RT-11
Software Support Manual

Order No. DEC-11-ORPGA-B-D, DN1

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 document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

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

Copyright © 1973, 1975, 1976 by Digital Equipment Corporation

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL    DECsystem-10    MASSBUS
DEC    DECTape    OMNIBUS
PDP    DIBOL    OS/8
DECUS    EDUSYSTEM    PHA
UNIBUS    FLIP CHIP    RSTS
COMPUTER LABS    FOCAL    RSX
COMTEX    INDAC    TYPESET-8
DDT    LAB-8    TYPESET-10
DECCOMM    DECSYSTEM-20    TYPESET-11
CONTENTS

CHAPTER 1  RT-11 OVERVIEW
1.1 INTRODUCTION  1-1
1.2 SYSTEM CONCEPTS AND TERMINOLOGY  1-1

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

2.7 MAKING TTY SET OPTIONS PERMANENT IN F/B MONITOR
2.7.1 Carriage Width  2-25
2.7.2 Other Options  2-26

CHAPTER 3  FILE STRUCTURES AND FILE FORMATS  3-1
3.1 DEVICE DIRECTORY SEGMENTS  3-1
3.1.1 Directory Header Format  3-1
3.1.2 Directory Entry Format  3-2
3.1.2.1 Status Word  3-3
3.1.2.2 Name and Extension  3-4
3.1.2.3 Total File Length  3-4
3.1.2.4 Job Number and Channel Number  3-4
3.1.2.5 Date  3-5
3.1.2.6 Extra Words  3-7
3.2 SIZE AND NUMBER OF FILES  3-7
3.2.1 Directory Segment Extensions  3-8
5.6  ADDING A SET OPTION  5-21
5.7  CONVERTING USER-WRITTEN HANDLERS  5-23

CHAPTER 6  F/B MONITOR DESCRIPTION  6-1
6.1  INTERRUPT MECHANISM AND .INTEN ACTION  6-1
6.2  CONTEXT SWITCH  6-2
6.3  BLOCKING A JOB  6-3
6.4  JOB SCHEDULING AND USE OF .SYNCH REQUEST  6-3
6.5  USR CONTENTION  6-5
6.6  I/O TERMINATION  6-5

CHAPTER 7  RT-11 BATCH  7-1
7.1  CTL FORMAT  7-1
7.2  BATCH RUN-TIME HANDLER  7-2
7.3  BATCH COMPILER  7-4
7.3.1  BATCH Job Initiation  7-4
7.3.2  BATCH Job Termination  7-6
7.3.3  BATCH Compiler Construction  7-6
7.4  BATCH EXAMPLE  7-11
7.5  CTT TEMPORARY FILES  7-22

APPENDIX A  SAMPLE HANDLER LISTINGS  A-1
A.1  RC11/RS64 DEVICE HANDLER  A-2
A.2  RC11/RS64 BOOTSTRAP  A-9
A.3  LP/LS11 DEVICE HANDLER  A-28
A.4  CR11 DEVICE HANDLER  A-34
A.5  TC11 DEVICE HANDLER  A-47

APPENDIX B  FOREGROUND TERMINAL HANDLER  B-1

APPENDIX C  VERSION 1 EMT SUMMARY  C-1

APPENDIX D  FOREGROUND SPOOLER EXAMPLE  D-1

January 1976
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.1</td>
<td>KM0N (KEYBOARD MONITOR) FLOWCHARTS</td>
<td>E-3</td>
</tr>
<tr>
<td>E.1.1</td>
<td>KM0N Subroutines</td>
<td>E-11</td>
</tr>
<tr>
<td>E.1.2</td>
<td>KM0N Overlays</td>
<td>E-17</td>
</tr>
<tr>
<td>E.2</td>
<td>USR (USER SERVICE ROUTINES) FLOWCHARTS</td>
<td>E-27</td>
</tr>
<tr>
<td>E.3</td>
<td>CSI (COMMAND STRING INTERPRETER) FLOWCHARTS</td>
<td>E-45</td>
</tr>
<tr>
<td>E.3.1</td>
<td>CSI Subroutines</td>
<td>E-51</td>
</tr>
<tr>
<td>E.4</td>
<td>RM0N (RESIDENT MONITOR) FLOWCHARTS FOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SINGLE-JOB MONITOR</td>
<td></td>
</tr>
<tr>
<td>E.4.1</td>
<td>EMT Processors</td>
<td>E-63</td>
</tr>
<tr>
<td>E.4.2</td>
<td>Clock Interrupt Service</td>
<td>E-67</td>
</tr>
<tr>
<td>E.4.3</td>
<td>Console Terminal Interrupt Service</td>
<td>E-83</td>
</tr>
<tr>
<td>E.4.4</td>
<td>I/O Routines</td>
<td>E-85</td>
</tr>
<tr>
<td>E.5</td>
<td>RM0N (RESIDENT MONITOR) FLOWCHARTS FOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FOREGROUND/BACKGROUND MONITOR</td>
<td></td>
</tr>
<tr>
<td>E.5.1</td>
<td>EMT Processors</td>
<td>E-101</td>
</tr>
<tr>
<td>E.5.2</td>
<td>Job Arbitration, Error Processing</td>
<td>E-103</td>
</tr>
<tr>
<td>E.5.3</td>
<td>Queue Managers (I/O, USR, Completion)</td>
<td>E-121</td>
</tr>
<tr>
<td>E.5.4</td>
<td>Clock Interrupt Service</td>
<td>E-131</td>
</tr>
<tr>
<td>E.5.5</td>
<td>Console Terminal Interrupt Service</td>
<td>E-137</td>
</tr>
<tr>
<td>E.5.6</td>
<td>Resident Device Handlers (TT, Message)</td>
<td>E-139</td>
</tr>
<tr>
<td></td>
<td>Entry Point Index</td>
<td>E-147</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E-151</td>
</tr>
</tbody>
</table>
## 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-ll</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>

vii
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 and V02C 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 V02B and V02C and between V2 and V02B systems is included in RT-11 System Release Notes (V02C), (DEC-11-ORNRA-A-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 (.SAVSTATUS) of the RT-11 System Reference Manual.
<table>
<thead>
<tr>
<th>TERM</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSW</td>
<td>Job Status Word</td>
</tr>
<tr>
<td></td>
<td>The JSW contains information in bytes 44 and 45 about the job currently in memory.</td>
</tr>
<tr>
<td>F/B</td>
<td>The Foreground/Background version of the monitor</td>
</tr>
<tr>
<td>S/J</td>
<td>The Single-Job version of the monitor</td>
</tr>
<tr>
<td>B/G</td>
<td>The background job</td>
</tr>
<tr>
<td>F/G</td>
<td>The foreground job</td>
</tr>
<tr>
<td>&lt;CR&gt;</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>&lt;LF&gt;</td>
<td>Line Feed</td>
</tr>
</tbody>
</table>

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 Table 2 of RT-ll System Release Notes.
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.

![Diagram of Memory Layout]

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 database.)</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 FRUN /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.
The limit pointers for a foreground job are fixed once the job has been loaded into memory. A program that requires working space and uses a .SETTOP will fail if the space is not allocated with the /N switch (a FORTRAN program is a typical case; see Appendix G, Section G.1, of the RT-11 System Reference Manual). The high limit pointer (SYSLOW) for the background, however, is not fixed and will change as space is allocated for LOADed handlers, the text scroller, and foreground jobs. In addition, if the USR is made permanently resident (using the SET USR NOSWAP command), SYSLOW (BLIMIT HIGH) will again change. This is shown in Figure 2-4.
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

LIST HEAD

CORPTR:

∅ ← NEXT BLOCK

(One free block in 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:

\[
\text{OFFSET} = \text{the byte offset to the word desired}
\]

\[
\text{RMON} = 54
\]

\[
\text{MOV} \ @\text{RMON}, Rn \quad \text{;ANY GENERAL REGISTER}
\]

\[
\text{MOV} \ \text{OFFSET}(Rn), Rn
\]
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 2-1
Fixed Offsets

<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>4</td>
<td>Serves as a link to interrupt entry code.</td>
</tr>
<tr>
<td>4</td>
<td>$CSW</td>
<td>160₁₀</td>
<td>Default I/O channels for the background (16₁₀ @ 5 words each).</td>
</tr>
<tr>
<td>244 164</td>
<td>$SYSCH</td>
<td>10₁₀</td>
<td>Internal I/O channel used for system functions.</td>
</tr>
<tr>
<td>256 174</td>
<td>BLKEY</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>CHKEY</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>$DATE</td>
<td>2</td>
<td>Current date value. (The format is shown in Chapter 3, section 3.1.2.5.)</td>
</tr>
<tr>
<td>264 180</td>
<td>DFLG</td>
<td>2</td>
<td>&quot;Directory operation in progress&quot; flag. Used to inhibit ^C from aborting a job until directory operation is finished.</td>
</tr>
<tr>
<td>266 182</td>
<td>$USRLC</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)</th>
<th>Tag</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octal Decimal</td>
<td></td>
<td></td>
<td></td>
</tr>
<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 Versions 2, 2B, and 2C).</td>
</tr>
<tr>
<td>277 191</td>
<td>SYSUPD</td>
<td>1</td>
<td>Version release number (1 for V02, 2 for V02B, etc.)</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>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Bit #</strong></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0 → S/J Monitor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1 → F/B Monitor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1 → VT11 hardware exists</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1 → RT-11 BATCH controls the background</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0 → 60-cycle KG11L clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 → 50-cycle clock</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1 → 11/45 FPP present</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0 → No foreground job present</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 → Foreground job is in memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1 → User is linked to VT11 scroller</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1 → USR is resident via SET USR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0 → No PDP-11/03 processor</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 → PDP-11/03 processor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1 → KG11 clock is present (always set if bit 11 is 1)</td>
<td></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) Octal</th>
<th>Tag</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>310 200</td>
<td>TTPS</td>
<td>2</td>
<td>Address of console printer status.</td>
</tr>
<tr>
<td>312 202</td>
<td>TTPB</td>
<td>2</td>
<td>Address of console printer buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(See Section 2.6, Using Auxiliary Terminals as the Console Terminal.)</td>
</tr>
<tr>
<td>314 204</td>
<td>MAXBLK</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>316 206</td>
<td>E16LST</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>320 208</td>
<td>CNTXT</td>
<td>2</td>
<td>Pointer to the impure area for the current executing job.</td>
</tr>
<tr>
<td></td>
<td>(F/B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>322 210</td>
<td>JOBNUM</td>
<td>2</td>
<td>Executing job's number (0 = B/G, 2 = F/G).</td>
</tr>
<tr>
<td>320 208</td>
<td>$TIME</td>
<td>4</td>
<td>Two words of time of day in the S/J Monitor.</td>
</tr>
<tr>
<td></td>
<td>(S/J)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>324 212</td>
<td>SYNCH</td>
<td>2</td>
<td>Address of monitor routine to handle .SYNCH request.</td>
</tr>
<tr>
<td>326 214</td>
<td>LOWMAP</td>
<td>20</td>
<td>Start of low memory protection map. (This map protects vectors at locations 0-476.)</td>
</tr>
<tr>
<td>352 234</td>
<td>USRLOC</td>
<td>2</td>
<td>Pointer to current entry point of USR.</td>
</tr>
<tr>
<td>354 236</td>
<td>GTVECT</td>
<td>2</td>
<td>Pointer to VT11 vector. The vector is initially positioned at 320.</td>
</tr>
<tr>
<td>356 238</td>
<td>ERRCNT</td>
<td>1</td>
<td>Error count byte (for future use by system programs).</td>
</tr>
<tr>
<td>357 239</td>
<td>FUTURE</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
$HSIZE
$DVSIZ
$UNAM1,$UNAM2
$OWNER

The size of these tables is fixed and is governed by the $SLOT assignment; the default value is $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 = RF11 Disk
13 = TAll 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 Diskette

Odd byte - Bit flags with the following meanings:

Bit 15: 1 = Random-access device (disk, DECTape)
        0 = Sequential-access device (line printer, papertape, card reader, magtape, cassette, terminal)
Bit 14: 1 = Read-only device (card reader, papertape reader)
Bit 13: 1 = Write-only device (line printer, papertape punch)
Bit 12: 1 = NonRT-11 directory-structured device (magentape, cassette)
Bit 11: 1 = Enter handler abort entry every time a job is aborted.
        0 = Handler abort entry taken only if there is an active queue element belonging to aborted job.
Bit 10: 1 = Handler accepts .SPFUN requests (e.g., MT, CT, DX).
        0 = .SPFUN requests are rejected as illegal.
Bits 9-8: Reserved

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 $HSIZE (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.

2-14 January 1976
2.5.2.6 SDVSIZ (Device Directory Size Table) - Entries in this table are non-zero for file-structured devices only and reflect the number of 256\textsubscript{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>11300\textsubscript{8}</td>
<td>RP02</td>
<td>116300\textsubscript{8}</td>
</tr>
<tr>
<td>TC11</td>
<td>1102\textsubscript{8}</td>
<td>RJS03</td>
<td>2000\textsubscript{8}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RJS04</td>
<td>4000\textsubscript{8}</td>
</tr>
<tr>
<td>RF11</td>
<td>2000\textsubscript{8} (1 platter)</td>
<td>RX01</td>
<td>752\textsubscript{8}</td>
</tr>
<tr>
<td></td>
<td>4000\textsubscript{8} (2 platters)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6000\textsubscript{8} (3 platters)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10000\textsubscript{8} (4 platters)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The default for RF11 and RJS03/4 is one platter, or 2000\textsubscript{8} blocks. It is possible to alter the system to indicate the correct number of platters. Instructions are in Chapter 4 of the RT-11 System Generation Manual, (DEC-11-ORGMA-A-D).

2.5.2.7 $UNAML, $UNAM2 (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):

FILST$ = 100000  Random-access device (disk, DECtape)
RONLY$ = 40000   Read-only device
WONLY$ = 20000   Write-only device
SPECLS$ = 10000  Non RT-11 directory-structured device (including MT and CT)
HNDLR$ = 4000    Handler abort entry
SPFUN$ = 2000    Special function requests

ENTRY - the 2-character device name with the 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:

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 BKGND), the second is the foreground's pointer. The $IMPUR table is accessed by using IMPLOC, located at an offset of 422 into RMON. IMPLOC points beyond the end of $IMPUR to $IMPUR +4 to facilitate accessing the $IMPUR table from the top down in order of decreasing priority.

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.

A foreground program can determine whether the KMON is resident by testing KMONIN, located at an offset of 424 into RMON. KMONIN is non-zero if the KMON is resident and zero if a background job is running. In addition, the file name of the running foreground or background job is located in the job's impure area at offset I.NAME (376). Note that for a background job, KMONIN must first be tested to determine whether the name belongs to an active job since the file descriptor is not cleared when KMON is entered.

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-11 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 20,10 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 2-3
Bitmap Byte Table

<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>260-277</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>
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:

$$\text{MOV } @\#54, R0$$
$$\text{BISB } \#(B1110000), 342(R0)$$
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-11 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-11 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 Table 2 of RT-11 System Release Notes (V02C). The patching procedure is:

```
 patch name--
 *monitr.sys/m<cr>
 *base;0<cr>
 *65/ vectin<lf>
 62/ statin<lf>
 64/ vectout<lf>
 66/ statout<cr>
 *300/ nnnnn 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
```

[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 Table 2 of RT-ll System Release Notes (V02C). 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 21410. See Table 2 of RT-ll System Release Notes (V02C) for the exact locations of BASE and TTWIDT.

```
*R PATCH <CR>

PATCH Version number

FILE NAME--
*MONITR.SYS/M<CR>
*BASE;ØR<CR>
*Ø,21410\ 110 2Ø4<CR>  [The /M is necessary; set
*E

relocation registers; open
with backslash]
```

In this example, the width is changed from 72\textsubscript{10} to 132\textsubscript{10} (204\textsubscript{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 Table 2 of RT-11 System Release Notes is:

```
.R PATCH <CR>

PATCH Version number
FILE NAME--
*MONITR.SYS/M<CR>
^BASE;OR<CR>
^G,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; &R<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 directory-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 3-2
File Types

<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.)

<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.
3.1.2.5 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></td>
<td>MONTH (1-12.)</td>
<td>DAY (1-31.)</td>
<td>YEAR-110(8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNUSED</td>
<td></td>
<td></td>
<td></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
RAD₅₆ FOR "MON"
RAD₅₆ FOR "ITR"
RAD₅₆ 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 (1₈₈₈) BLOCKS LONG

PERMANENT
RAD₅₆ FOR "PIP"

RAD₅₆ FOR "MAC"

FILE IS 9₁₀ (1₁₈) BLOCKS LONG

NO CREATION DATE
TENTATIVE FILE ON CHANNEL 1
RAD₅₆ FOR "PIP"

RAD₅₆ FOR "MAC"

JOB #, CHANNEL #

EVERY TENTATIVE MUST BE FOLLOWED BY
AN EMPTY ENTRY

FILE IS 52₈₁₈ (1₂₂₈₈) 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-ll 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-ll 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:

\[n\]

This represents a directory segment with some number of directory entries. \(n\) is the segment number.

\[n\]

This represents a segment which is full, i.e., no more entries will fit in the segment.

Systems start out with entries entered into segment 1:

\[1\]

As entries are added, segment 1 fills:

\[1\]

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:

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:

```
1  |  In this case, segment 2 overflows into
Link to 2

2  |  segment 4 and the links are used to link
Link to 4

3  |  logical pieces rather than physical pieces.
Link to 3

4  |
```

3-10
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, Versions 2B and 2C. The structure is slightly different from that of Version 2. However, RT-11 V02B and V02C can read magtapes written under Version 2.

RT-11 magtapes use a subset of the VOL1, HDR1, and EOF1 ANSI standard labels. Each magtape file has the format:

```
HDR1*---data---*EOF1*
```

where each asterisk represents a tape mark.

A volume containing a single file has the following format:

```
VOL1 HDR1*---data---*EOF1**
```

A volume containing two files has the following format:

```
VOL1 HDR1*---data---*EOF1*HDR1*---data---*EOF1**
```

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

A zeroed magtape has the following format:

```
VOL1**
```

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 '1', the byte contains the ASCII code for '1'). Table 3-3 shows the contents of the first 80 bytes in the three labels. Note that VOL1, HDR1, 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>

First End-of-File Label (EOF1)

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).
3.3.1.1 Bootable Magtape File Structure - An RT-11 bootable magtape is a multi-file volume that has the following format:

VOLL BOOT HDR1*---data---*EOF1**

where BOOT is a 256-word physical block containing the magtape boot-strap loaders.

The format of the bootable magtape is not standard, because of the BOOT block, but other systems that will skip the BOOT block to HDR1 will be able to read RT-11 bootable magtapes if they can read regular RT-11 magtapes.
3.3.1.2 Moving MT to Other Industry-Compatible Environments - RT-11
VO2C 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 VO2B magtapes
once they have been mounted. RT-11 VO2C 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 VO2C magtapes may be read on IBM systems that support ANSI
standard label processing. RT-11 VO2C magtapes to be read by IBM
systems should consist of single file volumes (one file per magtape).
Important JCL parameters for reading RT-11 VO2C 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 6-character filenames and 3-character extensions; filenames less
than 6 characters should be enclosed in quotes.

3.3.1.3 Recovering From Bad Tape Errors - When a bad tape error
occurs on magtape, the magtape handler will retry the desired func-
tion, 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>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</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>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</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:

```
| Block Gap | Data Block | Block Gap | Clear Trailer |
```

or

```
| Block Gap | Data Block | Block Gap | Data Block | Clear Trailer |
```

Partially Written Block

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.

3-15
### 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 $128_{10}$ ($200_8$); 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 1</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 an .OBJ extension. Object modules are produced by language processors (such as MACRO or FORTRAN) and are processed by the Linker to become a runnable 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-ll file, on paper tape, or by some other means) and is arranged as illustrated in Figure 3-4.

<table>
<thead>
<tr>
<th>Byte containing octal value 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte containing octal value $$</td>
</tr>
<tr>
<td>Low-order byte of length</td>
</tr>
<tr>
<td>High-order byte of length</td>
</tr>
<tr>
<td>data bytes</td>
</tr>
<tr>
<td>Checksum byte</td>
</tr>
</tbody>
</table>

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 Library File Only</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:

![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 RLD 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 ASSIGN.

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

<table>
<thead>
<tr>
<th></th>
<th>Identification Word (TXT=3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Load Address of following data</td>
</tr>
<tr>
<td>4</td>
<td>Absolute Load Data (up to 38 bytes)</td>
</tr>
</tbody>
</table>

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>Global Name</td>
<td></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>Constant</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples:

a) ABS=17755$0
   TST ABS
   } both addresses cause internal displaced
b) CLR 17755$0
   relocation to occur

3-25
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>$/1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global Name</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
.GLOBL GLOBAL
MOV GLOBAL,R0
```

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>$/1</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Constant</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
.GLOBL GLOBAL
CMP #GLOBAL+6,R0
```

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>$/1</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Global Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Constant</td>
<td></td>
</tr>
</tbody>
</table>

Example:

```
.GLOBL 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.

```
  7
  ------------
  Program Section Name
  Constant
```

This command is generated whenever .ASECT or .CSECT 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.

```
  10
  Constant
```

Examples:

a) .=100 ;IF WE ARE IN THE ASEC
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.

```
  Relative Reference  11
```

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.

![Diagram of Library File Format]

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 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 - negative relocation only
   (MOV G,R) where the global is de-
   fined as absolute elsewhere

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

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

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 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Text</td>
</tr>
<tr>
<td>Relocation Information</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</td>
</tr>
<tr>
<td></td>
<td>overlays</td>
</tr>
<tr>
<td>56</td>
<td>REL file identification word, which must</td>
</tr>
<tr>
<td></td>
<td>contain the RAD50 value of the characters 'REL'</td>
</tr>
<tr>
<td>6Ø</td>
<td>Relative block number of relocation in-</td>
</tr>
</tbody>
</table>

3-33
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:

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELATIVE WORD OFFSET</td>
<td></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.
.TITLE  PTST
.MCALL ..V2,,REDEF,,LOOKUP,,READW,,QSET,,PRINT,,EXIT

..V2,,
..REDEF
.STI
.QSET  #01IRI,#7
.LOOKUP #AREA,#0,#PTR
.RCC  1S
.PRINT #LKFALI
.EXIT
.ISI
.READW #AREA,#0,#RUFF,#PS,#0
.RCC  PS
.PRINT #RNFALI
.EXIT
.2SI
.PRINT #OK
.EXIT

.OLISTI
.BLKW  #7
.AREA1
.BLKW  20,
.PTR1
.RADI0  /PR FILE12/
.BUFF1
.NLIST
.REPT  #PSA,
.WORD  0
.ENDR
.LIST
.LKFAIL1
.ASCIZ  /LOOKUP FAILED/
.RDFAIL1
.ASCIZ  /READW FAILED/
.OK1
.ASCIZ  /READW OK/
.EVEN
.NLIST
.REPT  <ST+1776>,#/2
.WORD  0
.ENDR
.NLIST
..END  8T

3-36
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.
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 REL 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-11 System Reference Manual.

