RT-11
System Reference Manual

Order No. DEC-11-ORUGA-C-D, DN1, DN2
RT-11
System Reference Manual

Order No. DEC-11-ORUGA-C-D, DN1, DN2
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, 1974, 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

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE</td>
<td></td>
<td></td>
<td>xxi</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>1.1</td>
<td>PROGRAM DEVELOPMENT</td>
<td>1-2</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>1.2</td>
<td>SYSTEM SOFTWARE COMPONENTS</td>
<td>1-3</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>1.3</td>
<td>SYSTEM HARDWARE COMPONENTS</td>
<td>1-5</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>1.4</td>
<td>USING THE RT-11 SYSTEM</td>
<td>1-7</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>1.4.1</td>
<td>RT-11 Single-Job Monitor</td>
<td>1-7</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>1.4.2</td>
<td>RT-11 Foreground/Background Monitor</td>
<td>1-7</td>
</tr>
<tr>
<td>CHAPTER 1</td>
<td>1.4.3</td>
<td>Facilities Available Only in RT-11 F/B</td>
<td>1-8</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.1</td>
<td>START PROCEDURE</td>
<td>2-1</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.2</td>
<td>SYSTEM CONVENTIONS</td>
<td>2-3</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.2.1</td>
<td>Data Formats</td>
<td>2-3</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.2.2</td>
<td>Prompting Characters</td>
<td>2-4</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.2.3</td>
<td>Physical Device Names</td>
<td>2-4</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.2.4</td>
<td>File Names and Extensions</td>
<td>2-5</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.2.5</td>
<td>Device Structures</td>
<td>2-7</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.3</td>
<td>MONITOR SOFTWARE COMPONENTS</td>
<td>2-7.1</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.3.1</td>
<td>Resident Monitor (RMON)</td>
<td>2-7.1</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.3.2</td>
<td>Keyboard Monitor (KMON)</td>
<td>2-7.1</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.3.3</td>
<td>User Service Routine (USR)</td>
<td>2-8</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.3.4</td>
<td>Device Handlers</td>
<td>2-8</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.4</td>
<td>GENERAL MEMORY LAYOUT</td>
<td>2-8</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.4.1</td>
<td>Component Sizes</td>
<td>2-9</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.5</td>
<td>ENTERING COMMAND INFORMATION</td>
<td>2-10</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.6</td>
<td>KEYBOARD COMMUNICATION (KMON)</td>
<td>2-11</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.6.1</td>
<td>Foreground/Background Terminal I/O</td>
<td>2-13</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.6.2</td>
<td>Type-Ahead</td>
<td>2-14</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.7</td>
<td>KEYBOARD COMMANDS</td>
<td>2-14</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.7.1</td>
<td>Commands to Control Terminal I/O (GT ON and GT OFF)</td>
<td>2-15</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.7.2</td>
<td>Commands to Allocate System Resources</td>
<td>2-16</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.7.2.1</td>
<td>DATE Command</td>
<td>2-16</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.7.2.2</td>
<td>TIME Command</td>
<td>2-17</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.7.2.3</td>
<td>INITIALIZE Command</td>
<td>2-18</td>
</tr>
<tr>
<td>CHAPTER 2</td>
<td>2.7.2.4</td>
<td>ASSIGN Command</td>
<td>2-18</td>
</tr>
</tbody>
</table>

iii January 1976
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7.2.5</td>
<td>CLOSE Command</td>
<td>2-20</td>
</tr>
<tr>
<td>2.7.2.6</td>
<td>LOAD Command</td>
<td>2-20</td>
</tr>
<tr>
<td>2.7.2.7</td>
<td>UNLOAD Command</td>
<td>2-21</td>
</tr>
<tr>
<td>2.7.2.8</td>
<td>SET Command</td>
<td>2-23</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Commands to Manipulate Memory Images</td>
<td>2-28</td>
</tr>
<tr>
<td>2.7.3.1</td>
<td>GET Command</td>
<td>2-28</td>
</tr>
<tr>
<td>2.7.3.2</td>
<td>Base Command</td>
<td>2-29</td>
</tr>
<tr>
<td>2.7.3.3</td>
<td>Examine Command</td>
<td>2-30</td>
</tr>
<tr>
<td>2.7.3.4</td>
<td>Deposit Command</td>
<td>2-30</td>
</tr>
<tr>
<td>2.7.3.5</td>
<td>SAVE Command</td>
<td>2-31</td>
</tr>
<tr>
<td>2.7.4</td>
<td>Commands to Start a Program</td>
<td>2-33</td>
</tr>
<tr>
<td>2.7.4.1</td>
<td>RUN Command</td>
<td>2-33</td>
</tr>
<tr>
<td>2.7.4.2</td>
<td>R Command</td>
<td>2-34</td>
</tr>
<tr>
<td>2.7.4.3</td>
<td>START Command</td>
<td>2-34</td>
</tr>
<tr>
<td>2.7.4.4</td>
<td>REENTER Command</td>
<td>2-35</td>
</tr>
<tr>
<td>2.7.5</td>
<td>Commands Used Only in a Foreground/Background Environment</td>
<td>2-35</td>
</tr>
<tr>
<td>2.7.5.1</td>
<td>FRUN Command</td>
<td>2-36</td>
</tr>
<tr>
<td>2.7.5.2</td>
<td>SUSPEND Command</td>
<td>2-37</td>
</tr>
<tr>
<td>2.7.5.3</td>
<td>RSUME Command</td>
<td>2-38</td>
</tr>
<tr>
<td>2.8</td>
<td>MONITOR ERROR MESSAGES</td>
<td>2-38</td>
</tr>
<tr>
<td>2.8.1</td>
<td>Monitor HALTS</td>
<td>2-41</td>
</tr>
</tbody>
</table>

CHAPTER 3  
TEXT EDITOR  

3.1  CALLING AND USING EDIT  

3.2  MODES OF OPERATION  

3.3  SPECIAL KEY COMMANDS  

3.4  COMMAND STRUCTURE  
3.4.1 Arguments  
3.4.2 Command Strings  
3.4.3 The Current Location Pointer  
3.4.4 Character- and Line-Oriented Command Properties  
3.4.5 Command Repetition  

3.5  MEMORY USAGE  

3.6  EDITING COMMANDS  
3.6.1 Input/Output Commands  
3.6.1.1 Edit Read  
3.6.1.2 Edit Write  
3.6.1.3 Edit Backup  
3.6.1.4 Read  
3.6.1.5 Write  
3.6.1.6 Next  
3.6.1.7 List  
3.6.1.8 Verify  
3.6.1.9 End File  
3.6.1.10 Exit  
3.6.2 Pointer Relocation Commands  
3.6.2.1 Beginning  
3.6.2.2 Jump  
3.6.2.3 Advance  
3.6.3 Search Commands  
3.6.3.1 Get  
3.6.3.2 Find  

January 1976
5.2.2.1 Permanent Symbols 5-9
5.2.2.2 User-Defined and Macro Symbols 5-9
5.2.3 Direct Assignment 5-10
5.2.4 Register Symbols 5-11
5.2.5 Local Symbols 5-12
5.2.6 Assembly Location Counter 5-14
5.2.7 Numbers 5-17
5.2.8 Terms 5-17
5.2.9 Expressions 5-18

5.3 RELOCATION AND LINKING 5-19

5.4 ADDRESSING MODES 5-20
5.4.1 Register Mode 5-21
5.4.2 Register Deferred Mode 5-21
5.4.3 Autoincrement Mode 5-21
5.4.4 Autoincrement Deferred Mode 5-22
5.4.5 Autodecrement Mode 5-23
5.4.6 Autodecrement Deferred Mode 5-23
5.4.7 Index Mode 5-23
5.4.8 Index Deferred Mode 5-23
5.4.9 Immediate Mode 5-24
5.4.10 Absolute Mode 5-24
5.4.11 Relative Mode 5-24
5.4.12 Relative Deferred Mode 5-25
5.4.13 Table of Mode Forms and Codes 5-25
5.4.14 Branch Instruction Addressing 5-26
5.4.15 EMT and TRAP Addressing 5-27

