field Name - reference name of field
- L - length of field in bytes
- Content - content of field
Table 3-3
ANSI MT Labels Under RT-11

<table>
<thead>
<tr>
<th>Volume-Header Label (VOL1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1-3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5-10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12-37</td>
</tr>
<tr>
<td>38-51</td>
</tr>
<tr>
<td>52-79</td>
</tr>
<tr>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First File Header Label (HDR1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1-3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5-21</td>
</tr>
<tr>
<td>22-27</td>
</tr>
<tr>
<td>28-31</td>
</tr>
<tr>
<td>32-35</td>
</tr>
<tr>
<td>36-39</td>
</tr>
<tr>
<td>40-41</td>
</tr>
<tr>
<td>42-47</td>
</tr>
<tr>
<td>48-53</td>
</tr>
<tr>
<td>54</td>
</tr>
<tr>
<td>55-60</td>
</tr>
<tr>
<td>61-73</td>
</tr>
<tr>
<td>74-80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First End-of-File Label (EOF1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same as HDR1 except that the label identifier (CP 1-3) is EOF, not HDR, and the block count field (CP 55-60) contains the number of blocks in the file as a decimal value encoded in ASCII characters (for example, if the file was 12 blocks long, the block count field would be 00012).</td>
</tr>
</tbody>
</table>

3-12
3.3.1.1 Moving MT to Other Industry-Compatible Environments - RT-11 V02B magtapes may be read by RSX-11D Version 6. RT-11 magtapes should be mounted under RSX-11D, by using the /OVR switch of the MOUNT command, or by specifying a volume label of "RT1101". RSX-11D Version 6 will not allow the user to write on RT-11 V02B magtapes once they have been mounted. RT-11 V02B can read RSX-11D Version 6 magtapes, but RT-11 users should not attempt to write on tapes created by RSX-11D. Users should note that data structures differ between the two systems and these differences must be handled by the user.

RT-11 V02B magtapes may be read on IBM systems that support ANSI standard label processing. RT-11 V02B magtapes to be read by IBM systems should consist of single file volumes (one file per magtape). Important JCL parameters for reading RT-11 V02B tapes under an IBM OS system are as follows:

(In the DD statement of the Job Control Language)

DISP = OLD
LABEL = (01,AL,,IN)
VOL = (,RETAIN,SER=RT1101)
DSN = RTFILE.MAC
BLKSIZE = 512
DEN = 2     (for 800 bpi 7-track or 9-track tape)

The DSN parameter is the Data Set Name or the RT-11 filename and extension. Files to be moved to other systems should be created with full six-character filenames and three-character extensions.

3.3.1.2 Recovering From Bad Tape Errors - When a bad tape error occurs on magtape, the magtape handler will retry the desired function, and, if the error persists, will attempt to save the tape's file structure. It does this on writes, for example, by retrying the write 10 times, using the write with extended file gap to space past the bad tape. If, after retrying, the error still exists, the file will be closed, containing all data written prior to the write on which the error occurred. The user should still be able to write additional files on the tape, since the bad portion of the tape will be within the area of the closed file.
If a bad tape error occurs when writing the file header during ENTER, and retry fails, the handler writes logical end of tape after the previous file on the tape. The remainder of the tape can be accessed only if the last complete file on the tape can be extended (or overwritten by a file of different length) so that the bad tape error does not occur on the file header when a subsequent file is ENTERed.

If a bad tape error occurs while writing the end of file label (EOFl) during CLOSE, the handler writes a triple tape mark to signify end of file and logical end of tape. Additional files can be added to the tape only if the last complete file can be extended (or overwritten by a file of different length) so that the bad tape error does not occur at the EOFl label.

3.3.2 Cassette File Structure

A blank (newly initialized) cassette appears in the format:

```
<table>
<thead>
<tr>
<th>Clear Leader</th>
<th>Extended File Gap</th>
<th>Sentinel File</th>
<th>Garbage</th>
</tr>
</thead>
</table>
```

$32_{10}$ bytes

while a cassette with a file on it appears as:

```
<table>
<thead>
<tr>
<th>Clear Leader</th>
<th>Extended File Gap</th>
<th>Header Block</th>
<th>Block Gap</th>
<th>Data Block</th>
<th>Block Gap</th>
<th>Data Block</th>
<th>File Gap</th>
<th>Sentinel File</th>
</tr>
</thead>
</table>
```

$32_{16}$ bytes $128_{16}$ bytes

3-14
Files normally have data written in 128$^{10}$-byte blocks. This can be altered by writing cassettes while in hardware mode. (In hardware mode, the user program must handle the processing of any headers and sentinel files; in software mode the handler automatically does this. Refer to Appendix H of the RT-11 System Reference Manual.)

The preceding diagram shows a file terminated in the usual manner (by a sentinel file). However, the physical end of cassette may occur before the actual end of the file. This format appears as:

![Diagram of file format](image)

In the latter case, for multi-volume processing the partially written block must be the first data block of the next volume.

3.3.2.1 File Header - The File Header is a 32$^{10}$-byte block that is the first block of any data file on a cassette. If the first byte of the header is null, the header is interpreted as a sentinel file, which is an indication of logical end of cassette. The format of the header is described in Table 3-4.
Table 3-4

CT File Header Format

<table>
<thead>
<tr>
<th>Byte Number</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-5</td>
<td>File name in ASCII characters (ASCII is assumed to imply a 7-bit code)</td>
</tr>
<tr>
<td>6-8</td>
<td>Extension in ASCII characters</td>
</tr>
<tr>
<td>9</td>
<td>Data type (0 for RT-11)</td>
</tr>
<tr>
<td>10,11</td>
<td>Block length of 128(<em>{10}) (200(</em>{8})); Note: byte 10=0 (high-order), byte 11=200(_{8}) (low-order)</td>
</tr>
<tr>
<td>12</td>
<td>File sequence number. (0 for a single-volume file or the first volume of a multi-volume file; successive numbers are used for continuations)</td>
</tr>
<tr>
<td>13</td>
<td>Level 1; this byte is a l</td>
</tr>
<tr>
<td>14-19</td>
<td>Date of file creation (6 ASCII digits representing day (01-31); month (01-12), and last two digits of the year; 0 or 40(_{8}) in first byte means no date present)</td>
</tr>
<tr>
<td>20,21</td>
<td>Zero</td>
</tr>
<tr>
<td>22</td>
<td>Record attributes (0 in RT-11 cassettes)</td>
</tr>
<tr>
<td>23-28</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>29-31</td>
<td>Reserved for user</td>
</tr>
</tbody>
</table>

3.4 RT-11 FILE FORMATS

3.4.1 Object Format (.OBJ)

An object module is a file containing a program or routine in a binary, relocatable form; object files normally have a .OBJ extension. Object modules are produced by language processors (such as MACRO or FORTRAN) and are processed by the Linker to become a run- nable program (in SAV, LDA, or REL format, discussed later). Object files may also be processed by the Librarian to produce library .OBJ files, which are then used by the Linker. Figure 3-3 illustrates this process.
Figure 3-3
Object Module Processing
Many different object modules may be combined to form one file; each object module remains complete and independent. However, object modules combined into a library by the Librarian are no longer independent -- they become part of the library's structure.

Object modules are made up of formatted binary blocks. A formatted binary block is a sequence of 8-bit bytes (stored in an RT-11 file, on paper tape, or by some other means) and is arranged as illustrated in Figure 3-4.

![Diagram of a formatted binary block](image)

**Figure 3-4**
Formatted Binary Block

Each formatted binary block has its length stored within it; the length includes all bytes of the block except the checksum byte. The data portion of each formatted binary block contains the actual object module information (described later). The checksum byte is computed such that the sum of all bytes in the formatted binary block, including the checksum byte, is zero when the sum is masked to 8 bits.

Formatted binary blocks are used to hold various kinds of information in an object module; this information is always contained completely in the data portion of the block, surrounded by the formatted binary block structure.
Eight types of data blocks may be present in an object module:

<table>
<thead>
<tr>
<th>Identification Code</th>
<th>Type of Block</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GSD blocks</td>
<td>hold the Global Symbol Directory information</td>
</tr>
<tr>
<td>2</td>
<td>ENDGSD block</td>
<td>signals the end of GSD blocks in a module</td>
</tr>
<tr>
<td>3</td>
<td>TXT blocks</td>
<td>hold the actual binary &quot;text&quot; of the program</td>
</tr>
<tr>
<td>4</td>
<td>RLD blocks</td>
<td>hold Relocation Directory information</td>
</tr>
<tr>
<td>5</td>
<td>ISD blocks</td>
<td>hold Internal Symbol Directory - not supported by RT-11</td>
</tr>
<tr>
<td>6</td>
<td>ENDMOD block</td>
<td>signals end of the object module</td>
</tr>
<tr>
<td>7</td>
<td>Librarian Header Block</td>
<td>17 words holding the status of the library file</td>
</tr>
<tr>
<td>8</td>
<td>Librarian End Block</td>
<td>signals the end of the library file</td>
</tr>
</tbody>
</table>

The structure of object modules produced by a language processor will be described first, followed by details specific only to Library .OBJ files.

The first block of an object module must be a GSD block, and all GSD blocks must appear before the ENDGSD block. The ENDMOD block must be the last block of the module. Except for these three restrictions, blocks may appear in any order within an object module.

When a 16-bit word is stored as part of the data in a block, it is always stored as two consecutive 8-bit bytes, with the low-order byte first.

The first word (data word) of each type of block mentioned above contains the identification code of that block type (1 = GSD block, etc.) with any information present following the identification word.
3.4.1.1 Global Symbol Directory - The object module's global symbol directory contains the following information:

0 - Module Name
1 - Program Section (CSECT) Definitions
2 - Internal Symbol Table Name (not supported by RT-11)
3 - Transfer (Start) Address
4 - Global Symbol Definitions or References

Each piece of information in the GSD is contained in a GSD item, formatted as shown in Figure 3-5:

![GSD Structure](image)

The code byte identifies the information contained in a GSD item according to the codes listed above (0 = Module Name, 1 = Program Section Definition, etc.). The first GSD item of an object module must contain the Module Name information (FLAGS, CODE, and SIZE = 0). There may be no more than five GSD items per GSD block (i.e., per formatted binary block). As many GSD blocks as necessary may be present, but all must appear before the ENDGSD block. GSD blocks need not be contiguous.

Flags are coded as follows:

<table>
<thead>
<tr>
<th>Bits 0,1,2,4,7</th>
<th>unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 3:</td>
<td>0 = undefined, 1 = defined (used only with Global Symbols)</td>
</tr>
<tr>
<td>Bit 5:</td>
<td>0 = absolute, 1 = relocatable</td>
</tr>
<tr>
<td>Bit 6:</td>
<td>0 = internal, 1 = global</td>
</tr>
</tbody>
</table>
All program sections (CSECTs) defined in a module must be declared in GSD items (code byte = 1). The size word of each program section definition should contain the size in bytes to be reserved for the section. Program sections may be declared more than once, in which case the largest declared size of the section will be used. All global symbols that are defined in a given program section must appear in the GSD items immediately following the definition item of that program section.

A special program section named "ABS." (where  represents a space) is called the absolute section. The absolute section has the special attribute that it is always allocated by the Linker beginning at location 0 of memory. All global symbols that contain absolute (non-relocatable) values should be declared immediately after the GSD item that defines the absolute section. If it is not desired to allocate any memory space to the absolute section, its size word may be specified as zero, even if absolute global symbol definitions occur after it. Flag bit 5 of each absolute global symbol is always set to zero. GSD items that contain the definitions of global symbols (code byte = 4) must immediately follow the program section declaration into which they are to be defined. Flag bit 3 is set to 1 to indicate a symbol definition, bit 5 is set if and only if the symbol is relocatable, and bit 6 is set to indicate that the symbol being defined is a global. In addition, the offset word is set to contain the defined value of the global symbol, relative to the base of the program section in which the global is defined. At link time, the Linker assigned section base is added to get the final value of the global symbol.

Global symbols that are referenced but not defined in the current object module must also appear in GSD items. These global references may appear in any GSD item except the very first (which contains the module name). Global references are recognized by code byte 4 with flag bit 3=0, bit 5 is undetermined, and bit 6=1. All global symbols used in the RLJ of the object module (described later) must appear in at least one Global Symbol or Program Section GSD item.

If RT-11 is to begin execution of a program within a particular object module of that program, then the information on where to start is given in a Transfer Address (code = 3) GSD item. The first even transfer address encountered by the Linker will be passed to RT-11 as the program start address. Whenever the resulting program is run (using R or RUN
for SAV images, FRUN for REL files, or the absolute loader for LDA files), the start address is used to indicate the first executable instruction. If no transfer address is present or if all are odd, the resulting program will not self-start when run. In a Transfer Address GSD item, the name field is used to specify a program section (or global name) and the offset word is used to indicate the offset from the base of that program section (or global) to the starting point of the program. The program section or global name referenced need not be defined in the current object module, but must be defined in some object module included at link time.

**NOTE**

Program Section and Global names must begin with an alphabetic or numeric character, except for the names .ABS, and .LOD.

3.4.1.2 ENGDGSD Block - The ENGDGSD block contains a single data word, and that is the identification code of the ENGDGSD block (2). All GSD blocks in an object module must precede the ENGDGSD block.

3.4.1.3 TXT Blocks and RLD Blocks - The first TXT block (3) in an object module (if present) must be preceded by an RLD block (4).

TXT blocks contain the actual binary form of the programs and are formatted as shown in Figure 3-6:

![Figure 3-6](image)

**Figure 3-6**
TXT Block Format

The load address of a TXT block gives the relative address of the first byte of the absolute load data. The address is relative to the base of the last program section given in a Location Counter Definition RLD command (explained later).
The Absolute Load Data contains the actual bytes that will be loaded into memory when the program is run (except for relocations, described later).

RLD blocks contain variable length RLD commands, used to modify and complete the information contained in TXT blocks. Except for the Location Counter commands, RLD information must appear in an RLD block immediately following the TXT block to be modified.

Available RLD commands are:

1. Internal Relocation
2. Global Relocation
3. Internal Displaced Relocation
4. Global Displaced Relocation
5. Global Additive Relocation
6. Global Additive Displaced Relocation
7. Location Counter Definition
8. Location Counter Modification (not used by RT-11)
9. Set Program Limits

The location counter commands (numbers 7 and 8) are the only two RLD commands that must appear in an RLD block preceding the text blocks modified. The first RLD block must precede the first TXT block and must contain only a location counter definition command (7) in order to declare a program section for loading the first text block. (The location counter modification command (8) is included for compatibility with other systems, but is not used by RT-11.)

The data portion of an RLD block must not be larger than 42 bytes including the identification word (RLD=4) and all RLD commands.

All global names and program section names that appear in RLD commands must appear in GSD items in the same object module. Figure 3-7 shows the format of each RLD command (each part except the first word is optional and may not appear in some commands):
An RLD command may be 1, 2, 3, or 4 words long.

The Command Field contains the command code (1 = Internal Relocation, etc.). The Command Field occupies bits 0-6 of the first word of the command. The B field (bit 7) indicates a word command if 0 or a byte command if 1 (only valid for commands 1 through 6). The Relative Reference Field is a pointer into the preceding TXT block and is used with RLD commands that require text locations for modification (commands 1 through 6 and 9). This field specifies the displacement from the beginning of the preceding TXT block to the referenced text data byte (or word). The beginning of the TXT block is the identification word (the first word of the data portion of the block). Thus, the smallest relative reference will normally be 4 (the first byte (word) of the preceding TXT block).

The Name Field is used to hold a Global or Program Section name if the command requires it.

The Constant Field is used to hold a relative address or additive quantity if the command requires it. RLD commands are processed by the Linker as shown in the following situations:

1. Internal Relocation (code 1) - Add the current program section's base to the specified constant and place the result where indicated. This command relocates a direct pointer to an internal relocatable symbol.
Examples:
   a) .WORD LOCAL
   b) MOV #LOCAL,%0

2. Global Relocation (code 2) - Place the value of the specified global symbol where indicated. This command generates a direct pointer to an external symbol.

<table>
<thead>
<tr>
<th>Relative Reference</th>
<th>$/1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global Name</td>
<td></td>
</tr>
</tbody>
</table>

Examples:
   a) .WORD GLOBAL
   b) MOV #GLOBAL,%0

3. Internal Displaced Relocation (code 3) - Calculate the displacement from the position of the current location plus two to the specified absolute address, and store the result where indicated. This command occurs only when there is a reference to an absolute (non-relocatable) location from a relocatable section.

<table>
<thead>
<tr>
<th>Relative Reference</th>
<th>$/1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constant</td>
<td></td>
</tr>
</tbody>
</table>

Examples:
   a) ABS=177550
      TST ABS
   b) CLR 177550
      both addresses cause internal displaced relocation to occur
4. Global Displaced Relocation (code 4) - Calculate the displacement from the current location plus two to the specified global address, and store the result where indicated.

<table>
<thead>
<tr>
<th>Relative Reference</th>
<th>G/1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
.GLOBAL GLOBAL
MOV GLOBAL,R6
```

5. Global Additive Relocation (code 5) - Add the value of the specified global symbol to the specified constant, and store the result where indicated.

<table>
<thead>
<tr>
<th>Relative Reference</th>
<th>G/1</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
.GLOBAL GLOBAL
CMP #GLOBAL+6,R6
```

6. Global Additive Displaced Relocation (code 6) - Calculate the displacement from the current location plus two to the address specified by the sum of the global symbol value and the given constant, and place the result where indicated.

<table>
<thead>
<tr>
<th>Relative Reference</th>
<th>G/1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
.GLOBAL GLOBAL
CLR GLOBAL+6
```
7. Location Counter Definition (code 7) - This command is used to specify the program section into which the following TXT blocks are to be loaded.

<table>
<thead>
<tr>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Section Name</td>
</tr>
<tr>
<td>Constant</td>
</tr>
</tbody>
</table>

This command is generated whenever .ASECTION or .CSECTION is used to initiate or continue a program section. The constant word is effectively ignored by RT-11 and may be used for diagnostic purposes to indicate the relative point at which a program section is being entered.

8. Location Counter Modification (code 10) - This command is used to enter the current program section at a different point. This command is effectively ignored by RT-11 and is used for diagnostic purposes only.

<table>
<thead>
<tr>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
</tr>
</tbody>
</table>

Examples:

a) .=100 ;IF WE ARE IN THE ASECTION
b) .=.-20 ;IF WE ARE IN A RELOCATABLE SECTION

9. Set Program Limits (code 11) - This command (generated by the .LIMIT assembler directive) causes two words in the preceding TXT block to be modified. The first word is to be set to the lowest relocated address of the program. The second word is to be set to the address of the first free location following the relocated code. Note that both words to be modified must appear in the same TXT block.

<table>
<thead>
<tr>
<th>Relative Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

In addition to the above commands, note that commands numbered 14, 15, and 16 can be generated by MACRO. These commands are identical to commands 4, 5, and 6 respectively, but are used when the global is really a program section name.
3.4.1.4 ISD Internal Symbol Directory - Not supported by RT-11.

3.4.1.5 ENDMOD Block - Every object module must end with an ENDMOD block. The ENDMOD block contains a single data word -- the identification code of the ENDMOD block (6).

3.4.1.6 Librarian Object Format - A library .OBJ file contains information additional to that previously defined. The object modules in a library file are preceded by a Library Header Block and Library Directory, and are followed by the Library End Block or trailer. This is illustrated in Figure 3-8.

![Figure 3-8 Library File Format](image)

Diagrams of each component in the library file structure are included here, but Chapter 7 of the RT-11 System Reference Manual should be consulted for details.

The library header is composed of 17\textsubscript{10} words describing the status of the file. The contents of the 17 words are shown in Figure 3-9.
The Entry Point Table (EPT), Figure 3-10, is composed of four-word entries which contain information related to all object modules in the library file.
Object modules follow the Entry Point Table and consist of the types of data blocks already discussed: GSD, ENDGSD, TXT, RLD, and ENDMOD. The information in these blocks is used by the Linker during creation of the load module.

Following all object modules is a specially coded Library End Block (trailer), which signifies the end of the file, shown in Figure 3-11.

<table>
<thead>
<tr>
<th>1</th>
<th>FORMATTED BINARY HEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>FORMATTED BINARY LENGTH</td>
</tr>
<tr>
<td>10</td>
<td>TYPE CODE</td>
</tr>
<tr>
<td>0</td>
<td>NOT USED (MUST BE ZERO)</td>
</tr>
<tr>
<td>357</td>
<td>CHECKSUM BYTE</td>
</tr>
</tbody>
</table>

Figure 3-11
Library End Trailer

3.4.2 Formatted Binary Format (.LDA)

The Linker /L switch produces output files in a paper tape compatible binary format.

Paper tape format, shown in Figure 3-12, is a sequence of formatted binary blocks (as explained in Section 3.4.1 and in Figure 3-4). Each formatted binary block represents the data to be loaded into a specific portion of memory. The data portion of each formatted binary block consists of the absolute load address of the block followed by the absolute data bytes to be loaded into memory beginning at the load address. There may be as many formatted binary blocks as necessary in an LDA file. The last formatted binary block of the file is special; it contains only the program start address in its data portion. If this address is even, the loader passes control to the loaded program at this address. If it is odd, the loader halts upon completion of loading. The final block of the LDA file is recognized by the fact that its length is 6.
The load module's binary blocks contain only absolute binary load data and absolute load addresses; all global references have been resolved and the appropriate relocation has been performed by the Linker.

3.4.3 Save Image Format (.SAV)

Save image format is used for programs that are to be run in the background. This format is essentially an image of the program as it would appear in memory (block 0 of the file corresponds to memory locations 0-776, block 1 to locations 1000-1776, and so forth).
Locations 360-377 in block 0 of the file are restricted for use by the system. The Linker stores the program memory usage bits in these eight words. Each bit represents one 256-word block of memory and is set if the program occupies that block of memory. This information is used by the R, RUN, and GET commands when loading the program.

When loading a save image program into memory, KMON reads block 0 of the file to extract the memory usage bits. These bits are used to determine whether the program will overlay either the KMON or the USR. If these portions of the monitor will not be overlaid, the entire program is loaded; if the USR and KMON must swap, KMON loads the resident portion of the program, up to the start of KMON. It then puts the portion of the program that overlays KMON/USR into the system swap blocks. When the program starts, the monitor swaps in the virtual portion of the program, overlaying KMON.

When block 0 of a save image file is loaded, each word is checked against the protection bit map (LOWMAP), which is resident in RMON. Locations that are protected in the map, such as location 54 and the system device vectors, are not loaded.

3.4.4 Relocatable Format (.REL)

A foreground job is linked using the Linker /R switch. This causes the Linker to produce output in a linked, relocatable format, with a REL file extension.

The object modules used to create a REL file have been linked and all global references have been resolved. The REL file is not relocated, so it has an effective start address of 0, with relocation information included to be used at FRUN time. The relocation information in the file is used to determine which words in the program must be relocated when the job is installed in memory.

In order to determine if the code to be relocated (as indicated in the relocation information blocks) is to have positive or negative relocation (relative to the start address of the program), the following criteria from the text modification commands is used (R = relative address, G = global address, C = constant):

1. Internal Relocation (.WORD R) - always positive relocation (absolute)

2. Global Relocation (.WORD G) - positive relocation only if the global is not absolute
3. Internal Displaced Relocation - always negative relocation (MOV 54,R)

4. Global Displaced Relocation (MOV G,R) - negative relocation only where the global is defined as absolute elsewhere

5. Global Additive Relocation (.WORD G + C) - same as 2 above

6. Global Additive Displaced (MOV G + C,R) - same as 4 above

7. Program Counter Commands - not applicable

8. Set Program Limits - always positive relocation (requires 2 RELs; limit is two words)

There are two types of REL files to consider, those programs with overlay segments and those without.

3.4.4.1 Non-Overlay Programs - A REL file for a non-overlaid program appears as shown in Figure 3-13:

```
<table>
<thead>
<tr>
<th>Block Ø</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Program Text</td>
</tr>
</tbody>
</table>
```

**Figure 3-13**
REL File Without Overlays

Block 0 (relative to start of the file) contains certain information required by the FRUN processor:

<table>
<thead>
<tr>
<th>Offset from Beginning of Block 0</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>Size of the program root segment in bytes</td>
</tr>
<tr>
<td>54</td>
<td>Size of the overlay region in words; 0 if no overlays</td>
</tr>
<tr>
<td>56</td>
<td>REL file identification word, which must contain the RAD50 value of the characters 'REL'</td>
</tr>
<tr>
<td>6Ø</td>
<td>Relative block number of relocation information</td>
</tr>
</tbody>
</table>
In addition, the system communication locations (34-50) contain the following information:

<table>
<thead>
<tr>
<th>Offset from Beginning of Block 0</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>34,36</td>
<td>TRAP vector</td>
</tr>
<tr>
<td>40</td>
<td>Start address of program</td>
</tr>
<tr>
<td>42</td>
<td>Initial setting of stack pointer</td>
</tr>
<tr>
<td>44</td>
<td>Job Status Word</td>
</tr>
<tr>
<td>46</td>
<td>USR swap address</td>
</tr>
<tr>
<td>50</td>
<td>Highest memory address in user's program</td>
</tr>
</tbody>
</table>

In the case of non-overlaid programs, the FRUN processor performs the following general steps to install a foreground job.

1. Block 0 of the file is read into an internal monitor buffer.

2. The amount of memory required for the job is obtained from location 52 of block 0 of the file, and the space is allocated.

3. The program text is read into the space just allocated for it.

4. The relocation information is read into an internal buffer.

5. The locations indicated in the relocation information area are relocated by adding the relocation quantity, which is the starting address the job occupies in memory.

The relocation information consists of a list of addresses relative to the start of the user's program. This list is scanned, and the appropriate locations in the user's program area are updated with a constant. The job is then ready to be started.
The relocation information is in the following format:

| 15 | 14 | $0$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIVE WORD OFFSET</td>
<td>RELATIVE WORD OFFSET</td>
<td>RELATIVE WORD OFFSET</td>
</tr>
<tr>
<td>RELATIVE WORD OFFSET</td>
<td>RELATIVE WORD OFFSET</td>
<td>RELATIVE WORD OFFSET</td>
</tr>
<tr>
<td>RELATIVE WORD OFFSET</td>
<td>RELATIVE WORD OFFSET</td>
<td>RELATIVE WORD OFFSET</td>
</tr>
</tbody>
</table>

Bits 0-14 represent the relative address to relocate divided by two. This implies that relocation is always done on a word boundary, which is the case. Bit 15 is used to indicate the type of relocation to perform, positive or negative. The relocation constant (which is the load address of the program) is added to or subtracted from the indicated location depending on the sense of bit 15; 0 implies addition, 1 implies subtraction. 177776 terminates the list of relocation information.

Following is an example of a simple, non-overlaid program linked to produce a REL file. A dump of the file follows the program.
In block 0, word 50 shows the highest, non-relocated, memory address in the user program. Word 52 shows the program size in bytes. Word 54 shows the size of the overlay region. The value is non-zero only for programs with overlays. Word 60 contains a 3, indicating that the relocation information begins at block 3 of the file.
<table>
<thead>
<tr>
<th>BLOCK NUMBER</th>
<th>0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>127890</td>
</tr>
<tr>
<td>0002</td>
<td>127890</td>
</tr>
<tr>
<td>0003</td>
<td>127890</td>
</tr>
<tr>
<td>0004</td>
<td>127890</td>
</tr>
<tr>
<td>0005</td>
<td>127890</td>
</tr>
</tbody>
</table>

This block corresponds to locations 0-776 in the assembly listing.
This block corresponds to locations 1000-1776 in the assembly listing.
This block shows the root relocation information. The first word of block 3 is a 3; since this is positive, positive relocation is indicated. Locations 6, 14, 30, 46, 56, 100, 126, and 136 must all be positively relocated at FRUN time. (On examination of the assembly listing, those locations marked with a ' need to be relocated.) The 177776 terminates the list.

Had negative relocation been indicated at relative location 6, block 3 would have shown 100003, 6, 14, 23, 27, 40, 53, 57, 177776.

3.4.4.2 RFL Files with Overlays - When overlays are included in a program, the file is similar to that of a non-overlaid program. However, the overlay segments must also be relocated. Since overlays are not permanently memory resident but are read in from the file as needed, they require an additional operation. Each overlay segment is relocated (by FRUN) and then rewritten into the file. Then, when the overlay is called in, it will be properly relocated. This process takes
place each time an overlaid file is run with FRUN. The relocation information for overlay files contains both the list of addresses to be modified and the original contents of each location. This allows the file to be FRUN after the first usage.

NOTE

.ASECTs are illegal above 1000, and restricted in an overlaid foreground job. Refer to Chapter 6 of the RT-ll System Reference Manual.

A REL file with overlays appears as shown in Figure 3-14:

Figure 3-14
REL File With Overlays
In this case, location 54 of block 0 of the REL file contains the size of the overlay region, in words. This is used to allocate space for the job when added to the size of the program base segment in location 52.

After the program base (root) code has been relocated, each existing overlay is read into the program overlay region in memory, relocated via the overlay relocation information, and then written back into the file.

The root relocation information section is terminated with a -1. This -1 is also an indication that an overlay segment relocation block follows. The overlay segment relocation block is shown in Figure 3-15:

```
-1
Overlay blk #
Overlay Size
Relative Word Offset
Text to Relocate
Relative Word Offset
Text to Relocate
|   |
|   |
Relative Word Offset
Text to Relocate
-1
```

-1: Root (or Previous Overlay) Terminator
Overlay blk #: Start of Overlay Relative to Start of File
Overlay Size: (words)
Relative Word Offset
Text to Relocate
Relative Word Offset
Text to Relocate
|   |
|   |
Relative Word Offset
Text to Relocate
-1: Flag Indicating Start of New Overlay

Figure 3-15
Overlay Segment Relocation Block
The displacement is relative to the start of the program and is interpreted as in the nonoverlaid file (i.e., bit 15 indicates the type of relocation, and the displacement is the true displacement divided by two). Encountering -1 indicates that a new overlay region begins here. A -2 indicates the termination of all relocation information.
CHAPTER 4
SYSTEM DEVICE

4.1 DETAILED STRUCTURE OF THE SYSTEM DEVICE

The RT-11 system device holds all the components of the system and is used by RT-11 to store device handlers and the monitor file. The layout of the system device is:

<table>
<thead>
<tr>
<th>Block #</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bootstrap</td>
</tr>
<tr>
<td>1</td>
<td>Reserved for volume identification information</td>
</tr>
<tr>
<td>2</td>
<td>Bootstrap</td>
</tr>
<tr>
<td>3 to 5</td>
<td>Reserved for monitor or bootstrap expansion</td>
</tr>
<tr>
<td>6 to (N*2)+5</td>
<td>Directory segments; N is the number of directory segments</td>
</tr>
<tr>
<td>(N*2)+6 to end</td>
<td>File storage</td>
</tr>
</tbody>
</table>

All other system components, i.e., the monitor and device handlers, are files on the system device:

<table>
<thead>
<tr>
<th>File</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITR.SYS</td>
<td>The current RT-11 monitor; contains bootstrap, KMON, USR/CSI, RMON, KMON overlays, scratch blocks</td>
</tr>
<tr>
<td>SYSMAC.SML</td>
<td>System Macro Library</td>
</tr>
<tr>
<td>SYSMAC.S8K</td>
<td>8K System Macro Library</td>
</tr>
<tr>
<td>LP.SYS</td>
<td>Line printer handler</td>
</tr>
</tbody>
</table>

4-1
<table>
<thead>
<tr>
<th>File</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT.SYS</td>
<td>DECTape handler</td>
</tr>
<tr>
<td>TT.SYS</td>
<td>Console handler (S/J only)</td>
</tr>
<tr>
<td>RK.SYS</td>
<td>RK disk handler</td>
</tr>
<tr>
<td>DS.SYS</td>
<td>RJS03/4 fixed-head disk handler</td>
</tr>
<tr>
<td>DX.SYS</td>
<td>RX01 flexible disk handler</td>
</tr>
<tr>
<td>DP.SYS</td>
<td>RP disk handler</td>
</tr>
<tr>
<td>PR.SYS</td>
<td>High-speed reader handler</td>
</tr>
<tr>
<td>PP.SYS</td>
<td>High-speed punch handler</td>
</tr>
<tr>
<td>CR.SYS</td>
<td>Card reader handler</td>
</tr>
<tr>
<td>RF.SYS</td>
<td>RF disk handler</td>
</tr>
<tr>
<td>CT.SYS</td>
<td>Cassette handler</td>
</tr>
<tr>
<td>MT.SYS</td>
<td>TM11 magtape handler</td>
</tr>
<tr>
<td>MM.SYS</td>
<td>TJU16 magtape handler</td>
</tr>
<tr>
<td>BA.SYS</td>
<td>BATCH run-time handler</td>
</tr>
</tbody>
</table>

In general, files with the .SYS extension are parts of the monitor system. The bootstrap records the block numbers of the relevant areas in the monitor tables at bootstrap time. Thus, RT-11 is extremely flexible with respect to the interchange and construction of systems.

4.2 CONTENTS OF MONITR.SYS

Following is the block layout of the RT-11 monitor file, MONITR.SYS. Block numbers are relative to the start of the file.

<table>
<thead>
<tr>
<th>F/B Monitor</th>
<th>Block # (decimal)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-1</td>
<td>Copy of system bootstrap (blocks 0 and 2 of the system device)</td>
</tr>
<tr>
<td></td>
<td>2-17</td>
<td>Swap blocks</td>
</tr>
<tr>
<td></td>
<td>18-24</td>
<td>KMON (includes 2-block KMON overlay area)</td>
</tr>
<tr>
<td></td>
<td>25-32</td>
<td>USR/CSI</td>
</tr>
<tr>
<td></td>
<td>33-47</td>
<td>RMON</td>
</tr>
<tr>
<td></td>
<td>48-57</td>
<td>KMON overlays</td>
</tr>
</tbody>
</table>

4-2
<table>
<thead>
<tr>
<th>S/J Monitor</th>
<th>Block # (decimal)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-1</td>
<td>Copy of system bootstrap</td>
</tr>
<tr>
<td></td>
<td>2-16</td>
<td>Swap blocks</td>
</tr>
<tr>
<td></td>
<td>17-22</td>
<td>KMON (includes 1-block KMON overlay area)</td>
</tr>
<tr>
<td></td>
<td>23-30</td>
<td>USR/CSI</td>
</tr>
<tr>
<td></td>
<td>31-37</td>
<td>RMON</td>
</tr>
<tr>
<td></td>
<td>38-44</td>
<td>KMON overlays</td>
</tr>
</tbody>
</table>

### 4.3 KMON OVERLAYS

The KMON overlays are one block in size in the S/J Monitor and two blocks in size in the F/B Monitor. The contents of each overlay are described in this list:

<table>
<thead>
<tr>
<th>Overlay #</th>
<th>S/J</th>
<th>F/B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DATE, TIME</td>
<td>DATE, TIME, SAVE, ASSIGN</td>
</tr>
<tr>
<td>1</td>
<td>SAVE, ASSIGN</td>
<td>LOAD, UNLOAD, SUSPEND, RESUME, CLOSE, FRUN (Part 1)</td>
</tr>
<tr>
<td>2</td>
<td>LOAD, UNLOAD, CLOSE</td>
<td>FRUN (Part 2)</td>
</tr>
<tr>
<td>3</td>
<td>GT ON/OFF</td>
<td>GT ON/OFF, SET</td>
</tr>
<tr>
<td>4</td>
<td>SET</td>
<td></td>
</tr>
</tbody>
</table>

### 4.4 DETAILED OPERATION OF THE BOOTSTRAP

Bootstrapping a system causes a fresh copy of that system to be installed in memory. In the RT-11 boot, certain system device resident tables are also updated. Following is a detailed description of the bootstrap.
<table>
<thead>
<tr>
<th>Action</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. User executes hardware bootstrap</td>
<td>This causes block 0 of the system device to be read into 0-777. Control then passes to location 0.</td>
</tr>
<tr>
<td>2. Second part of bootstrap is read</td>
<td>The first part of the boot reads the second half into 1000-1777.</td>
</tr>
<tr>
<td>3. Determine how much memory is available</td>
<td>Boot sets a trap at location 4 and then starts addressing memory. When the trap is taken, illegal memory has been addressed.</td>
</tr>
<tr>
<td>4. Look for special devices</td>
<td>Boot sets a trap at location 10 and then tries to address the clock, FPU, and VT11 display processor. Their presence or absence is indicated in the CONFIG word in RMON.</td>
</tr>
<tr>
<td>5. Check memory size</td>
<td>If memory is too small to read in the monitor, a message is printed and the boot halts.</td>
</tr>
<tr>
<td>6. Read in directory and find MONITR.SYS</td>
<td>The entire directory is searched. If MONITR.SYS is not found, a HALT occurs after the boot prints an error message.</td>
</tr>
<tr>
<td>7. Read the monitor into memory</td>
<td>The monitor file, MONITR.SYS, is read into the highest 4K of memory.</td>
</tr>
<tr>
<td>8. Put pointers to monitor file blocks into RMON</td>
<td>RMON references the monitor swap blocks directly. Thus, the position of the swap blocks varies as the placement of MONITR.SYS varies. The real position of the blocks is updated for each boot operation.</td>
</tr>
<tr>
<td>9. Update position-dependent areas in RMON</td>
<td>MONITR.SYS is initially linked at 8K. However, if more than 8K is available, RT-11 uses it. To do that, certain words must be updated to point to the actual areas of high memory where they will be. Boot contains a list of all words to be updated, located at RELLIST in BSTRAP.MAC.</td>
</tr>
<tr>
<td>10. LOOKUP the device handlers in system and store their record numbers in $DVREC</td>
<td>Boot looks at $PNAME table to find the names of the devices in the system. The extension .SYS is appended. Thus, the PR handler is a file called PR.SYS. The location of the handler is then placed in $DVREC. If the LOOKUP fails, the device gets a 0 in its $DVREC entry. That implies that the device handler does not exist.</td>
</tr>
</tbody>
</table>
11. Print bootstrap header

   Boot prints monitor identification message "RT-11" followed by monitor type ("FB" or "SJ") followed by version number.

12. Set up locations 0 and 2

   Boot puts a "BIC R$1,R$0" in location zero and an .EXIT EMT in location 2.

13. Turn on KW11-L Clock

   The bootstrap turns on the clock, if present in the configuration.

14. Exit to Keyboard Monitor

4.5 FIXING THE SIZE OF A SYSTEM

RT-11 is designed to automatically operate from the top of the highest available 4K memory bank. However, it is possible to force the system to operate from a specified area that is not necessarily the highest. For instance, the following series of commands causes RT-11 to run in a 16K environment, even though the configuration actually has 28K of memory:

   .R PATCH<CR>                      [Run RT-11 PATCH program.]

   PATCH Version number

   FILE NAME--
   *MONITR.SYS/M<CR>
   *BHALT/4070<CR>
   *E
   .R PIP
   *A=MONITR.SYS/U<CR>
   *SY:/O

   [Specifying MONITR.SYS/M indicates it is a monitor file. Change location "BHALT" from a 407 to a 0 (HALT). The correct address of BHALT can be found in Chapter 5 of Getting Started With RT-11 (V02B). E causes an exit to the monitor. Now run PIP to update the bootstrap and reboot the system.]

When the bootstrap is performed, the computer halts. The halt allows the user to enter the desired size in the switch register. With this patch installed, the V2 bootstrap uses the top five bits (bits 11-15) of the switch register to determine memory size. If the switch register contains the number 160000 or greater (e.g., if the register is unchanged after booting the system), a normal memory determination is performed. Otherwise, the top five bits are taken to be a number representing the number of 1K word blocks of memory. Each bit has the following value:
Switch Register

<table>
<thead>
<tr>
<th></th>
<th>Memory Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>1K</td>
</tr>
<tr>
<td>10000</td>
<td>2K</td>
</tr>
<tr>
<td>20000</td>
<td>4K</td>
</tr>
<tr>
<td>40000</td>
<td>8K</td>
</tr>
<tr>
<td>100000</td>
<td>16K</td>
</tr>
</tbody>
</table>

A combination of the bits will produce the range of system sizes from 8K through 28K, in 1K increments.

Examples:

1. To boot a system into 24K on a 28K configuration, use the combination:

   \[140000 = 100000 \text{(16K)} + 40000 \text{(8K)}\]

2. To boot the S/J Monitor into 11K, use the combination:

   \[54000 = 40000 \text{(8K)} + 10000 \text{(2K)} + 4000 \text{(1K)}\]

When the switch register is set properly, press the CONTINUE switch and the bootstrap will be executed.

If the CONTINUE switch is pressed immediately following the halt without changing the switch settings, a normal memory determination is done. To change the bootstrap back to its original (non-halting) form, execute the same commands as above, but change the 0 at BHALT back to a 407.

This procedure allows the user to 'protect' memory areas, since RT-ll never accesses memory outside the bounds within which it runs.

Another useful procedure, when desiring to always boot a system into a specific memory size or when the console switch register is not available, is to determine the bit combination corresponding to the choice of memory size, as explained above. Then enter the following commands, where xxxxxx represents the bit pattern just determined:

\[\_R \text{ PATCH}<CR>\]

[Run RT-ll PATCH program.]

PATCH Version number

FILE NAME--

MONITR.SYS/M<CR>
*BHALT/ 4F7  248<LF>
*BHALT+2/ 137B2  127B2<LF>
*BHALT+4/ 177570  xxxxxx<CR>
*E

[NOP the branch at BHALT Change MOV #SP,R2 to MOV #VAL,R2. Address of switch register is replaced with one of the bit combinations described previously.]
For the patch addresses for other system devices, and for the address of BHALT, consult the Getting Started with RT-11 (V02B) manual.
CHAPTER 5
I/O SYSTEM, QUEUES, AND HANDLERS

I/O transfers in RT-11 are handled by the monitor through routines known as device handlers. Device handlers are resident on the system mass storage device and can be called into memory at a location specified by the user (via a .FETCH handler request or KMON LOAD command). Only the device handlers distributed with the system in use (V2 or V02B) may be used; the system will malfunction otherwise.

This chapter describes how to write a new device handler and add it to the system. A summary of differences between Version 1 and Version 2 Device Handler requirements is included for the user who wishes to update old device handlers. Instructions and examples for making a device the system device and for writing a new bootstrap for the device are also included.

5.1 QUEUED I/O IN RT-11

Once a device handler is in memory, any .READ/.WRITE requests for the corresponding devices are interpreted by the monitor and translated into a call to the I/O device handler. To facilitate overlapped I/O and computation, all I/O requests to RT-11 are done through an I/O queue. This section details the structure of the I/O queueing system.

5.1.1 I/O Queue Elements

The RT-11 I/O queue is made up of a linked list of queue elements. A single element has the structure shown in Figure 5-1:
Figure 5-1
I/O Queue Element

RT-11 maintains one queue element in the Resident Monitor. (In F/B, one element per job is maintained in the job's impure area.) This is sufficient for any program that uses wait-mode I/O (.READ/.WRITW). However, for maximum throughput, the .QSET programmed request should be used to create additional queue elements.

If an I/O operation is requested and a queue element is not available, RT-11 must wait until an element is free to queue the request. This obviously slows up program execution. If asynchronous I/O is desired, extra queue elements should be allocated. It is always sufficient to allocate N new queue elements, where N is the total number of pending requests that can be outstanding at one time in a particular program. This produces a total of N+1 available elements, since the Resident Monitor element is added to the list of available elements.

Diagrammatically, the I/O queue appears as follows:
AVAIL is the list header. It always contains a pointer to an available element. If AVAIL is 0, no elements are currently available.

When an I/O request is initiated, an element is allocated (removed from the list of available elements) and is linked into the appropriate device handler's I/O queue. The handler's queue header consists of two pointers: the current queue element (CQE) pointer, pointing to the element at the top of the list, and the last queue element (LQE) pointer, pointing to the last element entered in the queue. The LQE pointer is used for fast insertion of new elements into the queue.

Device Handler I/O Queue

In this case, the device is associated with element Q1. If another request comes in for that same device before the first completes, a waiting queue is built up for that device.
When the I/O transfer in progress completes, Q1 is returned to the list of available elements, and the transfer indicated by Q2 will be initiated:
When Q2 is completed, it too is returned to the list of available elements.

Note that the order of the queue element linkages may be altered.

A distinction between S/J and F/B operation is that F/B maintains two separate queue structures, one for each active job. The queue headers (AVAIL) are words in the user's impure area. The centralized queue manager dispatches transfers in accordance with job priority. Thus, if two requests are queued waiting for a particular device, the foreground request is honored first. At no time, however, will an I/O request already in progress be aborted in favor of a higher priority request; the operation in progress will complete before the next transfer is initiated.

Another difference between S/J and F/B operation is that the F/B scheduler will suspend a job pending the availability of a free queue element and will try to run another job.

5.1.2 Completion Queue Elements

The F/B Monitor maintains, in addition to the queue of I/O transfer requests, a queue of I/O completion requests. When an I/O transfer completes and a completion routine has been specified in the request (i.e., the seventh word of the I/O queue element is even and non-zero), the queue completion logic in the F/B Monitor transfers the request node (element) to the completion queue, placing the channel
status word and channel offset in the node. This has the effect of serializing completion routines, rather than nesting them. Completion routines are called by the completion queue manager on a first-in/first-out basis, and the completion routines are entered at priority level 0 rather than at interrupt level.

The .SYNCH request also makes use of the completion queue. When the .SYNCH request is entered, the seven-word area supplied with the request is linked into the head of the completion queue, where it appears to be a request for a completion routine. The .SYNCH request then does an interrupt exit. The code following the .SYNCH request is next called at priority level 0 by the completion queue manager. To prevent the .SYNCH block from being linked into AVAIL (the queue of available elements), the word count is set to -1. The completion queue manager checks the word count before linking a queue element back into the list of available elements, and skips elements with the -1 word count.

Figures 5-2 and 5-3 show the format of the completion queue and .SYNCH elements.

```
OFFSET
0  QUEUE LINK
2
4
6
10  CHANNEL STATUS WORD
12  CHANNEL OFFSET
14  COMPLETION ROUTINE ADDRESS
```

Figure 5-2
Completion Queue Element
5.1.3 Timer Queue Elements

Another queue maintained by the F/B Monitor is the timer queue. This queue is used to implement the .MRKT request, which schedules a completion routine to be entered after a specified period of time. The first two words of the element are the high- and low-order time and the seventh word is the completion routine address. An optional sequence number can be added to the request to distinguish this timer request from others issued by the same job.

The F/B Monitor uses the timer queue internally to implement the .TWAIT request. The .TWAIT request causes the issuing job to be suspended and a timer request is placed in the queue with the .RSUM logic as the completion routine. Refer to Figure 5-4 for the format of the timer queue element.

---

**Figure 5-3**
.SYNCH Element

---

**Figure 5-4**
Timer Queue Element
5.2 DEVICE HANDLERS

This section contains the information necessary to write an RT-11 device handler. It is illustrated with an example, a driver for the RS64 fixed-head disk (with RC11 controller). A source listing is included in Appendix A, Section A.1; portions of this listing are referenced throughout the remainder of this section and in future sections.

The user should refer to the PDP-11 Peripherals Handbook for details regarding the operation of any particular peripheral.

NOTE

All RT-11 handlers must be written in position independent code (PIC). Consult the PDP-11 Processor Handbook for information on writing PIC.

5.2.1 Device Handler Format

The first five words of any device handler are header words. The format is:

<table>
<thead>
<tr>
<th>Word #</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of first word of device's interrupt vector.</td>
</tr>
<tr>
<td>2</td>
<td>Offset from current PC to interrupt handler.</td>
</tr>
<tr>
<td>3</td>
<td>Processor status word to be used when interrupt occurs. Must be 340 (priority 7).</td>
</tr>
<tr>
<td>4, 5</td>
<td>Zero. These are the queue pointers.</td>
</tr>
</tbody>
</table>

See area C in the example handler (Section A.1).

A word must be provided at the end of the handler. When the handler is .FETCHed, the monitor places a pointer to the monitor common interrupt entry code in the last word of the handler. This requires that the handler size in the monitor's SHSIZE table be exact or the handler will malfunction. See area M in the example in Section A.1.

The word preceding the interrupt handler entry point must be an unconditional branch to the handler's abort code. The abort code is used by the F/B Monitor to stop I/O for the device. The abort entry point is shown at area G in the example and the abort code is at area K.
5.2.2 Entry Conditions

The device handler is entered directly from the monitor I/O queue manager, at which time it initiates the data transfer. The fifth word of the header contains a pointer into the queue element to be processed. This word (called CQE, for Current Queue Element) points to the third word of the queue element, which is the block number to be read or written. Referring to the example, location RCCQE contains the address of the third word of the queue element to be processed. It is generally advisable to put the pointer into a register, as that greatly facilitates picking up arguments to initiate the transfer. In the example, the entry point is at the location marked by E. Notice that registers need not be saved.

5.2.3 Data Transfer

Most handlers use the interrupt mechanism when transferring data. The handler initiates the transfer and then returns immediately to the monitor with an RTS PC, shown at area F. When the transfer is completed, the device interrupts. When the interrupt routine determines that I/O is complete or that an error has occurred, it jumps to the monitor completion routine in the manner shown at area J in the listing.

If the interrupt mechanism is not used, the data transfer must be completed before returning to the monitor. The handler must loop on a device flag with the interrupt disabled. When I/O is complete, the driver returns to the monitor with a jump to the monitor completion code, similar to that shown at area J in the example.

5.2.4 Interrupt Handler

Once the transfer has been initiated and control has passed back to the monitor, data interrupts will occur.

Information in the header of the handler causes the interrupt to be vectored to the interrupt handling code within the handler. The code at the interrupt location should keep the transfer going, determine when the transfer is complete, and detect errors.

When the transfer is done, control must be passed to the monitor's I/O queue manager, which performs a cleanup operation on the I/O queue.
Restrictions that apply to the interrupt code are:

1. The common interrupt entry into the monitor must be taken. Interrupt routines linked into a program use the .INTEN request described in Chapter 9 of the RT-11 System Reference Manual. Handlers made part of the system have a more efficient method of entry. The last word of the handler is set to point to the monitor common interrupt entry code when the handler is fetched. Upon reception of an interrupt, the handler must execute this code by performing a JSR R5, @SINPTR, where $INPTR is the tag commonly used by RT-11 handlers for the pointer word. See areas I and N in the example. The JSR instruction must be followed by the complement of the priority at which the handler will operate. See area I for an easy method to make the assembler compute the complement. On return from the monitor's interrupt entry code, R4 and R5 have been saved and may be used by the handler. Other registers must be saved and restored if they are to be used.

2. A check must be made to determine if the transfer is complete. However, with nonfile-oriented devices, such as paper tape, line printer, etc., an interrupt occurs whenever a character has been processed. For these devices, the byte count, which is in the queue element, is used as a character count.

Nonfile-structured input devices should be able to detect an end of file condition, and pass that on to the monitor.

NOTE

The queue element contains a word count, not a byte count. The initial entry to the handler should change the word count to a byte count if the device interrupts at each character. The transfer is complete when the byte count decrements to 0.

Before the conversion to bytes is made, the sign of the word count must be determined since it specifies whether this transfer is a Read or Write. A negative word count implies a Write and should be complemented before converting to bytes.

3. Check for occurrence of an error. If a hardware error occurred, the hard error bit in the channel status word (CSW) should be set, the transfer should be aborted, and the monitor completion code executed. The address of the channel status word is in word 2 of the queue element. The error bit is bit 0 of the CSW. Generally, it is advisable to retry a certain number of times if an error occurs. RT-11 currently retries up to eight times before deciding an error has occurred. (Note that this is true for file-structured devices only.) It is
desirable, in case an error occurs, to do a drive
or control reset, where appropriate, to clear the
error condition before a retry is initiated. See
the area between I and N in the example.

4. If the transfer is not complete and no error has
occurred, registers used should be restored, and
an RTS PC executed.

To pass an EOF (End of File) to the monitor, the
2000 bit in the CSW should be set. Refer to the
sample handler in Appendix A for an example of
setting the EOF bit. When EOF is detected on non-
file structured devices, the remainder of the input
buffer must be zeroed.

5. When the transfer is complete, whether an error oc-
curred or not, the monitor I/O completion code must
be entered to terminate activity and/or enter a com-
pletion routine. When return is made to the monitor,
R4 points to the fifth word of the handler (XCCQE
in the example). See area J in the example for the
method of returning to the monitor completion routine.

Handlers should check for special error conditions
that can be detected on the initial entry to the
handler. For example, trying to write on a read-
only device should produce a hard error. It must
be emphasized that the user handlers should inter-
face to the system in substantially the same way
as the handler in Section A.1. This handler is
included as a guide and an example.

5.3 ADDING A HANDLER TO THE SYSTEM

When the handler has been written and debugged, it may be installed
in the system by following the procedures in this section. The process
consists of inserting information about the handler into the monitor
tables listed below.

<table>
<thead>
<tr>
<th>Table to be Changed</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$HSIZE</td>
<td>Size of handler (in bytes).</td>
</tr>
<tr>
<td>$DVSIZ</td>
<td>Size of device in 256-word blocks. If nonfile device, entry = 0.</td>
</tr>
<tr>
<td>$PNAME</td>
<td>Permanent name of the device (should be two alphanumeric characters entered in .RAD50 notation, left-justified).</td>
</tr>
<tr>
<td>$STAT</td>
<td>Device status table. Refer to Section 2.5.2.2 for the format of $STAT table.</td>
</tr>
<tr>
<td>LOWMAP</td>
<td>Low memory protection map; refer to Section 2.5.4.</td>
</tr>
</tbody>
</table>
There is no restriction on handler names; any 2-letter combination not currently in use may be chosen for the new handler and the name may be inserted in any unused slot in the $SPNAME table, or in a slot occupied by a nonexistent device (i.e., a device not installed on the user's system). Note that the name must be entered in .RAD50. Since PATCH does not have a .RAD50 interpretation switch, the name must be entered to PATCH in its numerical form. Appendix C of the RT-11 System Reference Manual contains a .RAD50 conversion table; ODT can also be used to perform .RAD50 conversions.

As an example, assume again the handler for the RC11/RS64 disk (the sample handler in Section A.1) is to be inserted in the system. First, the values of the table entries for this device are determined (the addresses used in the example are for illustrative purposes only; consult Getting Started With RT-11 (V02B) for the correct table addresses for the version in use):

$HSIZE: 316
After assembly, the handler was found to take up 316 bytes. See area 0 in the example listing.

$DVSIZ: 2000
The disk has 1024 (decimal) 256-word blocks for storage.

$SPNAME: .RAD50 /RC/ or 70370
The name assigned is RC. The .RAD50 value of RC is 70370.

$STAT: 100023
The device is file-structured, is a read/write device, and uses the standard RT-11 file structure. The identifier (selected by the user) is 23. Refer to Section 2.5.2.2 for the format of the $STAT table.

LOWMAP: 14
Protect RC vector 210,212 at byte 336 of LOWMAP (refer to Section 2.5.4.).

Once these values have been decided, the steps for inserting the device handler are:

1. Assemble the handler, using either MACRO or ASEMBL.

2. Link the handler at 1000. The name of the handler should be whatever the $SPNAME entry is, with the .SYS extension appended:

```
.R LINK
*RC.SYS=RC
UNDEF GLBLs
```
where RC.OBJ is the handler object module. The default link address is 1000.

5-12
NOTE

If the handler being linked is one that could also be a system device handler, the user can expect one undefined global, $INTEN.

3. Run PATCH to modify the tables and protect the interrupt vectors.

For this example, assume that the table addresses are found to be:

<table>
<thead>
<tr>
<th>Table</th>
<th>S/J Address</th>
<th>F/B Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>$HSIZE</td>
<td>13674</td>
<td>14602</td>
</tr>
<tr>
<td>$DV$</td>
<td>13730</td>
<td>14636</td>
</tr>
<tr>
<td>$PNAME</td>
<td>16516</td>
<td>17640</td>
</tr>
<tr>
<td>$STAT</td>
<td>16552</td>
<td>17674</td>
</tr>
</tbody>
</table>

NOTE

The addresses above are for illustration only. Consult the Getting Started With RT-11 (VF2B) manual for current table addresses and for the address of the monitor base location, BASE. ~ 17660

The tables have room for fourteen (decimal) device entries; all are already assigned by the monitor. Assuming that a given configuration never has all supported devices, however, at least one slot should be available to be overlaid. For example, assume the twelfth slot is occupied by a device not installed on the system, and therefore available for change. The octal offset is 26, which, added to the table addresses above, gives the address of the empty slot:

S/J Monitor:

_R PATCH<CR>

PATCH Version number

FILE NAME--
*MONITRSY$M<CR>[/M is necessary; BASE; FR<CR> Monitor base; 13674/4000 316<CR> $HSIZE table; 13730/0 2000<CR> $DV$ table; 16516/6250 7037<CR> $PNAME table; 16552/4 100023<CR> $STAT table; Check that vectors in 16336/77<CR> permanent map are protected; Exit to monitor]
F/B Monitor

\_R PATCH

PATCH Version number

FILE NAME--
MONITR.SYS/M<CR>
\$BASE;/R<CR>
\$0,14662/4000316<CR>
\$0,14636/2000<CR>
\$0,17648/62507037<CR>
\$0,17674/410023<CR>
\$0,17336/77<CR>
\$ is necessary;
Monitor base;
$SIZE table;
$DVSIZ table;
$NAME table;
$STAT table;
Check that vectors in
permanent map are protected;
Exit to monitor

At this point, the system should be re-bootstrapped to make the modified monitor resident. The device RC will then be available for use.

5.4 WRITING A SYSTEM DEVICE HANDLER

This section describes the procedures for writing a new system device handler. A system device is the device on which the monitor and handlers are resident. RT-11 currently supports the RK, RF, DP, DS, and DX disks, and DEctape as system devices. The procedures for writing the handler and creating a new monitor are explained, illustrated by the example in Section A.1, the RC11/RS64 handler.

The basic requirements for a system device are random access and read/write capability. These requirements are met by the RC11 disk, which is a multiple platter, fixed-head disk. When writing the driver, the procedures in Section 5.2 should be followed. Because the system handler is linked with the monitor, the additional tagging and global conventions described here must also be followed.

5.4.1 The Device Handler

The following conditions must be observed when writing a system handler. Refer to the example listing in Section A.1.

1. The handler entry point must tagged xxSYS, where xx is the 2-letter device name. For the RC disk, this is RCSYS. See area D in the listing.
   Important: Note that the tag is placed after the third word of the header block.

2. The entry points of all current system devices must be referenced in a global statement. These
currently include RKSYS, RFYS, DSSYS, DXSYS, DPSYS, DTSYS and RCSYS. See Area A.

3. The entry point tags of all other system devices must be equated to zero. See area B in the listing.

4. A .CSECT SYSHND must be included at the top of the handler code. It is located above area C in the example.

5. The last word of the handler is used for the common interrupt entry address. This should have the tag $INPTR and should be set to the value $INTEN. See areas M and N in the example listing. These tags should be global. See area A.

6. The interrupt entry point should have the tag xxINT, or RCINT for this example, and this must be a global. See areas A and H.

7. The handler size must be global, with the symbolic name xxSIZE, or RCSRZE. See area A. This step is not necessary if the monitor sources are available and are being reassembled, since the global will be generated by the HSIZE macro. See Step 3 in Section 5.4.3.

5.4.2 The Bootstrap

This section describes the procedure for modifying the system bootstrap to operate with a new system device. Either the bootstrap source must be acquired, or the listing in Section A.2 may be used. Again, the RC11/RS64 disk is used for an example. The references in this section, however, are to the bootstrap listing found in Section A.2 of Appendix A.

The following changes must be made to the bootstrap to support a new system device:

1. Add a new conditional, $xxSYS, to the list at point AA. Here xx is the 2-letter device name, and in this case the conditional is $RCSYS.

2. Add a simple device driver for the device inside a $xxSYS conditional. This is shown at area CC. Because the RC11 is similar to the other disks, it is possible to share code with the other device drivers, reducing the implementation effort. To do this, the $RCSYS conditional is added at area BB and the device specific code is at area FF. This code merges with the common code at area GG.
3. The device driver has these characteristics:
   
a. The SYSDEV macro must be invoked for the device. The macro arguments are the 2-
   letter device name and the interrupt vector address. For this example, the arguments
   are "RC" and "210", shown at area DD on the listing.

b. The device driver entry point must have the tag READ. See area EE.

c. When the driver is entered:

   \[ \begin{align*}
   \text{R0} &= \text{Physical Block Number} \\
   \text{R1} &= \text{Word Count} \\
   \text{R2} &= \text{Buffer Address} \\
   \text{R3, R4, R5} &= \text{are available for use by the driver routine}
   \end{align*} \]

d. The driver must branch to BIOERR if a fatal I/O error occurs.

5.4.3 Building the New System

This section describes the procedure for building a new monitor using
the system device handler and bootstrap just developed. Again, the
example used is the RCl1/RS64 disk, and the appropriate listings are
those in Sections A.1 and A.2.

The procedure is:

1. Assemble the handler, producing an object module with
   the name xx.OBJ, where xx is the 2-letter device name.
   In this example, the name is RC.

   \[ \text{R MACRO} \]
   \[ \text{RC.OBJ=RC.MAC} \]

2. Assemble the bootstrap, defining the conditional $xxSYS
   (where xx is again the device name; e.g., $RCSYS). De-
   fine the conditional BF if an F/B bootstrap is desired.
   Let BF be undefined for an S/J bootstrap. For the S/J
   bootstrap:

   \[ \text{R MACRO<CR>} \]
   \[ \text{RCBTSJ=TT:,DK: BSTRAP<CR>} \]
   \[ \text{Z$RCSYS}=1<CR>} \]
   \[ \text{ZERRORS DETECTED: } \emptyset \]
   \[ \text{FREE CORE: 1568. WORDS} \]
For the F/B bootstrap:

```
.R MACRO<CR>
*RCBTFB=TT:,DK:BSTRAP<CR>
^$RCSYS=1<CR>
BF=1<CR>
^Z$RCSYS=1<CR>
BF=1<CR>
^ZERRORS DETECTED: 0
FREE CORE: 1558. WORDS
```

3. If the monitor sources are available, the DEVICE macro described in Section 2.5.2.9 can be invoked for the new device by editing the macro call into RMONFB.MAC and RMONSJ.MAC and reassembling the monitor. For the RC device, the macro would be:

```
DEVICE RC 2000 100020 RCSYS
```

The HSIZE macro, described in the same section, must also be invoked. For the RC device, the macro would be:

```
HSIZE RC,316,SYS
```

Monitor assembly instructions are in Appendix A of the RT-11 System Reference Manual. If this approach is used, the table patching procedure in step 5 is not necessary.

4. Link the monitor with the new bootstrap and device handler.

For S/J:

```
.R LINK
*RCMNSJ.SYS,MAP=RCBTSJ,RT11SJ,RC
```

For F/B:

```
.R LINK
*RCMNFB.SYS,MAP=RCBTFB,RT11FB,RC
```

5. If step 3 was not done and step 4 used the current monitor object modules, then the monitor tables must be patched to enter the device information. The monitor device tables are located using the procedure in Section 5.3. An additional table, the $ENTRY entry point table, must also be patched. For this example, assume the table addresses are:

<table>
<thead>
<tr>
<th>Table</th>
<th>S/J Address</th>
<th>F/B Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>$HSIZE</td>
<td>13674</td>
<td>14602</td>
</tr>
<tr>
<td>$DVSIZ</td>
<td>13730</td>
<td>14636</td>
</tr>
<tr>
<td>$PNAME</td>
<td>16516</td>
<td>17640</td>
</tr>
<tr>
<td>$STAT</td>
<td>16552</td>
<td>17674</td>
</tr>
<tr>
<td>$ENTRY</td>
<td>16612</td>
<td>17612</td>
</tr>
</tbody>
</table>
NOTE

These table addresses are for illustration only. Consult the Getting Started with RT-11 (V02B) manual for the table addresses of the current monitor release and for the address of BASE.

A link map was made during the linking sequence in Step 4. Locate the value of the system handler entry point, xxSYS. For this example, the tag is RCSYS and its value is found to be 56266 for F/B. This value is put in the $ENTRY table. The other values were determined in Section 5.3:

\[
\begin{align*}
$SHSIZE &= 316 \\
$DVSIZ &= 2000 \\
$PNAME &= 70370 \\
$STAT &= 100023 \\
$ENTRY &= 56266 \ (F/B) \ 45056 \ (S/J)
\end{align*}
\]

The patch procedure for the S/J monitor, using the twelfth slot, would then be:

```
.R PATCH<CR>
PATCH Version number
FILE NAME--
  *RCMNSJ.SYS/M<CR>  [The /M is necessary; Monitor base;
  *BASE; 0<CR>    $SHSIZE table; $DVSIZ table; $PNAME table; $STAT table; $ENTRY table; Exit to monitor]
  *0,13674/  4000  316<CR>
  *0,13730/  0  2000<CR>
  *0,16516/ 6250  70370<CR>
  *0,16552/  4 100023<CR>
  *0,16612/  0  45056<CR>
  *E
```

For the F/B monitor:

```
.R PATCH<CR>
PATCH Version number
FILE NAME--
  *RCMNFBSY/M<CR>  [$SHSIZE table; $DVSIZ table; $PNAME table; $STAT table; $ENTRY table; Exit to monitor]
  *BASE; 0<CR>
  *0,14602/  4000  316<CR>
  *0,14636/  0  2000<CR>
  *0,17649/ 6250  70370<CR>
  *0,17674/  4 100023<CR>
  *0,17564/  0  56266<CR>
  *E
```

The new monitor is now complete and may be used by transferring it to an RC disk and renaming it to MONITR.SYS.
5.5 DEVICES WITH SPECIAL DIRECTORIES

The RT-11 monitor can interface to devices having nonstandard (that is, non RT-11) directories. This section discusses the interface to this type of device.

5.5.1 Special Devices

Special devices are file-structured devices that do not use an RT-11 directory format. Examples are magtape and cassette as supported under RT-11. They are identified by setting bit 14 in the device status word. The USR processes directory operations for standard RT-11 devices; for special devices, the handler must process directory operations, as well as data transfers.

5.5.1.1 Interfacing to Special Device Handlers - There are three types of processes that a special device handler must perform:

1. Directory operations (.LOOKUP, .ENTER, etc.)
2. Data transfer operations (.READ, .WRITE)
3. Special operations (rewind, backspace, etc.)

The particular process required is passed to the handler in the form of a function code, located in the even byte of the fourth word of the I/O queue element (see Section 5.1.1). The function code may be positive or negative. Positive codes are used for processes of types 1 and 2 above; negative codes indicate device-dependent special functions.

The positive function codes are standard for all devices and include:

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read/Write</td>
</tr>
<tr>
<td>1</td>
<td>Close</td>
</tr>
<tr>
<td>2</td>
<td>Delete</td>
</tr>
<tr>
<td>3</td>
<td>Lookup</td>
</tr>
<tr>
<td>4</td>
<td>Enter</td>
</tr>
</tbody>
</table>

These functions correspond to the programmed requests .READ/.WRITE, .CLOSE, .DELETE, .LOOKUP, and .ENTER, described in Chapter 9 of the RT-11 System Reference Manual. The .RENAME request is not supported for special devices.
A queue element for a special handler will look identical to an element for a standard RT-11 handler when the function is a .READ/.WRITE (negative word count implies a .WRITE). For the remaining positive functions, word 5 of the queue element (the buffer address word discussed in Section 5.1.1) will contain a pointer to the file descriptor block, containing the device name, file name, and file extension in .RAD5Ø format.

Negative function codes are used for device-dependent special functions. Examples of these are backspace and rewind for magtape. Because these functions are characteristic of each device type, no standard definition of negative codes is made; they are defined uniquely for each device.

Software errors (for example, file not found or directory full) occurring in special device handlers during directory operations are returned to the monitor through the procedure described next. A unique error code is chosen for each type of error. This error code is directly returned by placing it in SPUSR (special device USR error), located at a fixed offset (272) into RMON. (Section 2.5.1 discusses monitor fixed offsets.) Hardware errors are returned in the usual manner by setting bit 0 in the channel status word pointed to by the second word of the queue element.

5.5.1.2 Programmed Requests to Special Devices - Programmed requests for directory operations and data transfers to special devices are handled by the standard programmed requests. When a .LOOKUP is done, for example, the monitor checks the device status word for the special device bit. If the device has a special directory structure, the proper function code is inserted into the queue element and the element is directly queued to the handler, by-passing any processing by the RT-11 USR. Device independence is maintained, since .READ, .WRITE, .LOOKUP, .ENTER, .CLOSE, and .DELETE operations are transparent to the user.

Requests for device-dependent special functions having negative function codes, must be issued by using the .SPFUN special function programmed request, described in Chapter 9 of the RT-11 System Reference Manual.
5.6 ADDING A SET OPTION

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

\[
\text{SET LP WIDTH}=80
\]

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

\[
\begin{align*}
\text{SET LP FORM} & \quad \text{(generate initial form feed)} \\
\text{SET LP NOFORM} & \quad \text{(suppress initial form feed)}
\end{align*}
\]

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

The SET command is entirely driven by tables contained in the device handler itself. Making additions to the list of SET options for a device is easy, requiring changes only to the handler, and not to the monitor. This section describes the method of creating or extending the list of SET options for a handler. The example handler used is the LP/LS11 line printer handler, listed in Appendix A in Section A.3. The SET command is described in Chapter 2 of the RT-11 System Reference Manual.

Device handlers have a file name in the form xx.SYS, where xx is the 2-letter device name; e.g., LP.SYS. Handler files are linked in save image format at a base address of 1000, in which a portion of block 0 of the file is used for system parameters. The rest of the block is unused, and block 0 is never FETCHed into memory. The SET command uses the area in block 0 of a handler from 400 to 776 (octal) as the SET command parameter table. The first argument of a SET command must always be the device name; e.g., LP in the previous example command lines. SET looks for a file named xx.SYS (in this case LP.SYS) and reads the first two blocks into the USR buffer area. The first block contains the SET parameter table, and the second block contains handler code to be modified. When the modification is made, the two blocks are written out to the handler file, effectively changing the handler.
The SET parameter table consists of a sequence of 4-word entries.
The table is terminated with a zero word; if there are no options
available, location 400 must be zero. Each table entry has the form:

```
.WORD   value
.RADS0  /option/
.BYTE   <routine-400>/2
.BYTE   mode
```

where:

- **value** is a parameter passed to the routine in
  register 3.
- **option** is the name of the SET option; e.g.,
  WIDTH or FORM.
- **routine** is the name of a routine following the
  SET table that does the actual handler
  modification.
- **mode** indicates the type of SET parameter:
  
  a. Numeric argument - byte value of 100
  b. NO prefix valid - byte value of 200

The SET command scans the table until it finds an option name matching
the input argument (stripped of any NO prefix). For the first example
command string, the WIDTH entry would be found (area 2 in the listing
in Section A.3). The information in this table entry tells the SET
processor that 0.WIDTH is the routine to call, that the prefix NO is
illegal and that a numeric argument is required. Routine 0.WIDTH is
located at area 4 on the listing. It uses the numeric argument passed
to it to modify the column count constant in the handler. The value
passed to it in R3 from the table is the minimum width and is used for
error checking.

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

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

2. Each table entry is four words long, as described pre-
   viously. The option name may be up to six .RAD50 char-
   acters long, and must be left-justified and filled with
   spaces if necessary. The table terminates with a zero
   (area 3 on the listing).
3. The routine that does the modification must follow the SET table in block 0 (area 4 on the listing). It is called as a subroutine and terminates with an RTS PC instruction. If the NO prefix was present and valid, the routine is entered at entry point +4. An error is returned by setting the C bit before exit. If a numeric argument is required, it is converted from decimal to octal and passed in R0. The first word of the option table entry is passed in R3.

4. The code in the handler that is modified must be in block 1 of the handler file, i.e., in the first 256 words of the handler. See areas 6 and 7 on the listing for code modified by the WIDTH option.

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

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

5.7 CONVERTING USER-WRITTEN HANDLERS

User-written device handlers must, in all cases, conform to the standard practices for Version 2 (2B). General programming information is discussed in Appendix H of the RT-ll System Reference Manual. Points to consider when converting user-written device handlers (written under Version 1 of the RT-ll System) follow; the details of these procedures have already been discussed.

1. The last word of a device handler is used by the monitor, thus the user must be sure to include one extra word at the end of his program when indicating the handler size.

2. The third header word of the handler should be 340, indicating that the interrupt should be taken at level 7.

3. It is not necessary to save/restore registers when the handler is first entered, although to do so is not harmful.

4. When an interrupt occurs, the handler must execute an .INTEN request or its equivalent. On return from .INTEN, R4 and R5 may be used as scratch registers. Device handlers may not do EMT requests without executing a .SYNCH request.

5. The handler must return from an interrupt via an RTS PC.

6. When the transfer is complete, the handler must exit to the monitor to terminate the transfer or enter a completion routine. When return is made to the monitor, R4 should point to the fifth word of the handler.
7. The handler should contain an abort entry point (located at INTERRUPT SERVICE -2) to which control is transferred on forced exit. The abort entry point should contain a BR instruction to code that will perform the necessary operations (stop device action and exit to monitor completion code).
CHAPTER 6

F/B MONITOR DESCRIPTION

The RT-11 Foreground/Background Monitor permits two jobs to simultaneously share memory and other system resources. The foreground job has priority and executes until it is blocked (i.e., execution is suspended pending satisfaction of some condition, such as I/O completion). When the foreground job is blocked, the background job is activated and executes until it finishes or until the foreground blocking condition is removed.

6.1 INTERRUPT MECHANISM AND .INTEN ACTION

All interrupt handlers must be entered at priority level 7 and must execute a .INTEN request on entry. The handler will then be called (as a co-routine of the monitor in system state) at its normal priority level. This is essential to the operation of RT11 for two reasons:

1. As a co-routine of the monitor, the interrupt handler exits to the monitor, which then does job scheduling.

2. Because of the above condition, there is a danger that interrupt processing may be postponed due to a context switch. For example, if a disk interrupts a lower priority device handler and goes to I/O completion, the monitor may switch to the foreground job and delay the lower priority interrupt until the foreground job is again blocked. By requiring the .INTEN request of all interrupt handlers, the monitor can assure that all interrupts are processed before the context switch is made.

The .INTEN request is implemented as a JSR R5 to the first fixed-offset location of RMON, which contains a jump to the interrupt entry code. This code saves R4 (R5 was saved by the JSR) and increments the system state counter. If the interrupt occurred on a job stack, the stack pointer is switched to use the system stack. The priority is lowered
to the handler's requested priority and control returns to the handler via another JSR instruction.

The handler interrupt code now executes in system state, with several results: any further interrupts are handled on the system stack, preventing their loss by a context switch to another job's stack; a context switch or completion routine cannot occur until all pending interrupts are processed; any error occurring in the handler occurs in system state, causing a fatal halt. When the handler exits via an RTS PC instruction, control returns to the monitor, which can now enter the scheduling loop if all interrupts have been processed.

6.2 CONTEXT SWITCH

When passing control from one job to another, the F/B Monitor does a complete context switch, changing the machine environment to that of the new job. The current context is saved on the stack of the current job and is replaced by the context of the new job.

The information saved on the stack includes:

1. The general registers (R0-R5)
2. The system communication area (memory locations 34-52)
3. The FPP registers, if used
4. The list of special locations supplied by the job (via .CNTXSW), if any

In addition, the stack pointer (R6) is saved in the job's impure area at offset I.SP (=50). The switch requires a minimum of $23_{10}$ words of stack, not including the special swap list.

The following are the minimum calculated times to context switch between jobs. The assumptions are that the F/G job is waiting for I/O completion, the handler completes an I/O request, and there are no user I/O completion routines.

<table>
<thead>
<tr>
<th>Processor (core memory)</th>
<th>11/20</th>
<th>11/40</th>
<th>11/45</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.66 ms</td>
<td>.36 ms</td>
<td>.28 ms</td>
</tr>
</tbody>
</table>

6-2
6.3 BLOCKING A JOB

The F/B Monitor gives priority to the foreground job, which runs until it is blocked by some condition. In this case, the background, if runnable (i.e., not blocked itself), is scheduled. The conditions which may block a job are flagged in the I.JSTA word, which is located in the job's impure area:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Bit in I.JSTA Word</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTIWTS$</td>
<td>14</td>
<td>Waiting for terminal input</td>
</tr>
<tr>
<td>TTOWT$</td>
<td>13</td>
<td>Waiting for room in output buffer</td>
</tr>
<tr>
<td>CHNWTS$</td>
<td>11</td>
<td>Waiting for channel to complete</td>
</tr>
<tr>
<td>SPNDS</td>
<td>10</td>
<td>Suspended</td>
</tr>
<tr>
<td>NORUNS</td>
<td>9</td>
<td>Not loaded</td>
</tr>
<tr>
<td>EXITS</td>
<td>8</td>
<td>Waiting for all I/O to stop</td>
</tr>
<tr>
<td>KSPNDS$</td>
<td>6</td>
<td>Suspended from KMON</td>
</tr>
<tr>
<td>USRWT$</td>
<td>4</td>
<td>Waiting for the USR</td>
</tr>
</tbody>
</table>

6.4 JOB SCHEDULING AND USE OF .SYNCH REQUEST

The F/B Monitor uses a scheduling algorithm to share system facilities between two jobs. The goal of the scheduler is to maximize system utilization, with priority given to the foreground job. The scheduler is generalized to use job numbers for scheduling, the higher job number having the higher priority. The background job is assigned job number 0 and the foreground job number 2. Job numbers must be even.

The foreground job runs until it is blocked by some condition (see Section 6.3), at which point the scheduler is initiated. The job list is scanned top down (from highest to lowest priority) for the highest priority job that is runnable. A job is runnable if it is not blocked, or if it is only blocked pending completion and is not suspended. If no jobs are currently runnable, the idle loop is entered.

If the new job is runnable, a context switch is made. The context switch routine tests for the completion pending condition (i.e., I/O is finished and a user completion routine was queued). In this case, a pseudo-interrupt is placed on the job's stack to call the completion queue manager when the scheduler exits to the job.
The scheduler is event driven and is entered from the common interrupt exit path whenever an event has occurred which requires action by the scheduler. The set of such events include:

1. An .EXIT or .CHAIN request
2. A job abort from the console, or an error abort
3. I/O transfer completed
4. Expiration of timed wait
5. A blocking condition encountered:
   a. .TWAIT request or SUSPEND command
   b. .TTYIN or .CSI waiting for end of line
   c. .TTYOUT or .PRINT waiting for room in output buffer
   d. Attempt to use busy channel
6. A blocking condition removed
7. No queue elements available
8. .SYNCH request (see below).

The .SYNCH request is used in interrupt routines to permit the issuing of other programmed requests. The .SYNCH macro is expanded as a JSR R5 to the .SYNCH code in the F/B resident monitor. The .SYNCH routine uses the associated 7-word block as a queue element for the completion queue.

If the .SYNCH block is not in use, register R5 is incremented to the successful return address and placed in the block as the completion address. The word count is set to -1 to prevent the block from being linked into the AVAIL queue. The block is placed in the completion queue, at its head, and the job associated with the .SYNCH request is flagged to have a completion routine pending. A request for a job switch is entered before the .SYNCH logic exits with an interrupt return.

On exit from the interrupt with a job switch pending, the scheduler is entered and the completion queue manager is called. When control finally returns to the code following the .SYNCH request, it is executing as a completion routine at priority level 0. It can now issue programmed requests without fear of being interrupted. If another interrupt comes in, it will be queued pending return of the current
completion routine, since the .SYNCH block is freed before calling the completion routine. Further interrupts will be rejected by the .SYNCH code, unless provision is made for supplying extra .SYNCH blocks.

6.5 USR CONTENTION

The directory operations handled by the USR are not re-entrant, particularly since the directory segment is buffered within the USR. Therefore, to use the USR in F/B, a job must have ownership of the USR. To facilitate this, the F/B monitor maintains a USR queuing mechanism.

Before issuing a USR request, a job must request ownership of the USR. If the USR is in use by another job, even of lower priority, the requesting job is blocked and must wait for the USR. The USRMWT$ flag is set in the I.JSTA word (see Section 6.3) and the job cannot continue until the USR is released and the blocking bit cleared. When the USR is released, the job list is scanned for jobs waiting for the USR, starting with the job having highest priority.

Because of the impact this may have on system performance, CSI requests are handled differently in the F/B system than in the S/J Monitor. If the command string is to come from the console keyboard, the prompting asterisk is printed and then the USR is released, pending completion of command line input. This prevents a job doing a CSI request from locking up the USR and blocking another, perhaps higher priority, job from executing. A job can determine if the USR is available by doing a .TLOCK request (see Chapter 9 of the RT-ll System Reference Manual).

6.6 I/O TERMINATION

Because of the multi-job capabilities of RT-ll F/B, termination of I/O on job exit or abort must be handled differently than in the S/J Monitor. The use of the RESET instruction is unacceptable, and a form of I/O rundown must be used. This is done by the IORSET routine, called when doing an abort or hard exit.

The IORSET routine searches the queue of every resident handler for elements belonging to the aborted job. If a handler is found to be resident and active (i.e., there are elements on its queue), the IORSET routine "holds" the handler from initiating a new transfer by setting bit 15 of the LQE word (entry point) in the handler. The
current transfer may complete, but the hold bit will prevent the queue manager from initiating a new transfer.

While it is held, the handler's queue is examined for the current request. If it belongs to the aborted job, the handler's abort entry point is called to stop the transfer. The queue of pending I/O requests is then examined and any elements belonging to the aborted job are discarded. The hold flag is cleared and a test is made to see if the current transfer completed while the handler was held. If it did, the completion queue manager, COMPLE, is again called to return the completed element and initiate the next transfer. At this point, any elements belonging to the aborted job will have been removed from the queue.

After the device handlers are purged, the internal message handler is examined for waiting messages that were originated by the aborted job. All such messages are discarded. Finally, all mark time requests belonging to the aborted job are cancelled.
CHAPTER 7

RT-11 BATCH

The RT-11 BATCH system is composed of a BATCH compiler and a run-time handler. The BATCH compiler converts BATCH Job Control language into a format comprehensible to the BATCH run-time handler. The compiler creates a control (CTL) file (from the BATCH language statements) which is then scanned by the handler; the CTL format is a versatile programming language in its own right. The result is a BATCH system that is simple to use, and yet easily customized to handle different situations.

7.1 CTL FORMAT

The BATCH run-time handler uses a unique language format that includes many programming features, such as labels, variables, and conditional branches. The directives are explained in detail in Chapter 12 of the RT-11 System Reference Manual.

Each directive consists of a backslash character followed by one or more other characters. For example, to run PIP and generate a listing, the CTL directives \E (execute) and \D (data line) are used:

\ER PIP
\DLP:=/L

Messages are sent to the console device by using the \@ directive:

\@ PLEASE MOUNT DT2
Labels and unconditional branches are implemented with the \L (label) and \J (jump) directives:

\JEND 1
  :
  :
\LEND

Each BATCH command is sent to the log as it is executed, using the \C (comment) directive:

\C $JOB

In this case, every character up to the next backslash is sent to the log.

7.2 BATCH RUN-TIME HANDLER

The BATCH run-time handler (BA.SYS) is constructed as a standard RT-ll device handler. To use the handler, it must be made permanently resident via the monitor LOAD command. The handler links itself into the monitor, intercepting certain EMTs described later.

The linking occurs the first time the BATCH compiler is run after the BA handler is loaded. The compiler does a .READW to the BA handler, which then links itself to the monitor and returns a table of addresses to the BATCH compiler. The linking is achieved by replacing the addresses of monitor EMT routines with corresponding addresses in the BATCH handler. Those EMTs that are diverted include:

<table>
<thead>
<tr>
<th>EMT</th>
<th>BATCH Handler Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>.TTYIN</td>
<td>B$TIN</td>
</tr>
<tr>
<td>.TTYOUT</td>
<td>B$TOT</td>
</tr>
<tr>
<td>.EXIT</td>
<td>B$EXT</td>
</tr>
<tr>
<td>.PRINT</td>
<td>B$PRN</td>
</tr>
</tbody>
</table>

Once the link is established, the BATCH handler cannot be unloaded. The links must first be undone by again running the BATCH compiler and specifying the /U switch. The compiler removes the links and prints a prompting message, after which the UNI BA command can be issued.
With the BA handler linked to the monitor, all console terminal communication is diverted to BA, along with program exits. The BA handler then dispatches the program request to the monitor routine or diverts it to a routine in BA, depending on the values of switches in BATSW1. The switches are:

<table>
<thead>
<tr>
<th>TAG</th>
<th>BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HELP</td>
<td>0</td>
<td>0 = Do not log terminal input (.TTYIN)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1 = Log terminal input</td>
</tr>
<tr>
<td>DESTON</td>
<td>1</td>
<td>0 = EMT is going directly to monitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = BA intercepts the EMT</td>
</tr>
<tr>
<td>SOURCE</td>
<td>2</td>
<td>0 = Character input by monitor from console terminal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Character input comes from BATCH stream</td>
</tr>
<tr>
<td>COMWAT</td>
<td>3</td>
<td>0 = No command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Command is waiting</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>4</td>
<td>0 = Console terminal inactive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Console terminal is active; i.e., BA is waiting for input from console terminal</td>
</tr>
<tr>
<td>DATA</td>
<td>5</td>
<td>0 = Characters are going to KMON; i.e., KMON is active in B/G</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Characters are going to B/G programs</td>
</tr>
<tr>
<td>BDESTN</td>
<td>6</td>
<td>0 = Output characters are going to console terminal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Output characters are going to LOG</td>
</tr>
<tr>
<td>BGET</td>
<td>7</td>
<td>0 = Normal mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Get mode (\G); input comes from console terminal until &lt;CR&gt;&lt;LF&gt; is encountered</td>
</tr>
<tr>
<td>NOTTY</td>
<td>8</td>
<td>0 = Log terminal output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Do not log terminal output (.TTYOUT, .PRINT)</td>
</tr>
<tr>
<td></td>
<td>9-13</td>
<td>Reserved</td>
</tr>
<tr>
<td>BSOURC</td>
<td>14</td>
<td>0 = BA directives come from console terminal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = BA directives come from CTL file</td>
</tr>
<tr>
<td>BEXIT</td>
<td>15</td>
<td>1 = A program has done an .EXIT while DATA switch was set</td>
</tr>
</tbody>
</table>

The BATSW1 word, located six bytes past the handler entry point, determines the state of the system at any given moment. If the word is zero, RT-11 operates normally. When the DESTON bit is set, EMTs are diverted to routines in BA for action, but the specific action taken by those routines is determined by the other switch bits.
For example, if the BDESTN bit is set, output from .TTYOUT and .PRINT is diverted from the console terminal to the log device. If SOURCE is set, the characters for the .TTYIN request are taken from the BATCH stream rather than from the console terminal via the monitor ring buffer. Directives for the BA handler itself may come from either the CTL file or the console terminal, depending on the state of the BSOURC bit.

The state of the background is reflected in the DATA bit. Either the KMON is active (DATA=0) or a program is active (DATA=1). If a program issues an .EXIT request while in DATA mode, the BEXIT state is entered until the BA handler encounters the next KMON directive (\E) in the BATCH stream, causing any unused \D lines to be ignored. A program can be aborted by diverting any of the .TTYIN, .TTYOUT or .PRINT requests to the .EXIT code in the monitor.

7.3 BATCH COMPILER

The obvious function of the BATCH compiler is to convert BATCH Standard Commands into the BA handler directives mentioned in Section 7.1, creating a control (CTL) file. BATCH jobs entered from a card reader or a file-structured device are compiled into a CTL file stored on a file-structured device for execution by the BA handler. However, the BATCH Compiler has other important functions; these are described in this section along with details on the initiation and termination of BATCH jobs.

7.3.1 BATCH Job Initiation

The following sequence of actions is performed by the BATCH Compiler when setting up a job for execution:

1. A check is made to ensure that LOG and BA device handlers are loaded and assigned properly. The LOG handler must be assigned the logical name LOG;; the BATCH Compiler may be run several times during the course of a job to do special tasks for the BA handler, and it will reference LOG:.

2. A nonfile-structured .LOOKUP is done on BA and a .READW is issued. If this is the first time BATCH has been run since BA was loaded, the handler links itself to the monitor (see Section 7.2). BA returns a list of
eleven pointers to important parameters within BA. These include:

- BA state word (BATSW1)
- CTL file savestatus area (INDATA)
- LOG file savestatus area (ODATA)
- Output (LOG) buffer (OUTBUF)
- Output buffer pointer (BATOPT)
- Output character counter (BATOCR)
- Input character counter (BATICT)
- Monitor EMT dispatch address save areas

3. A command string is collected from the console terminal and is processed by .CSISP C. An input file must be specified.

4. If the input file is a .BAT file to be compiled, a .CTL file is entered. If the LOG: device is file-structured, a fixed-size enter is done and then the file is initialized by writing zeroes in all blocks.

5. A .LOOKUP is done on all input files.

6. The .LOG file is .CLOSED so that a .LOOKUP and .SAVESTATUS may be done. The savestatus data is placed in the ODATA area in BA.

7. If the input file is a .BAT file, it is now compiled, with output going into the .CTL file.

8. The .CTL file is closed, again so that a .LOOKUP and .SAVESTATUS may be done. The .SAVESTATUS data is transferred to the INDATA area in BA. Buffer pointers and counters in BA are initialized.

9. The BA handler is activated by setting the SOURCE, DESTON, BSOURCE and BDESTN bits in the BATSW1 state word in BA. Control passes to BA when the compiler does an .EXIT, assuming an abort is not requested.

10. If an abort is requested (an error occurred during compilation or the /N switch was used), the .LOG file is .REOPENed and all $ command lines are logged out with any error diagnostics. The BATSW1 word is then cleared before exiting, preventing the execution of the job.

The following switches are used by the BATCH system during job initiation and continuation, and should not be typed by the user:

- /B: BATCH continuation of jobs in input stream
- /D: Print the physical device name assigned a logical device name in a $DISMOUNT command
- /M: Make a temporary source file
- /R: Return from $CALL
- /S: $CALL subroutine
7.3.2 BATCH Job Termination

Every BATCH job must be terminated with an $EOJ statement. The $EOJ statement causes the compiler to insert the CTL directives:

\R BATCH
\D/R

The /R switch for the BATCH compiler, which is legal only when entered from a BATCH stream, is used to terminate a BATCH job. This switch causes the compiler to pop the BATCH stack up a level. If the stack was empty, the stream is finished and the compiler cleans up, clears the BATSW1 word in BA, and exits. If the stack is not empty, the /R switch implies a return from a $CALL. The stack contents are used to restore parameters in the BA handler so that control will return to the calling BATCH stream at the next statement after the $CALL.

7.3.3 BATCH Compiler Construction

The BATCH Compiler is constructed in two pieces: a data area and a program area. The data area is located in low memory, in a .CSECT named UNPURE. The contents are described in the accompanying table (Table 7-1). The program section, located in the .CSECT named PROGRAM, starts at the symbol START. The general register R4 always points to UNPURE and all references to the data base are made as indexed references relative to R4.

Locations in the data base are created with the ENTRLO macro. For example,

    ENTRLO BOTLCT, #

allocates one word in the data base and initializes it to zero. The symbol BOTLCT is an offset into the data base, so that references to BOTLCT are made in the form BOTLCT(R4).

7-6
Table 7-1  
BATCH Compiler Data Base Description

<table>
<thead>
<tr>
<th>Tag</th>
<th>Byte Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATSWT</td>
<td>0</td>
<td>BATCH Control Switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABORT = 100000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATDOL = 40000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO = 20000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTYOUB = 10000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOGOUB = 4000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATOUB = 2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMOUB = 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JOB = 400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAKEB = 200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMMA = 100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BFORLI = 40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNIQUE = 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BANNER = 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RT11 = 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIME = 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAKE = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BATSW2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>More BATCH Control Switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABORT = 100000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIRST = 10000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SBIT = 4000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEQ = 2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTBIT = 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMSWB = 400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAKEB = 200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARFD = 100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STAROK = 40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BNOEOJ = 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTDAT = 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BEOF = 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSWT = 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EOJ = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPSWT</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Current command switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMSWT</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>BINLCT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSTAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICHRPT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BINCTR</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Tag</th>
<th>Byte Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARG</td>
<td>22</td>
<td>Input file EMT argument list</td>
</tr>
<tr>
<td>BATIBK</td>
<td>24</td>
<td>Input file block number</td>
</tr>
<tr>
<td>BATIBP</td>
<td>26</td>
<td>Input buffer address</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>Input buffer size</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>Wait I/O</td>
</tr>
<tr>
<td>BOTLCT</td>
<td>34</td>
<td>Last output buffer character count</td>
</tr>
<tr>
<td>OTSTAT</td>
<td>36</td>
<td>Output buffer status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BFREE = 1 0 → Buffer is free</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BWAIT = 2 In I/O wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BEOF = 4 End of file</td>
</tr>
<tr>
<td>OCHRPT</td>
<td>40</td>
<td>Output character pointer</td>
</tr>
<tr>
<td>BOTCTR</td>
<td>42</td>
<td>Output character count</td>
</tr>
<tr>
<td>BOTARG</td>
<td>44</td>
<td>Output file EMT argument list</td>
</tr>
<tr>
<td>BATOBK</td>
<td>46</td>
<td>Output file block number</td>
</tr>
<tr>
<td>BATOBP</td>
<td>50</td>
<td>Output buffer address</td>
</tr>
<tr>
<td></td>
<td>52</td>
<td>Output buffer size</td>
</tr>
<tr>
<td></td>
<td>54</td>
<td>Wait I/O</td>
</tr>
<tr>
<td>STACK</td>
<td>56</td>
<td>Compiler stack pointer save area</td>
</tr>
</tbody>
</table>
|         |             | These are the arguments passed between BATCH and BA:
| BATSW1  | 60          | Pointer to BATSW1 in BA.SYS                      |
| INDATA  | 62          | Pointer to INDATA                                |
| ODATA   | 64          | Pointer to ODATA                                 |
| OUTBUF  | 66          | Pointer to BATCH handler output buffer           |
| BATOPT  | 70          | Pointer to output character pointer              |
| BATOCT  | 72          | Pointer to output character counter              |
| BATICT  | 74          | Pointer to input character counter              |

(continued on next page)
### Table 7-1 (Cont.)

**BATCH Compiler Data Base Description**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Byte Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O$EXT</td>
<td>76</td>
<td>.EXIT</td>
</tr>
<tr>
<td>O$TIN</td>
<td>100</td>
<td>.TTYIN</td>
</tr>
<tr>
<td>O$TOT</td>
<td>102</td>
<td>.TTYOUT</td>
</tr>
<tr>
<td>O$PRN</td>
<td>104</td>
<td>.PRINT</td>
</tr>
<tr>
<td><strong>CSI Buffer:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPC0</td>
<td>106</td>
<td>Channel 0</td>
</tr>
<tr>
<td>SPC1</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>SPC2</td>
<td>132</td>
<td>2</td>
</tr>
<tr>
<td>SPC3</td>
<td>144</td>
<td>3</td>
</tr>
<tr>
<td>SPC4</td>
<td>154</td>
<td>4</td>
</tr>
<tr>
<td>SPC5</td>
<td>164</td>
<td>5</td>
</tr>
<tr>
<td>SPC6</td>
<td>174</td>
<td>6</td>
</tr>
<tr>
<td>SPC7</td>
<td>204</td>
<td>7</td>
</tr>
<tr>
<td>SPC8</td>
<td>214</td>
<td>10</td>
</tr>
<tr>
<td>LINIMP</td>
<td>224</td>
<td>Pointer to command line buffer (LINIMM)</td>
</tr>
<tr>
<td>LINIMM</td>
<td>226</td>
<td>Command line input buffer</td>
</tr>
<tr>
<td>LINIMS</td>
<td>350</td>
<td>Command line buffer save area</td>
</tr>
<tr>
<td>LIBLST</td>
<td>470</td>
<td>ASCIZ name of FORTRAN default library plus a line buffer</td>
</tr>
<tr>
<td>BATIBF</td>
<td>610</td>
<td>BATCH Compiler input buffers (INBSIZ * 2)</td>
</tr>
<tr>
<td>BATOBF</td>
<td>2610</td>
<td>BATCH Compiler output buffers (OTBSIZ * 2)</td>
</tr>
<tr>
<td>QSET</td>
<td>4610</td>
<td>Seven I/O queue elements for double/buffering</td>
</tr>
<tr>
<td>SOUTMP</td>
<td>4700</td>
<td>Source temporary file descriptor</td>
</tr>
<tr>
<td>OBJTMP</td>
<td>4714</td>
<td>Object temporary file descriptor</td>
</tr>
<tr>
<td>LOGTYP</td>
<td>4730</td>
<td>LOG device status word (word $ of .DSTATUS)</td>
</tr>
<tr>
<td>ARGARG</td>
<td>4732</td>
<td>EMT argument list for BA handler initialization</td>
</tr>
</tbody>
</table>

(concluded on next page)
<table>
<thead>
<tr>
<th>Tag</th>
<th>Byte Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STKBLK</td>
<td>4744</td>
<td>EMT argument list for READ/WRITE of BATCH stack</td>
</tr>
<tr>
<td>DEFCHN</td>
<td>4756</td>
<td>Default channel numbers</td>
</tr>
<tr>
<td>DEVSPC</td>
<td>4770</td>
<td>Pointer to device handler space</td>
</tr>
<tr>
<td>WDBLK2</td>
<td>4772</td>
<td>Two-word EMT argument block</td>
</tr>
<tr>
<td>WDBLK5</td>
<td>5000</td>
<td>Five-word EMT argument block</td>
</tr>
<tr>
<td>FTLPC</td>
<td>5012</td>
<td>Contents of PC on BATCH fatal error</td>
</tr>
<tr>
<td>AREA0</td>
<td>5014</td>
<td>Pointer to impure area</td>
</tr>
<tr>
<td>LSTTMP</td>
<td>5016</td>
<td>Listing temporary file descriptor</td>
</tr>
<tr>
<td>SWTMSK</td>
<td>5026</td>
<td>Switch mask for this BATCH directive</td>
</tr>
<tr>
<td>FD0</td>
<td>5030</td>
<td>File descriptor 0 for BATCH directive</td>
</tr>
<tr>
<td>FD1</td>
<td>5034</td>
<td>1</td>
</tr>
<tr>
<td>FD2</td>
<td>5040</td>
<td>2</td>
</tr>
<tr>
<td>FD3</td>
<td>5044</td>
<td>3</td>
</tr>
<tr>
<td>FD4</td>
<td>5050</td>
<td>4</td>
</tr>
<tr>
<td>FD5</td>
<td>5054</td>
<td>5</td>
</tr>
</tbody>
</table>
7.4 BATCH EXAMPLE

The following example demonstrates how the compiler converts BATCH Standard Commands into RT-11 BATCH handler directives. The example consists of a main BATCH stream, EXAMPL.BAT, and a BATCH subroutine file, EDITIT.BAT. EXAMPL creates a program, assembles and runs it. The program, called FILE.MAC, prints a message that is diverted to the log. The listing file from the assembly is printed and then deleted. The BATCH variable S is then tested and, if it is zero, the BATCH subroutine EDITIT is called. The EDITIT stream uses EDIT to edit the file FILE.MAC, changing the message to be printed. After return from EDITIT, the stream branches unconditionally to label L1, repeating the assembly and execution of FILE.MAC. EDITIT increments the variable S before returning, so that the BATCH stream, on encountering the IF statement again, now branches to label L2, skipping the call to EDITIT. SDIRECTORY and SDELETE operations are performed before finally exiting from BATCH.

Note the following about the .CTL files created:

1. The $JOB command produces a comment for the log (the \C directive, but no action directives). Its function is to initialize the BATCH compiler.

2. The $CREATE command produces directives that run the BATCH compiler, using the file name to be created with a /M switch. This is a special function of the BATCH compiler used to create data files. The compiler will enter the data that follows in the CTL file into the newly created file, until an EOF (CTRL/Z) is encountered. The data is fed to the compiler by the BATCH handler through the TTYIN programmed request. After the EOF character is encountered, the BATCH compiler closes the new file and exits, returning control to the BATCH handler through the .EXIT request. In this example, the file created is called FILE.MAC.

3. The $MACRO command has the /RUN switch appended, which forces the compiler to generate a series of assembly, link and execute instructions. A temporary execution file, 000000.SAV, is created from the assembled object module, FILE.OBJ. After execution with the monitor R command, the temporary execution file is deleted with PIP.

4. PIP is used to implement $PRINT, $DELETE, $COPY, and $DIRECTORY. The compiler translates these commands into the appropriate PIP command strings.
5. The variable S is defined to be zero with the LET statement. This translates into the BATCH handler directive,

\KS1\null

which instructs the BATCH handler to set variable S to the value in the byte following the character 1.

6. Labels are implemented by inserting a \L directive followed by the 6-character label name into the CTL stream where the label was declared. The label is also logged out with the \C directive so that the labels will appear in the log.

7. The unconditional branch, or GOTO command, is implemented with the \J directive immediately followed by the label. Note that the BATCH programmer must indicate whether the branch is forward or reverse. In this case, the branch is a backward reference and a minus sign is prefixed to the label:

GOTO -L1

There is no error checking done by the compiler. If an error is made (e.g., the minus sign is left off the L1), the BATCH handler searches forward in the CTL stream until it finds the label. Since an error was made, the label will not be found. The search (and consequently the BATCH job) terminates when the label stopper (\L$$$$$$) is encountered at the end of the CTL file.

8. The IF conditional branch is implemented with the \I directive. The \I directive is followed by the name of the variable to be tested, the value to be tested against, and three label fields. Each label field consists of the 6-character label name with a reference character appended. The character 1 indicates the label is a forward reference, a 0 indicates a backward reference. The test value is subtracted from the current value of the variable and the appropriate branch is taken. If no label is specified for a field, it is filled with spaces and causes the BATCH stream to fall through to the next command if that branch is elected.

9. The $CALL command is very useful and permits a BATCH stream to call another BATCH file as a subroutine, with control returning to the command following the $CALL. The $CALL is implemented by simply running the BATCH compiler, passing it the name of the SCALLED routine with a /S switch appended. Another BATCH compile/execute sequence will follow, but the /S switch will cause the compiler to save certain locations in the BATCH handler in an internal stack in the BA.SYS file. In this example, the $CALL EDITIT statement causes the file EDITIT.BAT to be compiled and executed.
10. BATCH variables may be used to enter ASCII values into a job stream. In the file EDITIT, the variable A is set equal to the value of the ESC (or ALT MODE) character. The variable A is inserted into a string of EDIT commands in place of the ALT MODE character.

11. The $EOJ must terminate every BATCH job. The $EOJ command generates the stopper label, \L$$$$$, and then produces directives to run the BATCH compiler again, this time with a /R switch. The compiler, when given a /R switch, checks the BATCH stack. If it is empty, the compiler exits. Otherwise, the stack is popped to restore conditions in the BATCH handler prior to the $CALL causing the push, and the BATCH stream continues. The $EOJ finally generates a \E to bring in the XMON and a \F<CR> to terminate the BATCH stream.
EXAMPL.BAT

$JOB
$MESSAGE EXAMPLE BATCH STRFAM
$CREATE FILE.MAC
   .MCALL .RFQDEF,.PRINT,.EXIT
   .REGDEF

START:
   .PRINT .MSG
   .EXIT
   .NLST REY
   .ASCIZ /THIS MESSAGE COMES FROM THE BATCH STRFAM/
   .EVEN
   .LIST REY
   .END START

SEND

SRT11
   LET S=0

L11
$MACRO/RUN FILE.LST/LIST FILE.MAC/INPUT FILE/OBJECT
$PRINT FILE.LST
$DELETE FILE.LST

SRT11
   IF(S=0),,L2
   $CALL EDITT,ICALL EDITT TO EDIT FILE.MAC

SRT11
   GOTO =L1

L21
$DIRECTORY FILE,*
$DELETE FILE,*
$ENDJ

EDITIT.BAT

$JOB/R11
$! JOB TO EDIT FILE.MAC
%S

$INCREMENT S TO PREVENT RECURSION
IA IS ALT MODE

$R EDIT
*ERFILE.MAC"APP"*
*GMSC"APP"*
*"APP"*
$ENDJ
EXAMPLE.CTL

\C
\*JOB
\$FN$AGE EXAMPLE BATCH STRFAM
\$E\* EXAMPLE BATCH STRFAM
\C
\$CREA\F FILE.MAC
\$ER BATCH
\$FILE.MAC/M=
\*MCALL .REGDEF, .PRINT, .FXTT
\*REGDEF
\START: .PRINT \$MSG
\EXIT
\$LIST REV
\$MSG: .ASCIZ /\THIS MESSAGE COMES FROM THE BATCH STRFAM/
\EVEN
\$LIST REV
\END START
\C
\S\EO\D
\SRT11
\LET 8=0
\$81 \$L1 \$CL11
\$MACRO/RUN FILE.LST/FILE.MAC/INPUT FILE/OBJFCT
\$ER MACRO
\$FILE,FILE.LST=FILE.MAC
\$D/ER LINK
\$D@DD@DD=FILE
\$ER \D@DD@DD0
\$ER PIP
\$D@DD@DD,SAVE
\C
\$PRINT FILE.LST
\$ER PIP
\$DST11,*/X=FILE.LST
\$D/IC
\$DELTFT FILE,LST
\$ER PIP
\$DFILE,LST/O
\C
\SRT11
\IF (8=0) ,1,2
\I8 1 1L2 1L \C
\$CALI EDITIT \$CALL EDITIT TO EDIT FILE.MAC
\$ER MATCH
\$EDITIT/S
\C
\SRT11
\GOTO -L1
\JL1 @\LL2 \$CLP1
EXAMPLE.CTL (Cont)

$DIRCCTORY FILE.*
$ER PIP
$DFILE.*/L
$DC
$DELFTP FILE.*
$ER PIP
$DFILE.*/D
$C
$EOJ
$LSSSSSS<F ER RATCH
$O/R
$E/F

EDITIT.CTL

$C
$JOB/RT11
$1 JOB TO EDIT FILE.MAC
%5 INCRTMENT 5 TO PREVENT RECURSION
%6 A IS ALT MODE
$F1 FOR EDIT
$DFFILE.MAC%K2%K2
$DMIN%K2%K2
$ARCIZ $MOTITED BY EDIT RUN BY RATCH
$C
$EOJ
$LSSSSSS<F ER RATCH
$O/R
$E/F
EXAMPL.LOG

$JOB
$MESSAGE EXAMPLE BATCH STREAM

$CREATE FILE.MAC

$END

$RT11
   LET S=0
L1       L1

$MACRO/RUN FILE.LST/LST FILE.MAC/INPUT FILE/OBJECT

*ERRORS DETECTED: 0
FREE COMPS 15100, WORDS

*
EXAMPL.LOG (Cont.)

.MAIN. RT-11 MACRO VM2-19  10-APR-75 10133145 PAGE 1

1 00000  .MCALL .RFGDEF .PRINT .EXIT
2 000000  .RFGDEF
3 000000  .PRINT #MSG
4 000000  .EXIT
5 000000  .NLIST REX
6 000010  124 MSG /THIS MESSAGE COMES FROM THE BATCH STREAM/
7 000000  .ARCTZ
8 000000  .EVEN
9 000000  .LIST REX
0 000000  .END  START
EXAMPLE.LOG (Cont.)

.Symbol Table

<table>
<thead>
<tr>
<th>MSG</th>
<th>%00010A</th>
<th>PC</th>
<th>%000007</th>
<th>R0</th>
<th>%000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>%000001</td>
<td>R2</td>
<td>%000002</td>
<td>R3</td>
<td>%000003</td>
</tr>
<tr>
<td>R4</td>
<td>%000004</td>
<td>R5</td>
<td>%000005</td>
<td>SP</td>
<td>%000006</td>
</tr>
<tr>
<td>START</td>
<td>%000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABS.</td>
<td>%000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000062</td>
<td>0R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERRORS DETECTED: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREE CORE: 15136, WORDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILE,FILE,LST=FILE,MAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THIS MESSAGE COMES FROM THE BATCH STREAM

SPRINT FILE,LST

SDELPUTF FILE,LST

SRT11 IF(S=0) ,,,12

$CALL EDITIT  ICALL EDITIT TO EDIT FILE,MAC

$JOB/RT11

$1 JOB TO EDIT FTLF,MAC

%5 1INCreMENT S TO PRFVENT RECURSION
LET A=33  IA IS ALT MODE

*ERPILF,MAC$**
* GHMSGISK  *AGCTZ /MODIFIED BY EDITOR RUN BY BATCH/SEX$3

$ENQ
$$SSSS$$

SRT11 GOTO =L1
L1: $MACRO/RUN FILE,LST/LIST FILE,MAC/INPUT FILE/ORJFCT

*ERRORS DETECTED: 0
FREE CORE: 15136, WORDS

7-19
.*MAIN.* RT-11 MACRO VWRZ-1R  18-APR-79 10:34:28 PAGE 1

1  000000  .MCALL  .REGDEF,.PRINT,.EXIT
2  000000  START  .PRINT #MSG
3  000006  .EXIT
4  000010  115 MSG
5  .ASCIZ /MODIFIED BY EDITOR RUN MY BATCH/
6  .EVEN
7  .LIST .REX
8  000000  .END  START
EXAMPLE.LOG (Cont.)

*MAIN, RT-11 MACRO VM03-1B  10-APR-75 10134198 PAGE 1*

SYMBOL TABLE

MSG  0000000000  PC  00000007  R0  00000000
R1  0000000001  PC  00000002  R0  00000003
R4  0000000004  R5  00000005  SP  00000006
START  00000000
* ABR, 00000000  VR0
     00000005  VR1
ERRORS DETECTED: 0
FREE CORE: 15136, WORDS

FILE,LST=FILE,MAC

MODIFIED BY EDITOR RUN BY BATCH

SPRINT FILE,LST

&DFLFTF FILE,LST

&RT11
TF(S=0) ,, ,2
L21
&DFINFACTORY FILF,*

10-APR-75
FILE .BAK 1 10-APR-75
FILE .MAC 1 10-APR-75
FILE .ORJ 1 10-APR-75
3 FILES, 3 BLOCKS
417 FREE BLOCKS
&DFLFTF FILE,*

SEOJ
7.5 CTT TEMPORARY FILES

In certain cases the BATCH compiler will produce temporary files with the extension CTT and the file name of the BAT file being compiled. These files occur when a multiple input file command string is issued, or when an unexpected $JOB or $SEQ statement occurs in a BATCH stream, or when multiple jobs are run from the card reader or a .BAT file.

The CTT file is actually a CTL file used to link together execution of several BATCH jobs. Each CTT file contains the BA directives:

```
\ER BATCH
\D/B
```

which execute the BATCH compiler, passing it the /B switch.

The CTT file also contains the following information:

1. Current input channel number (range is 3-10, 8)
2. Current input file block number
3. The CTL file descriptor block (device, file name and file size)
4. The LOG file descriptor block (device, file name, and file size)
5. The set of input (BAT) file descriptor blocks (device and file name)

When the CTT file is executed, the compiler restores the input channel number and block number and the entire set of file descriptor blocks from the CTT file. If, for example, the input channel number is 4, the second of a string of .BAT files is compiled and executed.
APPENDIX A

SAMPLE HANDLER LISTINGS
<table>
<thead>
<tr>
<th>Page 2</th>
<th>Handler Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INIT MP</td>
</tr>
<tr>
<td>2</td>
<td>INIT MP</td>
</tr>
<tr>
<td>3</td>
<td>INIT MP</td>
</tr>
<tr>
<td>4</td>
<td>INIT MP</td>
</tr>
<tr>
<td>5</td>
<td>INIT MP</td>
</tr>
<tr>
<td>6</td>
<td>INIT MP</td>
</tr>
<tr>
<td>7</td>
<td>INIT MP</td>
</tr>
<tr>
<td>8</td>
<td>INIT MP</td>
</tr>
<tr>
<td>9</td>
<td>INIT MP</td>
</tr>
<tr>
<td>10</td>
<td>INIT MP</td>
</tr>
<tr>
<td>11</td>
<td>INIT MP</td>
</tr>
<tr>
<td>12</td>
<td>INIT MP</td>
</tr>
<tr>
<td>13</td>
<td>INIT MP</td>
</tr>
<tr>
<td>14</td>
<td>INIT MP</td>
</tr>
<tr>
<td>15</td>
<td>INIT MP</td>
</tr>
<tr>
<td>16</td>
<td>INIT MP</td>
</tr>
<tr>
<td>17</td>
<td>INIT MP</td>
</tr>
<tr>
<td>18</td>
<td>INIT MP</td>
</tr>
<tr>
<td>19</td>
<td>INIT MP</td>
</tr>
<tr>
<td>20</td>
<td>INIT MP</td>
</tr>
<tr>
<td>21</td>
<td>INIT MP</td>
</tr>
<tr>
<td>22</td>
<td>INIT MP</td>
</tr>
<tr>
<td>23</td>
<td>INIT MP</td>
</tr>
<tr>
<td>24</td>
<td>INIT MP</td>
</tr>
<tr>
<td>25</td>
<td>INIT MP</td>
</tr>
<tr>
<td>26</td>
<td>INIT MP</td>
</tr>
<tr>
<td>27</td>
<td>INIT MP</td>
</tr>
<tr>
<td>28</td>
<td>INIT MP</td>
</tr>
<tr>
<td>29</td>
<td>INIT MP</td>
</tr>
<tr>
<td>30</td>
<td>INIT MP</td>
</tr>
<tr>
<td>31</td>
<td>INIT MP</td>
</tr>
<tr>
<td>32</td>
<td>INIT MP</td>
</tr>
<tr>
<td>33</td>
<td>INIT MP</td>
</tr>
<tr>
<td>34</td>
<td>INIT MP</td>
</tr>
<tr>
<td>35</td>
<td>INIT MP</td>
</tr>
<tr>
<td>36</td>
<td>INIT MP</td>
</tr>
<tr>
<td>37</td>
<td>INIT MP</td>
</tr>
<tr>
<td>38</td>
<td>INIT MP</td>
</tr>
<tr>
<td>39</td>
<td>INIT MP</td>
</tr>
<tr>
<td>40</td>
<td>INIT MP</td>
</tr>
</tbody>
</table>

## Monitor Defined Constants

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHLow</td>
<td>Monitor Base Pointer</td>
</tr>
<tr>
<td>MPTE</td>
<td>Monitor Page Table Entry</td>
</tr>
<tr>
<td>MPMTR</td>
<td>Monitor Trap Manager</td>
</tr>
<tr>
<td>MCHP</td>
<td>Monitor Checkpoint Header</td>
</tr>
<tr>
<td>MCR</td>
<td>Monitor Checkpoint</td>
</tr>
<tr>
<td>MDR</td>
<td>Monitor Description Register</td>
</tr>
<tr>
<td>MCR</td>
<td>Monitor Control Register</td>
</tr>
<tr>
<td>MDR</td>
<td>Monitor Debug Register</td>
</tr>
</tbody>
</table>

## Communication Constants

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRC</td>
<td>Monitor Read Count</td>
</tr>
<tr>
<td>MWR</td>
<td>Monitor Write Count</td>
</tr>
<tr>
<td>MIA</td>
<td>Monitor Instruction Address</td>
</tr>
<tr>
<td>MDA</td>
<td>Monitor Data Address</td>
</tr>
<tr>
<td>MB</td>
<td>Monitor Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Base Address</td>
</tr>
<tr>
<td>MIB</td>
<td>Monitor Index Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Index Base Address</td>
</tr>
</tbody>
</table>

## Control Register

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>Data Register</td>
</tr>
<tr>
<td>MRC</td>
<td>Monitor Read Count</td>
</tr>
<tr>
<td>MWR</td>
<td>Monitor Write Count</td>
</tr>
<tr>
<td>MIA</td>
<td>Monitor Instruction Address</td>
</tr>
<tr>
<td>MDA</td>
<td>Monitor Data Address</td>
</tr>
<tr>
<td>MB</td>
<td>Monitor Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Base Address</td>
</tr>
<tr>
<td>MIB</td>
<td>Monitor Index Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Index Base Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>Data Register</td>
</tr>
<tr>
<td>MRC</td>
<td>Monitor Read Count</td>
</tr>
<tr>
<td>MWR</td>
<td>Monitor Write Count</td>
</tr>
<tr>
<td>MIA</td>
<td>Monitor Instruction Address</td>
</tr>
<tr>
<td>MDA</td>
<td>Monitor Data Address</td>
</tr>
<tr>
<td>MB</td>
<td>Monitor Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Base Address</td>
</tr>
<tr>
<td>MIB</td>
<td>Monitor Index Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Index Base Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>Data Register</td>
</tr>
<tr>
<td>MRC</td>
<td>Monitor Read Count</td>
</tr>
<tr>
<td>MWR</td>
<td>Monitor Write Count</td>
</tr>
<tr>
<td>MIA</td>
<td>Monitor Instruction Address</td>
</tr>
<tr>
<td>MDA</td>
<td>Monitor Data Address</td>
</tr>
<tr>
<td>MB</td>
<td>Monitor Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Base Address</td>
</tr>
<tr>
<td>MIB</td>
<td>Monitor Index Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Index Base Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>Data Register</td>
</tr>
<tr>
<td>MRC</td>
<td>Monitor Read Count</td>
</tr>
<tr>
<td>MWR</td>
<td>Monitor Write Count</td>
</tr>
<tr>
<td>MIA</td>
<td>Monitor Instruction Address</td>
</tr>
<tr>
<td>MDA</td>
<td>Monitor Data Address</td>
</tr>
<tr>
<td>MB</td>
<td>Monitor Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Base Address</td>
</tr>
<tr>
<td>MIB</td>
<td>Monitor Index Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Index Base Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>Data Register</td>
</tr>
<tr>
<td>MRC</td>
<td>Monitor Read Count</td>
</tr>
<tr>
<td>MWR</td>
<td>Monitor Write Count</td>
</tr>
<tr>
<td>MIA</td>
<td>Monitor Instruction Address</td>
</tr>
<tr>
<td>MDA</td>
<td>Monitor Data Address</td>
</tr>
<tr>
<td>MB</td>
<td>Monitor Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Base Address</td>
</tr>
<tr>
<td>MIB</td>
<td>Monitor Index Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Index Base Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>Data Register</td>
</tr>
<tr>
<td>MRC</td>
<td>Monitor Read Count</td>
</tr>
<tr>
<td>MWR</td>
<td>Monitor Write Count</td>
</tr>
<tr>
<td>MIA</td>
<td>Monitor Instruction Address</td>
</tr>
<tr>
<td>MDA</td>
<td>Monitor Data Address</td>
</tr>
<tr>
<td>MB</td>
<td>Monitor Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Base Address</td>
</tr>
<tr>
<td>MIB</td>
<td>Monitor Index Base Address</td>
</tr>
<tr>
<td>MEB</td>
<td>Monitor End Index Base Address</td>
</tr>
</tbody>
</table>
A.2
RC11/R664 BOOTSTRAP
.SRTL MACROS, GLOBALS

;MCALL ;.VI.
;..VI.
;MCALL .EXIT, .LOOKUP, .PRINT, .SAVESTATUS

/ --------------------------------------------------------
/ CONDITIONAL ASSEMBLY OF ROOT FOR SINGLE USER OR RF SYSTEM
/ IF NORF RF  DEFAULT TO SINGLE USER
/ ALGOB REFERENCE TO MONITORI
/ GLOBAL ADREC, SENTRY, SIFTPR, RHLOC, SHONBL, SPNAME, SSLOT
/ GLOBAL SINFPR, SUBRF, SPNAME
/ GLOBAL RSTPR, COPTR, RXBBG, FILLER, NHFPUS, NHDBPS, RKLDC
/ GLOBAL HKDN, HKDN2, HKDILS, MAPOR, OGPU, HTPIS
/ GLOBAL SYLEN, RTSIZE, SWAPSZ, SVENTO, SVINDO, SVCHN, SVASBO
/ GLOBAL SYBLOW, TTBUF, TTAMBF, USRLOC, USRMS, MAXSYM
/ GLOBAL RELSET
/ FOLLOWING ARE GLOBALS FOR EITHER RF OR SU SYSTEM, BUT NOT BOTH
/ IF NE RF
/ GLOBAL RCHXTT, RHDOS1, RHDOS2, RHDOS3, CNXTT, FUDBES, FUDBES
/ GLOBAL HSRSP, SHINFPR, SHFPR, SHFPTT, TTSRR, TTUSRR, SRBNN
/ IFNP
/ GLOBAL AVAIL, T.CSW, FFPRD, FFPPRN, HSWL, TRPLC, TRPR
/ ENDC

perm = 2048 :STATUS WORD FOR PERMANENT FILE
fnblk = 4989 :STATUS OF END OF SEGMENT MARK
jaw = 177578 :ADDRESS OF JOB STATUS
sr = 177578 :CONSOLE SWITCH REGISTER

/ REGISTER OFFSETS

r0 = 0
r1 = 1
r2 = 2
r3 = 3
r4 = 4
r5 = 5
r6 = 6
r7 = 7
sp = 16
pc = 17

/ MONITOR OFFSET CONSTANTS
ROOT VR2R=61 RT-11 ROUTATRAP RT-11 MACRO VR2R=69 A-APR=75 11149184 PAGE 3

A8TTL ASPECT

IF NDF RdVSY8

IF NDF RDRV8

IF NDF RDOV8

IF NDF RDIY8

IF NDF RNVS8

RNVS = R

IT MUST BE AN RK SYSTEM

A8ECT

= A

PAR

AR

ROTI

BRANCH TO REAL ROIT

A8ECT

= A

PAR

AR

ROTI

BOOT VALIDATION PATTERN

A8TTL

JMP ROIT

PUT THE JUMP ROIT IN TRAP VECTOR

A8F

IFF

CS00 = 1

START FUNCTION

CS01 = 2

EMPTY BUFFER

CS02 = A

IFAR SECTOR

CS03 = A

UNIT 1 SELECTION

CS04 = A

IXY NONE

CS05 = A

IXY TRANSFER READY

CS06 = 177170

IXYR STATUS REGISTER
INITIALIZE RPT AND TOT VECTORS

READS: MOV #RC3S,#4  \( N_A \) RX STATUS REGISTER

MOV R4,#R5  \( N_A \) WILL POINT TO RX DATA BUFFER

ROCMD: MOV (R4),(R5)  \( N_A \) INITIAL READ FUNCTION

TOD: MOV #3,#R5  \( N_A \) GETS FILLED WITH READ COMMAND

TOD: CALL WAIT SUBROUTINE

TOD: MOV #R6,#R6  \( N_A \) ILOAD SECTOR NUMBER INTO R6DX

TOD: CALL WAIT SUBROUTINE

TOD: MOV #CSGO,#CSBIF,OR4  \( N_A \) ILOAD PARTY BUFFER FUNCTION INTO RXCB

TOD: CALL WAIT SUBROUTINE

TOD: IN TRANSFER READY UP?

RPL: IF NOT, SECTOR MUST BE LOADED

MOV R3,#R3  \( N_A \) MOVE DATA BYTE TO MEMORY

DEC #1  \( N_A \) CHECK BYTE COUNT

RET  \( N_A \) FLOOR AS LONG AS WORD COUNT NOT UP

CLR #2  \( N_A \) KLUDGE TO SLUFF BUFFER IF SHORT WR COUNT

RR #3  \( N_A \) FLOOR

MATT: RET #3  \( N_A \) ITS TR, ERR, DONT lPPT TTT ERR CAN'T BE

MATT: WAIT  \( N_A \) START AGAIN IF ERROR

RETRY: RET  \( N_A \) RETURN
; SECTOR 2 OF RT MONT
; INCALL READS SUBROUTINE
; Ijmp to sector 7
; Ijmp to sector 8
; IBranch if booting unit i, R6=1
; ISet to unit 0
; ISave unit booter from for later
; ISave the unit here
; ISet wait, #38
; Jmp R0
; IKnow WP are ready to do the real boot

ENDC

; root VR2=51
; RT=11 MONTSTRAP RT=11 MACRO VMPR=59
; A=APG=75 1149184 PAGE 4

; Following are the rootstrap I/O drivers for each valid
; system device.
; calling sequence:
; r0 = physical block to read/write
; r1 = word count
; r2 = buffer address
; r3-r5 are available and may be destroyed by the driver
; the driver must set R0 to RINERR if a fatal I/O error occurs.
; it must also invoke the macro SYSEV

; macro SYSEV NAME,VECTOR
; .GLOBAL NAME.INT, NAME.SIZF
; .define SYSTEM.DEVICE INTERRUPT & SIZE
; .sysname = NAME
; .tpec = <NAME>
; .sysname = <NAME+18R+5R
; .endr
; .sysvec = VECTOR
; = SYSEV
; .word NAME.INT, 368
; = put a vector to the system handler
; .word NAME.SIZF
; = put handler size where it can be used
; = GPR
; = VECTOR / 2
; = put vector to 4 in GPR
; = compute actual R0
; = compute actual R1
; = VECTOR is a multiple of 4
; = shift right 2 and R0
; .endr
; .endm SYSEV
; IP OF RPRES
; START: ROOTSTRAP T/S0 DRIVER - RP11

1 rp11 disk driver

2 sydfy m 250; rp11 device control reg
3 rpb= 176714; rp03 device status reg
4 rpa= 176718; rp05 device status reg
5 c8=0; rp05=0; rp01=1; ign bit in control & status
6 cb,ra= a000a; iread function code
7 cb,orv= a034a; unit select bits
8 db,att= a0377; unit att bits

read:

1 mov r8,r9; r9 = 0
2 jsr r2,div; get sector number
3 lmem 16; by dividing by 16
4 mov r4=(sp); save sector
5 mov r5,r5; soft mp divide
6 jsr r2,div; and compute cyl & track
7 lmem 20; by dividing by 20
8 smab ra; pbnit track in high byte
9 rra ra,ra; and install sector
10 mov #rp04,r3; and disk ads reg
11 mov r4,r3; soft track & sector
12 mov r5,-(r3); and cylinder
13 mov r2,-(r3); and bus address
14 mov r1,-(r3); and word count
15 neg #r1; mark negative
16 rra #c4,c8,orv,-,(r3); clear all but unit 0
17 rra #c5,ro+cr,ro,#r3; and start read

181 tset #r3; wait until transfer complete
19 rp; at
20 tst #r3; any error?
21 rra r0; error
22 movs #orv,##p0b; clear unit att for both
23 lclr #p0b; old & new controllers
24 rts pc; elsef just return

1 division routine for rp handler:

1 r5 = r5 / r0, rpmainop in ra

43 div
44 clr r5; iquot = 0
45 clr r4; #rpm1 = r
46 tst r3; ifs division 0?
47 reg r3; #ps = just return
48 com r5; iquot = -1 & set carry

151 rol r3; normalize
ROOT VR2E-O1 RT-11 MONTSTRAP RT-11 MACRO VNR2-E9 A=APR=75 11149186 PAGE 6

31
PSI ROL R4
COP R0, R2

SHIFT & SUBTRACT

33
PSI ROL R3

A3L R3

SHIFT QUOTIENT

:ENDC

ROOT VR2E-O1 RT-11 MONTSTRAP RT-11 MACRO VNR2-E9 A=APR=75 11149186 PAGE 7

1
:IP OF SRCYS81SRFSYS

BB

7

BB

2

OFFENDIF FOR RF DISK

6

RF DISK HANDLER

4

RF

2

RFCS

1

177448

177448

177448

177448

177448

177476

177472

1

RFCS

1

RFMC

1

177442

177442

177442

177442

177474

1

READY

1

MOV SRFDEL, R3

POINT TO DISK ADDRESS

11

MOV R0, R5

COPY BLOCK NUMBER

12

MOV R5, R4

MULTIPLY BY 256 TO GET WORD # ON DISK

13

CLER R5

MAKE RA AN EVEN BLOCK NUMBER

14

MOV R5, (R3)+

PUT LOW ORDER ADDRESS IN CONTROLLER

15

REC 177448, R4

ISOLATE HIGH ORDER ADDRESS

16

MOV R4, (R3)

PUT IT IN CONTROLLER

17

TST = (R3)

IFSET POINTER
ENDC

; THIS CODE IS COMMON TO RN05, RN11 AND RP11 HANDLERS

MOV R2,-(R3) ; BUFFER ADDR
MOV R1,-(R4) ; WORD COUNT
NEG (R3) ; (NEGATIVE)
MOV #5,-(R3) ; START DISK READ
TSB (R3) ; WAIT UNTIL COMPLETE
RPL 99
TAT (R3) ; ANY ERROR?
RTS RIOERR ; HARD WALT ON ERROR
RTS PC

*ENDC

ROOT VR2R=61 RT=11 BOOTSTRAP RT=11 MACROM VR2-84 A=APR-75 1149184 PAGE 8

BOOTSTRAP I/O DRVPR = RC11

.STRTL ROOTSTRAP I/O DRVPR = DFCTAPE

; DFCTAPE ROOTSTRAP HANDLER

SYSDV = 177342 ; COMMAND REGISTER
TCMN = 177342 ; DATA REGISTER
TCMT = 177342 ; STATUS REGISTER

READI MOV #TCMN,R4 ; R4 < COMMAND REG
MOV #TCMT,R3 ; R3 < DATA REG
NTARCH MOV #6,R5 ; ICOPY BLOCK NUMBER
RUR #2,R5 ; SEARCH FOR 2 EARLIER
MOV #4803,R4 ; IMPERSE,RNUM
PSI RIT #1R2800,R4 ; WAIT TILL BLOCK FOUND
REG 28
RMT NTTERR
CMP #5,R3 ; IS IT THE DESIRED BLOCK
BLY NTARCH ; NO, CONTINUE SEARCHING
DTPWRI MOV #3,R4 ; SEARCH FORWARD (RNUM)
RHI MOV #1R2800,R4 ; WAIT
RREG 48
RMT NTTERR
CMP #6,R3 ; DESIRED BLOCK
NTY DTPWRI ; NO SEARCH FORWARD
BLY NTARCH ; NO SEARCH REVERSE
MOV #2,-(R3) ; BUFFER ADDRESS
NEG R1
MOV R1,-(R3) ; WORD COUNT
ROOT VS259-81
RT-11 BOOTSTRAP RT-11 MACRO VNR2-84
READLS 11119186 PAGE 9
BOOTSTRAP I/O DRIVER = RC11

.2ENDC

;SBSYP
SYSPRV

;FLOPPY SYSTEM
;FLOPPY VECTORS THROUGH 264

READI
ABLI
ABLI
ABLI
18I
MOV
MOV
MOV
CLRI
AR
28I
SUR
34I
TNC
SUR
SUR
RPL
CMPI
RPI
45I
RPI
ADD
RPI
MOV
TNC
TST
RPT
;FLOPPY BLOCK TO LOGICAL SECTOR
;FLOPPY BLOCK
;MAKE WORD COUNT BYTE COUNT
;SAVE LSN FOR LATER
;MPF NEED 2 COPIES OF LSN FOR HAPPER
;INIT FOR TRACK QUITIENT
;JUMP INTO DIVIDE LOOP
;BUMP QUITIENT, STARTS AT TRACK 1
;STRACK=INTEGER(LSN/26)
;FLMP = R4*REM(LSN/26)+26
;SFTT C IF SECTOR MAPS TO 1-13
;IPPERFORM 211 INTERLEAVE
;SANJUST SECTOR INTO RANGE -1,-26
;S(divide for remainder only)
;SANPUT SECTOR INTO RANGE 1-26
;CALL READS SUBROUTINE
;GET THE LSN AGAIN
;SET UP FOR NEXT LSN
;WHAT'S LEFT IN THE WORD COUNT
;BRANCH TO TRANSFER ANOTHER SECTOR
;RETURN PC
;TST WAIT SUBROUTINE, PRINTS ERRORS
;RETURN FROM INTERRUPT
***** THIS MUST FALL INTO RINERR *****

ENDC

RINERR J&F RB.REPORT ISAY THAT WE GOT ERROR

R00459 015 012 077 \A0CTZ <15>12/>T/O ERROR<12>
R00460 102 055 111
R00461 057 117 049
R00462 195 172 122
R00463 117 172 012

\EVEN

RO0T VM2=81 RT-11 ROOTSTRAP RT-11 MACRO VMR2-89 A=APR=75 11149184 PAGE 16

ROOTSTRAP CORE DETERMINATION

1 \Serial ROOTSTRAP CORE DETERMINATION

2

3 R00474 11P037 177544
4 R00508 104757 177544
5 R00509 105575
6 R00510 104971
7 R00512 103171
8 R00512 104988
9 R00512 104988
10 R00512 104988

11 R00516 104976 RR \-> KEEP HIM FROM CONTINUING

12 R00520 101876
13 R00520 101876 R00582
14 R00530 101871 100882
15 R00534 101872 100882
16 R00534 101872 100882
17 \If GT -16BP, ERROR ROOTSTRAP BLOCK IS TOO BIG
18 R00544 101872 100882 MOV #1888, R3
19 R00550 101872 100882 MOV #1888, R3
20 R00552 101872 100882 MOV #1888, R3
21 R00556 104913

22

23

24 IF THIS ROOTSTRAP CAN SIMULATE ANY SIZE PDP=11.
25 IF LOCATION 'FINDL' IS A WALT, THE CPU WILL STOP DURING THE BOOT.
26 ON CONTINUE, THE TOP 5 BYTES OF THE SWITCH REGISTER ARE USED TO
27 SET THE TOP OF AVAILABLE CORE AS A MULTIPLE OF 1K.
28 IF THE 5TH IS >= 16800 OR IF FINDL IS A SR 10;
29 THE ROOTSTRAP WILL DO A NORMAL CORE DETERMINATION.
30
31 R00560 R00560 FINDL RR 10 CHANGE TO WALT FOR FINDLING
26 27 .IF NE BP
28             M8RENT
29             TTTUVR
30             TTTUAR
31             FUDGE1
32             FUDGE2
33             RKBND1
34             RKBND2
35             RKBND3
36             CNTXT
37             RNNKBTX
38             RNN8SP
39             SWPTR
40             SWRPTN
41             .SCRTN
42 .IF
43             001362 AHB8886
44             001364 AHB8886
45             001366 AHB8886
46             001370 AHB8886
47             001372 AHB8886
48             001374 AHB8886
49             001376 AHB8886
50             .ENDC
51             A
52             .END OF LIST
53 001602 AHB8886
54 001604 AHB8886 AHB8886 AHB8886 TLIST1 .WARM KM11S KM8SP KM8PUR 8BITS IN CONFIG WORD
55 001612 879259
56 001614 AHB8886 AHB8886
57 001616 .WARM P.R
58 001618 .RADES /BY/
59 001620 .BLOND .BLKWM 9
60 001622 .SAVESTATUS GOES HERE

BOOT VR2R-81 RT-11 MONTSTRAP RT-11 MACRO VMR2-89 APR-75 11149184 PAGE 12+ RELOCATION LIST
58 001614 AHB8886 AHB8886 .FILENAME .WARM P.R .FILENAME GOES HERE
59 001620 879259 .RADES /BY/
60 001622 .BLOND .BLKWM 9 .SAVESTATUS GOES HERE
TITLE LP V2=03 P5=JJ=78

; BT=11 LINE PRINTER (LP/LP11) HANDLER
; NCR=11-ORTI A=0
; DGR/FP/ARC/EF
; MARCH 1973/FEBRUARY 1974
; COPYRIGHT (C) 1974,1975
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS 01754

; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
; ON A SINGLE COMPUTER SYSTEM AND MAY NOT BE COPIED EXCEPT WITH
; THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE, THIS SOFTWARE,
; OR ANY OTHER PORTION THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
; ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
; SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.

; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
; OR RELIABILITY OF THE SOFTWARE ON EQUIPMENT
; WHICH IS NOT SUPPLIED BY DIGITAL.
LOADPT1  .WORD  LPVEC
LOADPT  .WORD  LPINT-.
PRI    .WORD  PRP
LPINT   .WORD  P
LPOF   .WORD  P
ENTRY POINT

LPI  MOV  LPOF, R4
A@  A(R4)
RCF  LPFR
LPFR REQUEST TO ILLEGAL
RA  #LPR, #LPS
RTA  PC

INTERRUPT SERVICE

AR  IPRONE
ENTRY POINT

LPINT1  JOB  $5, $10EH
WORD  $C<$PRAPRPT
MOV  LPOF, R4
RT  #LPR
FAOPT1  RMT  RET
TST  (R4)+
FFPPT1  REG  RKB
ITB  RKB BLOCK BY
LPIEXT1  TST  (R4)+
LPLRT  #LPS
RPL  RET
INPFR  RETURN FROM INTERRUPT
RPL  RET
TAB IN PROGRESS
TARFLG1  .WORD  R

TAR  IPRONE
TB  OPING TB

TIGNOR1  MOV  #R(R3)+, R5
R6  #17749, R5
TST  (R4)
REDF  LPRONE
TNC  (R4)
TNC  #R4-4
PMPB  R5, #40
RLN  PMST
CHPRT  INF-ZO TRST FOR SPECIAL CHARS.
CHPBR  CHPBR
INH

LCOPT1  ROP
SCHPR  ROP (R5)+
FOINT1  #LPR
RET  IRTGORE
# OF PRINTER COLUMN LEFT
INF MORE ROOM ON LINE, DON'T PRINT CHAR
This software is furnished under a license agreement for use only
on a single computer system and may be copied only with
the inclusion of the above copyright notice, this software,
or any other copy thereof, may not be provided or otherwise
made available to any other person except for use on such
system and to
one who agrees to these license terms. Title to any ownership
of the
software shall at all times remain in DEC.

The information in this document is subject to change without
notice and should not be construed as a commitment by DEC.
DEC assumes no responsibility for the use of
or reliance on the software on equipment
which is not supplied by DEC.
.ARTTL CONFIGURATION SECTION

1 THE FOLLOWING CODE IS EXECUTED WHEN A "SET CR" CONSOLE COMMAND IS
2 GIVEN.

3 .AREA CONFIGURATION AREA

4 .WP

5 .WP

6 /WP

/.AREA

1 /WP

11 /WP

12 /WP

13 /WP

14 /WP

15 /WP

16 /WP

17 /WP

18 /WP

19 /WP

20 /WP

21 /WP

22 /WP

23 /WP

24 /WP

25 /WP

26 /WP

27 /WP

28 /WP

29 /WP

30 /WP

31 /WP

32 /WP

33 /WP

34 /WP

35 /WP

36 /WP

37 /WP

38 /WP

39 /WP

40 /WP

41 /WP

42 /WP

43 /WP

44 /WP

45 /WP

46 /WP

47 /WP

48 /WP
### 826 Conversion Table
1. **Normal Character Table**
2. **APOSTROPHE** (8-2)
3. **COLON** (8-2)
4. **DOLLAR** (8-2)
5. **EQUAL** (8-3)
6. **FORWARD** (8-3)
7. **SINGLE Q** (8-4)
8. **SINGLE T** (8-4)
9. **TILDE** (8-5)
10. **UNDERBAR** (8-5)
11. **VIRUS** (8-5)
12. **WAVE** (8-5)

### 829 Conversion Table
1. **ERROR** (8-5)
2. **APRIL** (8-5)
3. **JUNE** (8-5)
4. **ERROR** (8-5)
5. **ERROR** (8-5)
The following table takes two arguments, the ASCII translation and the list of punch configurations for that character.

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Translation</th>
<th>Punch Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
<td>4, 5</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
<td>6, 7</td>
</tr>
</tbody>
</table>

The following table translates ASCII to punch code.

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Translation</th>
<th>Punch Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>65</td>
<td>11</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
<td>12</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
<td>13</td>
</tr>
</tbody>
</table>

The following table translates punch code to ASCII.

<table>
<thead>
<tr>
<th>Punch Code</th>
<th>ASCII Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
</tr>
<tr>
<td>13</td>
<td>C</td>
</tr>
<tr>
<td>Octet</td>
<td>Character</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>0062</td>
<td>$\text{#CHAP} #$</td>
</tr>
<tr>
<td>007A</td>
<td></td>
</tr>
<tr>
<td>007E</td>
<td></td>
</tr>
<tr>
<td>007F</td>
<td></td>
</tr>
<tr>
<td>0080</td>
<td></td>
</tr>
<tr>
<td>0081</td>
<td></td>
</tr>
<tr>
<td>0082</td>
<td></td>
</tr>
<tr>
<td>0083</td>
<td></td>
</tr>
<tr>
<td>0084</td>
<td></td>
</tr>
<tr>
<td>0085</td>
<td></td>
</tr>
<tr>
<td>0086</td>
<td></td>
</tr>
<tr>
<td>0087</td>
<td></td>
</tr>
<tr>
<td>0088</td>
<td></td>
</tr>
<tr>
<td>0089</td>
<td></td>
</tr>
<tr>
<td>008A</td>
<td></td>
</tr>
<tr>
<td>008B</td>
<td></td>
</tr>
<tr>
<td>008C</td>
<td></td>
</tr>
<tr>
<td>008D</td>
<td></td>
</tr>
<tr>
<td>008E</td>
<td></td>
</tr>
<tr>
<td>008F</td>
<td></td>
</tr>
</tbody>
</table>

**Escape Table**

<table>
<thead>
<tr>
<th>Octet</th>
<th>Character</th>
<th>Start at Octal</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>006A</td>
<td></td>
<td>9, 8, 4</td>
<td>ICTRI-T</td>
</tr>
<tr>
<td>006B</td>
<td></td>
<td>9, 8, 5</td>
<td>ICTRI-U</td>
</tr>
<tr>
<td>006C</td>
<td></td>
<td>9, 8, 6</td>
<td>ICTRI-V</td>
</tr>
<tr>
<td>006D</td>
<td></td>
<td>9, 8, 7</td>
<td>ICTRI-W</td>
</tr>
<tr>
<td>006E</td>
<td></td>
<td>11, 0, 8</td>
<td>ICTRI-X</td>
</tr>
<tr>
<td>006F</td>
<td></td>
<td>11, 0, 9, 1</td>
<td>ICTRI-Y</td>
</tr>
<tr>
<td>0070</td>
<td></td>
<td>11, 0, 2</td>
<td>ICTRI-Z</td>
</tr>
<tr>
<td>0071</td>
<td></td>
<td>11, 0, 3</td>
<td>ICTRI-2</td>
</tr>
<tr>
<td>0072</td>
<td></td>
<td>11, 0, 4</td>
<td>ICTRI-3</td>
</tr>
<tr>
<td>0073</td>
<td></td>
<td>11, 0, 5</td>
<td>ICTRI-4</td>
</tr>
<tr>
<td>0074</td>
<td></td>
<td>11, 0, 6</td>
<td>ICTRI-5</td>
</tr>
<tr>
<td>0075</td>
<td></td>
<td>11, 0, 7</td>
<td>ICTRI-6</td>
</tr>
<tr>
<td>0076</td>
<td></td>
<td>12, 7, 7</td>
<td>I (SPC)</td>
</tr>
<tr>
<td>0077</td>
<td></td>
<td>8, 7</td>
<td>I</td>
</tr>
<tr>
<td>0078</td>
<td></td>
<td>8, 6</td>
<td>I</td>
</tr>
<tr>
<td>0079</td>
<td></td>
<td>8, 5</td>
<td>I</td>
</tr>
<tr>
<td>007A</td>
<td></td>
<td>11, 8, 6</td>
<td>I</td>
</tr>
<tr>
<td>007B</td>
<td></td>
<td>11, 8, 5</td>
<td>I</td>
</tr>
<tr>
<td>007C</td>
<td></td>
<td>11, 8, 4</td>
<td>I</td>
</tr>
<tr>
<td>007D</td>
<td></td>
<td>11, 8, 3</td>
<td>I</td>
</tr>
<tr>
<td>007E</td>
<td></td>
<td>11, 8, 2</td>
<td>I</td>
</tr>
<tr>
<td>007F</td>
<td></td>
<td>11, 8, 1</td>
<td>I</td>
</tr>
</tbody>
</table>
TITLE

AT-11 DEFFP (TP11) HANDLER

AT-11-DEFFP-D

OBJECT

AUG91, 1974

COPYRIGHT (C) 1974, 1975

DIGITAL EQUIPMENT CORPORATION

Maynard, Massachusetts 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
ON A STAGE V COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
OR ANY OTHER COPY THEROF, MAY NOT BE PRODUCED OR DISTRIBUTED
AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
ANP WHO AGREES TO THEF LICENS EP TERMS. TITLE TO AND OWNERSHIP OF THE
SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.

THIS INFORMATION IN THIS DOCUMENT IS SUBJECT TO
CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSIDERED
AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USB
OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
WHICH IS NOT SUPPLIED BY DIGITAL.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\text{GLOBAL NTV8, B, F, P8, F, P8}$</td>
</tr>
<tr>
<td>2</td>
<td>$\text{IRK IS NOT RESIDENT}$</td>
</tr>
<tr>
<td>3</td>
<td>$\text{AUTO IN RF}$</td>
</tr>
<tr>
<td>4</td>
<td>$\text{IFURB AT LEVEL 7}$</td>
</tr>
<tr>
<td>Symbol Table</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>REC</td>
<td></td>
</tr>
<tr>
<td>PSY</td>
<td></td>
</tr>
<tr>
<td>NTIDF</td>
<td></td>
</tr>
<tr>
<td>NT17E</td>
<td></td>
</tr>
<tr>
<td>NT949</td>
<td></td>
</tr>
<tr>
<td>MONLINS</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>R949</td>
<td></td>
</tr>
<tr>
<td>TCNT</td>
<td></td>
</tr>
<tr>
<td>SINTEN</td>
<td></td>
</tr>
<tr>
<td>PRA</td>
<td></td>
</tr>
<tr>
<td>SYRAMD</td>
<td></td>
</tr>
<tr>
<td>FRM003</td>
<td></td>
</tr>
<tr>
<td>,LP: /NINTH/CODT</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX B

FOREGROUND TERMINAL HANDLER

The following listing is a terminal handler for the foreground. The user can write his own handler using this code as an example, or use the copy provided in the software kit. Instructions for its use are found on the second and third pages of the listing.
TITLE KB;MAC V8I=R1
RT=11 V2 DPTCP INDEPENDENT TERMINAL HANDLER, KB

REO=11-ORKRA=0

COPYRIGHT (C) 1975

DIGITAL EQUIPMENT CORPORATION

MAYNARD, MASSACHUSETTS 01754

THS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
WHICH IS NOT SUPPLIED BY DIGITAL.

MARCH 1975

RGR

KB;MAC V8I=R1 RT=11 MACRO VMRZ-B9 R=APR=75 1P173151 PAGE 2

RT=11 VP DEVICE INDEPENDENT TERMINAL HANDLER, KB, KR

CAN BE USED BY EITHER THE FRONTEND OR BACKGROUND (BUT NOT
BOTH SIMULTANEOUSLY) TO READ AND WRITE TO ANY DL=11A OR KL=11A
CONTROLLED TERMINAL.

THIS HANDLER HAS THE FOLLOWING CHARACTERISTICS:

1) PARSE RETURN CAUSES THE RESTIORDER
2) OF THE INPUT BUFFER FOR THE CALLING READ REQUEST TO BE
3) ZERO-FILLED, AND THE READ IS COMPLETED. THUS, THE HANDLER
TRANSFERS ONE LINE AT A TIME NO MATTER HOW LONG THE

TRYOUT BUFFER IS FOR THE READ REQUESTS THE UNUSED PORTION
OF THE BUFFER IS ZERO-FILLED, CARRIAGE RETURN ECHOS
CARRIAGE RETURN, LINE FEED, AND INSERTS SR AND UP CHARACTERS
IN THE BUFFER IF THERE IS ROOM, ELSE ONLY CR IS PLACED IN THE BUFFER.

IF THE BUFFER REMAINS EMPTY SELECT A FF CHARACTER IN
THE BUFFER.

3) PUTOUT ECHOS "\" AND DELETES THE LAST CHARACTER IN THE BUFFER.
IF THERE ARE NO CHARACTERS IN THE BUFFER, PUTOUT DOES NOT ECHO
AND IS IGNORED.

4) TAB ECHOS ENOUGH SPACES TO POSITION THE POINT HEAD AT THE
NEXT TAB STOP AND INSERTS A TAB CHARACTER IN THE BUFFER.

5) CTRL U ECHOS "\u" AND ERASES THE CURRENT LINE.
6) CTRL Z ECHOS "\z" AND CAUSES THE Handler TO REPORT END-OF-FILE.

THE CTRL Z CHARACTER IS NOT INSERTED IN THE BUFFER.

7) THE LOW-SPEED READER WILL USE IF IT IS TURNED ON WHILE A READ
REQUEST IS PENDING TO THE HANDLER. IF THE TAPE BEING READ HAS
MANY TABS, HOWEVER, THE TIME NEEDED TO ECHO THE TAB WILL
CAUSE CHARACTERS FOLLOWING THE TAB TO BE LOST, TO DISABLE THE
ECHOSING OF TABS, THE "SET NR لو" COMMAND CAN BE USED AS FOLLOWS:
"SET NR لو" WILL REEnable TAB ECHOS, ALLOWING A TAPE
TO BE READ WITHOUT CHARACTER LOSS.

"SET NR لو" WILL ENABLE TAB ECHOS FOR NORMAL KEYBOARD
INPUT. THIS IS THE DEFAULT.

8) WHEN THE HANDLER RECEIVES A READ REQUEST A "\0" CHARACTER IS
PRINTED IN THE LEFT HANDLER OF THE TERMINAL TO SIGNIFY THAT THE
HANDLER IS READY FOR INPUT, THIS CHARACTER CAN BE ERASED OR THE
PROMPT FEATURE CAN BE REMOVED BY RE-ASSIGNING THE SYMBOL
"PROMPT" TO THE ASCII VALUE OF THE
RECEIVED CHARACTER, SETTING PROMPT TO "" WILL CAUSE NO CHARACTER TO
BE PRINTED.

9) IF NO READ REQUEST IS ACTIVE, THE HANDLER WILL NOT ACCEPT INPUT.
AND THE KEYPAD WILL NOT ECHO. IF IT DOES ECHO, THE HANDLER IS
ACCEPTING INPUT.

THIS HANDLER CONTAINS CONDITIONAL CODE TO SUPPORT TERMINALS THAT
REQUEST FILLER CHARACTERS AFTER A PARTICULAR CHARACTER. TO ENABLE THE
FILLER FUNCTION, DEFINE THE SYMBOL "FILCHR" EQUAL TO THE ASCII
VALUE FOR THE CHARACTER TO BE FILLED AFTER, AND THE SYMBOL "FILCHR"
ITSELF TO BE THE OCTAL NUMBER OF NULLS TO BE ISSUED AFTER EACH OCCURRENCE
OF THE CHARACTER DEFINED BY "FILCHR." FOR EXAMPLE, TO PROVIDE
A FILLER CHARACTERS AFTER A CARRIAGE RETURN, SET "FILCHR=15" AND
"FILCHR=15."
THE HANDLER IS INSTALLED VIA THE FOLLOWING PROCEDURE:

1) ASSEMBLE IT AS FOLLOWS:
2) DEFINING FILLER CONDITIONAL IF NECESSARY
3) IN MACRO
4) END
5) END

2) LINK IT AS FOLLOWS:
3) .END
4) END
5) END

3) INSTALL IT AS DEVICES "KSI", AS DESCRIBED IN SECTION XXXIII
6) OF THE RT-11 V6 SOFTWARE SUPPORT MANUAL. REMEMBER THAT
7) THE VECTORS FOR THE TERMINAL MUST BE PROTECTED IN THE RT-11
8) MAP AS DESCRIBED IN THAT SECTION.
9) THE VALUES FOR THE VARIOUS TABLE ENTRIES SHOULD BE
10) NOTED AND THE VALUE OF SYMBOL "KSI" ON LAST LINE OF LISTING
11) VSIZE=0 (NON-FILE STRUCTURED DEVICES)
12) NAME=CAR (RADIO FOR "KI")
13) STATE=HIGH ORDER BYTE=0, LOW ORDER BYTE=ANY DEVICE NUMBER
14) AVAILABLE. NOTE THAT IT CANNOT BE 4. A VALUE > 10
15) IS RECOMMENDED.

4) ONCE INSTALLED, KSI WILL BE AVAILABLE WHEN THE SYSTEM IS RESTARTED.

THE HANDLER ITSELF IS ACTIVATED WITH READ AND WRITE REQUESTS, AS ARE ALL
18) RT-11 DEVICE HANDLERS, WHEN USING SYSTEM PROGRAMS WHICH OPERATE ON
19) FILE BUFFERS. SEVERAL LINES MAY ACCUMULATE IN THE BUFFER BEFORE
20) THEY APPEAR ON THE TERMINAL, AND THEN ALL AT ONCE. TO AVOID THIS
21) PROBLEM, EACH OUTPUT BUFFER CAN BE ZERO-FILLED AND SENT TO THE TERMINAL TO PREVENT
22) EACH LINE THE HANDLER WILL TERMINATE NULLS ON OUTPUT.

7) IN FORMULA, EACH LINE CAN BE FORCED IN OR OUT BY USING A REWIND
8) FOLLOWING EACH READ OR WRITE TO THE DEVICE. FOR EXAMPLE:
9) LOCALES, INPUT(S)
10) CALL ASTION (F, "KSI")
11) WRITE (F, 1)
12) REWIND 1
13) WRITE (F, 2)
14) REWIND 1
15) READ (F, 1) INPUT
16) REWIND 1
17) 
18) 
19) 
20) 
21) FORMAT ...
22) FORMAT ...
23) FORMAT ...

THE HANDLER CAN BE "RE-CONFIGURED" FOR VARIOUS VECTOR AND
26) REGISTER ADDRESSES BY CHANGING THE ASSIGNMENTS OF THE SYMBOLS
27) AVERAGE AND "KSI" ON THE FOLLOWING PAGE, EDITING THESE TWO
28) ENTRIES TO CHANGE ALL PLACING ADDRESSES.
IVECTOR AND DEVICE REGISTER ADDRESSES; REARRANGE THESE TWO TO RECONFIGURE

IKEYBOARD VECTOR

KEYBOARD CONTROL REGISTER

IPRINTER VECTOR

IPRINTER BUFFER

IPRINTER CONTROL REGISTER

IPRINTER BUFFER

OFFSET=278

IPMP BIT IN CM

IPM VALUE FOR PRIORITY 7

IPM VALUE FOR PRIORITY 6

WT=11

ITAB

LPF=1F

FALL FEED

FOM FEED

CLEAR RETURN=14

ETRNL/U

ETRNL/2

ETRNL/1

ETRNL/0

SPACE

RENAME=77

RENAME=76

PROMPT=*

PROMPT=

IOSR LAR CODE

THE FOLLOWING IN THE HANDLER INTERFACE TO THE MONITOR SET COMMAND;

FOR DETAILS OF INTERACTING TO THE SFT COMMAND, SEE THE RT=11 V2 SOFTWARE

SUPPORT MANUAL.

IFAR NALSR, SET LSRPT TO *MT2

IFAR LSR, SET LSRPT TO *MT2

OP SR1 MOV (RC)+, R3

FOR LSR, SFT LSRPT TO 377

FOR LSR, SFT LSRPT TO 377

MODIFY OPTION VARIABLE IN HANDLER

RETURN TO SET PROCESSOR
IThis is the handler header area, used by FFCH and the.

IQueue manager to store variables critical to handler operation.

KBSTRT1 "HORN TPVEC", IPRTNER VECTOR ADDRESS

"HORN TPINT", IOPPET TO PRINTER INTERRUPT SERVICE

"HORN PRY", IINPUT

KBGF1 "HORN A", ILAST QUEUE ENTRY

KBGF2 "HORN A", ICURRENT QUEUE ENTRY

IInput.

Ifollowing is the transfer initiation core:

IThe first word of this routine is the entry point for all

I transfer requests. The keyboard vector is set up by FFCH ONLY. SETS UP THE

I PRINTER VECTOR), AND THE PARAMETERS FOR THE TRANSFER ARE ESTABLISHED.

IF THE REQUEST IS A WRITE, CONTROL TRANSFERS TO THE PRINTER ROUTINE TO

IF THE REQUEST IS A TRANSFER, THE FIRST CHARACTER FROM THE USER BUFFER IF IT IS A READ,

IF THE ENTER USER BUFFER IS ZERO, THE FLAG (READFL) IS SET TO

IF THE READ IN PROGRESS, AND A PROMPT CHARACTER IS ECHOED.

IF THE TERMINAL BUFFER INTERRUPT IS ENABLED.

ADO KBINT-, RR

MOV #KBINT-, RR

CALCULATE ABSOLUTE ADDRESS OF KEYBOARD INTERRUPT SERVICE

MOV RB, #KBVFC

ISET UP KEYBOARD VECTOR

MOV #KBVF2 RETRY1, #KBVF3 #KBVF4

IINIT READ FLR AND TAR COUNT

MOV KBVF1, RB

IPRTNER TO CURRENT & ELMPT

MOV #KBVF5, RB

IF NEGATIVE, WRITE TO PRINTER

MOV #KBVF6, RB

IAPPEND ORIGINOAL PRINTER FOR LATER.

MOV #KBVF7, RB

MAKE WIDTNT INTO BYTE COUNT

MOV #KBVF8, RB

IIF ERROR, WRITE TO PRINTER

MOV #KBVF9, RB

USER BUFFER PRINTER IN BS

IIF USER BUFFER BEFORE STARTING TRANSFER

ERRO RB, #KBVF10

DEP #4

MP #4

ITRANCH IF NOT OMAP

IREADFL

IIF "READ IN PROGRESS" FLAG

JB #1, &KBVF10

IPROMPT INPUT WITH OPP

JMP KBTF

ISEND IF KEYBOARD INTERRUPT AND RETURN
THIS IS THE ADRPT ENTRY POINT. THE HANDLER IS ENTERED AT THIS ADDRESS.
IF THE MONITOR RECEIVES A REQUEST TO ADRPT ANY TO TRANSFER TN PROGRESS.

THIS IS THE TERMINAL OUTPUT INTERRUPT SERVICE. AFTER ENTERING SYSTEM STATP,
IT CHECKS IF THERE ARE ANY CHARACTERS IN THE PCHO BUFFER TO BE
PRINTED. IF NOT, IT THEN DETERMINES WHETHER A WRITE REQUEST IS IN PROGRESS
FOR NOT. IF SO, THE NEXT CHARACTER IN THE USER BUFFER IS PRINTED.
IF NOT, THE INTERRUPT IS DISMISSED.

IF THERE ARE CHARACTERS IN THE PCHO BUFFER, THE FIRST CHARACTER IN THE
LIST IS PEPCHD INTO RA, THE LIST IN THE ECHO BUFFER IS "BLIN UP"
BY ONE CHARACTER, AND THE CHARACTER IN RA IS THEN PRINTED.

IF THE FILLER CONDITIONAL CODE IS INCLUDED AT ASSEMBLY TIME,
THE CHARACTER IN RA IS COMPARED AGAINST THE CHARACTER TO BE FILLER AFTER.
IF THE SAME, A COUNT OF NECESSARY FILLS IS STUFFED IN "FILCH1" AND THE
FILLER IS PRINTED. THE INTERRUPT SERVICE THEN CHECKS THE NUMBER
OF FILLS LEFT IN THE FIRST TYPH, AND PRINTS NULLS IF ANY ARE LEFT.

TPNUT1 RSP 110.5
DC 150.5
ENTER SYSTEM STATP

TPNUT2 RIT 171A RGP 171B RPT 171C

IPDP FILCH1

IIF THERE IS A PRINT READY
IIF THE PRINTER READY
ICONDITONAL CODE FOR FILLER
MANY FILLS NEED TO BE OUTPUT

BR TPNUT3

IGN PRINT IT

MOV PC, RR
ADD #48, RR
MOV (RR), RR

IIF CH CHAR TO ECHO FROM ECHO BUFFER
IIF CHAR FROM USER BUFFER INTO RA

MVR 18
IIF CHAR FROM USER BUFFER INTO RA

MOV #LENGTH, (SP)
NUMBER OF CHARACTERS IN ECHO BUFFER ON STACK

MOV (RR), (SP)+(SP)
LINE ECHO LIST UP

IIF INCREASE COUNT OF CHARMS TO SLIDE
BRANCH IF NOT PINTER

RST (SP)
IIF NOP = CLEAN UP STACK

AND PRINT CHAR


AGAIN IF READS OR WRITING?

RST (SP)
BRANCH IF READING

RST (SP)
BRANCH IF TRANSFER COUNT

IIF OP INTERFERE TRANSFER COUNT

IIF CHAR FROM TRANSFER COMPLETE

IIF NOT PRINT NULLS
KB.MAC  V81-01  RT-11  MACRO  VMR2-69  A=APR-75  1P713151  PAGE 6

KB,MAC  V81-01  RT-11  MACRO  VMR2-69  A=APR-75  1P713151  PAGE 7
The code snippet seems to be written in a low-level programming language, possibly assembly language, and deals with buffer operations and keyboard interrupt handling. It includes instructions for moving characters between buffers, checking for specific conditions, and managing data flow between different parts of the system. The code appears to be part of a larger program designed for managing input/output operations, possibly in a real-time or embedded systems context.
38 LDA #$0F  ; move address into A
39 STA #X000  ; store address
40 STA #Y000  ; store address
41 STA #Z000  ; store address
42 STA #A000  ; store address
43 STA #B000  ; store address
44 STA #C000  ; store address
45 STA #D000  ; store address
46 STA #E000  ; store address
47 STA #F000  ; store address
48 BNE $F800  ; branch if not equal
49 BX $F800  ; branch to address
50 RET  ; return from subroutine
51  ; END
CORRECTIONS AND ENHANCEMENTS TO THE KB HANDLER AND INSTALLING HANDLERS

1. A CTRL U will double the byte count used by the handler.

2. The write interrupt is not disabled upon completion of a write.

3. The handler will not transfer a file to the KB terminal if it is 32K words or larger. This is due to the byte count overflowing a 16 bit word.

4. Execution time of both read and writes can be diminished with a minimum number of changes.

The following Edit changes correct the above problems. After the changes are made, KB.MAC must be reassembled, relinked, and reinstalled.