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

```
| Block Ø  | REL Control Block |
|          | Overlay Handler and Tables |
| Root Segment Text |
| Overlay 1 Data |
|              |
| Overlay N Data |
|              |
|              | Root Relocation Information |
| -1          |
|              | End of Root Relocation Information |
|              |
|              | Overlay 1 Relocation Information |
| -2          |
|              | Overlay N Relocation Information |
|              | End of all Relocation Information |
```

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  | → 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 |
|   -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>0-1</td>
<td>Copy of system bootstrap (blocks 0 and 2 of the system device)</td>
<td></td>
</tr>
<tr>
<td>2-17</td>
<td>Swap blocks</td>
<td></td>
</tr>
<tr>
<td>18-24</td>
<td>KMON (includes 2-block KMON overlay area)</td>
<td></td>
</tr>
<tr>
<td>25-32</td>
<td>USR/CSI</td>
<td></td>
</tr>
<tr>
<td>33-47</td>
<td>RMON</td>
<td></td>
</tr>
<tr>
<td>48-57</td>
<td>KMON overlays</td>
<td></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.
1. User executes hardware bootstrap

   On all system devices except diskette, this causes block 0 of the
   system device to be read into 0-777. Control then passes to
   location 0. On diskette, causes logical block 0 to be read into
   0-777. Hardware bootstrap reads 64 words from track 1, sector 1.
   Control passes to location 0, where 64 words from each of sectors 3,
   5, and 7 (track 1) are read.

2. Second part of bootstrap is read

   The first part of the boot reads the second half into 1000-1777.
   On diskette, the first part of the boot reads logical block 2 (sectors
   9, 11, 13, 15) into 1000-1777.

3. Determine how much memory is available

   Boot sets a trap at location 4 and then starts addressing memory.
   When the trap is taken, illegal memory has been addressed.

4. Look for special devices

   Boot sets a trap at location 10 and then tries to address the clock,
   FPU, and VT11 display processor. Their presence or absence is indi-
   cated in the CONFIG word in RMON.
   (If a PDP-11/03 processor is present, the bootstrap assumes that a
   clock is present.)

5. Check memory size

   If memory is too small to read in the monitor, a message is printed
   and the boot halts.

6. Read in directory and find MONITR.SYS

   The entire directory is searched. If MONITR.SYS is not found, a HALT
   occurs after the boot prints an error message.

7. Read the monitor into memory

   The monitor file, MONITR.SYS, is read into the highest bank of
   memory.

8. Put pointers to monitor file blocks into RMON

   RMON references the monitor swap blocks directly. Thus, the posi-
   tion of the swap blocks varies as the placement of MONITR.SYS varies.
   The real position of the blocks is updated for each boot operation.

9. Update position-dependent areas in RMON

   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 RELST in
   BSTRAP.MAC.
10. Update processor-dependent area in RMON

If processor is a PDP-11/03, any PS references in the monitor are changed to use the MFPS and MTPS instructions.

11. LOOKUP the device handlers in system and store their record numbers in $DVREC

Boot looks at $SPNAME 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.

12. Print bootstrap header

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

13. Set up locations 0 and 2

Boot puts a "BIC R0,R0" in location zero and an .EXIT EMT in location 2.

14. Turn on KW11-L Clock

The bootstrap turns on the clock, if present in the configuration and processor is not a PDP-11/03.

15. 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/____4&&____0<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 Table 2 of RT-11 System Release Notes (V02C).
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 | Memory Size
---|---
4000 | 1K
10000 | 2K
20000 | 4K
40000 | 8K
100000 | 16K

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-11 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 PATCH<CR>

PATCH Version number
```

FILE NAME=

*MONITR.SYS/M<CR>

*BHALT/ 407 240<LF>

*BHALT+/ 13702 1202<LF>

*BHALT+/ 177570 xxxxxxx<CR>

*[NOP the branch at BHALT

Change MOV @SR,R2 to

MOV #VAL,R2. Address of

switch register is replaced

with one of the bit combina-

[Run RT-11 PATCH program.]

[tions described previously.]
For the patch addresses for other system devices, and for the address of BHALT, consult Table 2 of RT-11 System Release Notes (V02C).
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 (.READW/.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 by the S/J monitor for fast insertion of new elements into the queue.

Device Handler I/O Queue

LQE: Q1 (Pointer to last queue element)
CQE: Q1 (Pointer to current queue element)

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.

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QUEUE LINK</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CHANNEL STATUS WORD</td>
</tr>
<tr>
<td>12</td>
<td>CHANNEL OFFSET</td>
</tr>
<tr>
<td>14</td>
<td>COMPLETION ROUTINE ADDRESS</td>
</tr>
</tbody>
</table>

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 .MRRT 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.
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 $HSIZE 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. (See the RT-11 System Reference Manual, Section H.2, for further information.)
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, @$INPTR, 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-structured 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.)
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 H 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 occurred or not, the monitor I/O completion code must be entered to terminate activity and/or enter a completion routine. When return is made to the monitor, R4 must point to the fifth word of the handler (RCCQE 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 interface 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 $PNAME 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 R-T-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 ad-
dresses used in the example are for illustrative purposes only; con-
sult Table 2 of RT-11 System Release Notes (V02C) 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.

$DVISIZ: 2000

The disk has 1024 (decimal) 256-
word blocks for storage.

$PNAME: .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 de-
nvice 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 $PNAME 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.
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>$HSIZE</td>
<td>13624</td>
<td>14556</td>
</tr>
<tr>
<td>$DVSIZ</td>
<td>13660</td>
<td>14612</td>
</tr>
<tr>
<td>$PNAME</td>
<td>16470</td>
<td>17630</td>
</tr>
<tr>
<td>$STAT</td>
<td>16524</td>
<td>17664</td>
</tr>
</tbody>
</table>

NOTE

The addresses above are for illustration only. Consult Table 2 of RT-ll System Release Notes (V02C) for current table addresses and for the address of the monitor base location, BASE.

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--
MONITR.SYS/M<CR>
BASE;Ø<CR>

/[M is necessary; $HSIZE table;
Monitor base;
$DVSIZ table;
$PNAME table;
$STAT table;
Check that vectors in permanent map are protected;
Exit to monitor]
F/B Monitor

.R PATCH

PATCH Version number

FILE NAME--
.MONITR.SYS/M<CR>
.BASE;0R<CR>
0,14556/ 4000 316<CR>
0,14612/ 0 2000<CR>
0,17630/ 6250 70370<CR>
0,17664/ 4 100023<CR>
0,17336 77<CR>
E

[/M is necessary; Monitor base;
$HSIZE table;
$DVSIZ table;
$PNAME 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 DBCtape 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, RFSYS, 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 RCSIZE. 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:

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

   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 RC11/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). Define the conditional BF if an F/B bootstrap is desired. Let BF be undefined for an S/J bootstrap. For the S/J bootstrap:

   \[
   \text{
   .R MACRO<CR>} \\
   \text{\#RCBTJSJ=TT:,DK:BSTRAP<CR>} \\
   \text{\$RCSYS=1<CR>} \\
   \text{\Z\$RCSYS=1<CR>} \\
   \text{\ZERRORS DETECTED: g} \\
   \text{FREE CORE: 156\$8. WORDS}
   \]
For the F/B bootstrap:

```
.R MACRO<CR>
*RCFBFB=TT; DK: BSTRAP<CR>
^$RCSYS=1<CR>
BF=1<CR>
^Z$RCSYS=1<CR>
BF=1<CR>
^ZERRORS DETECTED: 0
FREE CORE: 15580 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 Chapter 5 of 
the M-11 System Generation 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
*RCMNFBSY.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 moni-
tor 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>$HSIZE</td>
<td>13674</td>
<td>14602</td>
</tr>
<tr>
<td>$DVISIZ</td>
<td>13730</td>
<td>14636</td>
</tr>
<tr>
<td>$PNNAME</td>
<td>16516</td>
<td>17640</td>
</tr>
<tr>
<td>$STAT</td>
<td>16552</td>
<td>17674</td>
</tr>
<tr>
<td>$ENTRY</td>
<td>16612</td>
<td>17612</td>
</tr>
</tbody>
</table>

5-17 January 1976
NOTE

These table addresses are for illustration only. Consult Table 2 of FT-ll System Release Notes (V02C) 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:

$SIZE = 316
$DSIZE = 2000
$PNAME = 70370
$STAT = 100023
$ENTRY = 56266 (F/B) 45056 (S/J)

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

.R PATCH<CR>

PATCH Version number

FILE NAME---
*RCMNJS/J.SYS/M<CR>
*BASE;ØR<CR>
*Ø,13674/ 4000 316<CR> [The /M is necessary; Monitor base; $SIZE table;]
*Ø,1373Ø/ Ø 2000<CR> $DSIZE table;
*Ø,16516/ 6250 70370<CR> $PNAME table;
*Ø,16552/ 4 100023<CR> $STAT table;
*Ø,16612/ Ø 45056<CR> $ENTRY table;
*E Exit to monitor]

For the F/B monitor:

.R PATCH<CR>

PATCH Version number

FILE NAME---
*RCMNFB.SYS/M<CR>
*BASE;ØR<CR>
*Ø,14602/ 4000 316<CR> [$SIZE table; Monitor base; $DSIZE table;
*Ø,14636/ Ø 2000<CR> $PNAME table;
*Ø,1764Ø/ 6250 70370<CR> $STAT table;
*Ø,17674/ 4 100023<CR> $ENTRY table;
*E Exit to monitor]

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

5-18
January 1976
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 12 in the device status word. The USR processes directory operations for RT-11 directory-structured devices; for special devices, the handler must process directory operations (LOOKUP, ENTER, CLOSE, DELETE), 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 Ø 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 .SPFUND special function programmed request, described in Chapter 9 of the RT-11 System Reference Manual. Devices which need to use the .SPFUN requests must have a bit set in the device status table (see Section 2.5.2.2).

5-20 January 1976
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:

\[ \text{SET LP FORM} \quad \text{(generate initial form feed)} \]
\[ \text{SET LP NOFORM} \quad \text{(suppress initial form feed)} \]

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
.RAD50  /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 .ASECTION 400 (area 1 on the listing).

2. Each table entry is four words long, as described previously. The option name may be up to six .RAD50 characters 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 and 2C). 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-ll 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 RTll 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 .CNTAXSW), 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 2310 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</th>
<th>11/20</th>
<th>11/40</th>
<th>11/45</th>
</tr>
</thead>
<tbody>
<tr>
<td>(core memory)</td>
<td>.66 ms</td>
<td>.36 ms</td>
<td>.28 ms</td>
</tr>
</tbody>
</table>
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>TTOWTS</td>
<td>13</td>
<td>Waiting for room in output buffer</td>
</tr>
<tr>
<td>CHNWT$</td>
<td>11</td>
<td>Waiting for channel to complete</td>
</tr>
<tr>
<td>SPNDS$</td>
<td>10</td>
<td>Suspended</td>
</tr>
<tr>
<td>NORUN$</td>
<td>9</td>
<td>Not loaded</td>
</tr>
<tr>
<td>EXITS$</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, and it requests a completion routine, the completion routine 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 USRWTS 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-11 System Reference Manual).

6.6 I/O TERMINATION

Because of the multi-job capabilities of RT-11 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, COMPLT, 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 UNL 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>1</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>1</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>1</td>
<td>1 = Command is waiting</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>4</td>
<td>0 = Console terminal inactive; i.e., BA is waiting for input from console terminal</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1 = Console terminal is active; i.e., BA is active in B/G</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>1</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>1</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>1</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>1</td>
<td>1 = Do not log terminal output (.TTYOUT, .PRINT)</td>
</tr>
<tr>
<td>9-13</td>
<td></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>1</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 BDDESTN 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 (BATS1W)
CTL file savestatus area (INDATA)
LOG file savestatus area (ODATA)
Output (LOG) buffer (OUTBUF)
Output buffer pointer (BATOPT)
Output character counter (BATOCT)
Input character counter (BATICT)
Monitor EMT dispatch address save areas

3. A command string is collected from the console terminal and is processed by .CSISPC. 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, BSOURC and BEDESTN bits in the BATS1W 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 BATS1W 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