5.5 ASSEMBLER DIRECTIVES 5-27
5.5.1 Listing Control Directives 5-27
5.5.1.1 .LIST and .NLIST 5-27
5.5.1.2 Page Headings 5-34
5.5.1.3 .TITLE 5-34
5.5.1.4 .SBTTL 5-34
5.5.1.5 .IDENT 5-36
5.5.1.6 Page Ejection 5-36
5.5.2 Functions: .ENABL and .DSABL Directives 5-36
5.5.3 Data Storage Directives 5-37
5.5.3.1 .BYTE 5-38
5.5.3.2 .WORD 5-39
5.5.3.3 ASCII Conversion of One or Two Characters 5-40
5.5.3.4 .ASCII 5-41
5.5.3.5 .ASCII2 5-42
5.5.3.6 .RAD50 5-43
5.5.4 Radix Control 5-44
5.5.4.1 .RADIX 5-44
5.5.4.2 Temporary Radix Control: ^D, ^O, and ^B 5-45
5.5.5 Location Counter Control 5-46
5.5.5.1 .EVEN 5-46
5.5.5.2 .ODD 5-46
5.5.5.3 .BLKB and .BLKW 5-47
5.5.6 Numeric Control 5-47
5.5.6.1 .FLT2 and .FLT4 5-48
5.5.6.2 Temporary Numeric Control: ^F and ^C 5-49
5.5.7 Terminating Directives 5-50
5.5.7.1 .END 5-50
5.5.7.2 .EOT 5-51
5.5.8 Program Boundaries Directive: .LIMIT 5-51
5.5.9 Program Section Directives 5-51
5.5.10 Symbol Control: .GLOBL 5-54
5.5.11 Conditional Assembly Directives
5.5.11.1 Subconditionals
5.5.11.2 Immediate Conditionals
5.5.11.3 PAL-11R and PAL-11S Conditional Assembly Directives

5.6 MACRO DIRECTIVES
5.6.1 Macro Definition
5.6.1.1 .MACRO
5.6.1.2 .ENDM
5.6.1.3 .MEXIT
5.6.1.4 MACRO Definition Formatting
5.6.2 Macro Calls
5.6.3 Arguments to Macro Calls and Definitions
5.6.3.1 Macro Nesting
5.6.3.2 Special Characters
5.6.3.3 Numeric Arguments Passed as Symbols
5.6.3.4 Number of Arguments
5.6.3.5 Automatically Created Symbols Within User-Defined Macros
5.6.3.6 Concatenation
5.6.4 .NARG, .NCHR, and .NTYPE
5.6.5 .ERROR and .PRINT
5.6.6 Indefinite Repeat Block: .IRP and .IRPC
5.6.7 Repeat Block: .REPT
5.6.8 Macro Libraries: .MCALL

5.7 CALLING AND USING MACRO
5.7.1 Switches
5.7.1.1 Listing Control Switches
5.7.1.2 Function Switches
5.7.1.3 Cross Reference Table Generation (CREF)

5.8 MACRO ERROR MESSAGES

CHAPTER 6
LINER

6.1 INTRODUCTION

6.2 CALLING AND USING THE LINER
6.2.1 Command String
6.2.2 Switches

6.3 ABSOLUTE AND RELOCATABLE PROGRAM SECTIONS

6.4 GLOBAL SYMBOLS

6.5 INPUT AND OUTPUT
6.5.1 Object Modules
6.5.2 Load Module
6.5.3 Load Map
6.5.4 Library Files

6.6 USING OVERLAYS

6.7 USING LIBRARIES
6.7.1 User Library Searches

6.8 SWITCH DESCRIPTION
6.8.1 Alphabetize Switch
6.8.2 Bottom Address Switch
6.8.3 Continue Switch 6-20
6.8.4 Default FORTRAN Library Switch 6-20
6.8.5 Include Switch 6-20
6.8.6 LDA Format Switch 6-21
6.8.7 Modify Stack Address 6-21
6.8.8 Overlay Switch 6-21
6.8.9 REL Format Switch 6-23
6.8.10 Symbol Table Switch 6-23
6.8.11 Transfer Address Switch 6-24

6.9 LINKER ERROR HANDLING AND MESSAGES 6-24

CHAPTER 7 LIBRARYAN 7-1

7.1 CALLING AND USING LIBR 7-1

7.2 USER SWITCH COMMANDS AND FUNCTIONS 7-2
7.2.1 Command Syntax 7-2
7.2.2 LIBR Switch Commands 7-2
7.2.2.1 Command Continuation Switch 7-3
7.2.2.2 Creating a Library File 7-4
7.2.2.3 Inserting Modules Into a Library 7-5
7.2.2.4 Replace Switch 7-5
7.2.2.5 Delete Switch 7-6
7.2.2.6 Delete Global Switch 7-7
7.2.2.7 Update Switch 7-9
7.2.2.8 Listing the Directory of a Library File 7-9
7.2.2.9 Merging Library Files 7-10

7.3 COMBINING LIBRARY SWITCH FUNCTIONS 7-11

7.4 FORMAT OF LIBRARY FILES 7-12
7.4.1 Library Header 7-12
7.4.2 Entry Point Table (Library Directory) 7-13
7.4.3 Object Modules 7-14
7.4.4 Library End Trailer 7-14

7.5 LIBR ERROR MESSAGES 7-14

CHAPTER 8 ON-LINE DEBUGGING TECHNIQUE 8-1

8.1 CALLING AND USING ODT 8-1
8.1.1 Return to Monitor, CTRL C 8-3
8.1.2 Terminate Search, CTRL U 8-4

8.2 RELOCATION 8-4
8.2.1 Relocatable Expressions 8-4

8.3 COMMANDS AND FUNCTIONS 8-5
8.3.1 Printout Formats 8-5
8.3.2 Opening, Changing and Closing Locations 8-6
8.3.3 Accessing General Registers 6-7 8-9
8.3.4 Accessing Internal Registers 8-10
8.3.5 Radix 50 Mode, X 8-10
8.3.6 Breakpoints 8-11
8.3.7 Running the Program, r;G and r;P 8-12
8.3.8 Single Instruction Mode 8-14
8.3.9 Searches 8-14
8.3.10 The Constant Register, r;C 8-16
8.3.11 Memory Block Initialization, ;F and ;I 8-16
8.3.12 Calculating Offsets, r;O 8-17

January 1976 viii
8.3.13 Relocation Register Commands, r;nR, ;nR, ;R 8-17
8.3.14 The Relocation Calculators, nR and n! 8-18
8.3.15 ODT Priority Level, $P 8-19
8.3.16 ASCII Input and Output, r;nA 8-20

8.4 PROGRAMMING CONSIDERATIONS 8-20
8.4.1 Functional Organization 8-20
8.4.2 Breakpoints 8-21
8.4.3 Searches 8-24
8.4.4 Terminal Interrupt 8-24

8.5 ODT ERROR DETECTION 8-25

CHAPTER 9 PROGRAMMED REQUESTS 9-1

9.1 FORMAT OF A PROGRAMMED REQUEST 9-2

9.2 SYSTEM CONCEPTS 9-5
9.2.1 Channel Number (chan) 9-5
9.2.2 Device Block (dbblk) 9-5
9.2.3 EMT Argument Blocks 9-5
9.2.4 Important Memory Areas 9-6
9.2.4.1 Vector Addresses 9-6
9.2.4.2 Resident Monitor 9-7
9.2.4.3 System Communication Area 9-7
9.2.5 Swapping Algorithm 9-9
9.2.6 Offset Words 9-11
9.2.7 File Structure 9-13
9.2.8 Completion Routines 9-13
9.2.9 Using The System Macro Library 9-14

9.3 TYPES OF PROGRAMMED REQUESTS 9-14
9.3.1 System Macros 9-20
9.3.1.1 .DATE 9-20
9.3.1.2 .INTEN 9-21
9.3.1.3 .MFPS/.MTPS 9-21.1
9.3.1.4 .REGDEF 9-22
9.3.1.5 .SYNCH 9-22
9.3.1.6 ..V1... .V2... 9-24

9.4 PROGRAMMED REQUEST USAGE 9-25
9.4.1 .CDFN 9-26
9.4.2 .CHAIN 9-27
9.4.3 .CHCOPY 9-28
9.4.4 .CLOSE 9-30
9.4.5 .CMKT 9-31
9.4.6 .CNTXSW 9-32
9.4.7 .CSIGEN 9-33
9.4.8 .CSISPC 9-36
9.4.8.1 Passing Switch Information 9-38
9.4.9 .CSTAT 9-41
9.4.10 .DELETE 9-42
9.4.11 .DEVICE 9-44
9.4.12 .DSTATUS 9-45
9.4.13 .ENTER 9-47
9.4.14 .EXIT 9-49
9.4.15 .FETCH 9-50
9.4.16 .GTIM 9-51
9.4.17 .GTJB 9-52
9.4.18 .HERR/.SERR 9-53
9.4.19 .HRESET 9-55
9.4.20 .LOCK/.UNLOCK 9-56
9.4.21 .LOOKUP 9-58
9.4.22 .MRKT 9-60
9.4.23 .MWAIT 9-62
9.4.24 .PRINT 9-63
9.4.25 .PROTECT 9-64
9.4.26 .PURGE 9-65
9.4.27 .QSET 9-65
9.4.28 .RCTRLQ 9-67
9.4.29 .RCVD/.RCVDC/.RCVDW 9-68
9.4.30 .READ/.READC/.READW 9-71
9.4.31 .RELEAS 9-74
9.4.32 .RENAME 9-75
9.4.33 .REOPEN 9-77
9.4.34 .SAVESTATUS 9-77
9.4.35 .SDAT/.SDATC/.SDATW 9-80
9.4.36 .SETTOP 9-82
9.4.37 .SFPQ 9-84
9.4.38 .SPFQ 9-85
9.4.39 .SPND/.RSUM 9-87
9.4.40 .SRESET 9-90
9.4.41 .TLOCK 9-91
9.4.42 .TRPSET 9-92
9.4.43 .TTYIN/.TTYN 9-93
9.4.44 .TTYOUT/.TTOUT 9-95
9.4.45 .TWAIT 9-98
9.4.46 .WAIT 9-99
9.4.47 .WRITE/.WRITC/.WRITW 9-100