```
R EDIT
*FEBK*$MACR$$
*FBYTCNTS$$
*I MOVR5),R4<CR>
ROR(R5)<CR>

$$
*GDONE$CFINS$$
*GB$CBM$$
*G2$C4S$$
*A$FPLE$$
*GAK62<AK>$S
*GBEG$CBEG$$
*GDONE$CABORT$$
*GPC$SB$$
*ITPBUT$ DEC BYTCNT<CR>
BR TPOUI2<CR>

$$
*GDONE$IASS$$
*IFINS$$
*F$444$$
*I BEQ PUT<CR>

$$
*41PUT$$
*FTEMPS=C-(SP)$$
*3<GETEMPS=C(SP)-$$S
*GETEMPS=C(SP)+$$
*FTEMPS$$
*GAK$$
*EX$$
```

RT-11 V02B-05, V02C-02
KB HANDLER
RT-11 SOFTWARE SUPPORT MANUAL DEC-11-ORPCA-B-D
RT-11 SYSTEM GENERATION MANUAL DEC-11-ORGMA-A-D

Seq 4*
1 of 2
It should also be noted that the slots occupied by the TT and BA handlers (slots 1 and 12) must not be used. On the page 5-13 and 5-14 of the RT-11 Software Support Manual (DEC-11-ORPCA-B-D) the aforementioned handler slots are used in the example patches. Replacement pages will be generated to correct this. Replacement pages will also be generated for pages 4-23 and 4-26 of the RT-11 System Generation Manual (DEC-11-ORGMA-A-D). The instructions on these pages should read:

The table entries to be patched are:

$WSIZE+CCTAL OFFSET
$DVSIZE+CCTAL OFFSET
$PKNAME+CCTAL OFFSET
$STAT+CCTAL OFFSET
ERRORS IN KB.MAC (SPR 11-10433, 11-10434, 11-10595 JM)

The following errors involve the operation of KB.MAC:

1. The output interrupt is not disabled when the echoing of characters on input is done.
2. Characters, upon requesting output, are sometimes lost.
3. Nulls decrement the character count but are not placed in the character buffer.
4. When KB is used for a second terminal on a LSI-11, the processor will trap to ODT micro-code upon enabling the printer interrupt.

The following edit changes will correct the above problems. Previous changes to KB.MAC (Article Seq. #4, published in the September, 1976, DIGITAL SOFTWARE NEWS) must be installed before applying the changes below. After the changes are made, KB.MAC must be reassembled, relinked, and reinstalled.

```
.R EDIT <CR>
*EBKB.MAC$R$$
*FTPOUT:$-A$$
*I
BEQ TPOUT <CR>
CLR J#TPCSR <CR>
RT3 PC <CR>

$K$$
*GTPOUT3:$$
*SKAU$$
*FPUT:$-40$$
*-$5A$GPUS=CKBIN$$
*EX$$
```

RT-11 V02B-05, V02C-02 for COS-350 V02 USERS
HANDLER
KB.MAC
APPENDIX C
VERSION 1 EMT SUMMARY

Although Version 1 programmed requests are supported by Versions 2 and 2B of RT-11, it is strongly recommended that the Version 1 formats not be used. For purposes of compatibility, however, this section provides a brief review of the V1 format. The V2/V2B format is covered in detail in Chapter 9 of the RT-11 System Reference Manual.

In brief, the major distinctions between V1 and V2/V2B formats are:

1. V1 format has arguments pushed on the stack and in R0. V2/V2B requests generally accept a set of arguments, or an argument in R0.

2. V1 channel numbers are restricted to 16. Also, the channel number in V1 is not a legal assembler argument; it is merely an integer in the range 0 to 15.

3. V1 requests are non-reentrant because the channel number and function code are embedded within the EMT instruction.

Table C-1 lists all the Version 1 macro calls. Those in the left column have the same format as the corresponding Version 2/2B request; those in the right column have a different format, shown after the table. The operations performed by the requests are the same in both versions.

<table>
<thead>
<tr>
<th>V1 - Format Same as V2/V2B</th>
<th>V1 - Format Different from V2/V2B</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CSIGN</td>
<td>.CLOSE</td>
</tr>
<tr>
<td>.CSISP</td>
<td>.DELETE</td>
</tr>
<tr>
<td>.DATE</td>
<td>.ENTER</td>
</tr>
<tr>
<td>.DSTA</td>
<td>.LOOKUP</td>
</tr>
<tr>
<td>.EXIT</td>
<td>.READ</td>
</tr>
<tr>
<td>.FETCH</td>
<td>.READC</td>
</tr>
</tbody>
</table>

(continued on next page)
Table C-1 (Cont.)
V1 Programmed Requests

<table>
<thead>
<tr>
<th>V1 - Format Same as V2/V2B</th>
<th>V1 - Format Different from V2/V2B</th>
</tr>
</thead>
<tbody>
<tr>
<td>.HRESET</td>
<td>.READEW</td>
</tr>
<tr>
<td>.LOCK</td>
<td>.RENAME</td>
</tr>
<tr>
<td>.PRINT</td>
<td>.REOPEN</td>
</tr>
<tr>
<td>.QSET</td>
<td>.SAVESTATUS</td>
</tr>
<tr>
<td>.RCTRLO</td>
<td>.WAIT</td>
</tr>
<tr>
<td>.RELEASE</td>
<td>.WRITE</td>
</tr>
<tr>
<td>.SETTOP</td>
<td>.WRITC</td>
</tr>
<tr>
<td>.SRESET</td>
<td>.WRITW</td>
</tr>
<tr>
<td>.TTINR</td>
<td></td>
</tr>
<tr>
<td>.TTOUTR</td>
<td></td>
</tr>
<tr>
<td>.TTYIN</td>
<td></td>
</tr>
<tr>
<td>.TTYOUT</td>
<td></td>
</tr>
<tr>
<td>.UNLOCK</td>
<td></td>
</tr>
</tbody>
</table>

The formats of V1-specific requests (those listed in the right column) follow. Definitions of arguments used in these macro calls are:

- **.blk**: A block number specifying the relative block in a file where an I/O transfer is to begin.
- **.buff**: A buffer address specifying a memory location into which or from which an I/O transfer is to be performed.
- **.cblk**: The address of the five words of user memory where the channel status will be stored.
- **.chan**: A channel number in the range 0-17 (octal).
- **.crtn**: The entry point of a completion routine.
- **.dblk**: The address of the 4-word RAD50 file description (dev:file.ext).
- **.length**: The number of blocks allocated to the file being opened.
- **.wcnt**: A word count specifying the number of words to be transferred to or from the buffer during an I/O operation.