7-5
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 BATSWL 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 PROGRM, 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,$0
```

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).
<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 ABORT after compile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATDOL = 40000 DATA or DOLLARS set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO = 20000 &quot;NO&quot; prefix on switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CTYOU8 = 10000 Output to CTY (@)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOGYOU8 = 4000 Output to LOG (\C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATYOU8 = 2000 Output to user prog (\D)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMYOU8 = 1000 Output to monitor (\E)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JOB = 400 $JOB encountered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAKEB = 200 /B switch on command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMMA = 100 Comma terminates command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BFORLI = 40 Next link requires FORTRAN library</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNIQUE = 20 UNIQUE command option set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BANNER = 10 Print BANNER on $JOB, $EOJ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTll = 4 RTll default on NO 'S' in Column 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIME = 2 Print time of day</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAKE = 1 Create a source file</td>
</tr>
<tr>
<td>BATSW2</td>
<td>2</td>
<td>More BATCH Control Switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABORT = 100000 Second time through ABORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FIRST = 10000 First card processed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SBIT = 4000 /S switch on command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEQ = 2000 $SEQ card processed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTBIT = 1000 Request temporary listing file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMSWB = 400 Command switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAKEB = 200 Same as BATSWT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARFD = 100 Asterisk in FD field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STAROK = 40 Wild card option is valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BNOEOJ = 20 $JOB or $SEQ before $EOJ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTDAT = 10 List DATA sections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BEOF = 4 EOF encountered on .BAT file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSWT = 2 /X switch set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EOJ = 1 $EOJ encountered</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMPSWT</td>
<td>4</td>
<td>Temporary command switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMSWT</td>
<td>6</td>
<td>Current command switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINSIZ</td>
<td>10</td>
<td>Input line buffer size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINLCT</td>
<td>12</td>
<td>Last buffer character count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSTAT</td>
<td>14</td>
<td>Input buffer status (see OTSTAT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICHRPT</td>
<td>16</td>
<td>Input character pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINCTR</td>
<td>20</td>
<td>Input buffer counter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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>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\quad 0 \rightarrow $ Buffer is free</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$BWAIT = 2\quad In\ I/O\ wait$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$BEOP = 4\quad End \ of \ file$</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>
<tr>
<td></td>
<td></td>
<td>These are the arguments passed between BATCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and BA:</td>
</tr>
<tr>
<td>BATS/1</td>
<td>60</td>
<td>Pointer to BATS/1 in BA.SYS</td>
</tr>
<tr>
<td>INDATA</td>
<td>62</td>
<td>Pointer to INDATA</td>
</tr>
<tr>
<td>ODATA</td>
<td>64</td>
<td>Pointer to ODATA</td>
</tr>
<tr>
<td>OUTBUF</td>
<td>66</td>
<td>Pointer to BATCH handler output buffer</td>
</tr>
<tr>
<td>BATOPT</td>
<td>70</td>
<td>Pointer to output character pointer</td>
</tr>
<tr>
<td>BATOCT</td>
<td>72</td>
<td>Pointer to output character counter</td>
</tr>
<tr>
<td>BATICT</td>
<td>74</td>
<td>Pointer to input character counter</td>
</tr>
</tbody>
</table>

(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$EXT</td>
<td>76</td>
<td>Pointers to EMT intercept pointers:</td>
</tr>
<tr>
<td>O$TIN</td>
<td>100</td>
<td>.EXIT</td>
</tr>
<tr>
<td>O$TOT</td>
<td>102</td>
<td>.TTYIN</td>
</tr>
<tr>
<td>O$PRN</td>
<td>104</td>
<td>.TTYOUT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.PRINT</td>
</tr>
<tr>
<td>SPC0</td>
<td>106</td>
<td>CSI Buffer:</td>
</tr>
<tr>
<td>SPC1</td>
<td>120</td>
<td>Channel 0</td>
</tr>
<tr>
<td>SPC2</td>
<td>132</td>
<td>1</td>
</tr>
<tr>
<td>SPC3</td>
<td>144</td>
<td>2</td>
</tr>
<tr>
<td>SPC4</td>
<td>154</td>
<td>3</td>
</tr>
<tr>
<td>SPC5</td>
<td>164</td>
<td>4</td>
</tr>
<tr>
<td>SPC6</td>
<td>174</td>
<td>5</td>
</tr>
<tr>
<td>SPC7</td>
<td>204</td>
<td>6</td>
</tr>
<tr>
<td>SPC8</td>
<td>214</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></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 0 of .DSTATUS)</td>
</tr>
<tr>
<td>ARGARG</td>
<td>4732</td>
<td>EMT argument list for BA handler initialization</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>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-ll 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 de-
leted. 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 encoun-
tering the IF statement again, now branches to label L2, skipping the
call to EDITIT. $DIRECTORY and $DELETE operations are performed be-
fore 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 func-
   tion 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 encoun-
   tered. 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,

\texttt{\textbackslash RS1<null>}

which instructs the BATCH handler to set variable S to the value in the byte following the character \texttt{L}.

6. Labels are implemented by inserting a \texttt{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 \texttt{C} directive so that the labels will appear in the log.

7. The unconditional branch, or GOTO command, is implemented with the \texttt{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:

\texttt{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 (\texttt{L$$$$}) is encountered at the end of the CTL file.

8. The IF conditional branch is implemented with the \texttt{I} directive. The \texttt{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 $CALLe d 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 KMON and a \F<CR> to terminate the BATCH stream.
EXAMPL.BAT

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

STARTI .PRINT #MSG
   .EXIT
   .NLIST REV

MSGI .ASCIZ /THIS MESSAGE COMES FROM THE BATCH STREAM/
   .EVEN
   .LIST REV
   .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),,12
SRT11  SCALL EDITIT ICALL EDITIT TO EDIT FILE.MAC
SRT11  GOTO   =L1

L21
$DIRECTORY FILE.*
$DELETE FILE.*
$END

EDITIT.BAT

$JOB/RT11
$! JOB TO EDIT FILE.MAC
%S
LET A=33
!INCREMENT S TO PREVENT RECURSION
!A IS ALT MODE
.R EDIT
*ERFILE.MAC*AR*A**A*
*GMSFILE*AKI .ASCIZ /MODIFIED BY EDITOR RUN BY BATCH/
*"A*EX*A**A*
$END
SJOB

MESSAGE EXAMPLE BATCH STREAM
\n\nSTART:
PRINT #MSG
EXIT
LST REV

MSG:
ASCIZ /THIS MESSAGE COMES FROM THE BATCH STREAM/
EVEN
LIST REV
END START

SRT11
LET S=0
\KS1 \LL1 \CL1:

$MACRO/RUN FILE,LST=LIST FILE,MAC/INPUT FILE,OBJ
\ER MACRO
\DFILE,FILE,LST=FILE,MAC
\F\ER LINK
\D00000=FILE
\ER D00000
\ER PIP
\D00000,SAV/D
\C
SPRINT FILE,LST
\ER PIP
\DFLIST=*,*/X=FILE,LST
\F\C
$DELETE FILE,LST
\ER PIP
\DFLIST=LIST/D
\C
SRT11
IF(S=0),,L2
\IS 1 1L2 \CL
SCALL EDITIT !CALL EDITIT TO EDIT FILE,MAC
\ER RATCH
\DFEDITIT=S
\C
SRT11
GOTO -L1
\JL1 0\LL2 \CL2:

7-15
EXAMPL.CTL (Cont)

$DIRECTORY FILE,*
\ER PIP
\DFILE,*/L
\FN\C
$DELETE FILE,*
\ER PIP
\DFILE,*/D
\C
\SEQ
\LSSSSSS\F\ER BATCH
\D/R
\E/F

EDITIT.CTL

\C
\JOB/RT11

$! JOB TO EDIT FILE.MAC
%5
\KSS\C LET A=33
%A IS ALT MODE
\KA1 \ER EDIT
\DFBFIELD,MAC\KA2R\KA2KA2
\DGMSG1\KA2KI ,ASCIZ /MODIFIED BY EDIT RUN BY BATCH/
\D\KA2EX\KA2KA2
\C
\SEQ
\LSSSSSS\F\ER BATCH
\D/R
\E/F
EXAMPL. LOG

$JOB
$MESSAGE EXAMPLE BATCH STREAM
$CREATE FILE.MAC
$EOE
$RT11
  LET S=0
L1    L11
$MACRO/RUN FILE.LST/LIST FILE.MAC/INPUT FILE/OBJEECT

*ERRORS DETECTED: 0
*FREE CORE: 151000, WORDS
*
EXAMPL.LOG (Cont.)

MAIN. RT-11 MACRO NM02-19 10-APR-75 10:33:45 PAGE 1

1 000000
2 000000
3 000000
4 000006
5 000010
6 000010
7 000000
8 000000
9 000000

10 000000
11 000000
12 000000
13 000000
14 000000
15 000000
16 000000
17 000000
18 000000
19 000000
20 000000
21 000000
22 000000
23 000000
24 000000
25 000000
26 000000
27 000000
28 000000
29 000000
30 000000
31 000000
32 000000
33 000000
34 000000
35 000000
36 000000
37 000000
38 000000
39 000000
40 000000
41 000000
42 000000
43 000000
44 000000
45 000000
46 000000
47 000000
48 000000
49 000000
50 000000
51 000000
52 000000
53 000000
54 000000
55 000000
56 000000
57 000000
58 000000
59 000000
60 000000
61 000000
62 000000
63 000000
64 000000
65 000000
66 000000
67 000000
68 000000
69 000000
70 000000
71 000000
72 000000
73 000000
74 000000
75 000000
76 000000
77 000000
78 000000
79 000000
80 000000
81 000000
82 000000
83 000000
84 000000
85 000000
86 000000
87 000000
88 000000
89 000000
90 000000
91 000000
92 000000
93 000000
94 000000
95 000000
96 000000
97 000000
98 000000
99 000000
100 000000
101 000000
102 000000
103 000000
104 000000
105 000000
106 000000
107 000000
108 000000
109 000000
110 000000
111 000000
112 000000
113 000000
114 000000
115 000000
116 000000
117 000000
118 000000
119 000000
120 000000
121 000000
122 000000
123 000000
124 000000
125 000000
126 000000
127 000000
128 000000
129 000000
130 000000
131 000000
132 000000
133 000000
134 000000
135 000000
136 000000
137 000000
138 000000
139 000000
140 000000
141 000000
142 000000
143 000000
144 000000
145 000000
146 000000
147 000000
148 000000
149 000000
150 000000
151 000000
152 000000
153 000000
154 000000
155 000000
156 000000
157 000000
158 000000
159 000000
160 000000
161 000000
162 000000
163 000000
164 000000
165 000000
166 000000
167 000000
168 000000
169 000000
170 000000
171 000000
172 000000
173 000000
174 000000
175 000000
176 000000
177 000000
178 000000
179 000000
180 000000
181 000000
182 000000
183 000000
184 000000
185 000000
186 000000
187 000000
188 000000
189 000000
190 000000
191 000000
192 000000
193 000000
194 000000
195 000000
196 000000
197 000000
198 000000
199 000000
200 000000
201 000000
202 000000
203 000000
204 000000
205 000000
206 000000
207 000000
208 000000
209 000000
210 000000
211 000000
212 000000
213 000000
214 000000
215 000000
216 000000
217 000000
218 000000
219 000000
220 000000
221 000000
222 000000
223 000000
224 000000
225 000000
226 000000
227 000000
228 000000
229 000000
230 000000
231 000000
232 000000
233 000000
234 000000
235 000000
236 000000
237 000000
238 000000
239 000000
240 000000
241 000000
242 000000
243 000000
244 000000
245 000000
246 000000
247 000000
248 000000
249 000000
250 000000
251 000000
252 000000
253 000000
254 000000
255 000000

7-18
EXAMPL.LOG (Cont.)

MAIN. RT=11 MACRO VM02-10 10-APR-75 10:13:45 PAGE 1+
SYMBOL TABLE

MSG 000010R  PC  =%000007  R0  =%000000
R1 =%000001  R2 =%000002  R3  =%000003
R4 =%000004  R5 =%000005  R6  =%000006
START 000000R
ABS, 000000  000006  001
ERRORS DETECTED: 0
FREE CORE: 15100. WORDS

FILE,FILE,LST=FILE,MAC

THIS MESSAGE COMES FROM THE BATCH STREAM

$PRINT FILE,LST

$DELETE FILE,LST

SRT11 IF(S=0) ,,L2
SCALL EDITIT ICALL EDITIT TO EDIT FILE,MAC

$JOB/RT11
S1 JOB TO EDIT FILE,MAC
% 
LET A=33 
INCREMENT S TO PREVENT RECURSION
IA IS ALT MODE

*ERFILE,MACRSS
*
GMSGISKI *ACIZ /MODIFIED BY EDITOR RUN BY BATCH/
SEXSS

SEND
SSSSS

SRT11 GOTO =L1
L1:
SMACRO/RUN FILE,LST/LIST FILE,MAC/INPUT FILE/OBJECT

*ERRORS DETECTED: 0
FREE CORE: 15136. WORDS
*

7-19
.MAIN.

000000 .MCALL .REGDEF.,PRINT.,EXIT
000000 .REGDEF
000000 START: .PRINT #MSG
000006 .EXIT
00010 .NLIST REX
115 MSG1 .ASCIZ /MODIFIED BY EDITOR RUN BY BATCH/
00010 .EVEN
0000010 .LIST REX
000000 .END START
EXAMPLE.LOG (Cont.)

;MAIN. RT-11 MACRO VM02-10 10-APR-75 10:34:08 PAGE 1+
SYMBOL TABLE

MS6  000010R  PC  %000007  R0  %000000
R1   %000001   R2  %000002  R3  %000003
R4   %000004   R5  %000005  SP  %000006
START 000000R
* ABS. 000000 000
   000050 001
ERRORS DETECTED: 0
FREE CORE: 15136 WORDS

FILE .FILE. LST = FILE.MAC

MODIFIED BY EDITOR RUN BY BATCH

SPRINT FILE.LST

SDELTF FILE.LST

SRT11 IF(S=0),12
L21
SDIRECTORY FILE.*

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
SDELTF FILE.*

SEOJ

7-21
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)
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
TITLE RC11 V01-01 (FIXED HEAD DISK)

IO2-11 RC11/RS64 DEVICE HANDLER

IDFC=11-YYYYY+A

IJDG

JULY 1974

COPYRIGHT (C) 1974

IDIGITAL EQUIPMENT CORPORATION

MAYNARD, MASSACHUSETTS 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY

FOR 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

FOR OTHER USES TO ANY OTHER PERSON EXCEPT

IF FOR USE ON SUCH SYSTEM AND TO ONE WHO AGREES TO THESE

LICENSE TERMS, TITLE TO AND OWNERSHIP OF THE SOFTWARE

IS MALL AT ALL TIMES REMAIN IN DIGITAL.

THE INFORMATION IN THIS DOCUMENT IS SUBJECT

T TO CHANGE WITHOUT NOTICE AND SHOULD NOT

BE CONSTRUED AS A COMMITMENT BY DIGITAL

EQUIPMENT CORPORATION. DIGITAL ASSUMES NO

RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR

IN THIS DOCUMENT.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE

SURE OR RELIABILITY OF ITS SOFTWARE ON

EQUIPMENT WHICH IS NOT SUPPLIED BY

DIGITAL.
; SATTL HANDLE DEFINITIONS
; MCALL LRGREF, .1VR.. ;.V2: ;

; IRGREGISTER DEFINITION
; LTST ME
; .RFGREF

; .NR1ST ME

; IRT-11 MONITOR DEFINED CONSTANTS
; MONLOW = 44 ; IMONITOR BASE POINTER
; NFSPTR = 7F ; IPRINTER TO B MANAGER
; FCOMPLET E ENTRY ; FCRAF = 1 ; FCHRM ERROR BIT
; FCTRY = A ; FCRY FOR ERRORS

; IPRIORITY CONSTANTS
; PR7 = 54 ; IHANDLER ENTERED AT PR7
; PR5 = 44 ; IHANDLER RUNS AT PR5

; IR-11 COMMUNICATION CONSTANTS
; WPE = 101 ; IGET INTERRUPT ENABLE, WRITE A INITIATE FUNC.
; RD = 100 ; IGET INTERRUPT ENABLE, READ & INITIATE FUNC.
; TNMC = 1000 ; FINITL, INREF, CURRENT ADDR REG (RCC0)
; ASRT = 40 ; FAPORT OPERATION IN PROGRESS (RCC9)
; STRYF = 08 ; FTRY AFTER ERROR MASK FOR RCCS
; SBTR = 1 ; FBIT 14 = 1 => DATA ERROR
; SBTR = 1 ; FBIT 15 = 1 => ADDRESS ERROR

; IRF-11 CONTROL REGISTER
; RCLA = 177440 ; FLOOK AHFAN REGISTER
; RCNA = 177442 ; IDISK ADDRESS REGISTER
; RCFA = 177444 ; IDISK ERROR STATUS REGISTER
; RCFB = 177446 ; IDISK CONTROL AND STATUS REGISTER
; RCFR = 177448 ; IDISK REGISTER
; RCUC = 177450 ; IMRWR COUNT REGISTER
; RCCA = 177452 ; FCURRENT ADDRESS REGISTER
; RCNM = 177464 ; IMAINTENANCE REGISTER
; RCNB = 177466 ; IODATA RUFFER REGISTER
RC11 VM=01 (FIXED HEAD DISK) RT-11 MACRO VM=2-09 A-APR-75 1214126 PAGE 2

RC11 DEVICE HANDLER

;SMART RC11 DEVICE HANDLER
; (MAXIMUM SUPPORT 1 CONTROLLER AND 4 RBA4 DISKS)
; (1926 BLOCKS OF 512 WORDS)
;CRECT RYHND

ENTRY POINT

MOV  #RET, (PC)+
ISPECIFY THE RETRY COUNT

RETRY COUNTER

ISPT UP THROUGH COMMON ROUTINE

ISPT ZERO FILE INITIATION

ISPT TO RCA

ISPT BUFFER ADDR (RCA)

ISPT WRG CNT ADDR (RCW)

ISASSUM WRITE FUNCTION
41 000172 003445
42 000174 010546
43 01 000176 013746 00085
44 000202 006285
45 000204 100046 0008P1
46 000206 104314
47 000212 001373
48 000214 009726
49 000216 001491
50 000220 004295
51 000222 068574
52 000224 013695
53 000226 053745 177408
54 000228 053745 177408
55 00022A 053745 177408
56 00022C 053745 177408
57 00022E 053745 177408
58 000230 053745 177408
59 000232 053745 177408
60 000234 053745 177408
61 000236 053745 177408
62 000238 053745 177408
63 00023A 053745 177408
64 00023C 053745 177408
65 00023E 053745 177408
66 000240 053745 177408

A-6

January 1976
COMMON SUBROUTINE

1  COMMON SUBROUTINE
2  IRDCOMNN
3  COMMON SUBROUTINE USED BY INTERRUPT
4  AND ENTRY ROUTINES

6   000252  012794  177456  IRDCOMNN
7   000256  031014  
8   000260  014912  MOV  SRC, RG
9   000264  019650  MOV  (SRN)+, RG
10   000268  004717  ADD  R4, R4
11   00026c  014755  177516  ADD  MOV  RCOF, R5
12   000270  015546  MOV  (R6)+, -(R5)
13   000274  00a316  ASL  SP
14   000278  00a316  ASL  SP
15   000300  00a316  ASL  SP
16   000304  050214  MOV  (R7)+, R4
17   000308  024444  CMP  -(R5), -(R7)
18   00030c  00a416  MOV  (SP)+, R4
19   000310  004725  XTH  (R5)+
20   000312  00a200  RTS  80

21
22
23
24   000314  000800  MONITOR ENTRY ADDR.
25
26   000316  RCRIZE= +RCRAT
27
28
29
30
31
TABLE OF CONTENTS

2- 2  MACROS, GLOBALS
3- 1  ASPECT
7- 89  ROOTSTRAP I/O DRIVER - RC11
10- 1  ROOTSTRAP CORE DETERMINATION
11- 1  READ MONITOR, LOADUP HANDLERS
12- 1  RELOCATION LIST

ROOT VR2R-01  RT-11 ROOTSTRAP RT-11 MACRO VHR2-09  APR-75 11149184

1  @BEGIN
2  SRMSYS=1
3  TITLE ROOT VR2R-01 RT-11 ROOTSTRAP
4  RT-11 ROOTSTRAP
5  
6  DEC-11-ORBTA-D
7  
8  COPYRIGHT DEC 1975
9  
10  DIGITAL EQUIPMENT CORPORATION
11  MAYNARD, MASSACHUSETTS 01754
12  
13  THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
14  ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
15  THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
16  OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
17  AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
18  ONE WHO AGREES TO THESE LICENSE TERMS, TITLE TO AND OWNERSHIP OF THE
19  SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
20  
21  THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
22  CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
23  AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
24  
25  DIGITAL DISCLAIMS ALL RESPONSIBILITY FOR THE USE
26  OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
27  WHICH IS NOT SUPPLIED BY DIGITAL.
.GLOBAL MACROS, GLOBALS

:MCALL .VI.


:GLOBAL REFERENCES TO MONITOR:
:GLOBAL DOVREC, SENTRY, SINPTR, SKMLOC, SHONBL, SPNAME, SLOT
:GLOBAL SWPFL, SUBRLC, SPNAME
:GLOBAL RSTRNG, CORPTR, PKRASTG, FILLER, HWFPUS, HWDBPS, KMLOC
:GLOBAL KMON, KMONSZ, KILLS, MAPOFF, GCOMP, RT1185
:GLOBAL RTLEN, RTSIZE, SWPFLZ, SYENTO, SYENNO, SYENCH, SYASSG
:GLOBAL SYSLOW, TTBUF, TTMBUF, USRLOC, USRZ, MAXSYM

:GLOBAL RELAT

:GLOBAL ANIL, T.CSW, FPPADD, FPPIGN, MONLOC, TRAPLC, TRAPER

PERM = 2560
PINFBLK = 4000
JSW = 4
SR = 177578

:REGISTER DEFINITIONS:

R0=16
R1=1
R2=2
R3=3
R4=4
R5=5
SP=6
PC=7

:MONITOR OFFSET CONSTANTS
INITIALIZE RTT AND TOT VECTORS
ION RTT INTERRUPT TO READS ROUTINE
ION TOT INTERRUPT TO WAIT ROUTINE

UNITRO DI
BYTE C$CGO+CSR0
READ FROM UNIT 0, SETS WEIRD BUT OK R8
BYTE C$CGO+CSR0+CSUNITRO FROM UNIT 1
134=92 USEABLE

ROOTI
MOVB UNITRO(R8),ROCHDISP RTT READ FUNCTION FOR CORRECT UNIT
RETRYI
MOV #PC,SP
INIT SP WITH NEXT INSTRUCTION
MOV #26B,RR2
AREA TO READ IN NEXT PART OF ROOT
CLR RR2
ISRT TRACK NUMBER
JMP OUT OF ROOM HERE, GO TO CONTINUATION
PAPER TAPE VECTORS

2B1
MOV SP,R1
SET TO BIG WORD COUNT
INC R0
ISRT TO ABSOLUTE TRACK 1
RR2 38
BRANCH TO CONTINUATION

3B1
MOV #PC,R3
ABSOLUTE SECTOR 3, FOR NEXT PART
RTT
CALL READS SUBROUTINE

8 B1
BRANCH TO CONTINUATION
FLOATS OF UNUSED VECTORS, (OR-11B7)

READS
MOV #RXCS,R4
RX => RX STATUS REGISTER
MOV R4,R5
R5 WILL POINT TO RX DATA BUFFER
ROCHDISP
LABM R
INITIATE READ FUNCTION
MOV R3,R5
GETS FILLED WITH READ COMMAND
TOT
CALL WAIT SUBROUTINE
MOV R3,R5
LOAD SECTOR NUMBER INTO RXDB
TOT
CALL WAIT SUBROUTINE
MOV R0,R5
LOAD TRACK NUMBER INTO RXDB
TOT
CALL WAIT SUBROUTINE
MOV #CSGO+CSEBIF,R4
LOAD EMPTY BUFFER FUNCTION INTO RXCS
TOT
CALL WAIT SUBROUTINE
STB R4
IS TRANSFER READY UP?
RTRET
BRANCH IF NOT, SECTOR MUST BE LOADED
MOVB R5,RR2
MOVE DATA BYTE TO MEMORY
DEC R1
CHECK BYTE COUNT
JMP R6
LOOP AS LONG AS WORD COUNT NOT UP
CLP RR2
KLUDGE TO SLUFF BUFFER IF SHORT WD COUNT
RR 43
LOOP
WATTI TST R4
IS TR ERR, DONE UP TNT ENR CAN'T BE
RED R4
LOOP TILL SOMETHING
RTRET R4
START AGAIN IF ERROR
RETURN
ROUTE1: CMPB (R3)+,(R3)+
RPT CMPB (R3)+,(R3)+
RPT RST #UNIT0,ROCMO
RNE 1S
CLR RO
MOV @RO,(PC)+
ISAVE UNIT BOOED FOR FER LATER
ISAVE THE UNIT HERE
ISAVE HANDLE ERRORS DIFFERENTLY
ISAVE THE UNIT HERE
NOW WE ARE READY TO DO THE REAL BOOT

ROOT VM25=81 RT-11 ROUTESTR RT-11 MACRO VM2=09 8=APR '75 1116684 PAGE 4
ASECT

1,2,3,4,5,6,7,8,9
10 11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27

; FOLLOWING ARE THE ROUTESTR I/O DRIVERS FOR EACH VALID
; SYSTEM DEVICE.
; CALLING SEQUENCE:
; R0 = PHYSICAL BLOCK TO READ/WRITE
; R1 = BLOCK COUNT