9.5 CONVERTING VERSION 1 MACRO CALLS TO VERSION 2 9-108
9.5.1 Macro Calls Requiring No Conversion 9-108
9.5.2 Macro Calls Which May Be Converted 9-108

CHAPTER 10 EXPAND UTILITY PROGRAM 10-1
10.1 LANGUAGE 10-1
10.2 RESTRICTIONS 10-1
10.3 CALLING AND USING EXPAND 10-2
10.4 EXPAND ERROR MESSAGES 10-6

CHAPTER 11 ASEMRL, THE 8K ASSEMBLER 11-1
11.1 CALLING AND USING ASEMRL 11-1
11.2 ASEMRL ERROR MESSAGES 11-7

CHAPTER 12 BATCH 12-1
12.1 INTRODUCTION TO RT-11 BATCH 12-1
12.1.1 Hardware Requirements to Run BATCH 12-1
12.1.2 Software Requirements to Run BATCH 12-2
12.2 BATCH CONTROL STATEMENT FORMAT 12-2
12.2.1 Command Fields 12-2
12.2.1.1 Command Names 12-2
12.2.1.2 Command Field Switches 12-3
12.2.2 Specification Fields 12-5
12.2.2.1 Physical Device Names 12-6

January 1976 x
APPENDIX B

COMMAND AND SWITCH SUMMARIES

B.1 KEYBOARD MONITOR
B.1.1 Command Summary
B.1.2 Special Function Keys
B.2 EDITOR
B.2.1 Command Arguments
B.2.2 Input and Output Commands
B.2.3 Pointer Relocation Commands
B.2.4 Search Commands
B.2.5 Text Modification Commands
B.2.6 Utility Commands
B.2.7 Immediate Mode Commands
B.2.8 Key Commands
B.3 PIP
B.3.1 Switch Summary
B.4 MACRO/CREF
B.5 LINKER
B.5.1 Switch Summary
B.6 LIBRARIAN
B.6.1 Switch Summary
B.7 ODT
B.7.1 Command Summary
B.8 PROGRAMMED REQUESTS
B.9 BATCH
B.9.1 Switch Summary
B.9.2 Command Summary
B.10 DUMP
B.10.1 Switch Summary
B.11 FILEX
B.11.1 Switch Summary
B.12 SRCOM
B.12.1 SWITCH SUMMARY
B.13 PATCH
B.13.1 Command Summary
B.14 PATCHO
B.14.1 Command Summary

APPENDIX C

MACRO ASSEMBLER, INSTRUCTION, AND CHARACTER CODE SUMMARIES

C.1 ASCII CHARACTER SET
C.2 RADIX-50 CHARACTER SET
C.3 MACRO SPECIAL CHARACTERS
C.4 ADDRESS MODE SYNTAX

January 1976 xii
L.2.4 Set Bottom Address
L.2.5 Set Relocation Registers
L.3 EXAMPLES USING PATCH
L.4 PATCH ERROR MESSAGES

APPENDIX M
PATCHO

M.1 CALLING AND USING PATCHO

M.2 PATCHO COMMANDS
M.2.1 OPEN Command
M.2.2 POINT Command
M.2.3 WORD Command
M.2.4 BYTE Command
M.2.5 DUMP Command
M.2.6 LIST Command
M.2.7 EXIT Command
M.2.8 DEC Command
M.2.9 HELP Command

M.3 PATCHO LIMITATIONS

M.4 EXAMPLES

M.5 PATCHO ERROR MESSAGES
M.5.1 Run-Time Error messages

APPENDIX N
DISPLAY FILE HANDLER

N.1 DESCRIPTION
N.1.1 Assembly Language Display Support
N.1.2 Monitor Display Support

N.2 DESCRIPTION OF GRAPHICS MACROS
N.2.1 .BLANK
N.2.2 .CLEAR
N.2.3 .INSRT
N.2.4 .LNKRT
N.2.5 .LPEN
N.2.6 .NAME
N.2.7 .REMOV
N.2.8 .RESTR
N.2.9 .SCROL
N.2.10 .START
N.2.11 .STAT
N.2.12 .STOP
N.2.13 .SYNC/.NOSYN
N.2.14 .TRACK
N.2.15 .UNL NK

N.3 EXTENDED DISPLAY INSTRUCTIONS
N.3.1 DJSR Subroutine Call Instruction
N.3.2 DRET Subroutine Return Instruction
N.3.3 DSTAT Display Status Instruction
N.3.4 DHALT Display Halt Instruction
N.3.5 DNAME Load Name Register Instruction

N.4 USING THE DISPLAY FILE HANDLER
N.4.1 Assembling Graphics Programs
N.4.2 Linking Graphics Programs

xv January 1976
0.3.26  IJCVT        0-47
0.3.27  ILUN        0-47
0.3.28  INDEX       0-48
0.3.29  INSERT      0-49
0.3.30  INTSET      0-50
0.3.31  IPEEK       0-52
0.3.32  IPOKE       0-52
0.3.33  IQSET       0-53
0.3.34  IRAD50      0-54
0.3.35  IRCVD/IRCVDC/IRCVDP/IRCVDW 0-55
0.3.36  IREAD/IREADC/IREADP/IREADW 0-58
0.3.37  IREMAN      0-63
0.3.38  IREOPN      0-64
0.3.39  ISAVES       0-65
0.3.40  ISCHED      0-66
0.3.41  ISDAT/ISDAC/ISDATF/ISDATW 0-68
0.3.42  ISLEEP      0-71
0.3.43  ISPFN/ISPFNC/ISPFNF/ISPFNW 0-72
0.3.44  ISPY        0-76
0.3.45  ITIMER       0-77
0.3.46  ITLOCK      0-79
0.3.47  ITTINR      0-79
0.3.48  ITTOUR      0-81
0.3.49  ITWAIT      0-82
0.3.50  IUNITL      0-83
0.3.51  IWAIT       0-84
0.3.52  IWRITC/IWRITE/IWRITF/IWRITW 0-84
0.3.53  JADD        0-88
0.3.54  JAFIX       0-88
0.3.55  JCOMP       0-89
0.3.56  JDFIX       0-90
0.3.57  JDIV        0-91
0.3.58  JICVT       0-92
0.3.59  JJCVT       0-92
0.3.60  JMOV        0-93
0.3.61  JMUL        0-94
0.3.62  JSUB        0-95
0.3.63  JTIME       0-96
0.3.64  LEN         0-97
0.3.65  LOCK        0-97
0.3.66  LOOKUP      0-99
0.3.67  MRKT        0-100
0.3.68  MWAIT       0-101
0.3.69  PRINT       0-102
0.3.70  PURGE       0-103
0.3.71  PUTSTR      0-103
0.3.72  R50ASC      0-104
0.3.73  RAD50       0-105
0.3.74  RCHAIN      0-105
0.3.75  RCTRL0      0-106
0.3.76  REPEAT      0-107
0.3.77  RESUME      0-108
0.3.78  SCOMP       0-108
0.3.79  SCOOPY      0-109
0.3.80  SECONDS     0-110
0.3.81  STRPAD      0-111
0.3.82  SUBSTR      0-112
0.3.83  SUSPND      0-113
0.3.84  TIMASC      0-114
0.3.85  TIME        0-115
0.3.86  TRANSL      0-116