```
.CLOSE .chan
.DELETE .chan,.dblk
.ENTER .chan,.dblk,.length
.LOOKUP .chan,.dblk

.READ .chan,.buff,.wcnt,.crtn,.blk

.READEC .crtn is required only for .READC

.RENAME .chan,.dblk
.REOPEN .chan,.cblk
.SAVESTATUS .chan,.cblk
```
.WAIT .chan

.WRITES
.WRITC }
.WRITW }
 ....chan,.buff,.wcnt,.crtn,.blk

 [ .crtn is required
  only for .WRITC ]

The system macro library (SYSMAC.SML) can be used with Versions 2 and
2B to generate Version 1 programmed requests.

Under Version 2, the ..V2.. macro is capable of handling V1 expansions.
..V2.. normally expands as:

    .MCALL ...CM1,...CM2,...CM3,...CM4
      ...V2=1

This causes Version 2 expansions in all cases. To allow expansion of
all V1 requests in their V1 format (and all new Version 2 requests in
V2 format) the ..V2.. macro should not be called, but the utility
macros must still be defined:

    .MCALL ...CM1,...CM2,...CM3,...CM4

Omitting both ..V2.. and the utility macros causes all old V1 requests
to be expanded in V1 format; no V2 requests can be used.

Under Version 2B, the ..V1.. macro call enables expansion of all macros
in Version 1 format. ..V1.. expands as:

    ...V1=1

To enable expansion of all Version 1 macros in V1 format and all new
Version 2 macros in V2 format, these statements must be included:

    .MCALL ..V1..,...CM1,...CM2,...CM3,...CM4
      ..V1..

A listing of SYSMAC.SML is provided in the RT-11 System Reference
Manual.
APPENDIX D
FOREGROUND SPOOLER EXAMPLE

The following program is an example of a line printer spooler for the foreground. Instructions for its use follow.

1. Create the program using the Editor and store it on the system device under the name LSPOOL.MAC.

2. Next assemble it under MACRO and then link it to create the REL format output file:
   
   .R MACRO
   _LSPOOL=LSPOOL
   
   .R LINK
   _LSPOOL=LSPOOL/R
   
3. Load the necessary handlers (in this case, LP and RF) and run the program. All files on device RF with the extension .LST are listed on the line printer and then deleted from RF:
   
   _LOA LP,RF<CR>
   _FRU LSPOOL<CR>
   
   F>
   DEVICE TO SPOOL?
   
   B>
   [Control must be redirected to the foreground via *F.]
   
   F>
   RF:* .LST<CR>

This program assumes device DK: and extension .LPT unless otherwise indicated.
TITLE  LSPNL = LINE  PRINTER  SPOOLER
SPOOL  =  LINE  PRINTER  SPOOLER
PARPAR  =  UISPUL  PARPAR

THIS  PROGRAM  FOR  THE  BACKGROUND  TO  A  LINE  PRINTER  SPOOLER.
IT  SPARCHES  A  SPPCITED  DEVICE  FOR  IPLES  WITH  A  PARTICULAR
EXTENTION  (THE  DEFAULT  IS  LPT)  AND  PRINTS  TYPH,  MELETING
AFTER  PRINTING.  IF  WHELP  ARE  FOUND,  IT  WILL  GO  TO  SLEEP  FOR
HALF  A  MINUTE,  PERMITTING  THE  BACKGROUND  TO  RUN.

TO  RUN  LSPNL,  FIRST  LOAD  LP  HANDLER  AND  INPUT  DPVTCF  HANDLER
IF  IT  IS  NOT  THE  SYSTEM  DEVICE  TYPE.

P,L.
RIO LP,RF
P,LAPNQI.

LSPNL  WILL  TYPE:  "DPVTCF  TO  SPOOL?"
TYPE  INPUT  DPVTCF  AND  FILE  DESCRIPTION,  P,L.

IF  FILE  NOT

IFAIL  ":,V2...,BEROFF
IFAIL  :RANR,WRAT,LOOKUP,DPVTCF,CAICAP,,TTY
IFAIL  SPINT,TYQ1,RESEPT,CTRL0,,CLMPF,EXIT
IFAIL  ORATL,.,,WAIT

..V2...
APREP

ISNSWP  =  46
ERROR  COEF
PR  =  15
LP  =  12
START:
MOV  #11FF, #1138SWP  IMAKE  USER  SWAP  OVER  RUFI
MOV  #18, (PC)  ISAVE  STACK  POINTER  FOR  RESET
STKSAV:  WRP
ORATL, TDR, LP  MUST  BE  IN  MEMORY
PC  =  18
ILLEGAL  DEVI
TST  TDR,4  TYPRT  ENTRY  POINT
ANP  MERIN  ISR  TO  BEGIN  IF  LOADED
SRTN  ORAN  IBACK  TO  USER  POP  &  LOAD  LP

47
LSPOOL = LINE PRINTER SPoolER
RT-11 MACRO VMACR-1A A=APR-75 P1=3114 PARE 1+
A USEFUL FOREGROUND PROGRAM

119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134

THIS ROUTINE PRINTS THE SPoolER FILE JUST FOUND AND THEN DElEtES IT.

COPY FILE NAME AND EXTENSION
INTO FILE DESCRIPTR BLOCK
FOR A LOOKUP.
FILE IS BLANK, COPY IT, FILE IS SECTOR 0 IRR.
CHECK FOR A NULL
ERROR ON READ
COPY THE BLOCK TO CHANNEL 1
SEND OPEN TO CHANNEL 1
SEND ERROR JUST AN EOF?
SEND UNTIL EOF.
ERROR JUST AN EOF?
SEND AN INPUT ERROR
SEND FINDLP
THEN FIND ANOTHER FILE.
APPENDIX E
S/J AND F/B MONITOR FLOWCHARTS

The following flowcharts are of the Single-Job and Foreground/Background Monitors. It is recommended that the reader have source listings available for reference. Steps inside [___] are performed only in the F/B or S/J Monitor, as noted.

An index of all entry points appears at the end of the appendix.
E.1  KMON (KEYBOARD MONITOR) FLOWCHARTS
SAVEVC - Entered to rewrite the current virtual block back into the system scratch area. It also acts as the exit point for Deposit; The RTS PC will return control to KMON.
MAKE DEFAULT DEVICE = SY

FILE

DEVICE OTHER THAN SY SPEC?

Y

BADCOM

N

CCBS READ BLOCK $

INIT TO READ FILE STARTING AT 1000

BEGIN

CHAINED TO?

Y

N

SOFT RESET

SET USER SP & LOC. 5$

PROG. TOO LARGE?

Y

N

OVR COR?

N

ENTRPG

N

THE PROGRAM MAY BE ALL
WITHIN 5$

SET UP A .READW FOR REMAINING

Y

CODE

INTO RMON TO READ THE

RDOVLY

REMAINDER; ENTER USER'S

PROGRAM

E-7
GET

GET - Used to load a .SAV image into memory. If parts of the file overlay KMON/USR, those parts are placed into system scratch blocks.

GET

FIRST CALL TO GET?

Y

CLRCCE
CLEAR INTERNAL CONTROL BLOCK, CLEAR LOC. 56

SET FLAG, NOT FIRST GET CALL

LOOKUP THE FILE

FOUND?

Y

CCBB6
GET FILE'S BLOCK 0 INTO KMON

EXTRACT LENGTH OF FILE FROM LOCS. 360-377 OF FILE

A

N

?FIL NOT FND?

N

_FOLLOWING_CONTAINS
A_BITMAP_OF
---_BLOCKS_USED_BY
THE_FILE.89

A

OVERLAY KMON OR USR?

Y

READ THE FILE INTO MEMORY

FIRST BLOCK OVERLAP KMON?

Y

READSF READ PART WHICH GOES INTO REAL MEMORY

READ BLOCKS FROM FILE; WRITE THEM INTO SYS. SCRATCH

DONE WITH FILE?

N

Y

SET BITS ON IN KMON MEMORY BLOCK

RTS PC
REENTER/RUN/START

PROG. RE-ENTERABLE?

Y

GET START ADD. MINUS TWO

--- PUT -2 INTO R2 FOR CODE AT STRE.

--- INITIATE THE PROGRAM AT START ADDR-2.

N

BADCOM

RUN

GET

--- GET THE MEMORY IMAGE INTO LOW MEMORY AND SCRATCH BLOCKS, IF NECESSARY.

STARTK

OCTNUM

GET SPECIFIED START ADDRESS

NULL?

N

Y

STRE

ADD LOC. 4$ TO R2 TO GET ADDRESS

SETUP TO READ PROG. INTO MEMORY

BEGIN
E.1.1 KMON Subroutines
OVREAD/OVLINK

OVREAD - Used to read overlay command processors into memory.

```
OVREAD

SET PARAMS FOR .READW OF OVERLAY

ALREADY IN MEMORY?

Y

MARK THIS OVERLAY AS RESIDENT

$SYS

--- INTO RMON TO READ THE OVERLAY

RTS PC

--- RETURN; OVERLAY NOW IN MEMORY

N
```

OVLINK - Called from overlay processors to allow linking from one overlay to the other.

```
OVLINK

OVREAD

--- READ IN THE OVERLAY

JMP @R3

--- RE-ENTER THE SECOND PART OF THE OVERLAY.
```
ADTRAN - Used to determine if a user-typed address is a) legal (i.e., address of RMON), b) in scratch blocks on system device.

---GENERATE ?ADDR?

---BACK TO KMON.

---BACK TO KMON.
SAVEVC - Rewrites a block of memory back to the system scratch area if the block's contents were altered with a Deposit.

FILE - Called to pick up the .RAD5@ representation of DEV:FILE.EXT. It will assume a default extension of .SAV.
CCBBØ - The CCBBØ routine reads the first block of a .SAV file into the USR buffer, then moves selected locations from that block into the corresponding physical memory locations. The words moved are those marked with Ø's in the RMON bitmap. This procedure protects the system from having its vectors overlaid. If a chain is being done to a program which does not accept a CHAIN, 5ØØ-776 will be loaded with the contents of the file.
SYSK - Used to read/write blocks into and out of the system scratch area.

--- THE USR BUFFER IS USED TO HOLD THE BLOCKS TO GO INTO THE SCRATCH AREA.

--- CHECK TO SEE THAT MONITOR IS NOT OVERWRITTEN.
E.1.2 KMON Overlays
DATE/TIME

DATE

PRINT THE DATE?

Y 190$ N --- SET DATE

NUMK

GET DAY & CHECK LIMITS

MATCH UP MONTH

FOUND MATCH IN TABLE?

? 190$ 190$ N --- SET IT

NUMK

GET YEAR & CHECK LIMITS

SAVE THE DATE

EXIT

TIME

CLOCK EXISTS?

N --- NO CLOCK?

SET FOR 50 OR 60 ~

PRINT THE TIME?

Y A

INPUT TIME, CONVERT AND SAVE.

EXIT

A

GET TIME AND PRINT

EXIT

IF ONE, PRINT IT

EXIT
GT ON/OFF

GT

VT11 HARDWARE PRESENT ?

Y

FG ACTIVE ?

Y

GT OFF?

N --- GT ON

N

ALREADY THERE ?

Y

?ILL CMD?

STOP DISPLAY IF RUNNING AND TEST ITS SIZE

PROCESS SWITCHES, IF ANY

DETERMINE TOTAL SIZE AND ALLOCATE MEMORY

READ SCROLLER CODE INTO MEMORY

SCINIT --- GO TO INITIALIZATION ROUTINE IN SCROLLER

A

SCROLLER IN MEMORY

Y

STOP DISPLAY AND PAUSE PRIORITY TO 7

CLEAR POINTER TO SCROLLER

RESTORE TERMINAL-SERVICE & LOWER PRIORITY

RECOVER MEMORY

RETURN

N

?ILL CMD?

RETURN

N

RETURN
E.2 USR (USER SERVICE ROUTINES) FLOWCHARTS
USRBUF/FATAL/CDFN

The first 2 blocks of the USR are used by the USR for directory operations. They are also used by the RMON at various points for a 2-block general purpose buffer. There is, however, executable code in the buffer that can be executed every time a fresh copy of the USR is read from the system device. The functions included in the buffer are:

1. USR Relocation

   This code is executed whenever the USR is newly read into memory. It serves to make certain pointers into RMON absolute.

2. Fatal error processor and fatal error messages
   (S/J only)

3. CDFN (channel define) EMT (S/J only)

   The CDFN EMT call forces a new copy of the USR into memory to guarantee the presence of the EMT processor.

The flows for these functions follow.

NOTE

Fatal error handler and CDFN processor are RMON functions in the F/B Monitor. The only code in the buffer in the F/B system will be the USR relocation code.
USRBUF is the initial entry point for USR calls when the USR has just been read into memory. LOCATE sets up pointers into RMON.

--- START OF USR BUFFER

--- R5 WILL BE POINTED TO THE START OF RESIDENT USR CODE.

The LOCATE routine is called to update the list of pointers at RELIST. The list is initially a list of address differences (i.e., VALUE-$RMON where VALUE is the desired location and $RMON is the address of the start of RMON). LOCATE then makes all the differences into absolute addresses. Any errors which would generate a ?M-error use the FATAL error processor code to generate the message in the S/J system. This is a resident function in F/B.

--- ENTERS WITH R4 INDICATING ERROR CODE.
USRBUF/FATAL/CFN (CONT.)

CFN - A resident function in the F/B system.

---IF REQUEST IS FOR FEWER THAN THAN ALREADY EXIST, IT IS AN ERROR.

---TAKE COMMON USR EXIT.
The following flowcharts detail the code contained in the main body of the USR. On entry to the USR, R2 contains an index representing the function to be performed. This is used to dispatch control to the proper processor.
CALLING THE USR FROM A COMPLETION ROUTINE IS ILLEGAL, AS THE USR COULD HAVE BEEN INTERRUPTED.

FATAL ERROR: M-ILL USR

CALLED FROM A COMPLETION?

S/J

SAVE CERTAIN PARAMS; GET POINTER TO FUNCTION

DISPATCH TO PROPER PROCESSOR

QSET
DELETE
FETCH/RELEASE
CLOSE
ENTER
LOOKUP
RENAME
DSTAT
CSI

CDFN

HARD/SOFT RESET

THESE ARE USR FUNCTIONS ONLY IN S/J. IN F/B THEY ARE RESIDENT FUNCTIONS.
RENAME

---THIS SERVES AS RENAME FLAG.

TURN ON RENAME
BIT IN CHANNEL WORD

LOOKUP

USRCOM

---COMMON OPERATION IN OPENING A CHANNEL

LNFILE

---DEVICE IS NONFILE-STRUCTURED

SPLOOK

---DEVICE HAS ITS OWN FILE STRUCTURE (MT, CT).

DLEET

-----GET A PERMANENT FILE OF THE SPECIFIED NAME.

EMT ERROR
$1; FILE NOT FOUND

COMXIT

THIS A RENAME?

Y

FILL IN NEW FILE NAME

N

FILL IN CSW AREA

DO COMMON CODE AT CLOSE.

CLOCOM

COMXIT
LOOKUP/RENAME (CONT.)

--- HERE ON NONFILE-STRUCTURED LOOKUP

DELOUT

--- CLEAR OUT STARTING BLOCK # IN CHANNEL AREA

LNFILE

--- CHANNEL AREA WILL HAVE A FILE OPEN, WITH STARTING BLOCK = Ø.

COMXIT

--- HERE ON LOOKUP/RENAME ON 'SPECIAL' DEVICE.

SPLOOK

RENAMES DONE ?

Y

ERROR #1; CLOSE CHANNEL

COMXIT

N

SPSHEL CODE 3

COMXIT

--- A RENAME ON A SPECIAL DEVICE IS CURRENTLY ILLEGAL.

--- DO THE INDICATED FUNCTION (3) ON THE DEVICE.

LKERL

SPDEL

SPSHEL CODE 2

(DELETE)

ERROR ?

Y

LKERL

N

DELOUT

E-34
These are resident functions in F/B; USR functions in S/J.

--- ENTRY FOR HARD RESET

**HDRSET**

--- ENTRY POINT FOR SOFT RESET.

**SOFRST**

--- DO THE SYSTEM RESET

**RSTSR**

--- COMXIT

--- RSTSR

**A**

--- IF JOB HAS OVERLAYS, CHAN 17 IS NOT CLEARED.

--- RESET TO NORMAL 16 I/O CHANNELS; ZERO CHANNELS

--- RELEASE NON-RESIDENT DEVICE HANDLERS --- THOSE RESIDENT VIA .LOAD ARE NOT RELEASED.

--- WAIT FOR ALL I/O TO QUIESCE; SET QUEUE TO 1 ELEMENT

--- TURN ON INTERRUPTS FOR TTY OUTPUT

--- RTS PC

**A**

--- MAKE SURE TTY IS QUIET SO RESET WON'T CLOBBER A CHARACTER

--- WAIT FOR 11/05 TTY TO SETTLE DOWN. RESET

--- IS THERE A CLOCK?

--- TURN ON THE CLOCK INTERRUPT

--- N

--- A

--- Y
DELETE

USRCOM

------DO COMMON CHANNEL SETUP.

NONFILE DEV.

DELOUT

------SPECIAL DEVICE (MT/CT)

SPDEL

DLEET

------FIND PERMANENT FILE OF THE
SPECIFIED NAME.

LKERL

------NOT FOUND; GENERATE
ERROR #1.

MAKE THE
ENTRY AN
'EMPTY'

CLSQSH

------FINISH UP IN CLOSE CODE.
CONSOL

ENTRY

GET NEXT EMPTY SPACE

NONE LEFT

NXBLK

READ IN THE NEXT DIRECTORY SEG

ZERO LENGTH REQUEST?

OK

TAKE 1/2 LARGEST OR ENTIRE SECOND LARGEST EMPTY

MAKE THIS REQUEST LOOK LIKE ENTER FOR THIS FIXED LENGTH

THE CORRECT SEGMENT WAS RECORDED WHEN THE EMPTY WAS FOUND.

115

125

RETRN

6S

UPDATE LARGEST & SECOND LARGEST FILES LIST

THIS KEEPS TRACK OF WHERE THE LARGEST AND SECOND LARGEST EMPTY SPACES ARE LOCATED.

INCR BUMP TO NEXT ENTRY IN DIRECTORY

3S

ENTER

USRCOM

DO CHANNEL SETUP

NONFILE

COMXIT

SPECIAL DEV.

SPENTR

---CONSOLIDATE THE DIRECTORY SEGMENT IN MEMORY.
ENTER (CONT.,)

11$  

WAS IT OF LENGTH -1 ?

Y  12$

N  EMT ERROR #1; DID NOT FIND EMPTY BIG ENOUGH

DELOUT  ---DEACTIVATE CHANNEL AND RETURN.

15$

---HERE WHEN AN EMPTY OF APPROPRIATE SIZE WAS FOUND.

THIS HOLE LARGE ENOUGH ?

Y  OVER-FLOW ON THIS ENTER ?

Y  EXTEND

N  PUT A TENTATIVE ENTRY AT THE CORRECT SPOT

FILL IN THE CHANNEL STATUS AREA

SEGROW2  REWRITE THIS SEGMENT

COMXIT

N  6$

WE MUST EXPAND THE DIRECTORY IN THIS CASE.

SPETR

SPESHL

CODE 4 (ENTER)

COMXIT

E-38
EXTEND

IS THERE ANOTHER SEG?

POINT TO THE ENTRY WHICH IS 1/2 WAY DOWN IN SEGMENT

GET FIRST PERMANENT ENTRY AFTER 1/2 WAY POINT

MARK THIS AS END OF SEG; LINK THIS SEG TO NEW ONE

SEGRW2 WRITE SHORTENED SEGMENT

SEGRW WRITE OUT THE NEW SEGMENT

UPDATE THE 'HIGH BLOCK IN USE' WORD IN SEG #1

RENTR

COMERR GENERATES A FATAL ERROR, AND WILL NOT RETURN.

--- THE NEW SEGMENT IS ADJUSTED IN MEMORY, AND THEN WRITTEN OUT.

--- THIS REQUIRES A READ & WRITE OF SEGMENT #1.

--- NOW RESTART THE ENTER WITH AN EXPANDED DIRECTORY.
DSTAT/GET DEVICE STATUS

DSTAT

GESTAT

LD4DEV

SEARCH TABLES FOR DEVICE NAME

FILL IN 4 WORDS FROM TABLES

COMXIT

EMT ERROR $ COMXIT

FETCH/RELEASE

PHETCH

LD4DEV

FIND NAME IN TABLES

EMT ERROR $ ILLEGAL DEVICE

COMXIT

LOAD ADD. 1400 ?

Y RELEASE

IS HANDLER PERM. RES?

Y COMXIT

ALREADY RESIDENT ?

CLEAR THIS HANDLER'S ENTRY POINT

N

A

Y

RELEASING SYSTEM HANDLER OR ONE LOADED IS A NO-OP.
F JOB CANNOT FETCH HANDLER
---WHICH WAS NOT LOADED.
CLOSE

KLOSE

IS THIS MT/CT?

Y

GO TO HANDLER FOR THE CLOSE

N

SPESHL

BLCKH

GET A DIRECTORY SEGMENT INTO MEMORY

2S

ENTRY

GET FIRST TENTATIVE ENTRY.

NO MORE

NXBLK

READ NEXT DIRECTORY SEGMENT

NO MORE

DELOUT

N

INCR1

POINT TO NEXT ENTRY IN SEGMENT

TENTATIVE NOT ASSOCIATED WITH OUR CHANNEL/JOB.

2S

CLOCOM

SAVE POINTERS TO THIS ENTRY.

(I.E., PERMANENT FILE WITH SAME NAME)

IS THERE A PERM. ALREADY ON DEVICE?

N

MARK THE OLD ONE AS AN EMPTY.

ARE OLD & NEW IN THE SAME SEGMENT

CONSOLIDATE THE SEGMENT AND THEN RERITE IT.

N

Y

A

GET THE CORRECT SEGMENT BACK IN MEMORY

NO TENTATIVE OF THAT NAME WAS FOUND.
CLOSE (CONT.)/QUEUE EXTEND

A

THIS A RENAME?

RE-INSERT NEW FILE NAME

Y

ADJUST FINAL LENGTH OF FILE & TRAILING EMPTY SPACE.

N

CONSOLIDATE AND REWRITE THIS SEGMENT

DELOUT

QUEUE EXTEND (QSET)

QSET

SET POINTER TO CURRENT HEAD OF I/O QUEUE.

LINK ELEMENTS OF USER'S SPACE TOGETHER.

SET PRIORITY LEVEL 7

LINK NEW ELEMENT INTO EXISTING QUEUE.

SET PRIORITY LEVEL 0

COMXIT
E.3 CSI (COMMAND STRING INTERPRETER) FLOWCHARTS
CSI CODE

CSI

IS THIS SPECIAL MODE?

CLOSE FIRST NINE CHANNELS.

USE TERMINAL INPUT?

OUTPUT A PROMPTING "*"

COLLECT CONSOLE STRING; PUT INTO CSI LINE BUFFER.

SPECIAL MODE?

ZERO THE 39-WORD OUTPUT AREA.

IS THERE OUTPUT SIDE

--- PROCESS OUTPUT SIDE

STRTIN

SET FLAGS FOR INPUT SIDE.
CSI CODE (CONT.)

SPECIAL
GETFD
GET FILE DESCRIBTOR.

INPUT FILE?
Y
N

OUSTUF
CHECK THAT OUTPUT FILE NOT BEING OPENED NONFILE

SWITCH
IS NEXT CHAR. A/ N
Y

IS THERE A SWITCH VALUE?
N
N

SET BIT 15 OF SWITCH WORD; SAVE VALUE OF SWITCH

PUT FILE NUMBER INTO SWITCH WORD; BUMP SWITCH COUNT

THIS ALLOWS FOR THE /X:1:2:3 CONSTRUCTION

NOSWIT
IS NEXT CHAR A COMMA?
Y
N

NXTFIL
--GET NEXT DESCRIPTOR

IS IT AN = ?
Y
N

STRTIN
--NEW PROCESS INPUT FILES

IS IT END OF LINE?
Y
N

RETURN

ERROR ?ILL CMD?
--RESTART CSI IF TERMINAL INPUT.
ELSE RETURN WITH USER ERROR.
CSI CODE (CONT.)

RETURN

RESTORE THE USER'S STACK SAVED ON ENTRY TO CSI

RE-ENABLE ADDRESS CHECKING FOR F/B

--- F/B DISABLES ADDRESS CHECKING WHEN THE CSI IS RUNNING, THIS RE-ENABLES IT

MONOUT --- RETURN TO USER PROGRAM
E.3.1 CSI Subroutines

These subroutines are used by the CSI, and, in certain cases by the KMON.
OUSTUF

- This routine verifies that an output descriptor has a file name. If not, a syntax error is generated. It also will scan off the size in [ ] if it was specified.

OUSTUF

IS THERE A NAME?

GET DEVICE STATUS WITH .DSTAT

A

A

IS DEVICE FILE STRUCT.?

ARE THERE [ ]?

GET SIZE REQUEST AND STORE IT.

ERROR: ILL DEV?

Y

1$

Y

1$

N

N

RTS PC
GETFD - Picks up a file descriptor (DEV:FILE.EXT) from an input string and packs it in 4 words of .RAD5$.  

GETFD

GETNAM

GET DEVICE NAME

IS NEXT CHAR A :

N

MAKE THIS DEVICE THE NEW DEFAULT DEVICE.

USE CURRENT DEFAULT DEVICE NAME.

5$

Y

USE DEFAULT EXTENSION

GET NAME.

GET EXTENSION

RTS PC

NOTE: GETNAM will pick up a string of $-6$ characters and pack them in up to 2 words of .RAD5$.

WE ALREADY HAVE THE FILENAME NOW. GET THE EXTENSION.

6$

SKIP OVER DEVICE & FILE NAME

RTS PC

GETNAM - Converts a string of $-6$ alphanumeric characters to a 2-word RAD5$ group. The two words are zero filled when necessary. See code at GETNAM in the source listing if greater detail is necessary.
USRCOM

USRCOM - This routine is used to prepare a channel for I/O operations.

SAVE THE CHANNEL #.

IS CHANNEL ALREADY OPEN?

GIVE EMT ERROR $; ACTIVE CHANNEL

COMXIT

LK4DEV

FIND DEVICE NAME IN TABLE.

ILLEGAL NAME

IS IT IN MEMORY?

FATAL ERROR; $M-NO DEV

MARK THE CHANNEL ACTIVE

A
A

POINT CHANNEL WORD TO CORRECT I/O DEVICE.

IS IT A FILE-STRUCT. DEVICE?

N

USRNF

Y

SET REWRITE DIRECTORY BIT IF RENAME OR ENTER.

IS THE NAME NULL?

Y

RTS R5 TAKE NONFILE RETURN.

N

BLKCHK

READ A DIRECTORY SEGMENT INTO MEMORY

USRNF

IS IT 'SPECIAL'?

Y

SET DIRECTORY REWRITE BIT FOR SPECIAL DEVICES

N

NONFILE DEVICE EXIT

RTS R5

' SPECIAL' EXIT

RTS R5

RTS R5
DLEET/NXBLK

**DLEET** - This routine scans a device directory to find a file of a specified name.

```
DLEET
    BLKCHK
      GET A SEGMENT INTO MEMORY
        ENTRY
          FIND A PERMANENT ENTRY
            IS IT THE CORRECT NAME?
              Y  RTS R5
              N  UPDATE FILE
                  START BLOCK; POINT TO NEXT ENTRY
          NOT IN THIS SEGMENT
            NXBLK
              GET NEXT DIRECTORY SEGMENT
                ALL DONE
                  RTS R5
                NO FILE FOUND
        OK
```

---

**NXBLK** - Gets the next in the series of directory segments, if one exists.

```
NXBLK
    IS THERE ONE MORE SEGMENT?
      Y  PUT SEGMENT # INTO CSW FOR BLKCHK
      N  RTS R5
    NO MORE SEGMENTS
```

BLKCHK IS TREATED AS A CONTINUATION OF NXBLK, ALTHOUGH IT IS A SUBROUTINE ITSELF

E-56
BLKCHK - This routine isolates the segment number contained in bits 8-12 of the CSW, and checks to see if that segment is in memory at the current time. If not, it is read in.

Note that not only must the segment numbers agree, but also the device and unit numbers must be the same.
SEGRW

SEGRW - Segment Read/Write. This routine read/writes selected directory segments. There are three entry points:

SEGRW1: Use segment #1

SEGRW2: Use the segment currently in memory (BLKEY)

SEGRW: Use the number in Rø as the segment #.

---

**SEGRW1**

```
MOV $1 INTO Rø
```

**SEGRW2**

```
MOV BLKEY TO Rø
```

**SEGRW**

```
BLOCK # = SEG # * 2 + 4
```

```
SET UP AN EMT 375 READ
DO THE READ
```

```
RTS R5
```

THE ARGS ARE PUT ON THE STACK IN THIS CASE.
ENTRY - This routine uses R1 as a pointer into a directory segment to find a specified file type (Permanent, Tentative, Empty) or the end of segment mark.

```
ENTRY
  THIS ENTRY A MATCH ?
      Y   RTS R5  TAKE 'FOUND' EXIT
      N
  IS IT THE END OF SEGMENT ?
      Y   RTS R5  TAKE 'NOT FOUND' EXIT
      N
  INCR1
  POINT TO NEXT ENTRY
```

INCR1 - This routine bumps R1 to the next entry in a directory segment.

COMERR - This routine generates a fatal error from the USR. The call is:

```
JSR R5,COMERR
```

code

Code is used to indicate which error is to be generated. If .SERR is in effect, control passes to COMXIT, which returns to RMON.

SPESHL - This routine is used to effect file operations on MT/CT. This is done by passing a READ request to the Q manager. The even byte of the completion function will contain a 377. The queue manager detects this, and modifies the I/O queue element to indicate that the handler should perform a USR function.
CONSOL - This routine is used to compact a directory segment. It combines consecutive empties into one, and makes empties out of tentative files which are not associated with an active channel.

CONSOL

POINT TO TOP OF THIS SEGMENT

ENTRY

FIND A TENTATIVE ENTRY

NO MORE

14$ ALL UNUSED TENTATIVES ARE NOW EMPTIES. NOW COMBINE MULTIPLE EMPTIES TOGETHER.

Y

IS THIS TENTATIVE ENTRY OK?

N

MARK THIS ENTRY EMPTY

ADVANCE TO NEXT ENTRY

14$

POINT TO TOP OF SEGMENT.

5$

ENTRY

DONE

POINT TO FIRST EMPTY.

RTS PC

8$

POINT TO THE NEXT ENTRY

N

IS THIS ONE LENGTH Ø ?

N

IS NEXT ENTRY AN EMPTY ?

Y

COMBINE THE EMPTIES' LENGTHS & SQUEEZE SEGMENT

A

5$
**LK4DEV** - This routine looks up a specified device name in the system tables. It first attempts to find the name in the user assigned name table; failing that, the permanent name table is searched.

```
LK4DEV

IS NAME NULL?

IS IT A USER ASSIGNED NAME?

IS DEVICE HE SPECIFIED LEGAL?

SET UP TABLE POINTERS & DEVICE INDEX FOR RETURN

ERROR

RTS R4

POINT TO PERMANENT NAME TABLE.