; R2 = BUFFER ADDRESS
; R3,R4,R5 ARE AVAILABLE AND MAY BE DESTROYED BY THE DRIVER
; THE DRIVER MUST GO TO #ERR IF A FATAL I/O ERROR OCCURS.
; IT MUST ALSO INVOKE THE MACRO SYSD
; MACRO SYSDV NAME,VENDOR
; GLOBAL NAME=INT NAME=SIZE
; DEFINE SYSTEM DEVICE INTERRUPT & SIZE
;
SYNAME = a
;IRPC y,SYNAME
SYNAME = $SYNAME+4=y=080009
;ENDR
;SYVEC = VECTOR
; = $SYVEC
; = $SYVEC
; = $SYVEC
; = 40P
;SYITO = VECTOR / 2P
;SYITM = $1110000B
;RPT VECTOR & 17 / 4
;SYITM = SYITM /4
;ENDR
;ENDM SYDEV
A BASIC program for the RT-11 operating system, designed to interact with RP11 disk drives. The program appears to be a disk I/O routine, possibly for initializing or formatting the disk. The program includes a section for reading data from the disk and another for handling errors or making decisions based on the state of the disk.

Key commands and functions used:
- **MOV** for moving or transferring data between registers.
- **JMP** for conditional jumps based on data.
- **ISET** for setting the status of the disk.
- **IBY** for dividing by 10.
- **IP** for performing arithmetic operations.

The program structure suggests it might be part of a larger system routine, possibly for a computer or a specific device within a larger system.
SHIFT & SUBTRACT

SHIFT QUOTIENT

IF DF SYSYS#ISRFSYS#SRCYS

CONDITIONAL FOR RF DISK

IFDF SYSYS#ISRFSYS

SWITCH BOOTSTRAP T/O DRIVER = RF11

RF11 DISK HANDLER

DEVICE IS RF. IT VECTORS TO 284:

SYSDEV RF, 284

CONTROL & STATUS REGISTER

RFCH = 177442

WORD COUNT

RFMA = 177444

MEMORY ADDRESS

RFNA = 177446

FDISK ADDRESS

RFNE = 177478

FDISK ADDRESS EXTENSION

RFBS = 177472

DATA BUFFER

READI MOV #RFDA, R3

POINT TO DISK ADDRESS

MOV R2, R5

COPY BLOCK NUMBER

SWAB R5

MULTIPLY BY 256 TO GET WORD # ON DISK

MOV R5, R4

SAVE HIGH ORDER DISK ADDRESS

CLRB R5

MAKE 0 AT AN EVEN BLOCK NUMBER

MOV R5, (R3)

PUT LOW ORDER ADDRESS IN CONTROLLER

RRC #177429, R4

ISOLATE HIGH ORDER ADDRESS

MOV R4, (R3)

PUT IT IN CONTROLLER

TST = (R3)

RESET POINTER
ENDEC

1 THIS CODE IS COMMON TO RK5, RC11 AND RF11 HANDLERS

MOV R2,=R3
1 BUFFER ADD
MOV R1,=R3
1 WORD COUNT
NEG (R3)
1 (NEGATIVE)
MOV #R4,=R3
1 START_DISK_READ
TSB (R3)
1 WAIT_UNTIL_COMPLETE
RPL 3
TST (R3)
1 ANY_ERRORS?
RMT MERROR
1 HARD_HALT_ON_ERROR
RTS PC

ENDEC

ROOT V90=01   RT-11 Bootstrap RT-11 MACRO VMK2=89   6=APR=75 11:41:04 PAGE 8
Bootstrap I/O Driver = RC11

IF OF 50SYS

SSTL Bootstrap I/O Driver = DEFCTAPE

DEFCTAPE Bootstrap Handler

SYSEDY: DT,214
DEVICE IS DT, IT VECTORS TO 214.

TCCM = 177342
COMMAND REGISTER

TCNT = 177340
DATA REGISTER

TCDT = 177340
STATUS REGISTER

READI MOV $TCCM,R4
1 RA => COMMAND REG
MOV $TCDT,R3
1 R3 => DATA REG

DTORSCH MOV R4,R5
1 COPY BLOCK NUMBER
SUR $2,R5
1 SEARCH FOR EARLIER

MOV $TMR3,R4
1 REVERSE,RNUM

BST RIT @1MB280,#R4
1 WAIT TILL BLOCK FOUND

REG $3

RMT MERROR
1 IS IT THE DESIRED BLOCK

CMP R4,R3
1 NO.CONTINUE SEARCHING
ALR DTORSCH

DTPWARS MOV $3,#R4
1 SEARCH FORWARD (RNUM)

48 RIT @1MB280,#R4
1 WAIT

REG $3

RMT MERROR
1 IDSTRED BLOCK

CMP R4,R3
1 NO-SEARCH_FORWARD
RGV DTPWARS
1 NO-SEARCH_REVERSE
ALR DTORSCH

MOV R2,=R3
1 BUFFER ADDRESS
NEG R1

MOV R1,=R3
1 WORD COUNT
MOV \#5,RA
RIT \#102DB,RA
READ

OUT
RIT
REQ
SEL
RIT
RIT
RIT
RTS
PC

DTERR
TST \#TCAST
WHAT KIND OF ERROR?

NOT END ZONE

REVERSE
THEN GO SEARCH FORWARD
ELSE SEARCH REVERSE

ENDC

IF OF \$DXXSYS
FLOPPY SYSTEM

SRWTL BOOTSTRAP I/O DRIVER = FLOPPY

SYSDRV DX,264
FLOPPY VECTORS THROUGH 264

READI

ASL R0
CONVERT BLOCK TO LOGICAL SECTOR

RLS=BLOCK=4

ASL R1
MAKE WORD COUNT BYTE COUNT

MOV R0-(SP)
SAVE LSN FOR LATER

MOV R0,R3
NEED 2 COPIES OF LSN FOR MAPPER

MOVR R0,R4

INIT FOR TRACK QUOTIENT

JUMP INTO DIVIDE LOOP

PERFORM MAGIC TRACK DISPLACEMENT

BUMP QUOTIENT, STARTS AT TRACK 1

REP=INTEGER(LSN/26)

LOOP = R4=REM(LSN/26)+26

SET C IF SECTOR MAPS TO 1-15

PERFORM 211 INTERLEAVE

ADJUST SECTOR INTO RANGE =1,26

INVERSE FOR REMAINDER ONLY

NOW PUT SECTOR INTO RANGE 1-26

CALL READS SUBROUTINE

NOW GET THE LSN AGAIN

SET UP FOR NEXT LSN

WHATS LEFT IN THE WORD COUNT

BRANCH TO TRANSFER ANOTHER SECTOR

RETURN

RETURN FROM INTERRUPT

RETURN FROM WAIT SUBROUTINE, PRINTS ERRORS

RETRY

RETURN FROM INTERRUPT
***** THIS MUST FALL INTO RIMERR *****

:END:

38 008450 00A047 00A024 RIMERRJ JSR RB,REPORT I SAY THAT WE GOT ERROR
39 008454 015 012 077 :ASCIZ \19=\12/\7=\0 ERROR/\12>
40 008457 192 095 111
41 008458 077 117 040
42 008465 105 1F2 122
43 008470 117 122 012
44 008473 000

:EVER:

ROOT VR2R-81 RT-11 BOOTSTRAP RT-11 MACRO VMR2-89 E-APR-75 11149184 PAGE 19

BOOTSTRAP CORE DETERMINATION

1
2
3 008474 112037 177566 REPORT1 MOVB (RR)+,#TPR I PUT ANOTHER CHARACTER OUT
4 008500 109737 177566 REPORT1 TSTB #TPS I WAIT FOR TYPER READY
5 008504 109375
6 008506 109710
7 008510 001371
8 008512 000005
9 008514 000000
10 008516 000776
11 RR \=-2\ I KEEP HIM FROM CONTINUING
12 008520 012706 010000 ROOT1 MOV #10000,SP I SET STACK POINTER
13 008522 012700 000002 MOV #2,RR I READ IN SECOND PART OF ROOT
14 008530 012701 000000 MOV #\<ROOTSZ-1\>+400,RI EVERY BLOCK BUT THE ONE WE ARE IN
15 008532 012702 010000 MOV #10000,SP I INTO LOCATION 10000
16 008540 004767 177636 JSR PC,READ
17 .1IF GT \=-1000, \ERROR IBOOTSTRAP BLOCK IS TOO BIG
18 008544 012703 000004 MOV #4,RR IPOINT TO TRAP LOCATIONS
19 008550 011305 MOV #R3,RR ISAVE TRAP LOC
20 008552 012703 000620 MOV #NYM,(R3)+ ISET TRAP FOR NON EXISTENT MEMORY
21 008556 000013 CLR R3
22
23
24
25
26
27
28
29
30
31 008560 000447

:FINDEL RR \=-18\ I CHANGE TO HALT FOR FINDELING

;SMTL BOOTSTRAP CORE DETERMINATION

;/THIS BOOTSTRAP CAN SIMULATE ANY SIZE PDP=11./
;/ IF LOCATION "FINDLE" IS A HALT, THE CPU WILL STOP DURING THE BOOT./
;/ ON CONTINUE, THE TOP 5 BYTS OF THE SWITCH REGISTER ARE USED TO/
;/ SET THE TOP OF AVAILABLE CORE AS A MULTIPLE OF 1K./
;/ IF THE RR IS \=-18/ OR IF FINDLE IS A \=-18/, /
;/ THE BOOTSTRAP WILL DO A NORMAL CORE DETERMINATION./

:EVER

/EVER:

;EVER:
RIT #PEM.(R1)+
RED 15
SUR (PC)+,(R1+)
.RANGR /MON/ 
SUR (PC)+,(R1+)
.RANGR /IR/ 
SUR (PC)+,(R1)

RIT #ENDBLK,(R2)
RED MONFND
FNDCMONITOR

RIT #ENDBLK,(R2)
RED MONFND
FNDCMONITOR

ADD 10(R2),R0
ADD #16,R2
GET NEXT ENTRY
ADD RUFF+6,R2
MOV R2,R1
POINT R1 TO NEXT

ADD RUFF+2,R0
SEE IF NEXT IS AVAILABLE
ADD #FF,R0
YES, CONTINUE

ASCIZ <15+12>/<9>=NO MONITOR,SYS<12>

.asciz <15+12>/<9>=NO MONITOR,SYS<12>

.asciz <15+12>/<9>=NOT ENOUGH CORE<12>

.asciz <15+12>/<9>=NOT ENOUGH CORE<12>

.asciz <15+12>/<9>=NOT ENOUGH CORE<12>

.asciz <15+12>/<9>=NOT ENOUGH CORE<12>

.asciz <15+12>/<9>=NOT ENOUGH CORE<12>

.asciz <15+12>/<9>=NOT ENOUGH CORE<12>
BASE READ MONITOR, LOOKUP HANDLERS  

1  801126  061692
2  801128  061692
3  801130  062700  000002
4  801132  011046
5  801134  012701  000000
6  801136  012701  017736
7  801140  060701  000000
8  801144  001721  000000
9  801148  001721  000000
10 80114C  001721  000000
11 801150  001721  000000
12 801154  001721  000000
13 801158  001721  000000
14 80115C  001721  000000
15 801160  001721  000000
16 801164  001721  000000
17 801168  001721  000000
18 80116C  001721  000000
19 801170  001721  000000
20 801174  001721  000000
21 801178  001721  015324
22 80117C  001721  015324
23 801180  001721  107774
24 801184  001721  107774
25 801188  001721  107774
26 80118C  001721  107774
27 801190  001721  107774
28 801194  001721  107774
29 801198  001721  107774
30 80119C  001721  107774
31 8011A0  001721  107774
32 8011A4  001721  107774
33 8011A8  001721  107774
34 8011AC  001721  107774
35 8011B0  001721  107774
36 8011B4  001721  107774
37 8011B8  001721  107774
38 8011BC  001721  107774
39 8011C0  001721  107774
40 8011C4  001721  107774
41 8011C8  001721  107774
42 8011CA  001721  107774
43 8011CE  001721  107774
44 8011D0  001721  107774
45 8011D4  001721  107774
46 8011D8  001721  107774
47 8011DA  001721  107774
48 8011DC  001721  107774

;IF OF SRKSYS;ISDSSYS;SDDSYS;SDDSYS 1THE RK,RX,RP,RJ303/4 CAN BOOT FROM ANY UNIT

;IF OF SRKSYS  ;CODE FOR RK
MOV  #SRKDA,R1  ;GET THE RK UNIT NUMBER
ROL  R1
ROL  R1
ROL  R1
ROL  R1
RCL  R1
INCR  R1
STRIP  TO  3  BITS

;IF OF SRKSYS  ;CODE FOR RJ303/4
MOV  #SDDSYS  ;UNIT # INTO R1
RCL  R1
STRIP  TO  3  BITS

;IF OF SRKSYS  ;IF P11
MOV  #SRKCS,R1  ;GET CONTROLLER STATUS REG INTO R1
RCL  R1
STRIP  TO  3  BITS

;IF OF SRKSYS  ;IF P11
MOV  #SDDSYS  ;UNIT # INTO R1
RCL  R1
STRIP  TO 3 BITS 2-6
MOV  RTUNIT,R1

IF OF SDXSYS

ENDC

IF FLOPPY

ADD  R1,OKASSG(R0)

ADD  R1,SYASSG(R0)

MOV  R1,SYUNIT+1(R0)  ;SET UNIT NUMBER WE BOOTED

START VR2R-01  RT-11 BOOTSTRAP RT-11 MACRO VMR2-09  R=APR7S 11149184 PAGE 11+

READ MONITOR, LOOKUP HANDLERS

ENDC

ENABL LSR

ENDC

IF SRKSYS1BDXSYS1DP SYS

001254  004760  004322  004300
001262  004903
001264  012701  000000
001270  006401
001272  004721
001274  001414
001276  004441
001300  004127  000000  07370
001308  004127  000000  07370
001310  003836
001314  003836  000000
001320  001140  000000
001324  004721
001326  005203
001330  002703  000000
001334  001356
001336  002703  000010
001342  015740  000014  000000
001350  0012701
001354  004001
001356  002706  000000
001360  0012701  000000
001366  0012701  000000
001372
001374  002022
001402  000024
001408  000046
00140C  004006
001414  004110  000204
001420  000224
001422  009303

R3
R3
R3
R3
R1
R1
R1
R1
R1
R1
R1
R1
R3
R3
R3
R3
R3
R3
R3
R3
R3
R3
R3
R3
R3
RELOCATE RR STUFF HERE

TRAPRC
TRAPER
FPADD
FPIGN
MHLOC
I.CSH
AVAIL

IFPP SERVICE FOR MONITOR
WHERE USR WILL SIT
SINGLE USER STUFF HERE
MONITOR FREE Q POINTER

IF

RELOCATION LIST

FILENAME GOES HERE
IFILENAME GOES HERE
SAVESTATUS GOES HERE
### Relocation List

<table>
<thead>
<tr>
<th>No.</th>
<th>Module</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>001634</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>000000</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>000000</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>000000</td>
</tr>
</tbody>
</table>

- `RUFFA = + 777 / 1000`
- `MONTZ = + ROOTSZ * 1000`

### Symbol Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVAL</td>
<td>+----------</td>
</tr>
<tr>
<td>BLOOP</td>
<td>001612</td>
</tr>
<tr>
<td>BUFR</td>
<td>001634</td>
</tr>
<tr>
<td>DKASSG</td>
<td>+----------</td>
</tr>
<tr>
<td>FPPANO</td>
<td>+----------</td>
</tr>
<tr>
<td>ICNN</td>
<td>+----------</td>
</tr>
<tr>
<td>KCHI</td>
<td>+----------</td>
</tr>
<tr>
<td>MONFND</td>
<td>001126</td>
</tr>
<tr>
<td>QCNOC</td>
<td>+----------</td>
</tr>
<tr>
<td>RCER</td>
<td>+----------</td>
</tr>
<tr>
<td>RCHC</td>
<td>177450</td>
</tr>
<tr>
<td>REPORT</td>
<td>1600475</td>
</tr>
<tr>
<td>R1</td>
<td>+----------</td>
</tr>
<tr>
<td>SP</td>
<td>000000</td>
</tr>
<tr>
<td>SYBITS</td>
<td>000814</td>
</tr>
<tr>
<td>SYMnemonic</td>
<td>000814</td>
</tr>
<tr>
<td>TKS</td>
<td>007560</td>
</tr>
<tr>
<td>TRAPCE</td>
<td>+----------</td>
</tr>
<tr>
<td>USBSET</td>
<td>+----------</td>
</tr>
<tr>
<td>SHMORG</td>
<td>+----------</td>
</tr>
<tr>
<td>SHMPLR</td>
<td>+----------</td>
</tr>
</tbody>
</table>

- `RCNF = 001652`
- `RF = 000880`
- `SIDFRA 000450`
- `STRM = +----------`
- `FILLR = +----------`
- `HWDPS = +----------`
- `KMON = +----------`
- `NOPSZ = +----------`
- `NC = %000087`
- `PERK = 002000`
- `RCDR = 177456`
- `RCS = +----------`
- `RELST = 001596`
- `REPLT = +----------`
- `RT1158 = +----------`
- `R4 = %000084`
- `SR = %000083`
- `SYASSG = +----------`
- `SYNAME = 078378`
- `SYNAME = +----------`
- `TPA = 177564`
- `Traper = +----------`
- `USBLOC = +----------`
- `USBCY = +----------`
- `USPLRZ = +----------`

- `SA = 002000`
- `Errors Detected 0`
- `FREE CPM 1650W, Words`

- `RCRTSJ, LP1/NITIN/C=RCJSY, BSTRAP`
TITLE LP V02-03 PS-JUN-7A RT-11 MACRO VM#2-12 16-APR-75 10:05:11 PAGE 1

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

This 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, and any other copyright notice. Any copies made of this software, or any portion thereof, may not be transferred to any other person, and may not be modified in any way, except as specified in the license agreement. 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 of its software or for its reliability or for the equipment on which its software is used, which is not supplied by Digital.
<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>R0 = 10</td>
</tr>
<tr>
<td>2</td>
<td>000000</td>
<td>R1 = 11</td>
</tr>
<tr>
<td>3</td>
<td>000001</td>
<td>R2 = 12</td>
</tr>
<tr>
<td>4</td>
<td>000002</td>
<td>R3 = 13</td>
</tr>
<tr>
<td>5</td>
<td>000003</td>
<td>R4 = 14</td>
</tr>
<tr>
<td>6</td>
<td>000004</td>
<td>R5 = 15</td>
</tr>
<tr>
<td>7</td>
<td>000005</td>
<td>R6 = 16</td>
</tr>
<tr>
<td>8</td>
<td>000006</td>
<td>R7 = 17</td>
</tr>
<tr>
<td>9</td>
<td>000007</td>
<td>PC = 18</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td><strong>LINE PRINTED CONTROL REGISTERS</strong></td>
</tr>
<tr>
<td>11</td>
<td>177514</td>
<td>LPA = 177514</td>
</tr>
<tr>
<td>12</td>
<td>177516</td>
<td>LPRA = 177516</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td><strong>ILTNF PRINTER CONTROL REGISTER</strong></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td><strong>ILTNF PRINTER DATA BUFFER</strong></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td><strong>ILTNF PRINTER VECTOR ADDR</strong></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td><strong>CONSTANTS FOR MONITOR COMMUNICATION</strong></td>
</tr>
<tr>
<td>17</td>
<td>000281</td>
<td>MDFRR = 1</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td><strong>MDFRR ERROR BIT</strong></td>
</tr>
<tr>
<td>19</td>
<td>000280</td>
<td>MMLOW = 50</td>
</tr>
<tr>
<td>20</td>
<td>00027f</td>
<td>MFPSET = 57</td>
</tr>
<tr>
<td>21</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>22</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>23</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>24</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>25</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>26</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>27</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>28</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>29</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>30</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>31</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
<tr>
<td>32</td>
<td>00027f</td>
<td>MFPSET = 94</td>
</tr>
</tbody>
</table>

**ASCII CONSTANTS**

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>000015</td>
<td>CR = 15</td>
</tr>
<tr>
<td>34</td>
<td>000012</td>
<td>LF = 12</td>
</tr>
<tr>
<td>35</td>
<td>000014</td>
<td>FF = 14</td>
</tr>
<tr>
<td>36</td>
<td>000011</td>
<td>HT = 11</td>
</tr>
<tr>
<td>37</td>
<td>000214</td>
<td>COLSZ = 132</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>1132 COLORS</td>
</tr>
</tbody>
</table>
1  MOVL P0
2  #100H
3 LOAD P0  
   LDIPT  
   LPVEF  
   AND OF INTERRUPT VECTOR
4  LOPT  LDIPT  
   LPVEF  
   OFFSET TO INTERRUPT SERVICE
5  LDIPT  
   LPVEF  
   PRIORITY 7
6  MOV  LDIPT  
   LPVEF  
   POINTER TO LAST O ENTRY
7  LPVEF  
   MOV  
   POINTER TO CURRENT O ENTRY
8  ENTRY POINT
9  MOV  P0,LPVEF
10 MOV  P0,LPVEF
11 MOV  P0,LPVEF
12 MOV  P0,LPVEF
13 MOV  P0,LPVEF
14 MOV  P0,LPVEF
15 MOV  P0,LPVEF
16 MOV  P0,LPVEF
17 MOV  P0,LPVEF
18 MOV  P0,LPVEF
19 MOV  P0,LPVEF
20 MOV  P0,LPVEF
21 MOV  P0,LPVEF
22 MOV  P0,LPVEF
23 MOV  P0,LPVEF
24 MOV  P0,LPVEF
25 MOV  P0,LPVEF
26 MOV  P0,LPVEF
27 MOV  P0,LPVEF
28 MOV  P0,LPVEF
29 MOV  P0,LPVEF
30 MOV  P0,LPVEF
31 MOV  P0,LPVEF
32 MOV  P0,LPVEF
33 MOV  P0,LPVEF
34 MOV  P0,LPVEF
35 MOV  P0,LPVEF
36 MOV  P0,LPVEF
37 MOV  P0,LPVEF
38 MOV  P0,LPVEF
39 MOV  P0,LPVEF
40 MOV  P0,LPVEF
41 MOV  P0,LPVEF
42 MOV  P0,LPVEF
43 MOV  P0,LPVEF
44 MOV  P0,LPVEF
45 MOV  P0,LPVEF
46 MOV  P0,LPVEF
47 MOV  P0,LPVEF
48 MOV  P0,LPVEF
49 MOV  P0,LPVEF
50 MOV  P0,LPVEF
LP VR2-03 25-JUN-74 RT-11 MACRO VM92-10 16-APR-75 T010511 PAGE 4+

5A 001144 104377 TARCNTL,WRN 1 $L.B (PP)+ UPDATE TAB COUNT
5A 001146 000021
51 001150 001433 RED PSTTAB
52 001152 115537 PC1 MOV $PB,RS,##LB
53 001156 000742 AR LPNEXT TRY FOR NEXT CHAR
54 001160 000029 RETI RTA PC
55 001162 120527 000011 CHRTSTI CMPB RS,##HT
56 001166 001436 RED TARSET
57 001170 120527 000012 CMPB RS,##LF

5A 001074 001436 RED RSTC
5A 001076 120527 000015 CMPB RS,##CR
5A 001082 000248 CMPB RS,##FF
6A 001098 001333 CMPB IGNORE
6A 001120 013747 000024 177720 RSTCI MOV #000001,COLEN
6A 001122 013747 000011 177720 RSTTABI MOV S1,TARCNT
6A 001126 010751 AR PCI
6A 001130 013747 177712 177636 TARSETI MOV TARCNT,TABPLG
6A 001136 013745 000040 TARI MOV AR,RS
6A 001142 000735 AR PCHAR

70 001244 000240 RLPKI TNF -(R8)
71 001246 020240 CMPB (R8),=,(R8)+ MAKE SURE WE ONLY COMPL IN ONE CIF
72 001250 013745 000014 MOV $AR,RS PRINT TO ADS OF NEXT CHAR
73 001254 000756 AR RSTC PRINT INITIAL FF
74 001256 057754 000091 LRPRRI A15 $HNPFR,-,(R4)
75 001258 000091 OPERATION COMPLETE
7A 001262 005737 177754 LPMONE PLP $AR,PC
8A 001266 017744 MOV PC,R4
81 001270 062794 177520 ADD $COMP,-,(R4)
82 001274 011705 000054 MOV $A1,-,(R4)
83 001300 001755 000270 JMP 000051 JUMP TO Q MANAGER
84 001304 000000 TEXITI A
8A 001306 003596 LPBITE *:=LOADPT
8A 001308 000001 * END
A.4 CR11 DEVICE HANDLER
17 002230  CRVCT=278  INTERRUPT VECTOR
18 177140  CRST=177160  ICARD READER STATUS REGISTER
19 177160  CR1=177162  IDATA BUFFER 1
20 177142  CR2=177160  IDATA BUFFER 2
21 177144
22 18 1 CONSTANT FOR MONITOR COMMUNICATIONS
23 20  MONTR#1  MSEND ERROR
24 21  MONTR#2  IDWORD ADDRESS OF MONITOR
25 22  OFSET#2  OFFSET TO HANDLER RETURN
26 23  PS=177774  PROGRAM STATUS WORD
27 24  PRIORITY 1  PRIORITY 1
28 25  PRIORITY 2
29 26
30 27  ASCII CHARACTERS
31 28  CR=15  SCARRIAGE RETURN
32 29  LF=12  ILLNE FEED
33 30  SPACE#30  SPACE
34 31  EOF#31  END OF FILE
35 32
36 33  1 CARD READER CONTROL AND STATUS ATTR
37 34  READ#1  IREAD
38 35  EJECT#2  IEJECT CARD
39 36  INTERRU#3  Iinterrupt ENABLE
40 37  COLD#20  ICOLUMN NONE
41 38  READY#30  IREADY
42 39  BUSY1#10  IBUSY
43 40  ONLINE#200  IONLINE
44 41  MFILE#300  IDATA IATE
45 42  MERROR#200  IMERROR CHECK (CM11 ONLY)
46 43  MERROR#300  IMERROR CHECK (CM11 ONLY)
47 44  MERROR#4000  ICARD NONE
48 45  FRM#10000  IERROR
1 SET CR (NOT) CRF
2 APPEND/OR NOT APPEND CARriage RETURN/LTnE FEED TO EACH CARD IMAGE
3 SET CR (NOT) TRTM
4 TRTM/OR NOT TRTM TRAILING BLANKS FROM CARD IMAGES
5 SET CR (NOT) HANG
6 HANG/RETURN HARD ERROR IF READER NOT READY AT START OF TRANSFER
7 SET IO (NOT) IOF
8 IOF/OR NOT IOF COLUMN DATA IN IMAGE
9 SET IO (NOT) IOE
10 IOE/OR NOT IOE END-OF-IO OPTIONS
<table>
<thead>
<tr>
<th>Page</th>
<th>1 M24 CONVERSION TABLE TO ACCEPT M24 KEYPUNCH CODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A90606    012  117  SET0261.  BYTF  B1A, 137  TRACK ARROW  (\text{A-29})</td>
</tr>
<tr>
<td>7</td>
<td>B90606    013  075  EQUAL  (\text{A-31})</td>
</tr>
<tr>
<td>8</td>
<td>C90606    015  176  PLUS ARROW  (\text{A-42})</td>
</tr>
<tr>
<td>9</td>
<td>D90606    016  067  TAPOSTROPHE  (\text{A-41})</td>
</tr>
<tr>
<td>10</td>
<td>E90606    017  174  TRACKSLASH  (\text{A-37})</td>
</tr>
<tr>
<td>11</td>
<td>F90606    023  073  ISMETACOLON  (\text{A-21})</td>
</tr>
<tr>
<td>12</td>
<td>G90606    024  040  ILFFT PAREN  (\text{A-41})</td>
</tr>
<tr>
<td>13</td>
<td>H90606    025  042  EQUALS  (\text{A-51})</td>
</tr>
<tr>
<td>14</td>
<td>I90606    026  043  ILR, STGN  (\text{A-61})</td>
</tr>
<tr>
<td>15</td>
<td>J90606    027  044  IFPFRNT  (\text{A-71})</td>
</tr>
<tr>
<td>16</td>
<td>K90606    112  072  ICOLON  (\text{A-2})</td>
</tr>
<tr>
<td>17</td>
<td>L90604    115  173  ILBPCKFT  (\text{A-6})</td>
</tr>
<tr>
<td>18</td>
<td>M90604    116  074  IGREATERTHAN  (\text{A-6})</td>
</tr>
<tr>
<td>19</td>
<td>N90604    117  044  IAMPFRAND  (\text{A-7})</td>
</tr>
<tr>
<td>20</td>
<td>O90604    219  053  IPIUR  (\text{A-5})</td>
</tr>
<tr>
<td>21</td>
<td>P90670    219  077  IQDIETTON  (\text{A-6})</td>
</tr>
<tr>
<td>22</td>
<td>Q90670    218  051  IRIGHT PAREN  (\text{A-6})</td>
</tr>
<tr>
<td>23</td>
<td>R90670    215  174  IFRBPCKFT  (\text{A-6})</td>
</tr>
<tr>
<td>24</td>
<td>S90700    216  074  ILESSTHAN  (\text{A-6})</td>
</tr>
<tr>
<td>25</td>
<td>T90700    012  072  * END OF TABLE **</td>
</tr>
</tbody>
</table>

1 M29 CONVERSION TABLE TO ACCEPT M29 KEYPUNCH CODES
1 MODIFIES CHARACTER TABLE TO ACCEPT M29 KEYPUNCH CODES

<table>
<thead>
<tr>
<th>Page</th>
<th>1 M29 CONVERSION TABLE TO ACCEPT M29 KEYPUNCH CODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>190704    012  072  RETPOS.  BYTF  A13, 072  ICOLON  (\text{A-2})</td>
</tr>
<tr>
<td>31</td>
<td>190705    013  043  ILR, STGN  (\text{A-31})</td>
</tr>
<tr>
<td>32</td>
<td>190705    015  047  TAPOSTROPHE  (\text{A-41})</td>
</tr>
<tr>
<td>33</td>
<td>190705    016  047  EQUAL  (\text{A-42})</td>
</tr>
<tr>
<td>34</td>
<td>190705    017  042  PLUS ARROW  (\text{A-42})</td>
</tr>
<tr>
<td>35</td>
<td>190705    020  174  TRACKSLASH  (\text{A-37})</td>
</tr>
<tr>