xvii   January 1976
## TABLES

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>RT-11 Hardware Components</td>
<td>1-6</td>
</tr>
<tr>
<td>2-1</td>
<td>Prompting Characters</td>
<td>2-4</td>
</tr>
<tr>
<td>2-2</td>
<td>Permanent Device Names</td>
<td>2-5</td>
</tr>
<tr>
<td>2-3</td>
<td>File Name Extensions</td>
<td>2-6</td>
</tr>
<tr>
<td>2-4</td>
<td>Special Function Keys</td>
<td>2-12</td>
</tr>
<tr>
<td>2-5</td>
<td>SET Command Options</td>
<td>2-23</td>
</tr>
<tr>
<td>3-1</td>
<td>EDIT Key Commands</td>
<td>3-2</td>
</tr>
<tr>
<td>3-2</td>
<td>Command Arguments</td>
<td>3-5</td>
</tr>
<tr>
<td>3-3</td>
<td>Immediate Mode Commands</td>
<td>3-31</td>
</tr>
<tr>
<td>4-1</td>
<td>PIP Switches</td>
<td>4-3</td>
</tr>
<tr>
<td>5-1</td>
<td>Legal Separating Characters</td>
<td>5-6</td>
</tr>
<tr>
<td>6-1</td>
<td>Linker Switches</td>
<td>6-3</td>
</tr>
<tr>
<td>7-1</td>
<td>LIBR Switches</td>
<td>7-3</td>
</tr>
<tr>
<td>8-1</td>
<td>Forms of Relocatable Expressions</td>
<td>8-5</td>
</tr>
<tr>
<td>8-2</td>
<td>Internal Registers</td>
<td>8-10</td>
</tr>
<tr>
<td>8-3</td>
<td>Radix 50 Terminators</td>
<td>8-11</td>
</tr>
<tr>
<td>9-1</td>
<td>Summary of Programmed Requests</td>
<td>9-15</td>
</tr>
<tr>
<td>9-2</td>
<td>Requests Requiring the USR</td>
<td>9-19</td>
</tr>
<tr>
<td>11-1</td>
<td>Directives not Available in ASEMBL</td>
<td>11-2</td>
</tr>
<tr>
<td>12-1</td>
<td>Command Field Switches</td>
<td>12-3</td>
</tr>
<tr>
<td>12-2</td>
<td>File Name Extensions</td>
<td>12-7</td>
</tr>
<tr>
<td>12-3</td>
<td>Specification Field Switches</td>
<td>12-8</td>
</tr>
<tr>
<td>12-4</td>
<td>Character Interpretation</td>
<td>12-9</td>
</tr>
<tr>
<td>12-5</td>
<td>BATCH Commands</td>
<td>12-12</td>
</tr>
<tr>
<td>12-6</td>
<td>Operator Directives to BATCH Run-Time Handler</td>
<td>12-50</td>
</tr>
<tr>
<td>12-7</td>
<td>Differences Between RT-11 and RSX-11D BATCH</td>
<td>12-53</td>
</tr>
<tr>
<td>H-1</td>
<td>Card Code Conversions</td>
<td>H-23</td>
</tr>
<tr>
<td>I-1</td>
<td>DUMP Switches</td>
<td>I-2</td>
</tr>
<tr>
<td>J-1</td>
<td>FILEX Switch Options</td>
<td>J-2</td>
</tr>
</tbody>
</table>
K-1 SRCCOM Switches
L-1 PATCH Commands
N-1 Description of Display Status Words
O-1 Summary of SYSLIB Subprograms
O-2 Special Function Codes

FIGURES

Number Page
2-1 RT-11 System Memory Maps 2-8
2-2 RT-11 Memory Map (GT40) 2-9
3-1 Display Editor Format 3-28
5-1 Assembly Source Listing of MACRO 5-15
5-2 Code Showing Local Symbol Blocks 5-31
5-3 Example of MACRO Line Printer Listing (132-column Line Printer) 5-32
5-4 Example of Page Heading From MACRO 80-column Line Printer 5-33
5-5 Symbol Table 5-33
5-6 Assembly Listing Table of Contents 5-35
5-7 .IRP and .IRPC Example 5-73
5-8 MACRO Source Code 5-80
5-8 CREP Listing Output 5-81
6-1 Linker Load Map for Background Job 6-9
6-2 Overlay Scheme 6-10
6-3 Memory Diagram Showing BASIC Link with Overlay Regions 6-11
6-4 Run-Time Overlay Handler 6-12
6-5 Library Searches 6-17
6-6 Alphabetized Load Map for a Background Job 6-19
7-1 General Library File Format 7-12
7-2 Library Header Format 7-13
7-3 Format of Entry Point Table 7-13
7-4 Library End Trailer 7-14
12-1 EOF Card 12-45

xix January 1976
PREFACE

This manual describes the use of the RT-11 Operating System. It assumes the reader is familiar with computer software fundamentals and has had some exposure to assembly language programs. The section "Additional and Reference Material" later in this Preface lists documents that may prove helpful in reviewing those areas. The Glossary provides definitions of technical terms used in the manual.

The user who is unfamiliar with RT-11 should first read those chapters of interest (see "Chapter Summary" below) to become familiar with system conventions. Having gained familiarity with RT-11, the user can then reread the manual for specific information.

Chapter Summary

Chapter 1 discusses system hardware and software requirements. It describes general system operations and lists specific components available under RT-11.

Chapter 2 introduces the user to system conventions and monitor/memory layout. It describes in detail the keyboard commands for controlling jobs and implementing user programs.

Chapters 3 through 8 describe the system utility programs EDIT, PIP, MACRO, LINK, LIBR, and ODT, respectively. These programs (a text editor, file transfer program, assembler, linker, librarian, and debugging program) aid the user in creating text files and producing assembly-language programs.

Chapter 9, which describes programmed requests, is of particular interest to the experienced programmer. It describes call sequences that allow the user to access system monitor services from within assembly-language programs.

Chapters 10 and 11 describe the 8K Assembler and EXPAND programs, respectively. These programs are useful in RT-11 installations with minimum memory configurations.

Chapter 12 describes the BATCH command language for RT-11. In BATCH mode, the RT-11 system can be left to run unattended for long periods of time.

The appendixes summarize the contents of the manual and describe additional system utility programs that can be used for extended system operations. These programs include SRCOM (a source file comparison program); FILEX (a file translation program that allows
Preface

transfer of files between RT-11 and other DIGITAL operating systems; PATCH and PATCHO (patching programs); DUMP (a file dump program); and SYSLIB (a library of programmed requests for FORTRAN users).

Version History

The current RT-11 system (monitor) is Version 2C (V2C). Each system component (monitors and utilities) is assigned a software identification number in the form Vxx-xx. Current identification numbers for V2C are listed in the RT-11 System Release Notes (DEC-11-ORNRA-A-D). To determine whether the correct version of a component is in use, examine its identification number and compare it with the list. (The procedure for examining the version number varies. Most system programs provide a special command; others print the version number when an output listing is requested. Consult the appropriate chapter or appendix of this manual for each component.)

NOTE

Throughout this manual, any references to V2 or V2B of RT-11 will pertain also to V2C. The RT-11 System Release Notes contain a comprehensive list of differences between V2C and previous versions of RT-11 (V2B, V2, V1).

Change bars and asterisks in the outermost margins of the manual are used to denote changes made to the text since the Version 2 release (DEC-11-ORUGA-B-D). The date July 1975 in the lower outside corner of a page indicates that the page was changed as a result of a release-independent update that occurred in July, 1975. The date January 1976 in the lower outside corner of the page indicates that the page was changed specifically as a result of the V2C update.

The user who is already familiar with the Version 2B RT-11 System Reference Manual (DEC-11-ORUGA-C-D,DN1) should first read the RT-11 System Release Notes document to note the major differences between V2B and V2C, and then read those pages of the RT-11 System Reference Manual that have changed as a result of the V2C update (identified by the date January 1976). The RT-11 System Generation Manual (DEC-11-ORGMA-A-D) should also be read if customization for special devices and features is required.

The user who is familiar with only the Version 2 RT-11 System Reference Manual (DEC-11-ORUGA-B-D) should read the following in addition to those items mentioned in the preceding paragraph:

Chapter 2 (System Communication) - Tables 2-2, 2-3, and 2-5
Chapter 3 (Text Editor) - Section 3.6.5.6
Chapter 9 (Programmed Requests) - Sections 9.1 and 9.1.3.6
Chapter 12 (BATCH) - Entire Chapter
Appendix H (F/B Programming and Device Handlers) - Sections H.2.4 and H.2.5
Appendix O (SYSLIB) - Entire Appendix

Finally, the user familiar with only the Version 1 RT-11 System Reference Manual (DEC-11-ORUGA-A-D) should read this entire manual with these exceptions:

January 1976
xxii
Preface

Chapter 3 (Text Editor) - note Section 3.7
Chapter 5 (MACRO Assembler) - note Section 5.7
Chapter 8 (ODT) - note restrictions in Section 8.1
Chapter 10 (EXPAND)
Chapter 11 (ASEMBL)
Appendix L (PATCH)

While knowledge of Versions 2 and 2B is sufficient for use of V2C, knowledge of Version 1 is not; the user with Version 1 knowledge only should carefully read the manual.

Additional and Reference Material

The following manuals provide an introduction to the PDP-11 computer family and the basic PDP-11 instruction set:

PDP-11 Paper Tape Software Programming Handbook**
(DEC-11-XPTSA-B-D)
PDP-11 Processor Handbook*
PDP-11 Peripherals Handbook*

The following manual provides an introduction to the use of RT-11 by presenting a simple demonstration of basic operating procedures:


These manuals describe the capabilities of the optional high-level language components:

PDP-11 FORTRAN Language Reference Manual** (DEC-11-LPLRA-B-D)

Summaries of the features provided by each language appear in this manual in Appendices F and G respectively.