ERROR

RTS R4

NORMAL RETURN

RTS R4
```
E.4 RMON (RESIDENT MONITOR) FLOWCHARTS FOR SINGLE-JOB MONITOR
EMT DISPATCHER

The code of the EMT dispatcher is entered when an EMT instruction is executed. The EMT instruction is decoded and control passes to the appropriate code for processing.

EMTPRO

CLEAR C BIT IN USER'S PS SAVE REGS.

EMTOUT Y

IS IT EMT 377 ?

N

Y

NEW FORMAT EMT ?

N

GET 'OLD' V1 STYLE ARGS FROM EMT INSTRUCTION

USE FUNCTION CODE TO SELECT EMT PROCESSOR

GET ARGS. FOR A NEW (374,375) EMT

IS CHAN. # LEGAL ?

Y

N

DISPATCH TO PROPER EMT PROCESSOR

TOOBIG N

IS EMT CODE LEGAL ?

A

A

EMTPRO

Y

NEW FORMAT INCLUDES 374, 375, 376, 377

ARGUMENTS ARE TREATED THE SAME WHETHER THEY WERE NEW OR OLD FORMAT EMT CALLS.

E-64
The following EMT™ requests are no-ops in the S/J Monitor:

- Mark Time .MRKT
- Cancel Mark Time .CMKT
- Timed Wait .TWAIT
- Send Data .SDAT
- Receive Data .RCVD
- Channel Status .CSTAT
- Protect Vectors .PROTECT
- Channel Copy .CHCOPY
- Special Device .DEVICE

Executing these requests in S/J will cause an immediate successful returns with no action taken.
USR DISPATCHER TABLE FOR EMT's 340-357

The USR Dispatch code handles dispatching those EMT's which require the USR. At each entry point, an INC R2 is performed. Thus, R2 acts as a function identifier once the USR is entered.

<table>
<thead>
<tr>
<th>CSI-GENERAL MODE MODE</th>
<th>CSI-SPECIAL MODE</th>
<th>HARD RESET</th>
<th>SOFT RESET</th>
<th>DEVICE STATUS</th>
<th>RENAME</th>
<th>LOOKUP</th>
<th>ENTER</th>
<th>CLOSE</th>
<th>FETCH</th>
<th>DELETE</th>
<th>CDFN</th>
<th>SET I/O QUEUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C$SIGN</td>
<td>C$SIGN</td>
<td>H$RSET</td>
<td>S$RSET</td>
<td>D$STAT</td>
<td>R$NAME</td>
<td>L$OOK</td>
<td>E$ENTER</td>
<td>C$LOS2</td>
<td>F$ETCH</td>
<td>D$LETE</td>
<td>C$DFN</td>
<td>Q$SET</td>
</tr>
<tr>
<td>CLR R4.</td>
<td>R4 IS NORMALLY NON-ZERO; IT IS CLEARED HERE FOR DISTINCTION BETWEEN CSI GENERAL MODE AND CSI SPECIAL MODE.</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
<td>INC R2</td>
</tr>
</tbody>
</table>

CALUSR
GET USR INTO MEMORY
---USR NOW IN MEMORY
JMP @USRLOC
E.4.1 EMT Processors
SET TRAP/SAVE STATUS

SET TRAP ADDRESS

T$RPST

IS THERE AN ADDRESS?

STORE THE TRAP ADDRESS IN LINE

EMTOUT

PUT IN THE ADDRESS OF MONITOR'S INTERNAL ROUTINE

SAVESTATUS

S$AVST

IS CHANNEL OPEN?

EMT ERROR #1; SAVESTAT ILLEGAL

SAVE 5 WORDS OF CHANNEL; DEACTIVATE CHANNEL

EMTDON
REOPEN

REOPEN

R$OPEN

CHANNEL IN USE?

Y

EMT ERROR ∅
CHANNEL IS IN
USE ALREADY

N

RESTORE 5 WORDS
OF STATUS TO IN-
DIATED CHANNEL

EMTDON

CLOSE

C$LOSE

--- IF A LOOKUP WERE DONE, THE
USR IS NOT REQUIRED.

C$LOS2

DO WE NEED TO
REWRITE DIREC-
TORY?

Y

N

DEACTIVATE
CHANNEL

EMTDON

RDOVLY

RDOVLY

IOSR

READ USER IN
FROM SWAP
BLOCKS

ENTRPG

SET UP USER
STACK POINTER,
FLAG USER
RUNNING

GO TO USER

E-69
READ

TSWCNT
COMMON READ/WRITE CHECKS

PUT THE CORRECT WORD COUNT INTO USER'S R6

NFREAD

WAS THERE A HARD ERROR?

Y

CLEAR HARD ERROR BIT; GIVE EMT ERROR #1

N

IS EOF BIT SET?

Y

CLEAR EOF BIT; GIVE EMT ERROR #9

N

IS HANDLER IN MEMORY?

Y

CHANGE LOGICAL BLOCK TO PHYSICAL BLOCK

N

QMANGR
QUEUE THE I/O REQUEST

HARD ERROR?

Y

N

EMTDON

--- TSWCNT HAS 3 POSSIBLE RETURNS; READ ONLY REQUIRES THE NORMAL ONE.

EMTDON

IF .SERR IS IN EFFECT, RETURN IS MADE TO USER PROGRAM.
WRITE

IF THE LAST BLOCK BEING WRITTEN IS > THE CURRENT LAST BLOCK WRITTEN, CHANGE THE CLOSING FILE LENGTH.

E-71
WAIT/CDFN

WAIT

WAIT

W$AIT

IS

CHANNEL

ACTIVE

? Y

N

GENERATE EMT
ERROR $5$

THIS IS DONE BY WAITING
FOR THE NUMBER OF FREE
QUEUE ELEMENTS TO BE
EQUAL TO THE TOTAL
NUMBER AVAILABLE.

WAIT FOR ALL I/O
TO FINISH

GIVE EMT ERROR
#1; CLEAR HARD
ERROR BIT

Y

N

DID
HARD ER-
ROR OCCUR
?

EMTDON

CDFN

Channel Define - the resident portion of CDFN cuases a fresh copy of
the USR to be read in, then enters the USR.

CSDFN

MARK USR
NON-RESIDENT

THIS FORCES CALUSR TO
READ IN A NEW USR

CSDFN2
GET JOB PARAMETERS

G$TJB

MAKE JOB # = \emptyset

\emptyset = LOW LIMIT; MOVE HI LIM. ADDR OF CHANNELS

EMTDON

GET TIME OF DAY

G$TIM

MOVE HI ORDER, THEN LOW ORDER

EMTDON

SET FPP EXCEPTION

S$FPA

IS THERE A USER ADDRESS?

\text{N} \quad \text{MAKE ADDRESS THE MONITOR'S ROUTINE}

\text{Y} \quad \text{MOVE THE ADDRESS TO INTERNAL LOCATION}

EMTDON
SPECIAL FUNCTIONS/PURGE
SOFT/HARD ERRORS

SPECIAL FUNCTIONS (MAGTAPE/CASSETTE)

S$PFUN

IS
FUNCTION
CODE < $?
N

EMT ERROR $ → EMTDON

Y

PUT A 377 INTO LOW BYTE OF THE FUNCTION CODE WORD

→ R$EAD

PURGE

P$URGE

ZERO FIRST WORD OF CHANNEL AREA

→ EMTDON

SOFT/HARD ERRORS

S$ERR

SET SOFT ERROR ACTION BIT

→ EMTDON

H$ERR

CLEAR SOFT ACTION BIT

→ EMTDON
LOCK USR

L$OCK

CALUSR
READ IN USR
IF NEEDED

EMTDON

--- THIS BUMPS A COUNTER WHICH
IS DECREMENTED DURING A
.UNLOCK. THE USR IS REALLY
UNLOCKED ONLY WHEN THIS
COUNT IS $.

CHAIN

C$CHAIN

SET BIT 4/$
IN JOB STATUS
WORD; MAKE R$/
NON-ZERO

CHEXIT

JOIN EXIT CODE
AT CHEXIT

UNLOCK USR

U$NL0K

WAS
USR
LOCKED?

MONOUT

N

DECUMENT
.LOCK
COUNTER

EMTDON

Y

IS
LOCK
COUNT $?

Y

N

A

IS
KMON IN
MEMORY?

N

IS
A 'C
WAITING?

N

IS
USR
NON-RES?

N

Y

KILL DIRECTORY
NOW IN MEMORY;
READ USR

E-75
PRINT

PRINT - Causes a line to be output to the console terminal.

![Diagram showing the flow of the PRINT process]

1. **P$PRINT**
2. **IS CHAR. A NULL?**
   - If YES, go to **EMTDON**
   - If NO, go to next step
3. **IS IT A 200?**
   - If YES, go to **ECHO A CR/LF**
   - If NO, go to **TTOPT2**
4. **GET NEXT CHAR**
5. **OUTPUT THE CHARACTER**
SETTOP

MARK JSW TO NO SWAPPING

GOING BEYOND SYS. LIMIT?  

MAKE REQUEST = SYSTEM LIMIT - 2

BEYOND KMON?

MARK KMON NON-RESIDENT

BEYOND USR?

IS USR IN VIA SET?

MARK USR NON-RESIDENT

MAKE REQUEST FOR BOTTOM OF USR MINUS TWO

SET JSW TO INDICATE SWAPPING

RETURN TOP LOCATION IN $5$ AND $R$ $5$

EMTDON
EXIT

ESXIT

SET EXIT IN PROGRESS FLAG
CLEAR CHAIN BIT

WAIT FOR I/O TO QUIESCE (# AVAILABLE = TOTAL)

SET STACK TO RMON STACK

CLEAR .SERR
DISABLE .TRPSET
RESET FOR 16 IO CHANNELS

IS KMON IN MEMORY?

Y

MEXIT2

--- GO DIRECTLY TO KMON.

N

ARE WE IN SWAPPING STATE?

Y

N

IS USER NOW SWAPPED OUT?

Y

READ USER PROGRAM BACK IN

N

2$
EXIT (CONT.) / TTYIN

TTYIN

TSTIN

IN LINE MODE?

N

Y

IS A CHAR IN BUFF?

N

Y

A

AT END OF RING BUFF?

N

Y

EMT ERROR $; NO LINE/CHAR IN BUFFER

EMTDON

CYCLE TO HEAD OF RING BUFFER

3$
TTYIN (CONT.)/TTYOUT

3$

PUT CHAR INTO REG; DECREASE CHAR COUNT

EOLST
IS IT A LINE TERMINATOR

Y

DECREASE THE LINE COUNT

IS IT A ^C?

Y

.EXIT TO SYSTEM

N

EMTDON

POPREG
POP SAVED REGS, IF NECESSARY

RTI
BACK TO USER PROGRAM

T$TOUT

OUTPUT CHAR INTO OUTPUT RING BUFFER

DID IT FIT?

Y

EMTDON

N

EMT ERROR $; NO ROOM FOR OUTPUT

E-80
EMT 376 is reserved for reporting fatal monitor errors. When a fatal error condition is encountered, a call of the form:

```
EMT 376
code
```

is executed. This indicates to RT-11 that a fatal error has occurred. The normal response is to print a ?M-error message and then abort the job. However, if a .SERR request has been done, no message will occur and control will pass to the user's program. The error bit (C bit) will be set and byte 52 will contain the negative of the error code.

```
E376

SAVE ERROR CODE AND PC OF ERROR CALL

IS THE CODE <Ø ?

Y

N

WAS .SERR DONE ?

Y

N

INIT SP TO 1000; .SRESET
READ FRESH USR

JMP IOFFSET(R5)

-----IF CODE <Ø, IT IS ALWAYS FATAL AND WILL ALWAYS ABORT THE JOB.