<td>36</td>
<td>190705    023  044  ISMETACOLON  (\text{A-21})</td>
</tr>
<tr>
<td>37</td>
<td>190705    024  043  ILFFT PAREN  (\text{A-41})</td>
</tr>
<tr>
<td>38</td>
<td>190705    025  042  EQUALS  (\text{A-51})</td>
</tr>
<tr>
<td>39</td>
<td>190705    026  043  ILR, STGN  (\text{A-61})</td>
</tr>
<tr>
<td>40</td>
<td>190705    027  044  IFPFRNT  (\text{A-71})</td>
</tr>
<tr>
<td>41</td>
<td>190705    112  174  IAMPFRAND  (\text{A-7})</td>
</tr>
<tr>
<td>42</td>
<td>190705    115  173  IFRBPCKFT  (\text{A-6})</td>
</tr>
<tr>
<td>43</td>
<td>190705    116  073  IGREATERTHAN  (\text{A-6})</td>
</tr>
<tr>
<td>44</td>
<td>190705    117  040  IPIUR  (\text{A-5})</td>
</tr>
<tr>
<td>45</td>
<td>190705    219  051  IQDIETTON  (\text{A-6})</td>
</tr>
<tr>
<td>46</td>
<td>190705    218  051  IRIGHT PAREN  (\text{A-6})</td>
</tr>
<tr>
<td>47</td>
<td>190705    215  174  TRACKARROW  (\text{A-37})</td>
</tr>
<tr>
<td>48</td>
<td>190705    216  074  IFRBPCKFT  (\text{A-6})</td>
</tr>
<tr>
<td>49</td>
<td>190705    012  072  * END OF TABLE **</td>
</tr>
</tbody>
</table>

**IF GIF IS البشر - IDAAPT+216+1099, ERROR TABLE NOT IN BLOCK 1**
<table>
<thead>
<tr>
<th>Line</th>
<th>Memory Address</th>
<th>Machine Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
<td>000334</td>
<td>MOV BA,CHPTA</td>
<td>START FILLING FROM NEW CARD</td>
</tr>
<tr>
<td>92</td>
<td>000336</td>
<td>MOV BA,ENDPT</td>
<td>WHICH IS AS YET EMPTY</td>
</tr>
<tr>
<td>93</td>
<td>000338</td>
<td>MOV PA,HIPPT</td>
<td>INSERT BUFFER POINTER</td>
</tr>
<tr>
<td>94</td>
<td>000340</td>
<td>LOD CHRT1</td>
<td>CLEAR EOF FLAG</td>
</tr>
<tr>
<td>95</td>
<td>000342</td>
<td>MOV EY. (PC)+</td>
<td>ISPRT COLUMN POINT</td>
</tr>
<tr>
<td>96</td>
<td>000344</td>
<td>FOCNT: WARM</td>
<td>COUNT OF COLUMNS REMAINING IN CARD</td>
</tr>
<tr>
<td>97</td>
<td>000346</td>
<td>MOV SPE+INT+INT, START ISTART A CARD GIMN</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>000348</td>
<td>PTT PC</td>
<td>IAVE</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>ERROR MOV FROF, R5</td>
<td>IPOINT TO CURFELEMENT</td>
</tr>
<tr>
<td>102</td>
<td>000362</td>
<td>MOV SHERR, R5</td>
<td>ERROR MESSAGE</td>
</tr>
<tr>
<td>104</td>
<td>000364</td>
<td>MOV R5</td>
<td>IRENTEI</td>
</tr>
<tr>
<td>106</td>
<td>000366</td>
<td>MOV R6</td>
<td>IDATE IS ONLY CURABLE</td>
</tr>
<tr>
<td>108</td>
<td>000368</td>
<td>MOV IERROR</td>
<td>ISSUE HINT ERROR IF NOT</td>
</tr>
<tr>
<td>110</td>
<td>000370</td>
<td>MOV L4</td>
<td>IONE WITH DATA COLUMNS?</td>
</tr>
<tr>
<td>112</td>
<td>000372</td>
<td>MOV ERO</td>
<td>ISTART A NEW READ TO CORDER CONDITION</td>
</tr>
<tr>
<td>114</td>
<td>000374</td>
<td>MOV FPRD</td>
<td>IFE SF ASOMM CARD DONE</td>
</tr>
<tr>
<td>116</td>
<td>000376</td>
<td>MOV FPRD</td>
<td>IEND OF FILE CARD FOUND</td>
</tr>
</tbody>
</table>

CR.SYS RT-11 MPRN V4P2-1A 28-APR-75 16:01:17A PAGE 44
HANDLER proper

<table>
<thead>
<tr>
<th>Line</th>
<th>Memory Address</th>
<th>Machine Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>114</td>
<td>000412</td>
<td>ENTRT MMOV</td>
<td>PRINT INTO HIS BUFFER</td>
</tr>
<tr>
<td>116</td>
<td>000414</td>
<td>FPRD R4</td>
<td>CLEAR IT ALL</td>
</tr>
<tr>
<td>117</td>
<td>000416</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>000418</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>000420</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>000422</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>000424</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>000426</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>000428</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>000430</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>000432</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>000434</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>000436</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>000438</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>000440</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>000442</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>000444</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>000446</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>000448</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>000450</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>000452</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>000454</td>
<td>FPRD R4</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>000456</td>
<td>FPRD R4</td>
<td></td>
</tr>
</tbody>
</table>

RETURN TO MONITOR (REQUEST DONE, EOF, OR ERROR)
1 THE FOLLOWING MACRO TAKES AS ARGUMENTS THE ASCII TRANSLATION
2 OF A CHARACTER AND THE LIST OF PUNCH COMBINATIONS FOR THAT CHARACTER.
3
4 MACRO $LIST
5    $LIST
6      IF NE Y$;
7      IF IE Y$.
8      TST+Y$.
9      IFF
10      USTP Y$,
11      HENT
12      END
13      ENDC
14
15  ; THE FOLLOWING TABLE TRANSLATES 829 KEYPUNCH CODES TO ASCII.
16
17  ; CHRTBL:
18  ; REPT 55,
19  ; BYTE 130  ; DPC STANDARD FROMO CHARACTER
20  ; ENDR
21  ; CHRTBL
TITLE RT-V82-07 12-APR-76

RT-11 MARCH VM92-1N 28-APR-76 16:24:12 PAGE 1

AUGUST, 1976

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 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 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.
1 00000000  ;  CSECT SYMNAME
2          ； ENAMEL ISR
3 4 00000000
4          ； P0=29
5          ； P1=31
6          ； P2=32
7          ； P3=33
8          ； P4=34
9          ； P5=35
10         ； S0=29
11         ； SP=3A
12         ； PC=37
13         ； PA=17777A
14
15 17776   ； ADAPTER CONTROL REGISTERS
16          ； TCNT = 177760
17          ； TCST = 177760
18          ； TCSR = 177762
19          ； TCBO = 177764
20          ； TCA = 177766
21          ； TCVR = 310
22
23          ； CONSTANTS FOR MONITOR COMMUNICATION
24          ； M��F = 1
25          ； M��H = 50
26          ； OPESS = 7F
27          ； OPEST = 7F
28          ； GARYS, DSRYS, DRYS, DRYS, GRYS, GRYS, DSRYS, DRYS
29          ； GLOBAL INTFR, INTFN, INTNT
30
31         ； RX = 5
32          ； RX = 5
33          ； RX = 5
34          ； RX = 5
35          ； PR7 = 70
36          ； RX IS NOT RESIDENT
37          ； RX IS NOT RESIDENT
38          ； RX IS NOT RESIDENT
39          ； RX IS NOT RESIDENT
40          ； RX IS NOT RESIDENT
41          ； RX IS NOT RESIDENT
42
43
44
45
46
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 V01-01
/RT-11 V2 DRTCF INDEPENDENT TERMINAL HANDLER, KB
/
/DEC-11-ORKMA-D
/
/COPYRIGHT (C) 1975
/
/DIGITAL EQUIPMENT CORPORATION
/MAYNARD, MASSACHUSETTS 01754
/
/THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
/ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
/THF 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.
/
/THIS 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
/
RGB

IR-11 VP DEVICE INDEPENDENT TERMINAL HANDLER, KR, KR
/SCAN BE USED BY EITHER THE FOREGROUND 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) RETURN CAUSES THE REMAINDER
/OF THE INPUT BUFFER FOR THE CALLING READ REQUEST TO BE
/ZERO-FILLED, AND THE READ IS COMPLETED. THIS HANDLER
TRANSFERS ONE LINE AT A TIME, NO MATTER HOW LONG THE
INPUT BUFFER IS FOR THE READ REQUEST; THE UNUSED PORTION
OF THE BUFFER IS ZERO-FILLED; CARRIAGE RETURN ECHOS
CARRIAGE RETURN, LINE FEED, AND INSERTS CR AND LF CHARACTERS
IN THE BUFFER IF THERE IS ROOM; ELSE ONLY CR IS PLACED IN THE BUFFER.
2) Line FEED ECHOS IF LINE FEEDS AND INSERTS A FF CHARACTER IN
BUFFER.
3) ROUT ECHOES "\" AND DELETES THE LAST CHARACTER IN THE BUFFER.
4) IF THERE ARE NO CHARACTERS IN THE BUFFER, ROUT DOES NOT ECHO
AND IS IGNORED.
5) TAB ECHOS ENOUGH SPACES TO POSITION THE PRINT HEAD AT THE
NEXT TAB STOP, AND INSERTS A TAB CHARACTER IN THE BUFFER.
6) CTRL U ECHOS "^U" AND ERASES THE CURRENT LINE.
7) CTRL Z ECHOS "^Z" AND CAUSES THE HANDLER TO REPORT END-OF-FILE.
8) THE CTRL Z CHARACTER IS NOT INSERTED IN THE BUFFER.
9) THE LOW-SPEED READER WILL RUN 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 NECESSARY TO ECHO THE TABS WILL
CAUSE CHARACTERS FOLLOWING THE TABS TO BE LOST; TO DISABLE THE
ECHOING OF TABS, THE "SET" COMMAND CAN BE USED AS FOLLOWS:
"SET KB 10" WILL DISABLE TAB ECHOING, ALLOWING A TAPE
TO BE READ WITHOUT CHARACTER LOSS.
"SET KB NO" WILL ENABLE TAB ECHOING, FOR NORMAL KEYBOARD
INPUT; THIS IS THE DEFAULT.
10) WHEN THE HANDLER RECEIVE A READ REQUEST, A "^Z" CHARACTER IS
PRINTED IN THE LEFT MARGIN OF THE TERMINAL TO SIGNIFY THAT THE
HANDLER IS READY FOR INPUT; THIS CHARACTER CAN BE CHANGED, OR THE
PROMPT FEATURE CAN BE REMOVED, BY RE-ASSIGNING THE SYMBOL
"PROMPT" TO THE ASCII VALUE OF THE
DESIRED CHARACTER. SETTING PROMPT TO "^Z" WILL CAUSE NO CHARACTER
TO BE PRINTED.
11) IF NO READ REQUEST IS ACTIVE, THE HANDLER WILL NOT ACCEPT INPUT,
AND THE KEYBOARD WILL NOT ECHO. IF IT DOES ECHO, THE HANDLER IS
ACCEPTING INPUT.

THIS HANDLER CONTAINS CONDITIONAL CODE TO SUPPORT TERMINALS THAT
REQUIRE 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 "FILCNT"
TO BE THE TOTAL NUMBER OF NULLS TO BE ISSUED AFTER EACH OCCURRENCE
OF THE CHARACTER DEFINED BY "FILCHR". FOR EXAMPLE, TO PROVIDE
12 FILLER CHARACTERS AFTER A CARRIAGE RETURN, SET "FILCHR=12" AND
"FILCNT=12".
THE HANDLER IS INSTALLED VIA THE FOLLOWING PROCEDURES:

1. ASSEMBLE IT AS FOLLOWS:
   - DEFINE FILLER CONDITIONALS IF NECESSARY
   - IF MACRO
     *KR=KB
   - IF LINK
     .R LINK
     .R K.B.SYS=KB
   - INSTALL IT AS DEVICE "KB1", AS DESCRIBED IN SECTION XXXXX
     - OF THE RT=11 VP SOFTWARE SUPPORT MANUAL. REMEMBER THAT
     - THE VECTORS FOR THE TERMINAL MUST BE PROTECTED IN THE RT MAP
     - AS DESCRIBED IN THAT SECTION.
     - THE VALUES FOR THE VARIOUS TABLE ENTRIES SHOULD BE
       - HSIZE=VALUE OF SYMBOL "HSTZE" ON LAST LINE OF LISTING
       - PSIZE=0 (NON-FILE STRUCTURED DEVICE)
       - PRNAME=2429 (NAME FOR "KB1")
       - STAT=HIGH ORDER BYTE=0, LOW ORDER BYTE=ANY DEVICE NUMBER
     - AVAILABLE. NOT THAT IT CANNOT BE 4, A VALUE > 50
     - IS RECOMMENDED.

4. ONCE INSTALLED, KB1 WILL BE AVAILABLE WHEN THE SYSTEM IS REBOOTED.

THE HANDLER ITSELF IS ACTIVATED WITH READ AND WRITE REQUESTS, AS ARE ALL
RT=11 DEVICE HANDLERS, WHEN USING SYSTEM PROGRAMS WHICH OPERATE ON
LARGE BUFFERS. SEVERAL LINPS MAY ACUMULATE IN THE BUFFER BEFORE
THEY APPEAR ON THE TERMINAL AND THEN ALL AT ONCE. TO AVOID THIS PROBLEM,
EACH OUTPUT BUFFER CAN BE ZERO-FILLED AND SENT TO THE TERMINAL TO PRINT
EACH LINE. THE HANDLER WILL IGNORE NULLS ON OUTPUT.

IN FORTRAN, EACH LINE CAN BE FORCED IN OR OUT BY USING A REWIND
FOLLOWING EACH READ OR WRITE TO THE DEVICE. FOR EXAMPLE:

LOGICAL=1 INPUT=0
CALL ASSIGN (7, *KB1/C *)
WRITE (7,1)
REWIND 7
WRTEF (7,2)
REWIND 7
READ (7,3) INPUTL
REWIND 7
;
;
1 FORMAT = * * *
2 FORMAT = * * *
3 FORMAT = * * *

THE HANDLER CAN BE "RE-CONFIGURED" FOR VARIOUS VECTOR AND
ADDRESS ADDRESSES BY CHANGING THE ASSIGNMENTS OF THE SYMBOLS
IN B-B.E. AND "KB1" ON THE FOLLOWING PAGE. EDITING THESE TWO
SUFFIXES TO CHANGE ALL PLATING ADDRESSES.
VECTOR AND DEVICE REGISTER ADDRESSES: EDIT THESE TWO TO RECONFIGURE

80380  KBVEC=80B
80390  KBCSR=17A590

OTHER DEVICE ADDRESSES

803A0  TPVEC=KBVEC+4
803B0  KSRUF=KBCSR+2
803C0  TPCSR=KBCSR+6
803D0  TPRUF=KBCSR+6

CONSTANTS

OFFSET=278  OFFSET TO MONITOR COMPLETION ENTRY

EOF=0000B  EOF BIT IN CBW
PR=7548B  IPNH VALUE FOR PRIORITY 7
PR=6548B  IPNH VALUE FOR PRIORITY 4

HT=11  TAB
LF=12  LF IN FEED
FF=14  FORM FEED
CR=15  CARRIAGE RETURN=15
CTRU=25  ICTRL/U
CTRLL=32  ICTRL/Z
SPACE=40  SPACE
DEL=177  HIBOUT

ESLENGTH=29  LENGTH OF ECHO BUFFER

PROMPT=32  PRACTICE CHARACTER

80276  ISPT LS8 CODE

THE FOLLOWING IN THE HANDLER INTERFACE TO THE MONITOR SET_COMMAND.

FOR DETAILS OF INTERFACING TO THE ISPT COMMAND, SEE THE RT-11 V6 SOFTWARE

SUPPORT, MANUAL

80400  AECT
80400  =488
80400  MOV  (PC)+,R3

80400  OP=85H
80400  INC  R3

80400  MOV  R3,LSROPT
80400  RTS  PC
80400  RETURN TO SET PROCESSOR
THIS IS THE HANDLER HEADER AREA, USED BY FETCH AND THE
FQUIRE MANAGER TO STORE VARIABLES CRITICAL TO HANDLER OPERATION.

KBATR1: WORD  TPVEC  PRINTER VECTOR ADDRESS
WB: WORD  TPRINT  OFFSET TO PRINTER INTERRUPT SERVICE
WPR: WORD  TPRT
KBPGE1: WORD  $A  LAST QUEUE ENTRY
KBPGE1: WORD  $A  CURRENT QUEUE ENTRY

THE FOLLOWING IS THE TRANSFER INITIATION CODE.
THE FIRST WORD OF THIS ROUTINE IS THE ENTRY POINT, FOR ALL
TRANSFER REQUESTS, THE KEYBOARD VECTOR IS SET UP IF FETCH ONLY SETS THE
PRINTER VECTOR, AND THE PARAMETERS FOR THE TRANSFER ARE ESTABLISHED.

IF THE REQUEST IS A WRITE, CONTROL TRANSFERS TO THE PRINTER ROUTINE TO
OUTPUT THE FIRST CHARACTER FROM THE USER BUFFER. IF IT IS A READ,
THE WHOLE USER BUFFER IS ZEROED, A FLAG (READFL) IS SET TO
SAY THAT A READ IS IN PROGRESS, AND A PROMPT CHARACTER IS ECHOED.
THE TERMINAL BEFORE THE KEYBOARD INTERRUPT IS ENABLED.

MOV RC, RR
ADD #KINT++, RR
ICALCULATE ABSOLUTE ADDRESS OF KEYBOARD INTERRUPT SERVICE

MOV RB, #KRVFC
MOV #255, #KRVFC+2

CLRT: CLR READFL

MOV KBQGE, RS
POINT TO CURRENT ELEMENT

CMP (RS)+, (RS)+4
AND 4 TO (RS)+4

MOV (RS)+7, URBTR1
SET UP POINTER TO USER BUFFER

MOV (RS)+, URBTR1
SAVE ORIGINAL POINTER FOR LATER

ASL (RS)+
MAKE WORD COUNT INTO BYTE COUNT

MOV (RS)+, BYTCNT
SAVE COUNT

MOV (RS)+, BYTCNT
WHICH IS NEP IN THIS HANDLER

MOV TOUT, R4
IF NEGATIVE, WRITE TO PRINTER

MOV (RS)+, R4
BYTE COUNT TO R4

MOV $F5, (RS)+
USER BUFFER POINTER IN RS

CLRB (RS)+
ZERO USER BUFFER BEFORE STARTING TRANSFER

DEF R4

BRANCH IF NOT DONE

IFB "READ IN PROGRESS" FLAG

PROMPT INPUT WITH RH

JMP KBTN

ENABLE KEYBOARD INTERRUPT AND RETURN
IT THIS IS THE ADBORT ENTRY POINT, THE HANDEL IS ENTERED AT THIS ADDRESS.
IF THE MONITORS RECEIVES A REQUEST TO ABDORT ANY TO TRANSFER IN PROGRESS
PR ABORT.

IT THIS IS THE TERMINAL OUTPUT INTERRUPT SERVICE. AFTER ENTERING SYSTEM STATE,
IT DETERMINES IF THERE ARE ANY CHARACTERS IN THE ECHO BUFFER TO BE
PRINTED. IF NOT, IT THEN DETERMINES WHETHER A WRITE REQUEST IS IN PROGRESS
FOR A5, IF SO, THE NEXT CHARACTER IN THE USER BUFFER IS PRINTED.
IF NOT, THE INTERRUPT IS DISMISRSED.

IF THERE ARE CHARACTERS IN THE ECHO BUFFER, THE FIRST CHARACTER IN THE
ECHO BUFFER IS PRINTED INTO A5, THE LIST IN THE ECHO BUFFER IS "SLID UP"
BY ONE CHARACTER, AND THE CHARACTER IN A5 IS THEN PRINTED.

IF THE FILLER CONDITIONAL CODE IS INCLUDED AT ASSEMBLY TIME, THE
CHARACTER IN A5 IS COMPARED AGAINST THE CHARACTER TO BE FILLED AFTER.
IF THE SAME, A COUNT OF NECESSARY FILLS IS STUFFED IN "FILCN" AND THE
CHARACTER IS PRINTED. THE INTERRUPT SERVICE THEN CHECKS THE NUMBER
OF FILLS NECESSARY AT THE FIRST ITEM, AND PRINTS NULLS IF ANY ARE LEFT

TPRENT; SRP RS, SRP; SP
ENTER SYSTEM STATE

IF THE PRINTER READY
YES=THEN WAIT FOR INTERRUPT TO PRINT ANYTHING
YES=THEN WAIT FOR INTERRUPT TO PRINT ANYTHING

IFPC FILCH
FILLER CODE FOR FILLER

FBV FILCN1
FILLER NEED TO BE OUTPUT?

BRANCH IF NOT

YES=DECREASE NUMBER BY ONE

NULL IS FILLER

PRINT IT

ICALL: ABSOLUTE ADDRESS

MOV PC RS

MOV (RS),R4

MOV (RS),R4

MOV @LENGTH, (SP)

MOV @ECHO, (SP)

MOV (SP), (SP)+

MOV (SP)+

DONF=CLEAN UP STACK

AND PRINT CHAR

IF THERE ARE READING OR WRITING
IF BRANCH IF READING

GET CHAR FROM USER BUFFER INTO A4

BRANCH IF TRANSFER COMPLETE

DON'T PRINT NULLS

DON'T PRINT NULLS
KB.MAC V01-01 RT-11 MACRO VM2-89 APR-75 I193151 PAGE 6+

58  IRQUEST TERMINATION AND ABORT CODE
59  IF THIS ROUTINE IS ENTERED WHEN THE I/O TRANSFER IS
60  COMPLETED OR ABORTED, THE DEVICE INTERRUPTS ARE DISABLED, AND
61  THE STANDARD MONITOR COMPLETION ENTRY CODE IS EXECUTED.
62  ABRQT: CLR #TPCSR
63  DONE1: CLR #KCSR
64  MOV PC,RA
65  MOV #I4..RA
66  ADD #KCODE,.RA
67  MOV #I4,.R4
68  JMP #OFFSET(RS1)

KB.MAC V01-01 RT-11 MACRO VM2-89 APR-75 I193151 PAGE 7

1  KEYBOARD INTERRUPT SERVICE
2  THIS IS THE KEYBOARD INTERRUPT SERVICE ROUTINE, AFTER ENTERING
3  THE SYSTEM STATE, IT GETS THE TYPED CHARACTER INTO RA, THEN
4  PROCEEDS DOWN A CHAIN OF CHECKS FOR THE SPECIAL CASE CHARACTERS
5  ($040, Ctrl U, Ctrl Z, CR, FF). IF IT IS ONE OF THE SPECIAL
6  CHARACTERS, THE ROUTINE "ECHO" IS CALLED TO ECHO APPROPRIATE
7  CHARACTERS ON THE TERMINAL, THEN APPROPRIATE ACTION FOR THE SPECIAL CASE
8  IS TAKEN. IF A NORMAL CHARACTER IS TYPED, IT IS ECHOED AND PLACED
9  IN THE INDICATOR BUFFER BEFORE THE INTERRUPT IS DISMISSED.
10  INRTNT: JR #5,#INPTR
11  EENTER SYSTEM STATE
12  CLR #R4 & #PR
13  MOVB #KBBUF,.R4
14  MOV #I4,.R4
15  #I776,.R4
16  EENTER TO SEVEN BITS
17  #I8,.R4
18  #DELET
19  #I4 THIS CHARACTER A RUBOUT?
20  RNF 11
21  BRANCH IF NOT
18 080526 11##07 000004 2141 MOV R4,208
19 080532 001167 000030 JSR R1,ECHO
20 080536 000 377 2081 .BYTE 0,377
21 080540 11##07 000176 MOV R4,#SUBPTR
22 080544 000267 00172 INC SUBPTR
23 080550 004367 000142 DEC RTNCY
24 080554 001631 RED NONE
25 080556 012775 000101 175500 KBTHI MOV #THI,#KACAR
26 080560 000287 RT8 PC

KB.MAC V01-01 RT-11 MACRO VNR2-04 R-APR75 12133151 PAGE 9

18 SUBROUTINE ECHO:
19 THIS SUBROUTINE SERVES TO PLACE THE SPECIFIED CHARACTERS IN THE
20 ECHO BUFFER, AND START THE PRINTER IN CASE IT IS IDLE.
21 THE CALLING SEQUENCE IS
22 JSR R1,ECHO
23 BYTE CHARR1,CHAR2,CHAR3,...CHARN,377
24 ON ENTRY, R4 CONTAINS THE CHAR TYPED AT THE KEYBOARD.
25 NOTE THAT THERE MUST BE AN EVEN NUMBER OF BYTES IN THE ARGUMENT LIST
26 AND THEREFORE THE NUMBER OF CHARACTERS EXCLUDING THE 377
27 MUST BE ODD.
28 WHEN ENTERED, ECHO SCANS THE ECHO BUFFER TO FIND THE END OF THE
29 ECHO LIST, WHICH IS MARKED BY A NULL BYTE, WHEN THE END OF THE LIST
30 IS FOUND, IT DETERMINES IF THERE ARE AT LEAST 8 FREE SLOTS IN THE LIST
31 IT THEN ACCOMODATES A POSSIBLE LINE FEED OR FORM FEED, IF NOT, THE
32 CHARACTER JUST TYPED IS IGNORED. IF SO, THE CHARACTERS FROM THE
33 ARGUMENT LIST FOLLOWING THE CALL ARE INSERTED IN THE BUFFER.
34 THE PRINTER IS STARTED IF IT IS IDLE, AND THE ROUTINE RETURNS.
35 NOTE THAT IF A SPECIAL CASE, IF R4 CONTAINS A TAB CHARACTER
36 WHEN THIS ROUTINE IS ENTERED, THE ARGUMENT LIST IS NOT USED. RATHER,
37 AN APPROPRIATE NUMBER OF SPACES TO MOVE THE PRINT HEAD TO THE
38 NEXT TAB STOP ARE PLACED IN THE ECHO BUFFER, AND THE ROUTINE RETURNS
39
40 6N VENAL LSR
41 ECHO:
42 MOV PC,RS
43 ;CALC ABSOLUTE ADDRESS
44 MOV #RSRT-1,RS
45 ;SET ECHO BUFFER
46 ADI #0
47 ;SAVE ADDRESS OF ECHO BUFFER
48 MOV R5,TEMPP
49 ADD #LENGTH-1,TEMPP
50 ;TEMP POINTS TO END OF ECHO BUFFER
51 MOV (RS)+,R5
52 ;THIS END OF ECHO LIST?
53 JNZ 48
54 ;BRANCH IF NOT
55 MOV (SP)+,R1
56 ;YES=RS POINTS TO FIRST FREE SLOT IN ECHO LIST
57 MOV RS TEMP
58 ;FIND NUMBER OF FREE SLOTS IN ECHO LIST
59 MOV TEMP,#8
60 ;IS THERE ENOUGH ROOM TO ECHO TAB OR FF?
61 JNZ 55
62 ;BRANCH IF YES
63 MOV (SP)+,R1
64 ;IGNORE THIS CHAR THEN
65 MOV R1,RT8
66 ;RETURN TO MONITOR
67
SYMBOL TABLE

ABORT = 00234R
DELET = 000177
DELETE = 000177
SPACE = 000640
PROMPT = 000640
TPINT = 000720
TPVEC = 000720
UNITR = 000720
UNITR1 = 000720
UNITR2 = 000720
UNITR3 = 000720
UNITR4 = 000720

RTCNT = 00736R
NONE = 00248R
LF = 000360
PC = 000047
READFL = 000740
P2 = %00002
SPACE = 000640
TPINT = 000720
UNITR = 000720
UNITR1 = 000720
UNITR2 = 000720
UNITR3 = 000720
UNITR4 = 000720

CR = 000015
FBDEND = 000024
LFRT = 000012
PRTR = 000047
UNITR = 000047
UNITR1 = 000047
UNITR2 = 000047
UNITR3 = 000047
UNITR4 = 000047

CTRLU = 006275
CTRLA = 006275
CTRLB = 006275
CTRLC = 006275
CTRLD = 006275
CTRLE = 006275
CTRLF = 006275
CTRLG = 006275

EOF = 020090
KCON = 176592
KBSTRT = 000000
KDDOR = 000000
KBBSIZE = 000000
KBBST = 000000
OPDLR = 000000

ABSF = 000424
ERRORS DETECTED: 0
FREE STORAGE: 19440 WORDS

KB, LPI/NTW/C=64
APPENDIX C
VERSION 1 EMT SUMMARY

Although Version 1 programmed requests are supported by Versions 2, 2B, and 2C 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/V2C 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/V2C requests generally accept a set of arguments, or an argument in R0.

2. V1 channel numbers are restricted to $16_{10}$. Also, the channel number in V1 is not a legal assembler argument; it is merely an integer in the range 0 to $15_{10}$.

3. V1 requests are non-entrant 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/2C 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>Table C-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1 Programmed Requests</td>
</tr>
<tr>
<td>V1 - Format Same as V2/V2B</td>
</tr>
<tr>
<td>.CSIGEN</td>
</tr>
<tr>
<td>.CSISPC</td>
</tr>
<tr>
<td>.DATE</td>
</tr>
<tr>
<td>.DSTAT</td>
</tr>
<tr>
<td>.EXIT</td>
</tr>
<tr>
<td>.FETCH</td>
</tr>
</tbody>
</table>

(continued on next page)

C-1 January 1976
### 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/V2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>.HRESET</td>
<td>.READW</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>.RELEAS</td>
<td>.WRITE</td>
</tr>
<tr>
<td>.SETTOP</td>
<td>.WRITC</td>
</tr>
<tr>
<td>.RESET</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 .crtn is required only for .READC
.READC .chan,.buff,.wcnt,.crtn,.blk .crtn is required only for .READC
.RENAME .chan,.dblk
.REOPEN .chan,.cblk
.SAVESTATUS .chan,.cblk
```
.WAIT .chan