Two PDP-11 system manuals are helpful when using FILEX (Appendix J) to convert programs between DOS, RSTS, and RT-11 formats:

(DEC-11-ORSUA-D-D)
DOS/BATCH Handbook** (DEC-11-ODBHA-A-D)

Users of display hardware may wish to refer to the appropriate hardware manual:

GT40/42 User's Guide*** (39H150)
GT44 User's Guide*** (39H250)
VT11 Graphic Display Processor Manual*** (79H650)
DECscope User's Manual*** (EK-VT50-OP)

The experienced programmer will want to read the following manual:

RT-11 Software Support Manual* (DEC-11-ORPGA-B-D)

* Included in the RT-11 Software Kit
** May be ordered from the DIGITAL Software Distribution Center
*** May be ordered from DIGITAL Communication Services

xxiii January 1976
Preface

Consult the following for a list of all manuals available in the RT-11 software documentation set:


Documentation Conventions

Conventions used throughout this manual include the following:

1. Actual computer output is used in examples wherever possible. When necessary, computer output is underlined to differentiate from user responses.

2. A line feed (character or key) is represented in the text as <LF>; a carriage return (character or key) is represented as <CR>. Unless otherwise indicated, all commands and command strings are terminated by a carriage return.

3. Terminal, console terminal, and teleprinter are general terms used throughout all RT-11 documentation to represent any terminal device, including DECwriters, displays, and Teletypes**. RP02 is a generic term used to represent both the RP11C/RP02 and RP11E/RP02 disks.

4. Several characters in system commands are produced by typing a combination of keys concurrently; for example, the CTRL key is held down while typing an O to produce a command which causes suppression of teleprinter output. Key combinations such as this are documented as CTRL O, CTRL C, SHIFT N, and so forth.

* Included in the RT-11 Software Kit
****Teletype is a registered trademark of the Teletype Corporation.
CHAPTER 1
RT-11 OVERVIEW

RT-11 is a single-user programming and operating system designed for the PDP-11 series of computers. This system 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 operation, and a powerful Foreground/Background (F/B) capability(1).

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

In a Foreground/Background environment, two independent programs may reside in memory. The foreground program is given priority and executes until it relinquishes control to the background program; the background program is allowed to execute until control is again required by the foreground program, and so on. This sharing of system resources greatly increases the efficiency of processor usage.

To handle both operating environments, RT-11 offers two completely compatible and versatile monitors (Single-job and F/B); either monitor provides complete user control of the system from the console terminal keyboard. Monitor commands which allow the user to direct single-job, foreground, and background operations are described in Chapter 2.

In addition to the monitor facilities, RT-11 offers a full complement of system programs; these allow program development using high level languages such as FORTRAN IV and BASIC or assembly language (MACRO or EXPAND/ASEMGL). System programs are summarized in Section 1.2 and are discussed in detail in individual chapters and appendixes of this manual.

(1) The uses and advantages of each environment are outlined later in this chapter.
RT-11 Overview

1.1 PROGRAM DEVELOPMENT

Computer systems such as RT-11 are often used extensively for program development. The programmer makes use of the programming "tools" available on his system to develop programs which will perform functions specific to his needs. The number and type of "tools" available on any given system depend on a good many factors -- the size of the system, its application and its cost, to name a few. Most DIGITAL systems, however, provide several basic program development aids: these generally include an editor, assembler, linker, debugger, and often a librarian; a high level language (such as FORTRAN IV or BASIC) is also usually available.

An editor is used to create and modify textual material. Text may be the lines of code which make up a source program written in some programming language, or it may be data; text may be reports, or memos, or in fact may consist of any subject matter the user wishes. In this respect using an editor is analogous to using a typewriter -- the user sits at a keyboard and types text. But the advantages of an editor far exceed those of a typewriter because once text has been created, it can be modified, relocated, replaced, merged, or deleted --all by means of simple editing commands. When the user is satisfied with his text, he can save it on a storage device where it is available for later reference.

If the editor is used for the purpose of writing a source program, development does not stop with the creation of this program. Since the computer cannot understand any language but machine language (which is a set of binary command codes), an intermediary program is necessary which will convert source code into the instructions the computer can execute. This is the function of an assembler.

The assembler accepts alphanumerical representations of PDP-ll coding instructions (i.e., mnemonics), interprets the code, and produces as output the appropriate object code. The user can direct the assembler to generate a listing of both the source code and binary output, as well as more specific listings which are helpful during the program debugging process. In addition, the assembler is capable of detecting certain common coding errors and of issuing appropriate warnings.

The output produced by the assembler is called object output because it is composed of object (or binary) code. On PDP-ll systems, the object output is called a module and contains the user's source program in the binary language which is acceptable to a PDP-ll computer.

Source programs may be complete and functional by themselves; however, some programs are written in such a way that they must be used in conjunction with other programs (or modules) in order to form a complete and logical flow of instructions. For this reason the object code produced by the assembler must be relocatable -- that is, assignment of memory locations must be deferred until the code is combined with all other necessary object modules. It is the purpose of linker to perform this relocation.

The linker combines and relocates separately assembled object programs. The output produced by the linker consists of a load module, which is the final linked program ready for execution. The user can, at his option, request a load map which displays all addresses assigned by the linker.
RT-11 Overview

Very rarely is a program created which does not contain at least one unintentional error, either in the logic of the program or in its coding. Errors may be discovered by the programmer while he is editing his program, or the assembler may find errors during the assembly process and inform the programmer by means of error codes. The linker may also catch certain errors and issue appropriate messages. Often, however, it is not until execution that the user discovers his program is not working properly. Programming errors may be extremely difficult to find, and for this reason a debugging tool is usually available to aid the programmer in determining the cause of his error.

A debugging program allows the user to interactively control the execution of his program. With it, he can examine the contents of individual locations, search for specific bit patterns, set designated stopping points during execution, change the contents of locations, continue execution, and test the results, all without the need of re-editing and re-assembling.

When programs are successfully written and executed, they may be useful to other programmers. Often routines which are common to many programs (such as I/O routines) or sections of code which are used over and over again, are more useful if they are placed in a library where they can be retrieved by any interested user. A librarian provides such a service by allowing creation of a library file. Once created, the library can be expanded, updated, or listed.

High level languages simplify the programmer's work by providing an alternate means of writing a source program other than assembly language mnemonics. Generally, high level languages are easy to learn—a single command may cause the computer to perform many machine language instructions. The user does not need to know about the mechanics of the computer to use a high level language. In addition, some high level languages (like BASIC) offer a special immediate mode which allows the user to solve equations and formulas as though he were using a calculator. Assembling and linking are done automatically so that the user can concentrate on solving the problem rather than using the system.

These are a few of the programming tools offered by most computer systems. The next section summarizes specific programming aids available to the user of RT-11.

1.2 SYSTEM SOFTWARE COMPONENTS

The following is a brief summary of the RT-11 system programs:

1. The Text Editor (EDIT, described in Chapter 3) is used to create or modify source files for use as input to language processing programs such as the assembler or FORTRAN. EDIT contains powerful text manipulation commands for quick and easy editing of a text file. EDIT also allows use of a VT11 display processor (such as the GT44), if one is part of the hardware configuration (see Section 1.3).

2. The MACRO Assembler (Chapter 5) brings the capabilities of macros to the RT-11 system with 12K (or more) memory. (Macros are instructions in a source or command language which are equivalent to a specified sequence of machine
RT-11 Overview

instructions or commands.) The assembler accepts source files written in the MACRO language and generates a relocatable object module to be processed by the Linker before loading and execution. Cross reference listings of assembled programs may be produced using CREF in conjunction with the MACRO Assembler.

3. EXPAND (Chapter 10) is used in an 8K P/B job area or 8K systems (or in larger systems with programs of great size) to expand macros in an assembly language program into macro-free source code, thus allowing the program to be assembled in 8K using ASEMBL.

4. ASEMBl (Chapter 11) is an assembler designed for use in an 8K RT-11 system, an 8K P/B job area, or larger systems where symbol table space is a factor. ASEMBl is a subset of MACRO-11 with more limited features. (CREF is not available under ASEMBl.)

5. The Linker (LINK, described in Chapter 6) fixes (i.e., makes absolute) the values of relocatable symbols and converts the relocatable object modules of compiled or assembled programs and subroutines into a load module which can be loaded and executed by RT-11. LINK can automatically search library files for specified modules and entry points; it can produce a load map (which lists the assigned absolute addresses) and can provide automatic overlay capabilities to very large programs. The Linker can also produce files suitable for running in the foreground.

6. The Librarian (LIBR, see Chapter 7) allows the user to create and maintain his own library of functions and routines. These routines are stored on a random access device as library files, where they can be referenced by the Linker.