TURN ON THE C BIT IN HIS PS

ADJUST PC OF CALL TO BE PAST CODE WORD

EMTOUT

---INTO USR CODE TO HANDLE PRINTING THE ERROR.
```
CALUSR

CALUSR is used to ensure that the USR is in memory for a USR type request. It will handle the situation where the user program must be written to scratch blocks before the USR is read in. Entry is made at CLUSR2 when an error has occurred and the error processing code in the USR buffer is required.

EITHER 'NORMAL' VALUE OR WHAT THE USER HAS PUT IN LOCATION 46.
E.4.2 Clock Interrupt Service

The interrupt service for the clock is primitive. The clock vector is set up such that the interrupt routine is always entered with the C bit = 1. At the interrupt routine, the code is:

ADC $TIME+2
ADC $TIME
RTI

Since the C bit is 1, $TIME+2 is incremented by the ADC. When the low order word goes from 177777 to 0, the C bit remains on and $TIME is then incremented. No 24 hour wrap around is provided.
E.4.3 Console Terminal Interrupt Service
TT INPUT INTERRUPT SERVICE

--- HERE ON INPUT INTERRUPT

SAVE REGS; PUT CHAR INTO RS BACK TO PRIOR. S

----- THIS LOWERS THE PROCESSOR PRIORITY FROM LEVEL 4 TO LEVEL S.

IS IT LOWER CASE?

N

IS IT A SPECIAL CHAR?

Y

N

CONVERT IT TO UPPER CASE

DISPATCH TO CHARACTER PROCESSOR

CTRLC ^C
CTRLO ^O
CTRLQ ^Q
CTRLS ^S
CTRLU ^U
ALT 33,175,76
RUB RUBOUT

NOTE: THE CHARRS BELOW THE LINE ARE ONLY CHECKED WHEN TT IS IN LINE MODE. IN CHARACTER MODE THEY ARE NOT ACTED UPON.
TT INPUT INTERRUPT SERVICE (CONT.)

T.SPEC

Y

A PRINTING CHAR?

N

--- IF IT IS < 40, IT IS A NON-PRINTING CHARACTER.

ECHO ' FOLLOWED BY CHAR + 100

--- MAKES NON-PRINTS INTO EQUIVALENT CHARACTER

N

--- THIS ONE IS NOT A RUBOUT. IF LAST CHARACTER WAS, ECHO A CLOSING '\'.

WAS LAST CHAR A RUBOUT?

Y

ECHO A \  

TTINC3

N

MAKE CHARACTER A BELL  

3S

WILL THIS CHAR FIT?

Y

UPDATE RING BUFFER POINTERS, COUNTER

N

TERMINATOR

F-92

EOLTST TEST FOR LINE TERMINATOR

BUMP LINE COUNTER BY 1

UPDATE PREVIOUS CHAR TO BE THE CURRENT ONE

3S

E-87
TT IN P U T I N T E R R U PT S E R V I C E (C O N T.)

--- TT: USES HOOKS INTO THE RESIDENT SERVICE TO EFFECT ITS PROCESSING. SEE TT: HANDLER FOR DETAILS.

--- IF <CR>, PUT IN AUTOMATIC LINEFEED. WE SUBTRACT 3 FROM CHARACTER TO CHECK FOR <CR>. THAT MAKES THE CHARACTER A <LF> AT TTINC3.

--- THIS USES SAME EXIT SEQUENCE AS DOES AN EMT.
These routines are entered when any of the corresponding special characters are struck.

ALTMODE - 33,175,176

MAKE THE CHAR RECEIVED A 33

TTINC3

----- ALL POSSIBLE CODES ARE CONVERTED TO 33.
AT SUBROUTINE TTOUT, A 33 ABOUT TO BE ECHOED IS CHANGED TO A $.

CONTROL O

CTRLO

CLEAR OUTPUT RING BUFFER; ECHO 'O'

REVERSE THE CTRL O FLIP-FLOP

GENERATE <CR><LF>

EMTDON

CONTROL S, CONTROL Q

CTRLS

SET SYNC FLAG NON-ZERO

EMTDON

CTRLQ

CLEAR THE SYNC FLAG

FORCE AN OUTPUT INTERRUPT

EMTDON

---DO THIS BY CLEARING THEN SETTING THE OUTPUT INTERRUPT HIT

E-89
CONTROL C

CTRLC

GENERATE ^C <CR><LF>

WAS LAST CHAR A ^C?

Y

IS A .EXIT NOW IN PROGRESS?

Y

------EXIT IS ALREADY IN PROGRESS. LET IT CONTINUE UNMOLESTED.

N

IS DIRECT. OP IN PROGRESS?

Y

SET DELAYED ^C FLAG TO 140000

THIS WILL CAUSE THE ^C TO BE PICKED UP ON RETURN FROM THE INTERRUPTED OPERATION.

N

DELAY FOR 11/05 TT; DO A RESET

IS THERE A CLOCK?

Y

TURN ON THE CLOCK INTERRUPT

N

SET # FREE I/O QUEUE ELEMENTS= TOTAL AVAILABLE

---- THIS INDICATES TO .EXIT THAT ALL I/O IS DONE.

.EXIT
RUBCOM/RUBCM2

RUBCOM will update the input ring buffer pointers when a character is to be deleted.

RUBCM2 checks to see if the ring buffer is empty. The buffer is empty if either the count = ∅ or if the character to be deleted is a line terminator. This routine falls into routine EOLTST. The zero condition is returned if the buffer is empty.
TT OUTPUT INTERRUPT SERVICE

TTOUT INT --- HERE ON OUTPUT INTERRUPT

IS S FLAG UP ?

Y

TTPOXT

N

ARE WE DOING TABS OR FILLER ?

Y

PUT CHARACTER OUT TO TPB

TTPOXT --- (TELEPRINTER BUFFER)

N

INTO TT: TO SEE IF IT IS ACTIVE

Y

AT END OF RING BUFF ?

N

WRAP THE POINTER AROUND

Y

DECREASE COUNT IN RING

N

ALL DONE OUTPUT ?

Y

TTDON

N

GET CHARACTER FROM RING BUFFER

A

A

IS IT <48 ?

N

TCHKSP

CHAR. <48 WILL NOT PRINT.

Y

IS IT <177 ?

N

ADJUST TAB STOP, IF NECESSARY

Y

TPRNT2

BUMP BUFFER POINTER

TTPOXT

PUT CHARACTER OUT TO TPB
TTORUB and TTOPUT handle the printing of ALTMODE and RUBOUT. They print a $ for ALTMODE and \ for RUBOUT.
OPUT actually puts the output character into the ring buffer. It updates the ring pointers and sets the interrupt enable bit. If the buffer is full, it returns with the C bit set.

OPUT

IS "O SET ?

Y

N

CLEAR OUT PARITY BITS

WILL THIS CHAR FIT ?

Y

N

RTS PC

C BIT SET BY THE COMPARISON DONE HERE.

STORE THE CHARACTER. UPDATE POINTERS

AT END OF RING ?

Y

N

CYCLE BACK TO START OF BUFF. BUMP COUNT

SET OUTPUT INTERRUPT ENABLE

CLEAR C BIT IN CURRENT PS

RTS PC

BUMP COUNT
E.4.4 I/O Routines
I/O QUEUE MANAGEMENT Routines

QMANGR

I/O COMPLETION WILL FREE QUEUE ELEMENTS

IS THERE ROOM IN Q?

Y --- TO LEVEL 7

PICK UP THE FIRST AVAILABLE ELEMENT

MAKE THE NEXT ELEMENT THE FIRST AVAILABLE

--- TO LEVEL 9

FILL IN THE I/O QUEUE ELEMENT

ZERO THE FUNCTION WORD

----- IF THE BYTE OF THE COMPLETION ROUTINE WORD = 377, THIS IS INTERPRETED AS A FILE REQUEST ON MT OR CT.

IS THIS A CALL FROM MT/CT?

Y

FILL IN FUNCTION CODE

----- TO LEVEL 7

IS .EXIT GOING?

Y

N

IS DEVICE NOW BUSY?

Y

58

N

MARK THE DEVICE BUSY

TO LEVEL 9

A

E-98
I/O QUEUE MANAGEMENT Routines (cont.)

A

JSR PC, (R2)
GO TO DEVICE HANDLER

5$
LINK THIS ELEMENT INTO DEVICE'S Q

MAKE HANDLERS LAST QUEUE ELEMENT POINT TO THIS ELEMENT

to level $

- - - - A SYNCHRONOUS REQUEST WILL WAIT FOR I/O TO FINISH

DO WE WAIT FOR I/O?

Y

IS I/O DONE?

Y

N

RTS PC

RTS PC
I/O QUEUE COMPLETION

COMPLT is entered when an I/O transfer finishes.

- **A**
  - **DO A COMPLETION ROUTINE?**
    - **N**
      - **SET COMP. ROUTINE IN PROG. FLAG; SAVE 52**
      - **JSR PC, (R5)**
        - **TO COMPLETION ROUTINE**
      - **RESTORE 52; DECREMENT IN PROG. FLAG**
    - **Y**
      - **IS DEVICE FREE?**
        - **N**
          - **NOREPOP; RESTORE SAVED REGISTERS**
          - **RTS PC**
        - **Y**
          - **JSR PC, 2(R4)**
            - **RECALL HANDLER FOR NEXT TRANSFER**
          - **GO BACK TO PS WE ENTERED WITH**
          - **A**

- **COMPLT**
  - **SVREG**
    - **SAVE REGISTERS**
      - **WAS THERE A HARD ERROR?**
        - **Y**
          - **HALT ON HARD ERROR?**
            - **Y**
              - **HALT**
            - **N**
              - **CHANNEL # INTO R1**
              - **DECREASE THE I/O PENDING COUNT ON THE CHANNEL**
              - **--- TO LEVEL 7**
              - **MORE FOR DEVICE TO DO?**
                - **N**
                  - **SET 'DEVICE FREE' FLAG**
                - **Y**
                  - **PUT THIS Q ELEMENT BACK INTO FREE LIST**
                  - **BUMP THE ELEMENT FREE COUNTER**
                  - **GO BACK TO PS WE ENTERED WITH**

E-100
E.5 RMON (RESIDENT MONITOR) FLOWCHARTS FOR
FOREGROUND/BACKGROUND MONITOR
E.5.1 EMT Processors
EMT DISPATCHER

EMTPRO -- INTERRUPT ENTRY POINT. EMT'S ARE PROCESSED ON USER'S STACK. EXIT WITH RTI.

EXIT 377
E376 376
FATAL ERROR EMT

FETCH EMT CODE

CHECK ARG.
LIST ADDRESS

VALID ?
N
Y 3S

CHANNEL # TOO BIC?

LEGAL FUNCTION CODE ?

EMTCOM

LOCATE EMT IN DISPATCH TABLE

ARGUMENTS VALID ?
N
Y

POINT TO CHANNEL AREA

DISPATCH TO EMT

<374
3S

EXTRACT CHANNEL NUMBER

EMTCOM
EMT 16 DISPATCH, .RCTRL0, .PRINT

D$STAT
INC R2

R$NAME
INC R2

L$OOK
INC R2

E$NTER
INC R2

C$LOS2
INC R2

F$ETCH
INC R2

D$LETE
INC R2

Q$SET

EMTUSR

CALUSR
CALL USR INTO MEMORY

1$S

SET USR RUNNING BIT

JUMP TO USR

R$CTLO
--- .RCTRL0

CLEAR CTRL O FLAG

TTRSET
COPY SPECIAL MODE FLAG

EMTRTI

P$PRINT

FORCE CTY CONTEXT SWITCH

2$

GET A CHARACTER

CHAR A NULL ?

N

CHAR A 2$S ?

N

PRINT <CR><LF>

Y

EMTRTI

EMTRTI

TTOUT
PUT CHAR INTO OUTPUT BUFFER

2$

E-105
PROTECT, .CPYCH

P$ROTE ---- PROTECT VECTORS

ADDR < 5$0 ?
N 2$ Y
ADDR 0 MOD 4 ?
N 2$ Y
CONSTRUCT THE MASK

COMPUTE BYTE OFFSET INTO MAP

POINT TO MAP IN IMPURE AREA

ENTER SYSTEM STATE

BITS ALREADY SET ?
Y
RETURN ERROR 0
N
SET BITS IN MEMORY MAP

SET BITS IN TASK MAP

EMTRTI

C$PYCH ---- COPY CHANNELS

THIS CHANNEL ACTIVE ?
Y 2$ N
ENTER SYSTEM STATE

OTHERJB GET OTHER JOB'S IMPURE AREA

DOES IT EXIST ?
Y
ENOUGH CHANNELS ?
N 2$ Y
OTHER CHANNEL ACTIVE ?
N
RETURN ERROR 0

TRANSFER THE CHANNEL
CLEAR DWRIT$ REWRITE ON CLOSE BIT

WAS IT ENTERED ?
Y
MAKE IT A .LOOKUP

EMTRTI

EMTRTI

E-110
AN ADDRESS --- OF \$ MEANS MONITOR HANDLES ERROR
.TRPSET, .DEVICE (CONT.)

--- SUBROUTINE USED BY .MRKT AND .TWAIT

MARKTM

GET NODE FROM AVAIL QUEUE

SET UP HIGH AND LOW ORDER TIME

SET UP COMPLETION ADDRESS

CLEAR CHANNEL OFFSET

STORE I.D. AND JOB #

ENTER SYSTEM STATE

SUBTRACT PSEUDO-CLOCK TO MAKE TIME SYSTEM RELATIVE

LINK INTO CLOCK Q IN ORDER OF EXPIRATION

LEAVE SYSTEM STATE

RETURN

B

CHKSP

CHECK ADDRESS

WITHIN JOB LIMITS?

N — EMERG

Y

8$

BUMP POINTER

S$TRAP — .TRPSET

BUMP POINTER

D$VICE — .DEVICE

POINT TO WORD TO FILL

18$

MOVE ADDRESS FROM STACK TO IMPURE AREA

EMTRTI

CHKSP

Y

ADDRESS ZERO?

N

1$

USR RUNNING?

N

2$

ADDRESS ODD?

N

THIS JOB IS B/G?

N

Y

IS IT KMON?

Y

ASSUME NO ERRORS FROM KMON

N

WITHIN JOB LIMITS

Y

GOOD --- 1$

BUMP RETURN TO GOOD ADDRESS

ERROR --- 2$

RETURN

RETURN

E-113
.SYNCH, .GTIM

$SYNCH

NODE IN USE?

Y \rightarrow 20S

N

POINT R5 AT GOOD RETURN

SET UP NODE TO LOOK LIKE I/O QUEUE NODE. PUT SYNCH ADDR IN NODE. SET WORD COUNT TO -1 TO FLAG THIS A SYNCH NODE

VALID JOB #?

N \rightarrow A

Y \rightarrow $RQTSW REQUEST TASK SWITCH

POINT TO TASK'S IMPURE AREA

DOES TASK EXIST?

N \rightarrow A

Y \rightarrow IS IT ALIVE?

N \rightarrow A

Y \rightarrow SET CPENDS$ COMPLETION PENDING

B

GSTIM --- .GTIM

ENTER SYSTEM STATE

115

GET HIGH AND LOW ORDER PSEUDO-TIME

ADD IN ACCUMULATED TICKS TO GET REAL TIME

FAST MIDNIGHT?

N \rightarrow EMTRTI

Y

ADJUST TIME WORDS

BUMP THE DATE

11S -- DO THE .GTIM ALL OVER AGAIN

/\ CONTROL RETURNS TO CODE AFTER .SYNCH WHEN IT IS CALLED BY COMPLETION QUEUE MANAGER AT PRIORITY ZERO.

E-114
HARD AND SOFT RESET

HARD RESET ENTRY

ENTLR SYSTEM STATE WITH SOFT RESET CALLED AT EXIT

I/O RESET ENTRY

IORSET

SAVE REGS. Ø-3

SCAN HANDLER ENTRY POINT TABLE

5$

IS HANDLER RESIDENT?

5$

END OF TABLE?

SET HANDLER HOLD FLAG IN HANDLER

5$

ANY MORE Q ELEMENTS?

5$

THIS JOB'S?

9$

DISCARD IT

9$

5$

RESET HOLD FLAG

TOP ELEMENT COMPLETE?

5$

CLEAR COMPLETION FLAG

COMPLT

CALL QUEUE COMPL. FOR TOP ELEMENT

9$

5$

JBABPT

ABORT PENDING MESSAGES

CMARKT

GO CANCEL PENDING MARK TIMES
HARD AND SOFT RESET (CONT.)/RDOVLY

--- REVERT STOPS ALL I/O, RELEASES HANDLERS, REMOVES EXTRA CHANNELS AND RESETS THE I/O QUEUE

S$RESET

REVERT
RESET I/O

POINT TO CHANNELS

JOB WAS OVERLayed?

N 1$

SKIP CHANNEL 17

Y

1$

CLEAR ALL CHANNELS

EMTRTI

REVERT
QUIESCE
STOP I/O

POINT TO B/G CHANNELS

IS JOB THE B/G?

Y

POINT TO F/G CHANNELS

reset channels to original 16

RESET SUSPEND COUNT

reset queue of available nodes to point to the one internal node. CLEAR completion queue

RETURN

A

3$

Y

JOB IN F/G?

N

PURGE NON-RESIDENT HANDLERS FROM SENTRY

3$

N

ENABLE TT INPUT INTERRUPT

RETURN

3$

RDOVLY

FLAG USER PROGRAM RUNNING

$SYS READ IN USER FROM SWAP BLOCKS

ENTRPGB

SET UP USER STACK POINTER

FLAG USR GONE

DEQUUSR

E-117
CANCEL MARK TIME

1$ CSMKT

GO TO SYSTEM STATE

CMARKT

GET A QUEUE ELEMENT

1$ 4$

AQLINK:
RETURN ELEMENT TO FREE LIST

1$ 1$

CANCEL ALL?

EXIT (RTS PC)

3$

END OF Q?

Y

N

THIS JOB'S ELEMENT?

N

1$

Y

CANCEL ALL?

N

CORRECT ELEMENT?

N

1$

Y

REMOVE ELEMENT

4$

CANCEL ALL?

N

RETURN TIME REMAINING

4$
E.5.2 Job Arbitration, Error Processing
COMMON INTERRUPT
ENTRY AND EXIT

SET UP RETURN ADDRESS TO USE WHEN EXITING

SAVE R4, BUMP LEVEL COUNTER.

IN SYSTEM STATE?
Y
N
SWITCH STK PTRS.

LOWER TO HANDLER'S PRIORITY

CALL HANDLER BACK AS SUBROUTINE

RETURN HERE FROM HANDLER

EXIT TO USER ENTRY POINT (TIMER, EXSWAP RETURN HERE)

ANY CLOCKS TICKS?
Y
N

ANY FPU INTERRUPTS?
Y
N

SET PRIORITY TO 7
ANY ABORTS OR CONTEXT SWITCH?
Y
N
EXSWAP CALL SCHEDULER

SWITCH TO USER JOB'S STACK AND RESTORE ITS REGISTERS.

EXINT

FAKE FPP INTERRUPT

FPU HDW?
Y
N
SAVE FPU STATUS REGISTER

SET TO RETURN TO USER'S FPU ROUTINE

DOES JOB HAVE ONE?
Y
N
USE RMON'S FPP ROUTINE

RESTORE REGS
EXINT

-USED BY MONITOR TO GO TO SYSTEM STATE.
EXSWAP

CALLED BY COMMON EXIT TO USER
CODE TO PROCESS ACTION SWITCH

ACTION IS AN ABORT?

Y --> ABORT

N --- DO A TASK SWITCH

CLEAR ACTION SWITCH LOWER PRIORITY TO Ø
DECREMENT JOB NUMBER

--- TRY TO RUN JOB NEXT LOWER
IN PRIORITY (= JOB NUMBER)
I.E., RUN HIGHEST JOB RUNNABLE.

END OF JOB LIST?

Y --> PLAY WITH CONSOLE LIGHTS

N --> EXUSER

DECREMENT JOB NUMBER

N

THIS JOB EXISTS?

Y

IS JOB BLOCKED?

N --> 4$ TASK WAITING TO RUN ITS
I/O COMPLETION ROUTINE

Y --> 3$ BLOCKED IN COMPLETION?

N

COMPLETION PENDING?

Y

SUSPENDED?

N

4$ IF COMPLETION PENDING, CNTXSW
WILL FAKE INTERRUPT ON TASK
STACK TO CALL COMPLETION QUEUE
MANAGER ON EXIT.

3$

CNTXSW

SWITCH IN THIS JOB

EXUSER

3$
JOB ABORT

UABORT
---
REQUESTS ABORT OF CURRENTLY RUNNING JOB.

GO TO SYSTEM STATE AND SET ABORT REQUEST FLAG FOR CURRENTLY RUNNING JOB. DONE AT LEVEL 7.

ABORT
---
ABORTS ALL JOBS WITH ABORT REQUEST FLAG SET IN THEIR JOB STATUS WORD.

DROP TO PRI Ø

1$ SWAPME

FLAG ACTION FOR CURRENT JOB

CLEAR ABORT REQUEST FLAG FOR CURRENT JOB. SEARCH TABLE OF IMPURE POINTERS FOR JOBS IN MEMORY.

2$ FOUND A JOB ?

Y

ABORT ?

N

END OF TABLE ?

Y

EXUSER
JOB ABORT (CONT.)

AB1

DIRECTORY OP IN PROGRESS?

Y

DON'T ABORT JUST YET

N

BY THIS JOB?

Y

2$

N

CNTXSW

SWITCH TO ABORT CONTEXT

IORSET

RESET ACTIVE I/O

FORCE JOB TO EXIT WHEN RESUMED. CLEAN UP JOB'S IMPURE AREA

1$
BLOCK A TASK/UNBLOCK A TASK
REQUEST TASK SWITCH

$SYSWT

JOB STILL BLOCKED?

N
RETURN

I.E., BLOCKING CONDITION STILL EXISTS?

Y

TURN ON BLOCKING BIT

PREVIOUSLY UNBLOCKED?

N
RETURN

Y

SWAPME

UNBLOK

BLOCKING BIT ON?

N
RETURN

Y

CLEAR BLOCKING BIT. GET JOB NUMBER

$SRTSW

REQUEST A TASK SWITCH

RETURN

DLYUSR

--- WAIT UNTIL USR IS AVAILABLE

SET BLOCKING BIT TO WAIT FOR USR.

CAUSE SCHEDULER TO SCAN TASK LIST STARTING AT CURRENT JOB NUMBER

GET CURRENT JOB'S PRIORITY

$SRTSW

REQ. JOB PRIORITY < CURRENT?

Y
EXIT

N

$SRSIG

NEW REQUEST PRIORITY < PREV. REQ.?

Y
EXIT

N

SET TASK SWITCH ACTION FLAG

EXIT

RTS PC

E-126
CHANGE CURRENT CONTEXT

CNTXSW ---- ENTER IN SYSTEM STATE, PRIORITY 0

FETCH CURRENT JOB'S STK PTR

SAME AS REQUESTED JOB? Y 7$

SAVE REG 0-3 ON ITS STACK. SAVE 34-52 ON ITS STACK.

SWAP FPU? N 51$

FPU EXISTS? Y

SWAP FPU REGISTERS

51$

SWAP SPECIAL LIST? N 53$

SWAP ITEMS IN SPECIAL LIST

SAVE PTR TO TOP OF EXTRA LIST, OLD STACK PTR, NEW JOB CONTEXT.

53$

A

FETCH NEW JOB STACK POINTER

SWAP IN SPECIAL LIST? Y

55$

SWAP IN ITEMS IN SPECIAL LIST

6$

SWAP FPU? N

FPU EXISTS? Y

SWAP FPU REGISTERS

6$

6$

E-127
CHANGE CURRENT CONTEXT (CONT.)

6$

RESTORE LOC. 32-52.
RESTORE REGISTERS.
SAVE POINTER TO JOB STACK.

SET UP JOB NUMBER.
THAT'S IT!

8$

WAS JOB DOING COMPLETION ?

Y

8$

SHOULD IT ?

N

FAKE INTERRUPT ON STACK.
UNBLOCK JOB.
SET COMPLETION FLAG.
SAVE CHANNEL & AND ERROR BYTE IN IMPURE AREA.
EXITS TO COMPLETION QUEUE MANAGER.

EXIT (RTS PC)

E-128
E.5.3 Queue Managers (I/O, USR, Completion)
ENQUEUE/DEQUEUE USR

ENQUEUE USR

ENQUSR

--- REQUEST USE OF USR IF BUSY

IS IT OURS ALREADY?

Y

EXIT

N

IS IT FREE?

N

DLVUSR

--- SEE: REQUEST TASK SWITCH

Y

DECLARE OUR OWNERSHIP

EXIT

RIDUSR

--- SWAP OUT USR

CLEAR USR REQUEST LEVEL

ABORT REQUESTED DURING DIRECT OP?

Y

UABORT

N

IS USR SWAPPING?

N

DEQUSR

Y

WAS USER SWAPPED OUT?

N

READ USER BLOCKS BACK INTO MEMORY

FLAG USR NON-RESIDENT

EXIT

DEQUEUE USR

DEQUSR

--- GIVE UP THE USR

THIS JOE OWN USR?

N

EXIT

Y

FLAG USR NOT IN USE

GET NEXT JOB

IS IT ACTIVE?

N

WAITING FOR USR

UNBLOCK

LET THE JOB RUN

GIVE USR TO IT?

EXIT

E-132
I/O QUEUE MANAGER

QMNGR

INSERTS I/O REQUEST
NODE IN HANDLER'S
REQUEST QUEUE.

POINT TO Q
OF AVAILABLE
ELEMENTS

IS AN
ELEMENT
AVAILABLE?

Y

ADVANCE QUEUE
AND BUMP
# OF REQUESTS

BUMP CHANNEL
REQUEST COUNTER

INSERT: BLOCK #, PTR TO
CSW, UNIT #, JOB #,
BUFFER ADDR, WORD
COUNT, COMPI. ADDR.

MT/CT
SPECIAL
FUNCTION?

N

PUT IN FUNCTION
BYTE AND
COMPLETION
FUNCTION

ENTER SYSTEM
STATE, THEN
HOLD THE HANDLER

Y

HANDLER
ACTIVE?

N

UNHOLD HANDLER
AND INSERT
THE Q NODE

EXIT TO
HANDLER

N

QFULL

$2$

POINT TO
QUEUE HEAD

GET A
QUEUE ELEMENT

NEW
ELEM. PRIOR.
< THIS ELEM.?

Y

ELEMENTS
LINKED IN
PRIORITY
ORDER

N

LINK NEW
ELEMENT IN

DID
HANDLER
COMPLETE
WHILE HELD?

N

QFULL

LOWER
PRIORITY TO $\emptyset$

$1\&$

USWAP0

CMPLT2

DO THE
COMPLETION

Y

ENTER SYS.
STATE TO WAIT
FOR A Q ELEMENT

SRQSIG

$1\&$

WAIT
FOR COMPLETION?

N

EXIT

Y

UNHOLD THE
HANDLER

EXIT

WAIT UNTIL
DONE
COMPLETION QUEUE MANAGER

$CRTNE

--- ENTRY POINT (ENTERED WHEN CNTXSW FADES AN INTERRUPT ON STACK BEING SWITCHED IN.)

GET ANOTHER COMPLETION ROUTINE

N

NONE ?

Y

CLEAR COMPLETION FLAG

EXIT TO USER (RTI)

35$

GET COMPL. ROUT. ADDRESS

CALL COMPLETION ROUTINE AS A SUBROUTINE

1$

(2$

LINK FORWARD IN COMPL. QUEUE

END OF QUEUE ?

Y

CLEAR LAST O. ELEM. FLAG

 THIS IS SYNCH ELEMENT ?

Y

35$

RETURN TO AVAIL. QUEUE

N

E-135
E.5.4 Clock Interrupt Service
E.5.5 Console Terminal Interrupt Service
TT INPUT INTERRUPT ROUTINE

TTIINT

$INTEN
COMMON INTERRUPT ENTRY

FETCH CHAR. AND STRIP PARITY

NULL?
Y EXIT
N

CONVERT LOWER CASE TO UPPER CASE

SPECIFIC CONTROL CHAR.
Y DISPATCH TO PROCESSING ROUTINE

CTRL.C
CTRL.O
CTRL.S
CTRL.Q
CTRL.F
CTRL.B

TTIDSP
N
IN SPECIAL MODE?

Y TTINC3

N ESCAPE, CTRL U, OR RUBOUT?

Y DISPATCH TO PROCESSING ROUTINE

CTRL.U
ALT
RUB

ALT

PREVIOUS CHAR A RUBOUT?
N TTIINC3

Y

VT@5?

Y TTIINC3

N ECHO

TYPE A "\"

TTINC3

TTINC3

ROOM IN BUFFER?
N 25S
Y

BUMP COUNT & INPUT POINTER

END OF BUFFER?
N WRAP POINTER TO TOP

Y

INSERT CHAR INTO BUFFER

EDLTST
CHICK FOR END OF LINE

EOL?
N 7S
Y

BUFF LINE COUNT

SING TT: HANDLER?
N 7S
Y TTHIN

CALL HANDLER

7S

E-140
TT INPUT INTERRUPT ROUTINE (CONT.)

75
SAVE FOR PREVIOUS CHAR. TEST

CTRL.C
ECHO$C
ECHO "IC<CR><LF>"

CTRL.B
GIVE INPUT OWNERSHIP TO BG

TTOENB
ENABLE PRINTER INTERRUPT

UNBLOK
UNBLOCK USER IF WAITING FOR TT

SECOND CTRL/C ?

N
TTOPT3
ECHO THE CHARACTER

N
Y

Y

Y

REQUEST TASK ABORT

EXIT

105
$REQABT

CARRIAGE RETURN ?

Y

--GENERATE A <LF> ON RECEIPT OF <CR>

TTINC3
--GO INSERT IN BUFFER

CHANGE CODE TO LINE FEED

255
SET UP TO ECHO A BELL

SAVE CHAR. IN CASE OF IC IC

--ECHOES 'BELL' WHEN INPUT BUFFER IS FULL

TTOPT4

45
ANY FG JOB ?

Y

ACTIVE ?

N

N

GIVE INPUT OWNERSHIP TO FG

EXIT

FG OWNS OUTPUT ?

N

GIVE OUTPUT OWNERSHIP TO FG

SET TO PRINT ID ("F >")

EXIT

TTOENB

CTRL.F

ECHO

ECHO "F?"

EXIT
TT OUTPUT INTERRUPT ROUTINE

TTOINT

INTEN
DROP TO LEVEL 4

STILL FILLING NULLS ?
Y
OFILL
N
DECREMENT COUNT, THEN PRINT A NULL

XOFF FLAG SET ?
Y
EXIT
N

FOON

BUMP TAB FILL COUNT

STILL PRINTING A TAB ?
Y
8$
N
CLEAR TAB FILL COUNT

PRINTING AN ID ?
Y
3$
N
GET NEXT ID CHAR

END OF ID ?
Y
CLEAR ID FLAG (TTOID)

3$

3$

POINT TO USER'S IMPURE AREA

PRINTING EOL (I.E., A <LF>) ?
N
5$
Y
POINT TO JOB TABLE

4$

4$

GET A JOB POINTER

THIS JOB EXIST ?
N
4$
Y
WAS IT END OF TABLE ?
Y
TURN OFF TT INTERRUPTS
EXIT
N

4$

4$

JOE ACTIVE ?
Y
USING "T": HANDLER ?
Y
45$
N
WANTS TO TYPE ANYWAY ?
Y
45$
N

PRINTS JOB --- I.D. (F> OR B>)}
TT OUTPUT INTERRUPT ROUTINE (CONT.)

PROCESS ---
SPECIAL
CHARACTERS

TAB ?
Y 11$
N
BACKSPACE ?
Y 14$
N
FORM FEED ?
N
Y
HARDWARE FORM FEED ?
N
SIMULATE FF WITH 8 <LF> S
13$
11$
Y
HARDWARE TAB ?
N
MOVE LINE POS. TO FILL COUNTER. SET TO ECHO SPACES
12$
12$
Y
FIX LINE POSITION
13$
BUMP CHAR. POINTER

14$
SCOPE MODE ON ?
Y
N
OBUMP

BACK UP CHARACTER POINTER
OBUMP
TTHOUT
CTRL O STRUCK ?
Y
N
OUTPUT COMPLETE ?
TTHOCM
GET A BYTE CHECK FOR NULLS
N
Y
NULL?

BUMP BYTE PTR AND DECR. COUNT
TTHOCM

CALL COMPLETION FUNCTION FOR TT: OUTPUT
TOON
E.5.6 Resident Device Handlers (TT, Message)
TT: RESIDENT HANDLER

ENTRY POINT

FIND IMPUPE AREA, THEN GET WORD CT.

(WRITE) 5$ < Ø

WORD COUNT ? = g

(SEEK) TTCMPL

(READ) > Ø

BLOCK # = Ø?

N

Y

TTOPT2

PRINT A '//' PROMPT CHAR.

TTTHIN

--- ENTER HERE WHEN END OF LINE IS DETECTED.

1$

LINE IN INPUT BUFFER ?

N

6$

Y

IGET

GET A CHARACTER FROM BUFFER

DOUBLE CTRL C ?

Y

$SRQABT

N

EOF (CTRL Z) ?

Y

3$

WRITE

N

5$

PASS CHAR TO USER, DECR. THE COUNT

SET UP POINTER ENABLE PRINTER INTERRUPT

A

EXIT

DONE ?

Y

35$

BUMP BUFFER POINTER

N

1$ --- GET AS MANY LINES AS POSSIBLE

2$

BUMP BUFFER POINTER

3$

CLEAR AN UNFILLED BYTE AND COUNT DOWN

DONE ?

N

2$

SET EOF FLAG

Y

TTCMPL

CLEAR IMPURE AREA PTRS, POINT TO QUEUE

35$

TTTCMPL

DO COMPLETION

5$

EXIT

COMPLT

A

EXIT

E-148
IGET

BUMP GET POINTER

END OF BUFFER?

Y

WRAP POINTER TO TOP

GET CHAR & DECR. COUNT

EOLTST

CHECK FOR $C, $Z, LF

END OF LINE?

Y

DECREMENT LINE COUNT. SET Z IF CTRL C

EXIT

EXIT
MESSAGE HANDLER

ENTRY POINT

GET JOB # OF NEXT ELEMENT

IS IT A WRITE?

N

CHANGE JOB # TO # OF RECEIVING JOB

ANY MESSAGE WAITING?

Y

MATCH WITH SDAT

Y

IS IT A RECEIVE DATA?

N

IS NEW ELEMENT A RCVD?

Y

-- IGNORE SEEKS

(WORD COUNT ≠ 0)

A SEEK?

N

TRANSFER DATA, INDICATE AMOUNT SENT.

COMPLT

RETURN ELEMENT IN QUEUE TO FREE LIST.

--- FREE THE OTHER ELEMENT, RETURN FROM COMPLT

10S

15S

IS WORD COUNT ≠ 0?

Y

IS THERE A SDAT?

N

SET UP TO TRANSFER DATA

10S

20S

25S

3S

22S

26S

27S

20S

22S

25S

26S

27S

EXIT

EXIT

ABORT ENTRY POINT

FIND END OF QUEUE AND LINK IN ELEMENT

CLEAR LAST LINK WORD

POINT TO A JOB’S MSG QUEUE

GET AN ELEMENT AND FREE IT

DONE?

Y

ANOTHER JOB?

N

EXIT
ENTRY POINT INDEX

Page numbers marked by an asterisk indicate the flowchart of the entry point.

$CRTNE, E-135*
$SENSYS, E-122*
$SINTEN, E-122*
$SQABT, E-141*, E-148
$ROSIG, E-126*, E-133
$ROTSW, E-126*
$SSYNCH, E-114*
$SYSWT, E-126*
ABORT, E-123, E-124, E-150*
ADTRAN, E-13*
ALT, E-89*, E-140*
AQLINK, E-134*
ASSIGN, E-20*
B, E-4, E-5*
BADCOM, E-4*, E-7, E-9
BEGIN, E-7*, E-9
BLCHK, E-56, E-57*

C$DFN, E-72*, E-112*
C$DFN2, E-66*, E-72
C$HAIN, E-75*, E-115*
C$LOS2, E-66*, E-69, E-105*, E-106
C$LOSE, E-69*, E-106*
C$MKT, E-119*
C$PYCH, E-110*
C$SIGN, E-66*, E-106*
C$SSIP, E-66*, E-106*
C$STAT, E-111*
CALUSR, E-82*, E-120*
CCBB0, E-15*
CDFN, E-30*
CHANGER, E-64, E-65*
CHKSP, E-113*
CHXIT, E-75, E-78*
CLOC, E-33, E-42*
CLSOSH, E-36, E-42*
CMARKT, E-116, E-119*
CMPLT2, E-134*
CNTXSW, E-127*
COMERR, E-59
COMPLT, E-100*, E-134*, E-148, E-150
COMXIT, E-30, E-33, E-34, E-35, E-37, E-38, E-40, E-41, E-43, E-54, E-59*
CONSOL, E-60*
CSL, E-46*
CTRL.B, E-141*
CTRL.C, E-141*
CTRL.F, E-141*
CTRL.O, E-142*
CTRL.Q, E-142*
CTRL.S, E-142*
CTRL.U, E-142*
CTRL.V, E-89*
CTRLQ, E-89*

CTRLS, E-89*
CTRLU, E-91*

D$DTAT, E-105*
D$DELETE, E-66*, E-105*
D$STAT, E-66*
D$VICE, E-113
D, E-4, E-6*
DATE, E-18*
DELETE, E-36*
DELOUT, E-34*, E-36, E-38, E-42, E-43
DEQUSTR, E-117, E-132*
DLEET, E-56*
DLUSR, E-126*, E-132
DOSAVE, E-19

E$INTER, E-66*, E-105*
E$XIT, E-78*, E-115
E, E-4, E-5*
E376, E-64, E-81*, E-104
E5ERO, E-111*
E5ERRI, E-111*
ECHO, E-143*
ECHOCC, E-143*
ECHORO, E-143*
EMTCOM, E-104*
EMTER0, E-107*, E-112, E-113, E-120
EMTXT, E-64, E-65*, E-68, E-81
EMTPRO, E-66*, E-104*
EMTXUSR, E-105*, E-106
ENQUIR, E-132*
ENTER, E-37*
ENTPRG, E-7, E-69*, E-117*
ENTRY, E-59*
EOLTST, E-92*
ERRCOM, E-129*
EXINT, E-122*, E-138
EXSWAP, E-122, E-123*
EXTEND, E-38, E-39*
EXUSER, E-122*, E-123, E-124, E-138

FSGETCH, E-66*, E-105*
FATAL, E-29*
FILE, E-14*
FOON, E-144*, E-146
FRUN, E-24*

E-151
G$TIM, E-73*, E-114*
G$TJB, E-73*, E-111*
GESTAT, E-40*
GET, E-4, E-8*
GETBLK, E-23*
GETFD, E-53*
GETNAM, E-53*
GT, E-25*

H$ERR, E-74*
H$RSET, E-66*, E-116*
HDRSET, E-35*

IGET, E-149*
INCR, E-59
INIT, E-4*
IORSET, E-116*

KLOSE, E-42*

L$OCK, E-75*
L$OOK, E-66*, E-105*
LK4DEV, E-61*
LKERL, E-34*, E-36
LKNIT, E-138*
LNFFILE, E-33, E-34*
LOAD, E-21*
LOOKUP, E-33*

M$RKT, E-112*
MSWAIT, E-111*
MARKTM, E-113*
MEXIT, E-4*, E-79
MEXIT2, E-4*, E-78
MONOUT, E-49, E-75*, E-120*

NFREAD, E-70*, E-71
NFWRIT, E-71*, E-109
NOSWIT, E-48*
NXBLK, E-56*
NXTFIL, E-47*, E-48

OBUMP, E-145*, E-146
OCKCR, E-145*, E-146
OFILL, E-144*
OPRINT, E-5*, E-24
OPUT, E-96*
OUSTUF, E-52*
OVLINK, E-12*
OVREAD, E-12*

P$PRINT, E-76*, E-105*
P$ROTE, E-110*

PSURGE, E-74*, E-106
PHEATCH, E-40*
PUTBLK, E-23*
Q$SET, E-66*, E-105*
QFULL, E-133*
QMAGR, E-98*, E-133*
QRESET, E-117*
QSET, E-43*
QUIESC, E-117*
R$CTLO, E-105*
R$CVD, E-108*
R$READ, E-70*, E-74, E-108
R$NAME, E-66*, E-105*
R$OPEN, E-69*, E-111*
R$SUME, E-112*
R, E-4, E-7*
RDOVLY, E-7, E-69*, E-117*
REENTR, E-4, E-9*
RENAME, E-33*
RENT, E-37*, E-39
RESUM, E-112*
RESUME, E-22*
REVERT, E-117*
RIDUSR, E-132*
RSTSR, E-35*
RTORG, E-29*
RUB, E-91*, E-142
RUBCM2, E-92*
RUBCOM, E-92*
RUN, E-4, E-9*
RXNCT, E-108*, E-109
RXWTE0, E-108*

S$AVST, E-68*, E-111*
S$DAT, E-109*
S$ERR, E-74*
S$STOP, E-77*, E-118*
S$FP, E-73*
S$FPF, E-112*
S$FFUN, E-74*, E-108*
S$SET, E-66*
S$SPND, E-112*
SSSET, E-117*
S$SWAP, E-112*
S$STRAP, E-113*
SAVE, E-19*
SAVEVC, E-6, E-14*
SEGRI, E-58*
SEGRI1, E-58*
SEGRI2, E-58*
SOFRST, E-35*
SPDEL, E-34*, E-36
SPECIAL, E-47, E-48*
SPENTR, E-37, E-38*
SPESH, E-59
SPLOOK, E-33, E-34*
SPREAD, E-108*
STARTK, E-4, E-9*
STRE, E-9*
STRTIN, E-46*, E-48
SUSPEND, E-22*
SWAPME, E-126*
SWITCH, E-47, E-48*
SYNNRR, E-47
SYSK, E-16*

T$LOCK, E-120*
T$RPST, E-68*
T$TIN, E-79*, E-107*
T$TOUT, E-80*, E-107*
T$WAIT, E-112*
TCHKSP, E-93, E-94*
TIME, E-18*
TIMER, E-122, E-138*
TOOBIG, E-64, E-65*
TPRNT7, E-93*, E-94
TRAP10, E-129*
TRAP4, E-129*
TSWCNT, E-109*
TTCMPL, E-148*
TTIN, E-148*
TTIOCM, E-146*
TTIOUT, E-145, E-146*
TTIBUM, E-91*

TTIDSP, E-140*, E-142
TTIXZ, E-88*, E-91
TTINT, E-86*, E-140*
TTINC3, E-87*, E-88, E-89, E-90, E-140*, E-141
TTODON, E-93, E-94*
TTOENB, E-141*, E-143
TTOINT, E-93*, E-144*
TTOPT2, E-143*
TTOPT3, E-143*
TTOPT4, E-141, E-143*
TTOPUT, E-95*
TTORUB, E-95*
TTPOXT, E-93, E-94*

USNLOK, E-75*, E-120*
UABORT, E-115, E-124*, E-129, E-132
UNBLOK, E-126*
UNLOAD, E-22*
USR, E-32*
USRBUF, E-29*
USRCOM, E-54*
USRNF, E-55*
USWAP0, E-115, E-133*

WSAIT, E-72*, E-111
WSRITE, E-71*, E-109
INDEX

Abbreviations, 1-2
Abort,
code, 5-8
entry point, 6-4
Absolute section, 3-21
Adding a handler to system, 5-11
Allocating space for F/G, 2-4
ANSI MT labels under RT-11, 3-12
ASSIGN command, 2-15

Backslash character (\) (BATCH), 7-1
Bad tape errors, 3-13
BATCH, 7-1
compiler, 7-4, 7-6
CTL format, 7-1
CTT temporary files, 7-22
directives, 7-1, 7-2
documentation, 7-1
example, 7-11
job termination, 7-6
language, 7-1
run-time handler, 7-2
BATSWI switches, 7-3
B/G (definition), 1-3
Bitmap byte table, 2-21
BLIMIT table, 2-4
Block,
ENDGSD, 3-19
GSD, 3-19
Blocking a job, 6-3
Blocks, data, 3-19
Bootstrap, 2-24
operation, 4-3
system, 5-15
Building a new system, 5-16

$SCALL command (BATCH), 7-12
Carriage width, 2-26
Cassette,
file header, 3-15
file structure, 3-14
Channel number, 3-4
Channel status word, 1-2
Checksum byte, 3-18
Clock interrupt service,
flowcharts, E-83, E-137
Command field, 3-24
Command string interpreter (CSI), see CSI
Compatibility between RT-11
versions, C-1
Compiler (BATCH), 7-4
construction, 7-6
temporary files, 7-22
Completion queue elements, 5-5, 5-6

Console terminal,
interrupt service (flowcharts), E-85, E-139
substitution, 2-23
Constant field, 3-24
Context switch, 6-2, 6-3
Converting user-written handlers,
requirements, 5-23
$COPY command (BATCH), 7-11
<CR> (definition), 1-3
$CREATE command (BATCH), 7-11
CR11 device handler, A-35
CSECTS, 3-21
CSI (Command String Interpreter),
definition, 1-2
flowcharts, E-45
requests, 6-5
subroutines (flowcharts), E-51
$CSW (definition), 1-2
CT file header format, 3-16
CTRL file (BATCH), 7-1, 7-22
CTT file (BATCH), 7-22
Current queue element, 5-3

Data base description (BATCH), 7-7, 7-10
Data blocks, 3-19
Data transfer operations, 5-19
by device handlers, 5-9
Date, 3-5
Definitions, 1-2
$DELETE command (BATCH), 7-11
determining user program memory, 2-2
Device directory, 2-15, 3-1
Device Directory Size table,
$DVSI, 2-15
Device driver, 5-16
Device Handler Block table,
$DVREC, 2-14
Device handlers, 5-1, 5-14
CR11, A-35
format, 5-8
LP/LS11, A-28
RC11/RS64, A-2
TC11, A-47
Device handlers changed by SET
command, 5-21
Device identifier, 2-13
DEVICE macro, 2-16
Device Ownership table, $OWNER,
2-16
Device Status table, $STAT, 2-13
devices with special directories,
5-19
Differences between V1 and V2/
V2B EMTs, C-1

Index-1
Hardware mode, 3-15
Header block, 3-1
  format, library, 3-29
  words, device handler, 5-8
High limit pointer, 2-5
$HSIZE (Handler Size table), 2-14
HSIZE macro, 2-17

IF conditional branch (BATCH), 7-12
Impure area, 2-3, 2-18, 2-19, 2-21
Initiating a BATCH job, 7-4
.INTEN request, 6-1
Internal relocation, 3-24
Internal displaced relocation, 3-25
Interrupt,
  code restrictions, 5-10
  handler, 5-9, 6-1
  vector tables, 5-13
  vectors, 2-23
I/O queue elements, 5-1, 5-2
I/O queuing system, 5-1
I/O routines, E-97
I/O termination, 6-5
I/O transfers, 5-1
ISD (Internal Symbol Directory), 3-28

Job,
  arbitration, error processing, E-121
  blocking, 6-3
  boundaries (F/B), 2-4
  ID area, 2-21
  initiation (BATCH), 7-4
  number, 3-4, 6-3
  priority, 5-5, 6-3
  scheduling, 6-3
  status, 2-19
  termination (BATCH), 7-6
Job Status Word, 1-3
$JOB command (BATCH), 7-11
JSW (definition), 1-3

KMON (definition), 1-2
KMON (Keyboard Monitor), 1-2
  flowcharts, E-3
  overlays, 4-3, E-17
  subroutines, E-11

Label,
  first end-of-file, 3-12
  first file header, 3-12
  volume header, 3-12
Labels,
  BATCH, 7-12
  magtape, 3-12
Language processor, 3-16
  object modules, 3-19
Last queue element, 5-3
.LDA, formatted binary format, 3-30
<LF> (definition), 1-3
Library,
  end trailer, 3-30
  file format, 3-28
  header, 3-28
  object format, 3-28
Limit tables, 2-4
BLIMIT, 2-4
FLIMIT, 2-4
Linking BATCH, 7-2
Location counter, commands, 3-23
  definition, 3-27
  modification, 3-27
Low memory bitmap (LOWMAP), 2-21
LP/LSI1 device handler, A-28

Macro,
  DEVICE, 2-16
  HSIZE, 2-17
$MACRO command (BATCH), 7-11
Magtape
  compatibility, 3-13
  file structure, 3-11
Making SET TTY options permanent, 2-25
Mode, hardware, 3-15
  software, 3-15
Memory, 1-1, 2-1, 2-2
  allocation, 2-7, 2-8, 2-9
  areas, 2-9
Mnemonic names, 1-3
Monitor,
  description (F/B), 6-1
  device tables, 2-13
  fixed offsets, 2-9
  memory allocation, 2-7
  memory layout, 2-1
  operation, 1-1
MONITR.SYS contents, 4-2
.MRKT request, 5-7

Name field, 3-24
Names,
  and extensions, file, 3-4
  handler, 5-12
  mnemonic, 1-3
Negative relocation, 3-32
Object format (.OBJ), 3-16
Object module processing, 3-17
Offsets, fixed, 2-10-2-12
Operations,
  data transfer, 5-19
  directory, 5-19
  special, 5-19
Output (BATCH), 7-4
Overlay programs, 3-32
Overlay segment relocation block, 3-44
Overlays, 3-42
Overview, 1-1
$OWNER (Device Ownership table), 2-16
Ownership code, 2-16

Paper tape format, 3-30
Patch procedures, TTY options, 2-26
Permanent file, 3-4
Permanent name table ($PNAME), 2-13
$PNAME (Permanent Name table), 2-13
Positive relocation, 3-32
$PRINT command (BATCH), 7-11
Priority level of handlers, 6-1
Program sections, 3-21
Programmed requests, 5-20
  to special devices, 5-20
V1, C-1
Public device, 2-16

Queue element for a special handler, 5-20
Queue elements,
  completion, 5-5
  timer, 5-7
Queue header, handler, 5-3
Queue manager, 5-5
  flowcharts, E-131
Queue structures, 5-5
Queued I/O, 5-1

.REL50 conversions, 5-12
RC11/RS64,
  bootstrap, A-9
  device handler, A-2
REL file,
  without overlays, 3-33
  with overlays, 3-42, 3-43
Relocatable format (.REL), 3-32
Relocation, 3-32
  codes, 3-24, 3-25, 3-26
  global, 3-25, 3-32
  global additive, 3-26, 3-33
  global additive displaced, 3-26, 3-33
  global displaced, 3-26, 3-33
Relocation, (cont.)
  information, 3-34
  internal, 3-24, 3-32
  internal displaced, 3-25, 3-33
Resident device handlers
  (flowcharts), E-147
Resident monitor, 1-2
RLD,
  block, 3-22
  commands, 3-23
  format, 3-24
RMON (definition), 1-2
RMON (Resident Monitor) flowcharts,
  for F/B Monitor, E-101
  for S/J Monitor, E-63
Root relocation, 3-44
RT-ll file formats, see file formats, RT-l1

Sample handler listings, A-1
Save image format (.SAV), 3-31
Scheduling, job, 6-3
Segments of directories, 3-8
Sentinel file, 3-15
SET command, 5-21
SET command options, 5-22
  conventions for adding, 5-22
Set program limits, 3-27
S/J (definition), 1-3
S/J Monitor,
  flowcharts, E-1
  restrictions, 2-22
Software mode, 3-15
Special,
  devices, 5-19
  operations, 5-19
Square brackets, [ and ], 1-3
Stack, 6-2
  information, 6-2
  location, 2-3
  pointer, 6-2
$STAT (Device Status table), 2-13
Status,
  buffer, 2-23
  register, 2-23
  word, 2-13, 3-3
Symbolic names, 1-3
.SYNCH element, 5-7
.SYNCH request, 5-6, 6-3, 6-4
SYSLOW,
  examples (background), 2-6
  pointer, 2-4
System,
  bootstrap, 5-15
  communication locations, 3-34
  components, 4-1
  configuration word, 2-11
  date word, 3-5
  size, 4-5, 4-6

Index-4
System device handler, requirements, 5-14
writing a, 5-14
System device structure, 4-1

Tables, monitor device, 2-13
TC11 device handler, A-47
Temporary files, BATCH compiler, 7-22
Tentative file, 3-3
Terminating a BATCH job, 7-6
Terminology, 1-2
Timer queue element, 5-7
Trailer, library file, 3-30
Transfer address GSD item, 3-21
TTCNFG option bits, 2-27
TTY options, 2-25
.TWAIT request, 5-7
TXT block format, 3-22

$UNAM1, $UNAM2 (User Name tables), 2-15
Underlining, 1-3
User Name tables, $UNAM1, $UNAM2, 2-15
User service routines, 1-2
User-written handlers, 5-23
Using auxiliary terminals as console terminal, 2-23
USR (User Service Routines),
contention, 6-5
definition, 1-2
flowcharts, E-27
ownership, 6-5
permanently resident, 2-7
queuing mechanism, 6-5
swapping, 2-2, 2-6

Variables, BATCH, 7-13
Vector protection, 2-21, 2-22
Version 1 EMT summary, C-1
Volume-header label, 3-12

Writing a system device handler, 5-14

Index-5
HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes software newsletters for the various DIGITAL products. Newsletters are published monthly, and keep the user informed about customer software problems and solutions, new software products, documentation corrections, as well as programming notes and techniques.

There are two similar levels of service:

. The Software Dispatch
. The Digital Software News

The Software Dispatch is part of the Software Maintenance Service. This service applies to the following software products:

PDP-9/15
RSX-11D
DOS/BATCH
RSTS-E
DECsystem-10

A Digital Software News for the PDP-11 and a Digital Software News for the PDP-8/12 are available to any customer who has purchased PDP-11 or PDP-8/12 software.

A collection of existing problems and solutions for a given software system is published periodically. A customer receives this publication with his initial software kit with the delivery of his system. This collection would be either a Software Dispatch Review or Software Performance Summary depending on the system ordered.

A mailing list of users who receive software newsletters is also maintained by Software Communications. Users must sign-up for the newsletter they desire. This can be done by either completing the form supplied with the Review or Summary or by writing to:

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

SOFTWARE PROBLEMS

Questions or problems relating to DIGITAL's software should be reported as follows:

North and South American Submitters:
Upon completion of Software Performance Report (SPR) form remove last copy and send remainder to:

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

The acknowledgement copy will be returned along with a blank SPR form upon receipt. The acknowledgement will contain a DIGITAL assigned SPR number. The SPR number or the preprinted number should be referenced in any future correspondence. Additional SPR forms may be obtained from the above address.

All International Submitters:
Upon completion of the SPR form, reserve the last copy and send the remainder to the SPR Center in the nearest DIGITAL office. SPR forms are also available from our SPR Centers.

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 Computers Users Society, maintains a user exchange center for user-written programs and technical application information. The Library contains approximately 1,900 programs for all DIGITAL computer lines. Executive routines, editors, debuggers, special functions, games, maintenance and various other classes of programs are available.

DECUS Program Library Catalogs are routinely updated and contain lists and abstracts of all programs according to computer line:

- PDP-8, FOCAL-8, BASIC-8, PDP-12
- PDP-7/9, 9, 15
- PDP-11, RSTS-11
- PDP-6/10, 10

Forms and information on acquiring and submitting programs to the DECUS Library may be obtained from the DECUS office.

In addition to the catalogs, DECUS also publishes the following:

DECUSCOPE - The Society's technical newsletter, published bi-monthly, aimed at facilitating the interchange of technical information among users of DIGITAL computers and at disseminating news items concerning the Society. Circulation reached 19,000 in May, 1974.

PROCEEDINGS OF THE DIGITAL EQUIPMENT USERS SOCIETY - Contains technical papers presented at DECUS Symposia held twice a year in the United States, once a year in Europe, Australia, and Canada.

MINUTES OF THE DECSystem-10 SESSIONS - A report of the DECSystem-10 sessions held at the two United States DECSystem Symposiums.

COPY-N-Mail - A monthly mailed communiqué among DECsystem-10 users.

LUG/SIG - Mailing of Local User Group (LUG) and Special Interest Group (SIG) communiqué, aimed at providing closer communication among users of a specific product or application.

Further information on the DECUS Library, publications, and other DECUS activities is available from the DECUS offices listed below:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS EUROPE
Digital Equipment Corp. International (Europe)
P.O. Box 340
1211 Geneva 26
Switzerland
READER'S COMMENTS

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. ☐