.WRITE }
.WRITC } .chan,.buff,.wcnt,.crtn,.blk [ .crtn is required only for .WRITC ]
.WRITW }

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>  
   F> RF:*.LST<CR>

This program assumes device DK: and extension .LPT unless otherwise indicated.
**TITLE**  LSPOOL - LINE PRINTER SPOOLER

**SATTL A USEFUL FOREGROUND PROGRAM**

THIS PROGRAM FOR THE FOREGROUND IS A LINE PRINTER SPOOLER.
IT SEARCHES A SPECIFIED DEVICE FOR FILES WITH A PARTICULAR
EXTENSION (THE DEFAULT IS LPT) AND PRINTS THEM, DELETING
AFTER PRINTING. IF NONE ARE FOUND, IT WILL GO TO SLEEP FOR
HALF A MINUTE, PERMITTING THE BACKGROUND TO RUN.

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

F.G.

LOAD LP,RF

FRU LSPOOL

LSPOOL WILL TYPE: "DEVICE TO SPOOL?"
TYPE INPUT DEVICE AND FILE DESCRIPTION, F.G.:
RFI,*.LST

MCAILL V2,.RERDEF

MCAILL RFIND,.WRTH,.LOOKUP,.DELETE,.CPL,.TTYIN

MCAILL PRINT,.TTYOUT,.RESET,.PC alliances,.CLOSE,.EXIT

MCAILL STATUS,.WAIT

V2

RERDEF

USPWR = 46

FRBYT = 52

CR = 15

LF = 12

START: MOV #BUFF,#USPWR

SAVE STACK POINTER FOR RESET

MOV SP,(PC)+

MAKE USP SWAP OVER RUFF

STKSAV: MOV #DOR,LA

MUST BE IN MEMORY.

PC = 15

ILLLEGAL DEVICE

TST TOP+2

FIRST ENTRY POINT

TST #LP

HP TO BEGIN IF LOADED

LP NOT IN MEMORY!
.OPEN CHANNEL TO READ DIRECTORY

1P = » BUFFER
1NIT R3
1MULTIPLY R2 BY 2
1AND R4 AND 4 TO GET BLOCK NUMBER
1OF DIRECTORY SEGMENT (STARTING
1AT BLOCK 4 OF DEVICE).

.READW #1NB,#9,#R2,#10000,0,03 READ A DIRECTORY SEGMENT
1PC = RANEPR
1ERROR READING DIRECTORY!!
1COPY PAST DIRECTORY HEADER
1INT FOR END OF SEGMENT
1BRANCH IF MORE TO GO
1IFGHT LINK TO NEXT SEGMENT
1BRANCH IF ANOTHER SEGMENT EXISTS,
1ELSE WAIT A WHILE
1IF THERE A FILE
1IF Temporary FILE?
1THEN EXTENSION MATCH?
1FILE, GO PRINT IT.
1PRINT A MESSAGE
1THEN TRY NEXT SEGMENT

.LSPool = LINE PRINTER SPOOLER
A USEFUL FOREGROUND PROGRAM

1 THIS ROUTINE PRINTS THE SPOOLED FILE JUST FOUND AND THEN DELETES IT.

COPYR: MOV (R5)+,2,(R1)    ;COPY FILE NAME AND EXTENSION
MOV (R5)+,4,(R1)           ;INTO FILE DESCRIPTOR BLOCK
MOV (R5)+,6,(R1)           ;FOR A LOOKUP.
,LOOKUP #1NB,#2,#R1       ;LOOKUP THE FILE ON CHANNEL 2.
SOMETHING FUNNY HAPPENED
1OK, COPY IT, R5 IS BLOCK #
1READW #1NB,#2,#R2,#10000,0,03 READ 1000 WORDS
1COPY ACTUAL WORD COUNT TRANSFERRED
127 000630  WRTH #FINB,#1,PR,PR,PS1 AND WRTH IT TO CHANNEL 1
129 000672  TST (RS)+
131 000714  JMP 104
133 000756  JR 158
135 000798  JS: TSTB #FRBYT I WAS ERROR JUST AN EOF?
137 000840  JS: CLOSE #2 I NOCPOSE THE BLOCK AND
139 000882  JS: PRINT #ERRIN I REPORT AN INPUT ERROR
141 000924  JS: JMP FNDLP I THEN FIND ANOTHER FILE
143 000966  JS: JMP 104 I ION EOF, CLOSE THE FILE
145 001008  JS: CLOSE #2 I THEN CONTINUE
147 001050  JS: DELETE USING AN INACTIVE CHANNEL
149 001092  JS: ENTRY BLOCK

161 001130  JS: ISPOOL OUTPUT DEVICE
163 001172  JS: ENTRY ARGUMENT BLOCK
165 001214  JS: LIST RIN
167 001256  JS: TIMBLK  WRLN  0,60,15.
169 001308  JS: NDEFIX  RADS2 /LPT/
171 001348  JS: CSBLK  BLKW  16.
173 001390  JS: RUFF  BLKR  16000
175 001432  JS: END START

LSPOOL = LINE PRINTER SPOOLER
RT-11 MACRO VM02-11 26-NOV-75 0017121 PAGE 1+
SYMBOL TABLE

January 1976
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.
R

MAKE DEFAULT DEVICE = SY

FILE

---PICK UP FILE DESCRIPTOR

DEVICE OTHER THAN SY SPEC?

Y BADCOT

N

CCBBØ

READ BLOCK Ø

INIT TO READ FILE STARTING AT 1000

BEGIN

CHAINED TO ?

Y

N

SOFT RESET

SET USER SP & LOC. 5Ø

PROG. TOO LARGE ?

Y

?OVR COR?

N

ENTRPG

CODE LEFT TO READ ?

Y

SET UP A .READW FOR REMAINING CODE

RDOVLY

THE PROGRAM MAY BE ALL WITHIN 5ØØ-776

INTO RMON TO READ THE REMAINDER; ENTER USER'S PROGRAM
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?
N
Y

CLRCB
CLEAR INTERNAL CONTROL BLOCK, CLEAR LOC. 5Ø

SET FLAG, NOT FIRST GET CALL

.LOOKUP THE FILE

FOUND?
N
Y

CCBBØ
GET FILE'S BLOCK Ø INTO KMON

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

360-377 CONTAINS A BITMAP OF BLOCKS USED BY THE FILE.

A

READ THE FILE INTO MEMORY
N
OVERLAYS KMON OR USR?
Y

FIRST BLOCK OVERLAYS 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 RMON MEMORY BLOCK

RTS PC

A
```
REENTER/RUN/START

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

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

--- GET THE MEMORY IMAGE INTO LOW MEMORY AND SCRATCH BLOCKS, IF NECESSARY.
E.1.1 KMON Subroutines
OVREAD/OVLINK

OVREAD - Used to read overlay command processors into memory.

1. OVREAD
2. SET params for .READW of OVERLAY
3. ALREADY IN MEMORY?
   - Y
   - N
     1. MARK THIS OVERLAY AS RESIDENT
     2. $SYS
        1. INTO RMON TO READ THE OVERLAY
        2. RTS PC

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

1. OVLINK
2. OVREAD
3. JMP @R3
   1. READ IN THE OVERLAY
   2. 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.
SAVEVC/FILE

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.
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.

```
CCBBØ

READ A BLOCK INTO THE USR BUFFER (BLOCK Ø)

MOVE WORDS FROM FILE TO LOW MEMORY IF THE WORD IS UNPROTECTED

DONE UP TO 500 YET?

N

DOING A CHAIN?

Y

THIS PROG. ACCEPT CHAIN?

Y

COPY FILE'S MEMORY BLOCK INTO RMON'S MEMORY BLOCK

RTS PC

N

MOVE 500-776 OF FILE INTO REAL 500-776

A

A

E-15
```
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.

SYSK

MARK NO DIRECTORY IN USR BUFFER

?OVR COR?

N

IS OPERATION OF PROPER SIZE?

Y

$SYS DO THE I/O OPERATION

RTS PC
E.1.2 KMON Overlays
LOAD

A

GET A DEVICE NAME AND MAP TO PHYSICAL NAME

OWNER ASSIGN?

N

Y

DETERMINE OWNER'S JOB #

2ØS

DO .DSTAT ON DEVICE

.DSTAT ERROR?

N

Y

HANDLER RESIDENT?

N

Y

ALLOCATE SPACE AND FETCH THE HANDLER.

PROTECT THE HANDLER'S VECTORS

2ØS

6Ø

DETERMINE DEVICE TABLE INDEX #

COMPUTE UNIT #; NONE IMPLIES UNIT Ø

SYSTEM DEVICE?

N

Y

SYSTEM UNIT?

N

Y

?ILL DEV?

1ØS

FILE STRUCTURED DEV.?

N

Y

SET TO ASSIGN JOB Ø TO ALL UNITS OF DEV.

ASSIGN OWNERSHIP TO ALL OCCURRENCES OF DEVICE

A DEVICE MAY HAVE SEVERAL ENTRIES IN OWNERSHIP TABLE (E.G., SY:, DK:, RK:)

END OF COMMAND?

N

A

RETURN

F/B

F/B
GET/PUT A BLOCK OF MEMORY

GETBLK
ADD 2 BYTES TO REQUEST FOR SIZE WD.
POINT TO FREE MEMORY LIST
1$
GET NEXT ELEMENT IN LIST

END OF LIST?
Y 4$
N
BLOCK BIG ENOUGH?
Y
EXACTLY THE SAME?
Y
REMOVE FROM LIST
5$
PUT SIZE IN WORD Ø OF BLOCK
RETURN

PUTBLK
POINT TO SIZE IN FIRST WORD OF BLOCK
POINT TO FREE MEMORY LIST
1$
GET NEXT ELEMENT IN LIST

END OF LIST?
Y 3$
N
NEXT BLOCK HIGHER?
Y
HI BLOCK IS CONTIGUOUS?
Y
CONCATENATE NEW BLOCK WITH HI BLOCK
N
JUST linkage IN NEW BLOCK

CONTIGUOUS WITH LO BLOCK?
Y
CONCATENATE NEW BLOCK & LO BLOCK
N
CAN RECLAIM MEMORY?
Y
A
N
RETURN
GT ON/OFF

GT

VT11 HARDWARE PRESENT?

Y

FG ACTIVE?

Y

?ILL CMD?

N

GT OFF?

Y

A

N --- GT ON

ALREADY THERE?

Y

?ILL CMD?

N

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

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 KMON 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/FATAL/CDFN (CONT.)

USRBUF is the initial entry point for USR calls when the USR has just been read into memory. LOCATE sets up pointers into RMON.

- **RTORG**
  - USRBUF

- **LOCATE**
  - UPDATE POINTERS TO RMON

- **JMP (R5)**
  - TRANSFER TO BODY OF USR

--- **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.

- **FATAL**

--- **ENTERS WITH R4 INDICATING ERROR CODE.**

- **LOCATE**
  - UPDATE USR POINTERS

- **GET POINTER TO PROPER ERROR MESSAGE**

- **PRINT THE ERROR MESSAGE AND ERROR PC**

- **DISALLOW THE REENTER COMMAND EXECUTE .EXIT**
CDFN - 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.
USR CODE

CALLING THE USR FROM A COMPLETION ROUTINE IS ILLEGAL, AS THE USR COULD HAVE BEEN INTERRUPTED.

SAVE CERTAIN PARAMS; GET pointer to function

QSET
DELETE
FETCH/RELEASE
CLOSE
ENTER
LOOKUP
RENAME
DSTAT
CSI

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

CDFN
HARD/SOFT RESET
RENAME

TURN ON RENAME BIT IN CHANNEL WORD

--- THIS SERVES AS RENAME FLAG.

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

DO COMMON CODE AT CLOSE.

N

FILL IN CSW AREA

COMXIT

CLOCOM
LOOKUP/RENAME (CONT.)

----HERE ON NONFILE-STRUCTURED LOOKUP

DELOUT

LNFILE

----CHANNEL AREA WILL HAVE A FILE OPEN, WITH STARTING BLOCK ≠ θ.

CLEAR OUT STARTING BLOCK # IN CHANNEL AREA

COMXIT

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

SPLOOK

RENAMESDONE?

Y

SPSHLL1

CODE 3

COMXIT

N

GENERATE EMT ERROR #1; CLOSE CHANNEL

LKER1

A RENAME ON A SPECIAL DEVICE IS CURRENTLY ILLEGAL.

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

SPDEL

SPSHLL1

ERROR?

N

DELOUT

Y

LKER1

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

USRCOM

---- DO COMMON CHANNEL SETUP.

NONFILE DEV. \(\rightarrow\) DELOUT

SPDEL

---- SPECIAL DEVICE (MT/CT)

DLEET

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

LKER1

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

MAKE THE ENTRY AN 'EMPTY'

CLSQSH

---- FINISH UP IN CLOSE CODE.
ENTER

USRCOM
DO CHANNEL SETUP

NONFILE
SPECIAL DEV.
SPENTR

CONSOL

---CONSOLIDATE THE DIRECTORY SEGMENT IN MEMORY.

ENTRY
GET NEXT EMPTY SPACE
NONE LEFT

NXBLK
READ IN THE NEXT DIRECTORY SEG

ZERO LENGTH REQUEST
NO MORE
Y
OK

2S
TAKE 1/2 LARGEST OR ENTIRE SECOND LARGEST EMPTY

11S
MAKE THIS REQUEST LOOK LIKE ENTER FOR THIS FIXED LENGTH

2S
RENTX

12S

6S
UPDATE LARGEST & SECOND LARGEST FILES LIST

THESE KEPT TRACK OF WHERE THE LARGEST AND SECOND LARGEST EMPTY SPACES ARE LOCATED.

3S
INCR BUMP TO NEXT ENTRY IN DIRECTORY

THE CORRECT SEGMENT WAS RECORDED WHEN THE EMPTY WAS FOUND.
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

OVERFLOW ON THIS ENTER?

Y

EXTEND

WE MUST EXPAND THE DIRECTORY IN THIS CASE.

N

PUT A TENTATIVE ENTRY AT THE CORRECT SPOT

FILL IN THE CHANNEL STATUS AREA

SEGRW2 REWRITE THIS SEGMENT

COMXIT

6$

N

SPENTR

SPESHL

CODE 4 (ENTER)

COMXIT
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/FETCH/RELEASE

DSTAT - GET DEVICE STATUS

GESTAT

LX4DEV
SEARCH TABLES FOR DEVICE NAME

NOT FOUND

EMT ERROR

COMXIT

FILL IN 4 WORDS FROM TABLES

COMXIT

FETCH/RELEASE

PHETCH

LX4DEV
FIND NAME IN TABLES

NOT FOUND

EMT ERROR

COMXIT

ILLEGAL DEVICE

RELEASE

Y

COMXIT

IS HANDLER PERM. RESI?

Y

COMXIT

--- RELEASING SYSTEM HANDLER OR ONE LOADED IS A NO-OP.

N

CLEAR THIS HANDLER'S ENTRY POINT

ALREADY RESIDENT?

Y

N

LOAD ADD. -400?

A

COMXIT
F JOB CANNOT FETCH Handler
-----WHICH WAS NOT LOADED.
CLOSE

KLOSE

IS THIS MT/CT?

Y

SPESHL

GO TO HANDLER FOR THE CLOSE

DELOUT

N

BLKCHK

GET A DIRECTORY SEGMENT INTO MEMORY

START WITH SEGMENT #Ø.

2$

ENTRY

GET FIRST TENTATIVE ENTRY.

NO MORE

N

NXDLK

READ NEXT DIRECTORY SEGMENT

NO MORE

DELOUT

ENTRY

GET FIRST TENTATIVE ENTRY.

NO MORE

NXDLK

READ NEXT DIRECTORY SEGMENT

DELOUT

INCR1

POINT TO NEXT ENTRY IN SEGMENT

2$

TENTATIVE NOT ASSOCIATED WITH OUR CHANNEL/JOB.

CLOCOM

SAVE POINTERS TO THIS ENTRY.

(I.E., PERMANENT FILE WITH SAME NAME)

CLSQSH

IS THERE A PERM. ALREADY ON DEVICE?

Y

MARK THE OLD ONE AS AN EMPTY.

ARE OLD & NEW IN THE SAME SEGMENT

N

CONSOLIDATE THE SEGMENT AND THEN REWRITE IT.

N

GET THE CORRECT SEGMENT BACK IN MEMORY

A

NO TENTATIVE OF THAT NAME WAS FOUND.
CLOSE (CONT.)/QUEUE EXTEND

A

THIS A RENAME?

Y

RE-INSERT NEW FILE NAME

N

ADJUST FINAL LENGTH OF FILE & TRAILING EMPTY SPACE.

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?

Y

Close first nine channels.

N

Use terminal input?

N

Put user's string into CSI line buffer.

Y

Output a prompting "*"

Collect console string; put into CSI line buffer.

Special mode?

Y

Zero the 39-word output area.

N

Is there output side?

Y

--- Process output side

N

STRTIN

Set flags for input side.

A

E-46
CSI CODE (CONT.)

A

NXTFIL

>9 FILES?

Y

?ILL CMD? -- SYNTAX ERROR

N

SPECIAL MODE?

Y

SPECIAL

N

GETFD

GET A FILE DESCRPTOR

NULL FILE NAME?

Y

SWITCH -- CHECK FOR SWITCH VALUES

N

FETCH THE HANDLER REQUIRED.

AN INPUT FILE?

Y

LOOKUP THE FILE - POSSIBLE FILE NOT FOUND; RETURN

N

ENTER THE FILE

SWITCH

SWITCH
CSI CODE (CONT.)

SPECIAL

GETFD

GET FILE DESCRIPTOR.

INPUT FILE?

Y

OUTSTUF

CHECK THAT OUTPUT FILE NOT BEING OPENED NONFILE

N

SWITCH

IS NEXT CHAR. A / ?

N

NOSWIT

Y

IS THERE A SWITCH VALUE ??

N

SET BIT 15 OF SWITCH WORD; SAVE VALUE OF SWITCH

Y

PUT FILE NUMBER INTO SWITCH WORD; BUMP SWITCH COUNT

THIS ALLS FOR THE /X:1:2:3 CONSTRUCTION

NOSWIT

IS NEXT CHAR A COMMA ?

Y

NXTFIL

GET NEXT DESCRIPTOR

N

IS IT AN = ?

Y

STRTN

NEW PROCESS INPUT FILES

N

IS IT END OF LINE?

Y

RETURN

N

ERROR ?ILL CMD?

-- RESTART CSI IF TERMINAL INPUT, ELSE RETURN WITH USER ERROR.
RETURN

RESTORE THE USER'S STACK SAVED ON ENTRY TO CSI

F/B

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.
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:

MAKE THIS DEVICE THE NEW DEFAULT DEVICE.

GETNAM

GET FILE NAME.

IS THERE AN EXTENSION?

USE DEFAULT EXTENSION

GETNAM

GET EXTENSION

RTS PC

NOTE: GETNAM will pick up a string of 0-6 characters and pack them in up to 2 words of .RAD5@.

WE ALREADY HAVE THE FILENAME NOW. GET THE EXTENSION.

USE CURRENT DEFAULT DEVICE NAME.

SKIP OVER DEVICE & FILE NAME

RTS PC

GETNAM - Converts a string of 0-6 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 - This routine is used to prepare a channel for I/O operations.
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

' SPECIAL' EXIT

N

NONFILE
DEVICE EXIT

RTS R5

RTS R5

RTS R5
DLEET/NXBLK

DLEET - This routine scans a device directory to find a file of a specified name.

```
DLEET
  \->
  B|LKCHK
  \->
  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
    \-- THE START BLOCK OF THE FILE IS UPDATED FOR USE IN LOOKUP
  \->
  NOT IN THIS SEGMENT
  \->
  NXBLK
  \->
  GET NEXT DIRECTORY SEGMENT
  \->
  ALL DONE
  \->
  RTS R5
  \->
  NO FILE FOUND
```

NXBLK - Gets the next in the series of directory segments, if one exists.