7. The Peripheral Interchange Program (PIP, see Chapter 4) is the RT-11 file maintenance and utility program. It is used to transfer files between all devices which are part of the RT-11 system, to rename or delete files, and to obtain directory listings.

8. SRCCOM (Source Compare, described in Appendix K) allows the user to perform a character-by-character comparison of two or more text files. Differences can be listed in an output file or directly on the line printer or terminal, thus providing a fast method of determining, for example, if all edits to a file have been correctly made.

9. FILEX (Appendix J) allows file transfers to occur between DECTapes used under the DECSYSTEM-10 or PDP-11 RSTS system, and DECTape and disk used under the DOS/BATCH system, and any RT-11 device.

10. The PATCH utility program (Appendix L) is used to make minor modifications to memory image files (output files produced by the Linker); it is used on files which do or do not have overlays. PATCHO (Appendix M) is used to make minor modifications to files in object format (output files produced by the FORTRAN compiler and the Librarian, or MACRO and ASEMBl assemblers).

1-4
RT-11 Overview

11. ODT (On-line Debugging Technique, described in Chapter 8) aids in debugging assembled and linked object programs. It can print the contents of specified locations, execute all or part of the object program, single step through the object program, and search the object program for bit patterns.

12. DUMP (Appendix I) is used to print for examination all or any part of a file in octal words, octal bytes, ASCII and/or RAD50 characters (see Chapter 5).

13. BATCH (Chapter 12) is a complete job control language that allows RT-11 to operate unattended. The BATCH stream may be composed of RT-11 monitor commands or system-independent BATCH jobs (jobs that will run on any DIGITAL system supporting the BATCH standard; currently RT-11 and RSX-11D). BATCH streams can be executed under the Single-Job Monitor or in the background under the F/B Monitor.

14. The RT-11 FORTRAN System Subroutine Library (SYSLIB, Appendix O) is a collection of FORTRAN callable routines that make the programmed requests and various utility functions available to the FORTRAN programmer. SYSLIB also provides a complete string manipulation package and two-word integer package for RT-11 FORTRAN.

BASIC and FORTRAN IV are two high level languages available under RT-11. Summaries of their language features and commands are provided in Appendixes F and G of this manual.

1.3 SYSTEM HARDWARE COMPONENTS

The minimum RT-11 system (that is, one that does not use the F/B capability) requires a PDP-11 series computer with at least 8K of memory, a random-access device, and a console terminal. The F/B capability requires at least 16K of memory and a line frequency clock. For specific hardware/software interdependent requirements, refer to the RT-11 System Release Notes.

Devices supported by RT-11 are listed in Table 1-1. The third (middle) column lists devices for which support is initially provided in the system as distributed; these devices can be used with no modification (to either the monitor tables or the handlers) necessary. The devices in the fourth column are supported after simple modifications to the monitor tables or handlers. The system customization section of the RT-11 System Generation Manual describes how to make these modifications. The fifth column lists devices for which no support is provided, but which may be interfaced by the user. Currently, the 5S64 disk is the only device in this category, and instructions for its interface are provided in the RT-11 Software Support Manual.

Consult the RT-11 System Generation Manual for modifications that may be made to existing system devices (for example, varying the baud rate of a terminal).
## RT-11 Overview

### Table 1-1
RT-11 Hardware Components

<table>
<thead>
<tr>
<th>Category</th>
<th>Controller</th>
<th>System-Installed Devices</th>
<th>Devices Requiring System Modification</th>
<th>User-Installed Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECPack Cartridge</td>
<td>RK11</td>
<td>RK05</td>
<td></td>
<td>RS64</td>
</tr>
<tr>
<td>Fixed-head</td>
<td>RF11</td>
<td>RS11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RC11</td>
<td>RJS03</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RH11</td>
<td>RJS04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Removable Pack Disks</td>
<td>RF11</td>
<td>RP02</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RX11</td>
<td>RX01</td>
<td></td>
<td>RX01 (second controller)</td>
</tr>
<tr>
<td>DECTAPE</td>
<td>TC11</td>
<td>TU56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAGTAPE</td>
<td>TM11/TMA11</td>
<td>TU10,TS03</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NH11</td>
<td>TJU16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CASSETTE</td>
<td>TA11</td>
<td>TU60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIGH-SPEED PAPER TAPE READER/PUNCH</td>
<td>PC11</td>
<td>PC11 (both)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PR11</td>
<td>PR11 (reader only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE PRINTER</td>
<td>LS11</td>
<td>LS11, LA180</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LV11</td>
<td>LV11 (printer only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LP11</td>
<td>all LP11 controled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>printers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARD READER</td>
<td>CR11</td>
<td>CR11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CM11</td>
<td>CM11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERMINAL</td>
<td>DL11</td>
<td>LT33, LT35</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LA30P, LA36, VT50, VT52, VT05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY PROCESSOR</td>
<td>VT11</td>
<td>VR14-L, VR17-L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOCK</td>
<td></td>
<td>KW11-L</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RT-11 Overview

RT-11 operates in environments ranging from 8K to 28K words of memory. Reconfiguration for different memory sizes is unnecessary—the same system device operates on any PDP-11 processor with 8K to 28K of memory and makes use of all memory available.

1.4 USING THE RT-11 SYSTEM

As mentioned earlier in the chapter, the RT-11 system offers two complete operating environments. Each is controlled by a single user from the console terminal keyboard by means of an appropriate monitor—Single-Job or Foreground/Background. Both monitors are completely compatible and allow full user interaction with all features which are a part of the operating environment in use.

The choice of which environment to use, and, consequently, which monitor to run, depends upon the needs of the user. The next two sections provide information useful in determining which monitor is more suitable for certain applications.

1.4.1 RT-11 Single-Job Monitor

The RT-11 Single-Job Monitor provides a single-user, single-program system which can operate in as little as 8K of memory. Since the Single-Job Monitor itself requires approximately one-half the memory space needed by the Foreground/Background Monitor, this system is ideal for extensive program development work; a much larger area of memory is available for the user program and its buffers and tables. Programs requiring extremely high data rates are best run in the Single-Job environment, since interrupts can be serviced at a much higher rate.

All system programs (listed in Section 1.2) can be used under the Single-Job Monitor, and many of the features of the Foreground/Background Monitor (i.e., KMON commands and programmed requests not used to control foreground jobs) are supported.

In effect, the Single-Job Monitor is much smaller and slightly faster than the Foreground/Background Monitor; it can best be used when program size is the important factor.

1.4.2 RT-11 Foreground/Background Monitor

Quite often the central processor of a computer system may spend a large percentage of time waiting for some external event to occur, the most common event being the completion of an I/O transfer (this is particularly true of real time jobs). Many users would like to take advantage of this unused capacity to accomplish other lower-priority tasks such as further program development or complex data analysis. The Foreground/Background system provides this capability.

In a Foreground/Background system the foreground job is the time-critical, on-line job, and is given top priority; whenever possible the processor runs the foreground job. However, when the foreground job reaches a state in which no more processing can be done
RT-11 Overview

until some external event occurs, the monitor will try to run the lower priority background job. The background job then runs until the foreground job is again in a runnable state, at which point the processor will interrupt the background job and resume the foreground job.

In general, the RT-11 Foreground/Background System is designed to allow a time-critical job to run in the foreground, while the background does non-time-critical jobs, such as program development. (All RT-11 system programs run as the background job in a F/B system.) Thus, the user can run FORTRAN, BASIC, MACRO, etc. in the background while the foreground may be collecting data and storing and/or analyzing it.

Most user programs written for an RT-11 System can be linked (using the Linker described in Chapter 6) to run as the foreground job. There are a few coding restrictions, and these are explained in Appendix H, F/B Programming and Device Handlers. A foreground program has access to all of the features available to the background job (opening and closing files, reading and writing data, etc.). In addition, the F/B System gives the user the ability to set timer routines, suspend and resume F/B jobs, and send data and messages between the two jobs.

1.4.3 Facilities Available Only in RT-11 F/B

As mentioned previously, RT-11 F/B allows the user to write and execute two independent programs. Some features which are available only to the F/B user include:

1. Mark Time--This facility allows user programs to set clock timers to run for specified amounts of time. When the timer runs out, a routine specified by the user is entered. These may be as many mark time requests as desired, providing system queue space is reserved (see .QSET, Chapter 9).

2. Timed Wait--This feature allows the user program to "sleep" until the specified time increment elapses. Typically, a program may need to sample data every few seconds or even minutes. While the program is idle, the other job can run. The timed wait accomplishes this; when the time has elapsed, the issuing job is again runnable (see .TWAIT, Chapter 9).

3. Send Data/Receive Data--It is possible, under RT-11 F/B, to have the foreground and background programs communicate with one another. This is accomplished with the send/receive data functions. Using this facility, one program sends messages (or data) in variable size blocks to the other. This can be used, for example, to pass data from a foreground collection program directly to a background analysis program (see .SDAT/.RCVD, Chapter 9).
CHAPTER 2
SYSTEM COMMUNICATION

The monitor is the hub of RT-ll system communications; it provides access to system and user programs, performs input and output functions, and enables control of background and foreground jobs.

The user communicates with the monitor through programmed requests and keyboard commands. The keyboard commands (described in Section 2.7) are used to load and run programs, start or restart programs at specific addresses, modify the contents of memory, and assign and deassign alternate device names.

Programmed requests (described in detail in Chapter 9) are source program instructions which pass arguments to the monitor and request monitor services. These instructions allow user assembly language programs to utilize the available monitor features.

2.1 START PROCEDURE

After the system has been built (see the RT-ll System Generation Manual), the monitor can be loaded into memory from disk or DECtape as follows:

1. Press HALT.

2. Mount the system device on unit 0 (or the appropriate unit if a unit other than 0 is to be used).

3. WRITE PROTECT the system unit.

If the hardware configuration includes a hardware bootstrap capable of booting the system device,

1. Set the switch register to the appropriate address and press LOAD ADRS.

2. If a second address is required, set the switch register to that address.

3. Press START.

2-1 January 1976
System Communication

If a hardware bootstrap is not available, or if an RK disk unit other than Unit 0 is to be used as the system device, one of the following bootstraps must be entered manually using the Switch Register. First set the Switch Register to 1000 and press the LOAD ADRS switch. Then set the Switch Register to the first value shown for the appropriate bootstrap and raise the DEPosit switch. Continue depositing the values shown.

<table>
<thead>
<tr>
<th>DECTape (RK Disk other (RK11,RK05) than Unit 0)</th>
<th>Disk (RF11)</th>
<th>(RJ03/4)</th>
<th>(RF11/RP02)</th>
<th>(RX11/RX01)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12700</td>
<td>12700</td>
<td>12700</td>
<td>12705</td>
<td>12702</td>
</tr>
<tr>
<td>177344</td>
<td>177406</td>
<td>177406</td>
<td>177466</td>
<td>172044</td>
</tr>
<tr>
<td>12710</td>
<td>12710</td>
<td>12760</td>
<td>5010</td>
<td>12745</td>
</tr>
<tr>
<td>177400</td>
<td>177400</td>
<td>xxxxxx</td>
<td>4</td>
<td>12740</td>
</tr>
<tr>
<td>12740</td>
<td>12740</td>
<td></td>
<td>5</td>
<td>12740</td>
</tr>
<tr>
<td>4002</td>
<td>5</td>
<td>12700</td>
<td>177400</td>
<td>5</td>
</tr>
<tr>
<td>5710</td>
<td>105710</td>
<td>177406</td>
<td>12740</td>
<td>32715</td>
</tr>
<tr>
<td>105376</td>
<td>100376</td>
<td>12710</td>
<td>5</td>
<td>100200</td>
</tr>
<tr>
<td>12710</td>
<td>5007</td>
<td>177400</td>
<td>105710</td>
<td>1775</td>
</tr>
<tr>
<td>** n = 4 for unit 0</td>
<td>** n = 6 for unit 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>105710</td>
<td>5</td>
<td>12710</td>
<td>100376</td>
<td>5007</td>
</tr>
<tr>
<td>12710</td>
<td>5007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100376</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5007</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When all the values have been entered, set the switches to 1000 and press the LOAD ADRS and START switches.

The monitor loads into memory and prints one of the following identification messages followed by a dot (.) on the terminal:

RT-11SJ V02C-xx
RT-11FB V02C-xx

The message printed indicates which monitor (Single-Job or F/B) has been loaded; the user may determine which is to be loaded during the system build operation.

After the message has printed, the system device should be WRITE ENABLED. The monitor is ready to accept keyboard commands.
System Communication

To bring up an alternate monitor while under control of the one currently running (in this case, F/B), run PIP to perform the following operations:

1. Preserve the running monitor by renaming it to yyyyyy. SYS (the actual name yyyyyy is not significant, although it is suggested that yyMNSJ for Single-Job and yyMNFB for Foreground/Background be used to be consistent with system conventions; yy in this case represents the disk type):

   R PIP
   %RK0:RKNFB.SYS=RK0:MONITR.SYS/R/Y
   ?REBOOT?

2. Rename the desired monitor to MONITR.SYS:

   %RK0:MONITR.SYS=RK0:RKMNJSJ.SYS/R/Y
   ?REBOOT?

3. Write the new bootstrap from the new MONITR.SYS file (using the PIP /U option; A is a dummy filename, which must be present in the command line):

   %RK0:A=RK0:MONITR.SYS/U

4. Reboot the system.

   %RK0:/O

   RT-115J V02C-02

Refer to the RT-11 System Generation Manual for an example of switching monitors.
This page intentionally blank.
System Communication

2.2 SYSTEM CONVENTIONS

Special character commands, file naming procedures and other conventions that are standard for the RT-11 system are described in this section. The user should be familiar with these conventions before running the system.

2.2.1 Data Formats

The RT-11 system makes use of five types of data formats: ASCII, object, memory image, relocatable image, and load image.

Files in ASCII format conform to the American National Standard Code for Information Interchange, in which each character is represented by a 7-bit code. Files in ASCII format include program source files created by the Editor, listing and map files created by various system programs, and data files consisting of alphanumeric characters. A chart containing ASCII character codes appears in Appendix C.

Files in object format consist of data and PDP-11 machine language code. Object files are those output by the assembler or FORTRAN compiler and are used as input to the Linker.

The Linker can output files in memory image format (.SAV), relocatable image format (.REL), or load image format (.LDA).

A memory image file (.SAV) is a 'picture' of what memory will look like when a program is loaded. The file itself requires the same number of disk blocks as the corresponding number of 256-word memory blocks.

A relocatable image file (.REL) is one which can be run in the foreground. It differs from a memory image file in that the file is linked as though its bottom address were 0. When the program is called (using the monitor FRUN command), the file is relocated as it is loaded into memory. (A memory image file requires no such relocation.)
System Communication

A load image (or .LDA) file may be produced for compatibility with the PDF-11 Paper Tape System and is loaded by the absolute binary loader. LDA files can be loaded and executed in stand-alone environments without relocation.

2.2.2 Prompting Characters

The following table summarizes the characters typed by RT-11 to indicate to the user either that the system is awaiting user response or to specify which job (foreground or background) is producing output:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>The Keyboard Monitor is waiting for a command (see Section 2.3.2).</td>
</tr>
<tr>
<td>*</td>
<td>The Command String Interpreter is waiting for a command string specification as explained in Sections 2.3.3 and 2.5.</td>
</tr>
<tr>
<td>†</td>
<td>When the console terminal is being used as an input file, the uparrow prompts the user to enter information from the keyboard. If the input is entered under EDIT or BASIC (or any program that accepts input in special terminal mode as described in Chapter 9), the characters entered are not echoed. Typing a CTRL Z marks the end-of-file.</td>
</tr>
<tr>
<td>&gt;</td>
<td>The &gt; character is used (under the F/B Monitor and only if a foreground job is active) to identify which job, foreground or background, is producing the output currently appearing on the console terminal. Each time output from the background job is to appear, B&gt; is printed first, followed by the output. If the foreground job is to print output, F&gt; is typed first. B&gt; and F&gt; are also printed as a result of the CTRL B and CTRL F commands described in Table 2-4.</td>
</tr>
</tbody>
</table>

2.2.3 Physical Device Names

Devices are referenced by means of a standard two-character device name. Table 2-2 lists each name and its related device. If no unit number is specified for devices which have more than one unit, unit 0 is assumed.
### Table 2-2
Permanent Device Names