```
NXBLK
  \->
  IS THERE ONE MORE SEGMENT?
    N \->
    RTS R5
    Y \->
    PUT SEGMENT # INTO CSW FOR BLKCHK
    \-- BLKCHK IS TREATED AS A CONTINUATION OF NXBLK, ALTHOUGH IT IS A SUBROUTINE ITSELF
```
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 - 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

THE ARGS ARE PUT ON THE STACK IN THIS CASE.

RTS R5
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.

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

ALL UNUSED TENTATIVES ARE NOW EMPTIES. NOW COMBINE MULTIPLE EMPTIES TOGETHER.

IS THIS TENTATIVE ENTRY OK?

Y

MARK THIS ENTRY EMPTY

ADVANCE TO NEXT ENTRY

N

POINT TO TOP OF SEGMENT.

ENTRY

POINT TO FIRST EMPTY.

DONE

RTS PC

POINT TO THE NEXT ENTRY

IS THIS ONE LENGTH ≠ ?

N

IS NEXT ENTRY AN EMPTY ?

Y

COMBINE THE EMPTIES' LENGTHS & SQUEEZE SEGMENT

5$

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?

Y ERROR

RTS R4

N

IS IT A USER ASSIGNED NAME?

N

POINT TO PERMANENT NAME TABLE.

IS DEVICE HE SPECIFIED LEGAL?

N

ERROR

RTS R4

Y

SET UP TABLE POINTERS & DEVICE INDEX FOR RETURN

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.
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.

CSI-GENERAL MODE MODE

CSI-SPECIAL MODE

HARD RESET

SOFT RESET

DEVICE STATUS.

RENAME

LOOKUP

ENTER

CLOSE

FETCH

DELETE

CDFN

SET I/O QUEUE

CLR R4. FLAG FOR GEN. MODE OF CSI

C$SIGN

INC R2

INC R2

INC R2

INC R2

INC R2

INC R2

INC R2

INC R2

INC R2

A

R4 IS NORMALLY NON-ZERO; IT IS CLEARED HERE FOR DISTINCTION BETWEEN CSI GENERAL MODE AND CSI SPECIAL MODE.

CALUSR

GET USR INTO MEMORY

---USR NOW IN MEMORY

JMP @USRLOC

A
E.4.1 EMT Processors
SET TRAP/SAVE STATUS

SET TRAP ADDRESS

T$RPST

IS THERE AN ADDRESS?

N

Y

PUT IN THE ADDRESS OF MONITOR'S INTERNAL ROUTINE

STORE THE TRAP ADDRESS IN LINE

EMTOUT

SAV STATUS

S$AVST

IS CHANNEL OPEN?

N

Y

EMT ERROR #1; SAVESTAT ILLEGAL

Y

A ENTER DONE?

N

SAVE 5 WORDS OF CHANNEL; DEACTIVATE CHANNEL

EMTDON
REOPEN

R$OPEN

CHANNEL IN USE?

Y

EMT ERROR Ø
CHANNEL IS IN USE ALREADY

N

RESTORE 5 WORDS OF STATUS TO INDICATED CHANNEL

EMTDON

CLOSE

C$CLOSE

--- IF A LOOKUP WERE DONE, THE USR IS NOT REQUIRED.

C$LOS2

Y

DO WE NEED TO REWRITE DIRECTORY?

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
READ

R$READ

TSWCNT
COMMON
READ/WRIT
CHECKS

--- TSWCNT HAS 3 POSSIBLE RETURNS;
READ ONLY REQUIRES THE
NORMAL ONE.

PUT THE CORRECT
WORD COUNT INTO
USER'S RØ

NFREAD

WAS
THERE A
HARD
ERROR?

Y

CLEAR HARD ERROR
BIT; GIVE EMTE
ERROR #1

N

IS
EOF BIT
SET ?

Y

CLEAR EOF BIT;
GIVE EMTE ERROR
#$

N

IS
HANDLER IN
MEMORY

Y

FATAL ERROR
?M-NO DEV

N

CHANGE LOGICAL
BLOCK TO
PHYSICAL BLOCK

QMANGR
QUEUE THE
I/O REQUEST

HARD
ERROR ?

Y

6$ EMTE DON

N

IF .SERR IS IN EFFECT,
RETURN IS MADE TO USER
PROGRAM.
WRITE

WSRITE

SET EOF ERROR BIT

EOF

TSWCNT

R/W CHECKS

NONFILE DEVICE

NFWRIT

UPDATE CLOSE LENGTH

? Y

WAS FILE ENTERED?

N

N

UPDATE THE CLOSING LENGTH

PUT WORD COUNT WRITTEN INTO USER'S REG

NFWRIT

NFREAD

IF THE LAST BLOCK BEING WRITTEN IS > THE CURRENT LAST BLOCK WRITTEN, CHANGE THE CLOSING FILE LENGTH.
WAIT/CDFN

WAIT

W$AILT

IS CHANNEL ACTIVE?

Y

Y

GIVE EMT ERROR #1; CLEAR HARD ERROR BIT

DID HARD ERROR OCCUR?

N

N

GENERATE EMT ERROR #9

WAIT FOR ALL I/O TO FINISH

EMTDON

THIS IS DONE BY WAITING FOR THE NUMBER OF FREE QUEUE ELEMENTS TO BE EQUAL TO THE TOTAL NUMBER AVAILABLE.

CDFN

Channel Define - the resident portion of CDFN causes a fresh copy of the USR to be read in, then enters the USR.

C$DFN

MARK USR NON-RESIDENT

C$DFN2

THIS FORCES CALUSR TO READ IN A NEW USR
GET JOB PARAMETERS

MAKE JOB # = \emptyset

\emptyset = LOW LIMIT;
MOVE HI LIM.
ADDR OF
CHANNELS

EMTDON

GET TIME OF DAY

MOVE HI ORDER,
THEN LOW ORDER

EMTDON

SET FPP EXCEPTION

IS THERE A
USER ADDRESS
?

MAKE ADDRESS
THE MONITOR'S
ROUTINE

MOVE THE
ADDRESS TO
INTERNAL
LOCATION

EMTDON

E-73
SPECIAL FUNCTIONS/PURGE
SOFT/HARD ERRORS

SPECIAL FUNCTIONS (MAGTAPE/CASSETTE)

S$FUN

IS FUNCTION CODE < Ø ?

N

EMT ERROR Ø

Y

PUT A 377 INTO LOW BYTE OF THE FUNCTION CODE WORD

R$EAD

EMTDON

P$URGE

ZERO FIRST WORD OF CHANNEL AREA

EMTDON

S$ERR

SET SOFT ERROR ACTION BIT

EMTDON

H$ERR

CLEAR SOFT ACTION BIT

EMTDON

SPECIAL FUNCTIONS/PURGE
SOFT/HARD ERRORS
LOCK USR/CHAIN/UNLOCK USR

**LOCK USR**

1. **LOCK**
2. **CALUSR**
   - **READ IN USR IF NEEDED**
   - **THIS BUMPS A COUNTER WHICH IS DECREMENTED DURING A .UNLOCK. THE USR IS REALLY UNLOCKED ONLY WHEN THIS COUNT IS 0.**
3. **EMTDON**

**CHAIN**

1. **C$CHAIN**
2. **SET BIT 488 IN JOB STATUS WORD; MAKE NR NON-ZERO**
3. **JOIN EXIT CODE AT CHXIT**

**UNLOCK USR**

1. **USNLOK**
2. **IS USR .LOCKED ?**
   - **MONOUT**
   - **Y**
   - **DECREMENT .LOCK COUNTER**
3. **IS LOCK COUNT 0 ?**
   - **Y**
   - **EMTDON**
   - **N**

4. ** IS KMON IN MEMORY ?**
   - **N**
   - **IS A C WAITING ?**
     - **Y**
     - **.EXIT**
     - **N**

5. **IS USR NON-RES ?**
   - **Y**
   - **KILL DIRECTORY NOW IN MEMORY; READ USR**
6. **EMTDON**

**IF KMNO IS IN, SO IS USR.**

---

E-75
PRINT

PRINT - Causes a line to be output to the console terminal.

Diagram:

1. P$PRINT
2. IS CHAR. A NULL?
   - Y: ECHO A CR/LF
     - N: IS IT A 200?
       - Y: EMTDON
       - N: TTOPT2
         - GET NEXT CHAR

Diagram flow: P$PRINT -> IS CHAR. A NULL? (N) -> TTOPT2
           -> IS IT A 200? (Y) -> EMTDON
SETTOP

S$ETOP

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?

MAKE REQUEST FOR BOTTOM OF USR MINUS TWO

MARK USR NON-RESIDENT

SET JSW TO INDICATE SWAPPING

RETURN TOP LOCATION IN 5Ø AND RØ

EMTDON
EXIT

ESXIT

SET EXIT IN PROGRESS FLAG
CLEAR CHAIN BIT

CHXIT

WAIT FOR I/O TO QUIESCE (# AVAILABLE = TOTAL)

=== WAIT FOR AVAILABLE QUEUE ELEMENTS TO EQUAL TOTAL NUMBER. ===

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?

N

Y

IS USER NOW SWAPPED OUT?

N

READ USER PROGRAM BACK IN

Y

2$
EXIT (CONT.)/TTYIN

TTYIN

T$TIN

IN LINE MODE ?

Y

N

IS A CHAR IN BUFF ?

Y

N

AT END OF RING BUFF ?

Y

N

CYCLE TO HEAD OF RING BUFFER

EMT ERROR $; NO LINE/CHAR IN BUFFER

EMTDON

A

A

E-79
TTYIN (CONT.)/TTYOUT

3$

PUT CHAR INTO R$; DECREASE CHAR COUNT

EOLST
IS IT A LINE TERMINATOR

N
EMTDON

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
EMT 376 is reserved for reporting fatal monitor errors. When a fatal error condition is encountered, a call of the form:

```
EMT 376
```

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 ?
```

```
N
```

```
Y
```

```
-----IF CODE <Ø, IT IS ALWAYS FATAL AND WILL ALWAYS ABORT THE JOB.
```

```
N
```

```
WAS SERR DONE ?
```

```
Y
```

```
TURN ON THE C BIT IN HIS PS
```

```
N
```

```
INIT SP TO 1000; .SRESET READ FRESH USR
```

```
JMP IOFFSET(R5)
```

```
-----INTO USR CODE TO HANDLE PRINTING THE ERROR.
```

```
ADJUST PC OF CALL TO BE PAST CODE WORD
```

```
EMTOUT
```
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

TTIINT --- HERE ON INPUT INTERRUPT

SAVE REGS; PUT CHAR INTO RØ BACK TO PRIOR. Ø

----- THIS LOWERS THE PROCESSOR PRIORITY FROM LEVEL 4 TO LEVEL Ø.

IS IT LOWER CASE ?

Y

CONVERT IT TO UPPER CASE

N

IS IT A SPECIAL CHAR?

Y

DISPATCH TO CHARACTER PROCESSOR

N

T.SPEC

CTRLC ^C
CTRLO ^O
CTRLQ ^Q
CTRLS ^S
CTRLU ^U
ALT 33,175,176
RUB RUBOUT

NOTE: THE CHAR S 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

ECHO ^ FOLLOWED BY CHAR + 100

--- MAKES NON-PRINTS INTO EQUIVALENT CHARACTER

N

WAS LAST CHAR A RUBOUT?

Y

ECHO A \\

--- THIS ONE IS NOT A RUBOUT. IF LAST CHARACTER WAS, ECHO A CLOSING \\

N

MAKE CHARACTER A BELL

3$

UPDATE RING BUFFER POINTERS, COUNTER

TERMINATOR

EOLTST

TEST FOR LINE TERMINATOR

UPDATE PREVIOUS CHAR TO BE THE CURRENT ONE

3$

BUMP LINE COUNTER BY 1
TT INPUT INTERRUPT SERVICE (CONT.)

3$
IN LINE MODE ?
N
Y
IN CHARACTER MODE, NO ECHOING IS DONE.

Y
IS ^S SET ?

N
TTIEXZ

IF ^S IS UP, DON'T ECHO, AS OUTPUT IS TEMPORARILY STOPPED.

TTOPUT
ECHO THE CHARACTER

--- TT: USES HOOKS INTO THE RESIDENT SERVICE TO EFFECT ITS PROCESSING.
SEE TT: HANDLER FOR DETAILS.

TTINC3

--- IF <CR>, PUT IN AUTOMATIC LINEFEED.
WE SUBTRACT 3 FROM CHARACTER TO CHECK FOR <CR>. THAT MAKES THE CHARACTER A <LF> AT TTINC3.

<CR>

N
EMTDON

--- THIS USES SAME EXIT SEQUENCE AS DOES AN EMT.
These routines are entered when any of the corresponding special characters are struck.

--- ALL POSSIBLE CODES ARE CONVERTED TO 33. AT SUBROUTINE TTOPUT, A 33 ABOUT TO BE ECHOED IS CHANGED TO A $.

--- DO THIS BY CLEARING THEN SETTING THE OUTPUT INTERRUPT BIT
CONTROL C

CTRLC

GENERATE ^C <CR><LF>

WAS LAST CHAR A ^C ?

Y IS A .EXIT NOW IN PROGRESS?

N -----EXIT IS ALREADY IN PROGRESS.

        LET IT CONTINUE UNMOLESTED.

IS DIRECT. OP IN PROGRESS?

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

.EXIT

----- THIS INDICATES TO .EXIT THAT ALL I/O IS DONE.

THIS WILL CAUSE THE ^C TO BE PICKED UP ON RETURN FROM THE INTERRUPTED OPERATION.
RUBCON/RUBCM2

RUBCON 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 = \( \emptyset \) 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

TTINT --- HERE ON OUTPUT INTERRUPT

S' FLAG UP ?
Y

TTP0XT

N

ARE WE DOING TABS OR FILLER ?
Y

PUT CHARACTER OUT TO TPB
TTP0XT ---- (TELEPRINTER BUFFER)

N

INTO TT: TO SEE IF IT IS ACTIVE

AT END OF RING BUFF ?
Y

WRAP THE POINTER AROUND

DECREASE COUNT IN RING

ALL DONE OUTPUT ?
Y

TTODON

N

GET CHARACTER FROM RING BUFFER

A

TCHKSP

A

IS IT <4? ?
Y

CHAR. <4? WILL NOT PRINT.

N

IS IT <177 ?
Y

ADJUST TAB STOP, IF NECESSARY

TPRNT2

BUMP BUFFER POINTER

PUT CHARACTER OUT TO TPB
TTP0XT

E-93
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?

N

CLEAR OUT PARITY BITS

N

WILL THIS CHAR FIT?

Y

STORE THE CHARACTER. UPDATE POINTERS

N

AT END OF RING?

Y

CYCLE BACK TO START OF BUFF. BUMP COUNT

BUMP COUNT

SET OUTPUT INTERRUPT ENABLE

CLEAR C BIT IN CURRENT PS

RTS PC

C BIT SET BY THE COMPARISON DONE HERE.
E.4.4 I/O Routines
I/O QUEUE MANAGEMENT Routines

QMANGR

I/O COMPLETION WILL FREE QUEUE ELEMENTS

N

IS THERE ROOM IN Q?

Y --- TO LEVEL 7

PICK UP THE FIRST AVAILABLE ELEMENT

MAKE THE NEXT ELEMENT THE FIRST AVAILABLE

--- TO LEVEL Ø

FILL IN THE I/O QUEUE ELEMENT

N

IS THIS A CALL FROM MT/CS?

Y

ZERO THE FUNCTION WORD

----- IF THE BYTE OF THE COMPLETION ROUTINE WORD = 377, THIS IS INTERPRETED AS A FILE REQUEST ON MT OR CT.

FILL IN FUNCTION CODE

Y

IS .EXIT GOING?

N

IS DEVICE NOW BUSY?

Y

MARK THE DEVICE BUSY

5$

TO LEVEL Ø

A

----- TO LEVEL 7
I/O Queue Management Routines (Cont.)

A

JSR PC, (R2)
GO TO DEVICE HANDLER

5$

--- HERE WHEN DEVICE IS ALREADY BUSY

LINK THIS ELEMENT INTO DEVICE'S Q

MAKE HANDLERS LAST QUEUE ELEMENT POINT TO THIS ELEMENT

TO LEVEL Ø

DO WE WAIT FOR I/O?

Y

IS I/O DONE?

Y

RTS PC

N

N

A SYNCHRONOUS REQUEST WILL WAIT FOR I/O TO FINISH

RTS PC
COMPLT is entered when an I/O transfer finishes.

- **COMPLT**
  - **SVREG**
    - **SAVE REGISTERS**
      - **WAS THERE A HARD ERROR?**
        - **Y**
          - **HALT ON HARD ERROR?**
            - **Y**
              - **HALT**
            - **N**
              - **CHANNEL # INTO R1**
        - **N**
          - **--- 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**

- **SET COMPL. ROUTINE IN PROG. FLAG; SAVE 52**
  - **DO A COMPLETION ROUTINE?**
    - **N**
      - **JSR PC,(R5) TO COMPLETION ROUTINE**
    - **Y**
      - **RESTORE 52; DECREMENT IN PROG. FLAG**
        - **IS DEVICE FREE?**
          - **Y**
            - **JSR PC,2(R4) RECALL HANDLER FOR NEXT TRANSFER**
          - **N**
            - **NOP POP RESTORE SAVED REGISTERS**
              - **RTS PC**
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.

FETCH EMT CODE

<374

EXTRACT CHANNEL NUMBER

CHECK ARG. LIST ADDRESS

?ADDR? ERROR

VALID ?

Y

374

N

?CHAN? ERROR

377

EXIT

E376

FATAL ERROR EMT

376

?ADDR? ERROR

Y

3S

CHANNEL # TOO BIG?

Y

?CHAN? ERROR

N

LEGAL FUNCTION CODE ?

N

?EMT? ERROR

Y

EMTCOM

EMTCOM

LOCATE EMT IN DISPATCH TABLE

?ADDR? ERROR

ARGUMENTS VALID ?

N

POINT TO CHANNEL AREA

Y

DISPATCH TO EMT
```
.SDAT, .WRITE

--- READ/WRITE COMMON ERROR CHECK

TSWCNT
--- COMPUTE LAST WORD OF TRANSFER

CHKSP
--- CHECK IT WITH JOB LIMITS

WITHIN LIMITS?

?ADDR?

RETURN ERROR 2

FILENAME?

RETURN

RWXT

RETURN ERROR 2

RETURN

START BLOCK 

RETURN

N

SHORTEN IT

RETURN ERROR 0

RWXT

--- USE COMMON CODE WITH .READ

E-109
```
.SYNCH, .GTIM


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 # ?

REQUEST TASK SWITCH

POINT TO TASK'S IMPURE AREA

DOES TASK EXIST ?

IS IT ALIVE ?

SET CPENDS COMPLETION PENDING

CONTROL RETURNS TO CODE AFTER .SYNCH WHEN IT IS CALLED BY COMPLETION QUEUE MANAGER AT PRIORITY ZERO.

.GTIM

ENTER SYSTEM STATE

GET HIGH AND LOW ORDER PSEUDO-TIME

ADD IN ACCUMULATED TICKS TO GET REAL TIME

PAST MIDNIGHT ?

ADJUST TIME WORDS

BUMP THE DATE

--- DO THE .GTIM ALL OVER AGAIN
HARD AND SOFT RESET

HARD RESET ENTRY

ENTER SYSTEM STATE WITH SOFT RESET CALLED AT EXIT

DOES IORSET CODE FOLLOWED BY CALL TO $S$RESET?

I/O RESET ENTRY

IORSET

SAVE REGS. 0-3

SCAN HANDLER ENTRY POINT TABLE

IS HANDLER RESIDENT?

END OF TABLE?

SET HANDLER HOLD FLAG IN HANDLER

ANY MORE Q ELEMENTS?

THIS JOB'S?

DISCARD IT

RESET HOLD FLAG

TOP ELEMENT COMPLETE?

CLEAR COMPLETION FLAG

CALL QUEUE COMPLETION FOR TOP ELEMENT

GO CANCEL PENDING MESSAGES

CMARKT

E-116
--- 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 OVERLAYERED?**

**Y**

**SKIP CHANNEL 17**

**N**

**CLEAR ALL CHANNELS**

**EMTRTI**

**QUIESCE**

**POINT TO JOB'S IMPURE AREA**

**WAIT FOR I/O TO STOP (I.IOCT=$\phi$)**

**RETURN**

**REVERT**

**QUIESCE**

**STOP I/O**

**POINT TO B/G CHANNELS**

**Y**

**1$**

**IS JOB THE B/G?**

**N**

**POINT TO F/G CHANNELS**

**1$**

**RESET CHANNELS TO ORIGINAL 16**

**RESET SUSPEND COUNT**

**RESET QUEUE OF AVAILABLE NODES TO POINT TO THE ONE INTERNAL NODE. CLEAR COMPLETION QUEUE**

**A**

**JOB IN F/G?**

**Y**

**PURGE NON-RESIDENT HANDLERS FROM SENTRY**

**3$**

**ENABLE TT INPUT INTERRUPT**

**RETURN**

**N**

**RDOVLY**

**FLAG USER PROGRAM RUNNING**

**$SYS READ IN USER FROM SWAP BLOCKS**

**ENTRTPG**

**SET UP USER STACK POINTER**

**FLAG USR GONE**

**DEQUSR**
.SETTOP

SETOP

--- SET TOP MEMORY LIMIT OF JOB

REQUEST < JOB LOW LIMIT?

Y

SET EQUAL TO LOW LIMIT OF JOB

--- CAN'T BE OUTSIDE LIMITS SET BY MONITOR.

N

REQUEST > JOB HIGH LIMIT?

Y

SET EQUAL TO JOB HIGH LIMIT

N

JOB IN BACKGROUND?

N

3S

Y

ENQUSR

REQUEST USR OWNERSHIP

MUST OWN THE USR IN CASE IT MUST BE SWAPPED OUT

29S

REQUEST HIGHER THAN KMON?

Y

FLAG KMON NON-RESIDENT

N

REQUEST HIGHER THAN USR?

Y

CAN WE SWAP USR?

Y

RETURN TOP VALUE IN RØ AND 5Ø

N

EXIT

29S

MAKE USR NON-RESIDENT. SET USR-IS-SWAPPING FLAG

29S

JOB OWNED USR PRIOR TO HERE?

Y

DEQUSR

GIVE UP USR OWNERSHIP

N

3S

RETURN HI LIMIT ACTUALLY GRANTED

E-118
CANCEL MARK TIME

C$MKT

GO TO SYSTEM STATE

C$MARKT

GET A QUEUE ELEMENT

4$ Y

END OF Q?

N

THIS JOB'S ELEMENT?

N 1$

3$ Y

CANCEL ALL?

N

1$ N

CORRECT ELEMENT?

Y 3$

REMOVE ELEMENT

4$ Y

CANCEL ALL?

N

RETURN TIME REMAINING

5$

A$QLINK

RETURN ELEMENT TO FREE LIST

CANCE$ ALL?

Y 1$

EXIT (RTS PC)

5$

CANCE$ ALL?

Y 1$

EXIT (RTS PC)

3$

SET ERROR BIT.

E-119
SWAP IN USR, LOCK/UNLOCK USR

CALUSR

ENQUUSR
GET OWNERSHIP OF USR

BUMP USR USAGE COUNT

USR IN MEMORY?

USR FLOATING?

GET FLOATING ADDRESS

DOES ONE EXIST?

LOW LIMIT O.K.?

HIGH LIMIT O.K.?

FATAL ERROR

MONOUT

IS JOB IN BACKGROUND?

FATAL ERROR

WRITE OUT SWAP BLOCKS

READ IN USR

--- LOCK THE USR IN MEMORY

T$LOCK

USR FREE?

CLAIM IT AND READ INTO MEM.

EMTRTI

--- UNLOCK THE USR FROM MEMORY

USNLOK

THIS JOB OWNS USR?

DID IT LOCK IT?

DECREMENT USR LEVEL COUNT

USR CALLED ITSELF?

RIDUSR
GET RID OF USR

EMTDON

EMTDON

USE NORMAL USR LOCATION

EXIT

EXIT

EMTRTI

N
Y

N
Y

N
Y

N
Y

N
Y

N
Y

N
Y

N
Y
E.5.2 Job Arbitration, Error Processing
COMMON INTERRUPT
ENTRY AND EXIT

SEN SYS -- USED BY MONITOR TO
GO TO SYSTEM STATE.

SET UP RETURN
ADDRESS TO USE
WHEN EXITING

$INTEN,

INTERRUPT ROUTINE
ENTRY POINT

SAVE R4.
BUMP LEVEL
COUNTER.

2$ Y

IN SYSTEM
STATE?

SWITCH STK PTRS.

1$

SET PRIORITY
to 7

ANY ABORTS --
OR CONTEXT
SWITCH?

$ EXSWAP CALL SCHEDULER

N

SWITCH TO USER JOB'S
STACK AND RESTORE
ITS REGISTERS.

EXINT

EXINT

EXUSER

FAKE FPP
INTERRUPT

 Fey FDW

SAV FPU
STATUS
REGISTER

SET TO
RETURN TO
USER'S FPU
ROUTINE

ANY FPU
INTERRUPTS?

YES

YES

RETURN HERE
FROM HANDLER

A

A

ANY CLOCK
TICKS?

N

N

LEAVING
SYS. STATE?

YES

YES

N

N

EXIT TO USER --- ENTRY POINT
(TIMER, EXSWAP RETURN HERE)

RESTORE REGS

EXINT

3$ 2$

2$ 2$

YES

YES

N

N

N

USE RMN'S
FPP ROUTINE

2$
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
PROCESSOR IDLE LOOP

DECREMENT JOB
NUMBER

THIS JOB
EXISTS?
Y
EXUSER
N

IS JOB
BLOCKED?
Y
4$
N

3$

IS JOB
BLOCKED?
Y

3$

3$

3$

3$

3$

3$

3$

3$

3$

3$

3$

3$

3$

4$

CNTXSW
SWITCH IN
THIS JOB

--- IF COMPLETION PENDING, CNTXSW
WILL FAKE INTERRUPT ON TASK
STACK TO CALL COMPLETION QUEUE
MANAGER ON EXIT.

EXUSER
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 $0

SWAPME

FLAG ACTION FOR CURRENT JOB

CLEAR ABORT REQUEST FLAG FOR CURRENT JOB. SEARCH TABLE OF IMPURE POINTERS FOR JOBS IN MEMORY.

FOUND A JOB ?

Y

N

END OF TABLE ?

N

ABORT ?

Y

N

EXUSER

2$
JOB ABORT (CONT.)

1. ABL
2. DIRECTORY OP IN PROGRESS?
   - Y -> BY THIS JOB?
     - Y -> DON'T ABORT JUST YET
     - N -> CNTXSW
   - N -> 2$
3. CNTXSW
   - SWITCH TO ABORT CONTEXT
4. IORSET
   - RESET ACTIVE I/O
5. FORCE JOB TO EXIT WHEN RESUMED. CLEAN UP JOB'S IMPURE AREA
6. 1$
BLOCK A TASK/UNBLOCK A TASK
REQUEST TASK SWITCH

$SYSWT

JOB STILL BLOCKED?
N RETURN
Y

I.E., BLOCKING CONDITION STILL EXISTS?

TURN ON BLOCKING BIT

PREVIOUSLY UNBLOCKED?
N RETURN
Y

SWAPME

SWAPME

UNBLOK

BLOCKING BIT ON?
N RETURN
Y

CLEAR BLOCKING BIT. GET JOB NUMBER

$RQTWr

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

$RQTSW

REQ. JOB PRIORITY < CURRENT?
Y EXIT
N

$RQSIG

NEW REQUEST PRIORITY < PREV. REQ.?
Y EXIT
N

$RQTsw AND $RQsIG CAUSE SCAN OF TASK LIST STARTING AT REQUESTED JOB NUMBER.

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$

N

SAVE REG $0-3$ ON ITS STACK.
SAVE $34-52$ ON ITS STACK.

SWAP FPU?

N 51$

Y

FPU EXISTS?

N 51$

Y

SWAP FPU REGISTERS

51$

SWAP SPECIAL LIST?

N 53$

Y

SWAP ITEMS IN SPECIAL LIST

53$

SAVE PTR TO TOP OF EXTRA LIST, OLD STACK PTR,
NEW JOB CONTEXT.

A

A

FETCH NEW JOB STACK POINTER

SWAP IN SPECIAL LIST?

N 55$

Y

SWAP IN ITEMS IN SPECIAL LIST

55$

SWAP FPU?

N 6$

Y

FPU EXISTS?

N 6$

Y

SWAP FPU REGISTERS

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$ Y

WAS JOB DOING
COMPLETION ?

N

8$ N

SHOULD
IT ?

Y

FAKE INTERRUPT ON STACK.
UNBLOCK JOB.
SET COMPLETION FLAG.
SAVE CHANNEL $ AND
ERROR BYTE IN IMPURE AREA.
EXITS TO COMPLETION QUEUE MANAGER.

8$

EXIT
 Gül (RTS FC)
ERROR PROCESSING

TRAP4
TRAP10

SAVE CONDITION BIT IN LOC 52

TRAP FROM SYSTEM?

Y  FATAL HALT

N  PRIORITY + 0
RESTORE C BIT
SET TO PRINT
MESSAGE

A

3$
--- MERGE WITH
ERROR COMMON
CODE

1$
MAKE ERROR
CODE POSITIVE

B

SET TO PRINT
'DIR UNSAFE'

9$
PRINT ANY
MESSAGES

UABORT

13$
CLEAR USER
ADDRESS TO AVOID
RECURSION

USER INTERCEPT ADDR?

Y  CLEAR USER
INTERCEPT ADDR.
RESTORE C BIT TO FLAG
TRAP TO 4 OR 10

N  EXIT TO
USER

13$
CLEAN UP
OUR STACK

14$
FETCH ADDR.
OF ERROR.
RESET STACK

IS THERE A
USER STACK?

Y  FATAL
HALT

N  EXIT TO
USER

13$

2$
GET ADDRESS
OF ERROR
RESET STACK.

3$
ERRCOM

.HRESET
.RCTRLO

PRINT MSG.
THEN ERROR
ADDRESS

USING THE
USR?

N  TRAP
DURING DIRECTORY
OP?

Y  FATAL
HALT

N  EXIT TO
USER

9$
PRINT ANY
MESSAGES

UABORT
E.5.3 Queue Managers (I/O, USR, Completion)
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

N

BUMP CHANNEL REQUEST COUNTER

QFULL

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?

Y

UNHOLD THE HANDLER

N

CMPLT2

DO THE COMPLETION

1$Q

LOWER PRIORITY TO Ø

USWAP0

ENTER SYS. STATE TO WAIT FOR A Q ELEMENT

1$Q

WAIT FOR COMPLETION?

Y

WAIT UNTIL DONE

N

EXIT

EXIT TO HANDLER

EXIT TO HANDLER

M1/CT SPECIAL FUNCTION?

Y

PUT IN FUNCTION BYTE AND COMPLETION FUNCTION

N

ENTER SYSTEM STATE, THEN HOLD THE HANDLER

HANDLER ACTIVE?

Y

$RQSIG

GO WAIT

N

UNHOLD HANDLER AND INSERT THE Q NODE
QUEUE COMPLETION

ENTERED FROM DEVICE HANDLERS WHEN I/O REQUEST COMPLETED

COMPLT

HANDLER BEING HELD?

Y

EXIT

N

COMPLT2

POINT TO IMPURE AREA AND DECR. PENDING REQUESTS ON CHANNEL

CHANNEL NOW FREE?

N

1$ DOING COMPLETION?

N

AQLINK

SAVE CHANNEL STATUS WORD AND CHANNEL OFFSET IN THE Q ELEMENT.

Y

SET STACK TO CALL HANDLER ON EXIT

MORE REQUESTS FOR DEVICE?

Y

CLEAR LAST QUEUE ELEMENT FLAG IN HANDLER

N

1$

$SQRTSW REQUEST CONTEXT SWITCH FOR THE TASK

DEC. # OF I/O REQUESTS FOR THE TASK

ANY MORE TO DO?

Y

EXIT (RTS PC)

N

UNBLOK

UNBLOCK THE TASK WAITING FOR THE CHANNEL

A

UNBLOCK TASK IF WAITING FOR I/O TO FINISH

REMOVE USED ELEMENT FROM QUEUE
$CRTNE  --- ENTRY POINT (ENTERED WHEN CNTXSW TAKES AN INTERRUPT ON STACK BEING SWITCHED IN.)

1$

GET ANOTHER COMPLETION ROUTINE

NONE ?

N

2$

YES

CLEAR COMPLETION FLAG

EXIT TO USER (RTI)

3$

GET COMPL. ROUT. ADDRESS

CALL COMPLETION ROUTINE AS A SUBROUTINE

1$

2$

LINK FORWARD IN COMPL. QUEUE

. END OF QUEUE ?

N

Y

CLEAR LAST Q. ELEM. FLAG

THIS IS SYNCH ELEMENT?

Y

3$

N

RETURN TO AVAIL. QUEUE

3$
E.5.4 Clock Interrupt Service
CLOCK INTERRUPT HANDLER

LKINT

BUMP TICK COUNTER

FAST EXIT AT LEVEL 7

Y

INTERRUPT THE SYSTEM ?

N

---- INTERRUPTED A TASK, O.K.

SAVE R5 ON USER STACK.
SWITCH TO SYSTEM STACK.
PRIORITY + 0

SAVE R4 AND SP. THEN GET # OF TICKS

ENTRY POINT FROM INTERRUPT EXIT CODE

TIMER

ADD TICKS TO SYSTEM PSEUDO-CLOCK

1$

POINT TO CLOCK QUEUE

I.E., BRANCH TO 4$ IF -- CARRY SET

PSEUDO CLOCK OVERFLOW ?

Y (C = 1)

4$

(z = $ IF NON-EMPTY QUEUE)

N (C = $)

QUEUE EMPTY ?

Y (z = 1)

EXUSER

N (z = $)

TEST FIRST -- ELEMENT IN QUEUE

HIGH-ORDER EXPIRATION TIME = $?

Y

CAN'T HAVE EXPIRED

EXUSER

N (C = 1)

CLOCK LO ORDER TIME?

Y ---- TIMER EXPIRED FOR THIS JOB

( C = $)

2$

SAVE C BIT.
LINK QUEUE FORWARD

2$

$RQTSW

REQUEST A CONTEXT SWITCH
FOR THIS JOB

SET COMPLETION PENDING FLAG
IN JOB STATUS

CQLINK

LINK TIMER
NODE TO COMPLETION
QUEUE

RESTORE C BIT
VALUE

1$

SUBTRACT 1 FROM HI ORDER
TIME. MOVE PAST LO ORDER
TIME

EXPIRED?

Y (C = 1)

2$

LINK TO NEXT QUEUE ELEMENT

N (C = $)

4$
E.5.5 Console Terminal Interrupt Service
TT INPUT INTERRUPT ROUTINE

TTIINT

$INTEN
COMMON INTERRUPT ENTRY

FETCH CHAR. AND STRIP PARITY

Y
NULL?
N
EXIT

CONVERT LOWER CASE TO UPPER CASE

SPECIAL CONTROL CHAR.

TTIDSP
N
IN SPECIAL MODE?
Y
TTINC3

Y
DISPATCH TO PROCESSING ROUTINE

CTRL.C
CTRL.G
CTRL.S
CTRL.Q
CTRL.F
CTRL.B

ALT
N
PREVIOUS CHAR A RUBOUT?
Y
TTINC3

VT$5?
N
ECHO
Y
TYPE A '\\'

TTINC3

TTINC3

ROOM IN BUFFER?
Y
BUMP COUNT & INPUT POINTER
N
END OF BUFFER?
Y
WRAP POINTER TO TOP
N
INSERT CHAR INTO BUFFER

EOLTST
CHECK FOR END OF LINE

EOL?
Y
7$ BUMP LINE COUNT

N
7$ USING TT: HANDLER?
Y
TTHIN CALL HANDLER

7$
TT INPUT INTERRUPT ROUTINE (CONT.)

CTRL.S

SET XEDOFF SWITCH, IF ENABLED

Y

EXIT

N

TTIDSP

- PROCESS CHARACTER

CTRL.Q

CLEAR XEDOFF FLAG

N

WAS THIS ENABLED?

Y

TTIDSP

N

RE-ENABLE TT OUTPUT INTERRUPTS

EXIT

CTRL.O

CLEAR CHAR COUNT, SET BUFFER POINTERS EQUAL

ECHO

ECHO "+O CR LF"

TOGGLE +O FLAG

UNBLOK

UNBLOCK TASK IF STOPPING OUTPUT WITH +O

CTRL.U

ECOR$ECHO A 'U'

RUB

ANY CHARs. IN BUFFER?

Y

LINE TERMINATOR?

N

NEED TO WRAP PTRS?

Y

WRAP PTR AROUND

N

DECREMENT COUNT & PTR

DOING RUBOUT?

Y

VT05 TYPE DEVICE?

N

SAVE PREVIOUS CHARACTER

ECHO

ECHO BACKSPACE, SPACE, BACKSPACE

"WAS LAST A RUBOUT?"

Y

TYPE CHAR. DELETED

EXIT

A

CLEAR PREVIOUS CHAR, IF ONE

ECHO

ECHO A CR/LF

EXIT

E-142
TT OUTPUT INTERRUPT ROUTINE

1$ GET NEXT ID CHAR

3$ CLEAR ID FLAG (TTOID)

8$ END OF ID ?

PRINTS JOB --- I.D. (P> OR B>)
E.5.6 Resident Device Handlers (TT, Message)
TT: RESIDENT HANDLER

ENTRY POINT

FIND IMPURE AREA, THEN GET WORD CT.

(WRITE) 5S < ø

WORD COUNT ?

(SEEK) ø = ø TTCMPL

(READ) ø > ø

BLOCK # = ø?

TTOPT2

PRINT A '+' PROMPT CHAR.

TTHIN --- ENTER HERE WHEN END OF LINE IS DETECTED.

1S

LINE IN INPUT BUFFER ?

Y

IGET

GET A CHARACTER FROM BUFFER

DOUBLE CTRL C ?

Y

$ROABT

N

EOF (CTRL Z) ?

Y

PASS CHAR TO USER, DEC. THE COUNT

A

SET UP POINTER ENABLE PRINTER INTERRUPT

WRITE 3S

35S

5S

TTCMPL

DO COMPLETION

6S

EXIT

EXIT

DONE ? Y 35S

N

BUMP BUFFER POINTER

1S --- GET AS MANY LINES AS POSSIBLE

2S

BUMP BUFFER POINTER

3S

CLEAR AN UNFILLED BYTE AND COUNT DOWN

N

DONE ?

Y

SET EOF FLAG

TTCMPL

CLEAR IMPURE AREA PTRS, POINT TO QUEUE

COMPLT
MESSAGE HANDLER

ENTRY POINT

GET JOB # OF NEXT ELEMENT

IS IT A WRITE ?

N

CHANGE JOB # TO # OF RECEIVING JOB

N

ANY MESSAGE WAITING ?

Y

SET UP TO TRANSFER DATA

Y

IS IT A RECEIVE DATA ?

Y

MATCH WITH SDAT

IS WORD COUNT $\emptyset$ ?

Y

ABORT ENTRY POINT

N

IS THERE A SDAT ?

Y

FIND END OF QUEUE AND LINK IN ELEMENT

A SEEK ?

Y

CLEAR LAST LINK WORD

Y -- IGNORE SEEKS (WORD COUNT = $\emptyset$)

N

TRANSFER DATA, INDICATE AMOUNT SENT.

3$

COMPLT

RETURN ELEMENT IN QUEUE TO FREE LIST.

18$

--- FREE THE OTHER ELEMENT, RETURN FROM COMPLT

N

EXIT

10$

DONE ?

Y

GET AN ELEMENT AND FREE IT

Y

OTHER JOB ?

N

EXIT

26$

27$

20$

25$

22$

20$

22$

E-150
ENTRY POINT INDEX

Page numbers marked by an asterisk indicate the flowchart of the entry point.

$CRTNE, E-135*
$SYSNL, E-122*
$INTEN, E-122*
$RQABT, E-141*, E-148
$RQSIG, E-126*, E-133
$RQTSW, E-126*
$SYNCH, 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*
BADCMT, E-4*, E-7, E-9
BEGIN, E-7*, E-9
BLKCHK, E-56, E-57*

CSDFN, E-72*, E-112*
CSDFN2, E-66*, E-72
CSHAIN, E-75*, E-115*
CSLOS2, E-66*, E-69, E-105*, E-106
CSLOSE, E-69*, E-106*
CSMKT, E-119*
CSPYCH, E-110*
CSSIGN, E-66*, E-106*
CSSISP, E-66*, E-106*
CSSSTAT, E-111*
CALUSR, E-82*, E-120*
CCBB0, E-15*
CDFN, E-30*
CHANER, E-64, E-65*
CHKSP, E-113*
CHXIX, E-75, E-78*
CLCOM, E-33, E-42*
CLSQH, 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*
CONSOI, E-60*
CSI, E-46*
CTRL.B, E-141*
CTRL.C, E-141*
CTRL.F, E-141*
CTRL.Q, E-142*
CTRL.S, E-142*
CTRL.U, E-142*
CTRL.C, E-90*
CTRL.O, E-89*
CTRL.Q, 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
DEQUSR, E-117, E-132*
DLEET, E-56*
DLYUSR, E-126*, E-132
DOSAVE, E-19

E$ENTER, E-66*, E-105*
E$EXIT, E-78*, E-115
E, E-4, E-5*
E376, E-64, E-81*, E-104
E5ER0, E-111*
E5ER1, E-111*
ECHO, E-143*
ECHOOC, E-143*
ECHOR0, E-143*
EMTCOM, E-104*
EMTDON, E-65*, E-68, E-69, E-70,
E-72, E-73, E-74, E-75, E-76,
E-77, E-79, E-80*, E-88,
E-89, E-90, E-91, E-107*,
E-108, E-120
EMTERO, E-107*, E-112, E-113,
E-120
EMTOUT, E-64, E-65*, E-68, E-81
EMTFRO, E-64*, E-104*
EMTRT, E-105, E-106, E-107*,
E-110, E-111, E-112, E-113,
E-114, E-117, E-120
EMTUSR, E-105*, E-106
EQUSR, E-132*
ENTER, E-37*
ENTRP, E-7, E-69, E-117
ENTRY, E-59*
EOLTST, E-92*
ERRCOM, E-129*
EXIT, E-122*, E-138
EXSWAP, E-122, E-123*
EXTEND, E-38, E-39*
EXUSER, E-122*, E-123, E-124, E-138

ESETCH, E-66*, E-105*
FATAL, E-29*
FILE, E-14*
FOON, E-144*, E-146
FRUN, E-24*

E-151 January 1976
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*
INCR1, 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
LKNKT, E-138*
LNPFILE, E-33, E-34*
LOAD, E-21*
LOOKUP, E-33*

M$RKT, E-112*
M$WAIT, 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*
NXTFILE, 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*

P$URGE, E-74*, E-106
PHETCH, E-40*
PUTBLK, E-23*

Q$SET, E-66*, E-105*
QFULL, E-133*
QMANGR, E-98*, E-133*
QRESET, E-117*
QSET, E-43*
QUIESCE, 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*
ROVLY, E-7, E-69*, E-117*
REENTR, E-4, E-9*
RENAME, E-33*
RENR, E-37*, E-39
RESUM, E-112*
RESUME, E-22*
REVERT, E-117*
RIDUSR, E-132*
RSTSR, E-35*
RTOG, E-29*
RUB, E-91*, E-142
RUBCM2, E-92*
RUBCOM, E-92*
RUN, E-4, E-9*
RWXT, E-108*, E-109
RWXTE0, E-108*

$S$AVST, E-68*, E-111*
$S$DAT, E-109*
$S$ERR, E-74*
$S$ETOP, E-77*, E-118*
$S$FPA, E-73*
$S$FPP, E-112*
$S$PFUN, E-74*, E-108*
$S$RSET, E-66*
$SS$PND, E-112*
$SS$RET, E-117*
$SS$SNAP, E-112*
$ST$RAP, E-113*
SAVE, E-19*
SAVEVC, E-6, E-14*
SEGRW, E-58*
SEGRW1, E-58*
SEGRW2, E-58*
SOFRST, E-35*
SPDEL, E-34*, E-36
SPECIAL, E-47, E-48*
SPENTR, E-37, E-38*
SPESHL, E-59
SPLOOK, E-33, E-34*
TTIDSP, E-140*, E-142
TTIEXZ, E-88*, E-91
TTIINT, 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*
USRNFP, E-55*
USWAPO, E-115, E-133*

W$AIT, 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
CTRL format, 7-1
CTT temporary files, 7-22
directives, 7-1, 7-2
disexample, 7-11
directive termination, 7-6
language, 7-1
run-time handler, 7-2
BATSWL 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
Bootable magtape, 3-12.1
Bootstrap, 2-24
operation, 4-3
system, 5-15
Building a new system, 5-16

$CALL 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
CR1L 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,
$SDVSIZE, 2-15
Device driver, 5-16
Device Handler Block table,
$SDVREC, 2-14
Device handlers, 5-1, 5-14
CR1L, A-35
format, 5-8
LP/LS1L, A-28
RC1L/RS64, A-2
TC1L, 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 EM'Ts, C-1
Directive, 7-1
$DIRECTORY command (BATCH), 7-11
Directory entries, 3-2
format, 3-3
Directory header,
format, 3-1
words, 3-2
Directory of devices, 2-15
Directory operations, 5-19
Directory segment, 3-1, 3-6
extensions, 3-8
format, 3-1
Double tape mark, 3-11
$DVREC (Device Handler Block
table), 2-14
$DVSIZ (Device Directory Size
table), 2-15
Dynamic memory allocation, 2-7

Empty file, 3-3
EMT processors (flowcharts), E-67,
E-103
ENDGSD block, 3-19, 3-22
ENDMOD block, 3-28
Entry conditions, device handler,
5-9
$ENTRY (Handler Entry Point table),
2-14
Entry point,
index, E-151
table format (library), 3-29
$EOJ (BATCH)
command, 7-13
statement, 7-6
Error checking (BATCH), 7-12
Errors, bad tape, 3-13
in special device handlers, 5-20
Events, scheduler, 6-4
Example,
BATCH, 7-11
program linked to produce REL
file, 3-35
Extra words, 3-5

F/B (definition), 1-3
F/B monitor,
description, 6-1
flowcharts, E-1
F/G (definition), 1-3
File formats, RT-11, 3-1
formatted binary (.LDA), 3-30
object (.OBJ), 3-16
relocatable (.REL), 3-32
save image (.SAV), 3-31
File,
header, cassette, 3-15
length, 3-4
names and extensions, 3-4

File structure, 3-1
  cassette, 3-14
  magtape, 3-11
File types, 3-3
  tentative, 3-3
  empty, 3-3
  permanent, 3-4
Files, size and number, 3-7
Filling directory segments, 3-7
First end-of-file label, 3-12
First file header label, 3-12
Fixed offsets, 2-10-2-12
FLIMIT table, 2-4
Flowcharts, S/J and F/B monitor,
CSI (Command String
Interpreter), E-45
KMON (Keyboard Monitor), E-3
RMON (Resident Monitor), F/B,
E-101
RMON (Resident Monitor), S/J,
E-63
USR (User Service Routines),
E-27
Foreground job area, 2-3, 2-4
Foreground spooler example, D-1
Foreground terminal handler, B-1
Format, CT file header, 3-16
Formatted binary block, 3-18
Formatted binary format, 3-30,
3-31
Function codes, 5-19
  negative, 5-20
  positive, 5-19

Global,
additive displaced relocation,
3-26
additive relocation, 3-26
displaced relocation, 3-26
references, 3-21
relocation, 3-25
symbol directory (object
module), 3-20
symbols, 3-21
GOTO command (BATCH), 7-12
GSD
  block, 3-19
  item, 3-20
  structure, 3-20

Handler Entry Point table,
$ENTRY, 2-14
Handler,
installation, 5-11
names, 5-12
queue header, 5-3
Handler Size table, $HSIZE, 2-14
Handlers, 2-14
interrupt, 6-1
for special devices, 5-19
user-written, 5-23

Index-2
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/LS11 device handler, A-28

Macro,
  DEVICE, 2-16
  HSIZE, 2-17
  $MACRO command (BATCH), 7-11
Magtape
  bootable, 3-12.1
  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

Index-3
January 1976
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

.RAD50 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-11 file formats, see file formats, RT-11

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
System device handler,
  requirements, 5-14
  writing a, 5-14
System device structure, 4-1

Tables, monitor device, 2-13
TCI1 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
READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

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 require a written reply, please check here. ☐