<table>
<thead>
<tr>
<th>Permanent Name</th>
<th>I/O Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTn:</td>
<td>TA11 cassette (n is the unit number, 0 or 1).</td>
</tr>
<tr>
<td>DK:</td>
<td>The default logical storage device for all files. DK is initially the same as SY: (see below), but the assignment (as a logical device name) can be changed with the ASSIGN Command (Section 2.7.2.4).</td>
</tr>
<tr>
<td>DKn:</td>
<td>The specified unit of the same device type as DK.</td>
</tr>
<tr>
<td>DPn:</td>
<td>RP02 disk (n is an integer in the range 0-7).</td>
</tr>
<tr>
<td>DSn:</td>
<td>RJS03/4 fixed-head disks (n is in the range 0-7).</td>
</tr>
<tr>
<td>DTn:</td>
<td>DECTape n, where n is a unit number (an integer in the range 0 to 7, inclusive).</td>
</tr>
<tr>
<td>DXn:</td>
<td>RX01 Floppy disk (n is 0 or 1).</td>
</tr>
<tr>
<td>LP:</td>
<td>Line printer.</td>
</tr>
<tr>
<td>MMn:</td>
<td>TJU16 magtape (n is in the range 0-7).</td>
</tr>
<tr>
<td>MTn:</td>
<td>TM11 (industry compatible) magtape (n is an integer between 0 and 7, inclusive).</td>
</tr>
<tr>
<td>PP:</td>
<td>High-speed paper tape punch.</td>
</tr>
<tr>
<td>PR:</td>
<td>High-speed paper tape reader.</td>
</tr>
<tr>
<td>RF:</td>
<td>RF11 fixed-head disk drive.</td>
</tr>
<tr>
<td>RKn:</td>
<td>RK disk cartridge drive n (n is in the range 0 to 7 inclusive).</td>
</tr>
<tr>
<td>SY:</td>
<td>System device; the device and unit from which the system is bootstrapped. (RT-11 allows bootstrapping from any RK unit; refer to Section 2.1.) The assignment as a logical device name can be changed with the ASSIGN command (Section 2.7.2.4).</td>
</tr>
<tr>
<td>SYN:</td>
<td>The specified unit of the same device type as that from which the system was bootstrapped.</td>
</tr>
<tr>
<td>TT:</td>
<td>Terminal keyboard and printer.</td>
</tr>
</tbody>
</table>

In addition to the fixed names shown in Table 2-2, devices can be assigned logical names. A logical name takes precedence over a physical name and thus provides device independence. With this feature a program that is coded to use a specific device does not need to be rewritten if the device is unavailable. Refer to Section 2.7.2.4 for instructions on assigning logical names to devices.

#### 2.2.4 File Names and Extensions

Files are referenced symbolically by a name of one to six alphanumeric characters followed, optionally, by a period and an extension of up to three alphanumeric characters. (Excess characters in a filename may cause an error message.) The extension to a filename generally indicates the format of a file. It is a good practice to conform to...
System Communication

the standard filename extensions for RT-11. If an extension is not specified for an input or output file, most system programs assign appropriate default extensions. Table 2-3 lists the standard extensions used in RT-11.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.BAD</td>
<td>Files with bad (unreadable) blocks; this extension can be assigned by the user whenever bad areas occur on a device. The .BAD extension makes the file permanent in that area, preventing other files from using it and consequently becoming unreadable.</td>
</tr>
<tr>
<td>.BAK</td>
<td>Editor backup file.</td>
</tr>
<tr>
<td>.BAS</td>
<td>BASIC source file (BASIC input).</td>
</tr>
<tr>
<td>.BAT</td>
<td>BATCH command file.</td>
</tr>
<tr>
<td>.CTL</td>
<td>BATCH control file generated by the BATCH compiler.</td>
</tr>
<tr>
<td>.CTT</td>
<td>BATCH internal temporary file.</td>
</tr>
<tr>
<td>.DAT</td>
<td>BASIC or FORTRAN data file.</td>
</tr>
<tr>
<td>.DIR</td>
<td>Directory listing file</td>
</tr>
<tr>
<td>.DMP</td>
<td>DUMP output file.</td>
</tr>
<tr>
<td>.FOR</td>
<td>FORTRAN IV source file (FORTRAN input).</td>
</tr>
<tr>
<td>.LDA</td>
<td>Absolute binary file (optional Linker output).</td>
</tr>
<tr>
<td>.LLD</td>
<td>Library listing file.</td>
</tr>
<tr>
<td>.LOG</td>
<td>BATCH log file.</td>
</tr>
<tr>
<td>.LST</td>
<td>Listing file (MACRO or FORTRAN output).</td>
</tr>
<tr>
<td>.MAC</td>
<td>MACRO or EXPAND source file (MACRO, EXPAND, SRCCOM input).</td>
</tr>
<tr>
<td>.MAP</td>
<td>Map file (Linker output).</td>
</tr>
<tr>
<td>.OBJ</td>
<td>Relocatable binary file (MACRO, ASEMBL, FORTRAN IV output, Linker input, LIBR input and output).</td>
</tr>
<tr>
<td>.PAL</td>
<td>Output file of EXPAND (the MACRO expander program), input file of ASEMBL.</td>
</tr>
<tr>
<td>.REL</td>
<td>Foreground job relocatable image (Linker output, default for monitor PRUN command).</td>
</tr>
<tr>
<td>.SAV</td>
<td>Memory image or SAVE file; default for R, RUN, SAVE and GET Keyboard Monitor commands; also default for output of Linker.</td>
</tr>
<tr>
<td>.SOU</td>
<td>Temporary source file generated by BATCH.</td>
</tr>
<tr>
<td>.SYS</td>
<td>System files and handlers.</td>
</tr>
</tbody>
</table>
System Communication

If a filename with a blank extension is to be used in a command line in which a default extension is assumed (by either the monitor or a system program), the user must insert a period after the filename to indicate that there is no extension. For example, to run the file TEST, type:

```
.RUN TEST.
```

If the period after the filename is not given, the monitor assumes the .SAV extension and attempts to run a file named TEST.SAV.

2.2.5 Device Structures

RT-11 devices are categorized by the physical structure of the device and the way in which the device allows information to be processed.

All RT-11 devices are either random-access or sequential-access devices. Random-access devices allow blocks of data to be processed in a random order — that is, independent of the data's physical location on the device or its location relative to any other information. All disks and DECTape fall into this category. Random-access devices are sometimes also called block-replaceable devices, because individual data blocks can be manipulated (rewritten) without affecting other data blocks on the device. Sequential-access devices require that data be processed sequentially; the order of processing data must be the same as the physical order of the data. RT-11 devices that are considered sequential devices are magtape, cassette, paper tape, card reader, line printer, and terminal.

File-structured devices are those devices that allow the storage of data under assigned filenames. RT-11 devices that are file-structured include all disks, DECTape, magtape, and cassette. Nonfile-structured devices, on the other hand, are those used to contain a single logical collection of data. These devices are used generally for reading and listing information, and include line printer, card reader, terminal, and paper tape devices.

Finally, file-structured devices are classified further as RT-11 directory-structured devices if they provide a standard RT-11 directory at the beginning of the device (the standard RT-11 directory is defined in the RT-11 Software Support Manual). The directory contains information about all files stored on the device and is updated each time a file is moved, added, or deleted from the device. RT-11 directory-structured devices include all disks and DECTapes. NonRT-11 directory-structured devices are file-structured devices that do not have the standard RT-11 directory structure at their beginning. For example, some devices, such as magtape and cassette, have directory-type information stored at the beginning of each file; the device must be read sequentially to obtain all information about all files.

It is possible to interface a device to the RT-11 system with a user-defined directory structure; procedures are explained in the RT-11 Software Support Manual.
System Communication

2.3 MONITOR SOFTWARE COMPONENTS

The main RT-11 monitor software components are:

- Resident Monitor (RMON)
- Keyboard Monitor (KMON)
- User Service Routine (USR) and Command String Interpreter (CSI)
- Device Handlers

The reader may find Figure 2-1 helpful while reading the following descriptions.

2.3.1 Resident Monitor (RMON)

The Resident Monitor is the only permanently memory-resident part of RT-11. The programmed requests for all services of RT-11 are handled by RMON. RMON also contains the console terminal service, error processor, system device handler, EMT processor, and system tables.

2.3.2 Keyboard Monitor (KMON)

The Keyboard Monitor provides communication between the user at the console and the RT-11 system. Monitor commands allow the user to assign logical names to devices, run programs, load device handlers, and control F/B operations. A dot at the left margin of the console terminal page indicates that the Keyboard Monitor is in memory and is waiting for a user command.

2.3.3 User Service Routine (USR)

The User Service Routine provides support for the RT-11 file structure. It loads device handlers, opens files for read or write operations, deletes and renames files, and creates new files. The Command String Interpreter (the use of which is described in Section 2.5) is part of the USR and can be accessed by any program to interpret device and file I/O information.
This page intentionally blank.
System Communication

2.3.4 Device Handlers

Device handlers for the RT-ll system perform the actual transfer of data to and from peripheral devices. New handlers can be added to the system as files on the system device and can be interfaced to the system by modifying a few monitor tables (see the RT-ll Software Support Manual, DEC-ll-ORPGA-B-D for instructions on how to interface a new handler to the RT-ll monitor).

2.4 GENERAL MEMORY LAYOUT

When the RT-ll System is first bootstrapped from the system device, memory is arranged as shown in the left diagram of Figure 2-1 (this is the case for either the Single-Job or Foreground/Background Monitor, since no foreground job exists yet). The background job is the RT-ll module KMON.

When an RT-ll foreground job is initiated (via the monitor FRUN command, Section 2.7.5.1), room is created for the foreground job to be loaded by decreasing the amount of space available to the background job. The memory maps in Figure 2-1 illustrate the system layout before and after a foreground job is loaded. (Refer also to Chapter 6, Section 6.5.)

![Memory Layout Diagram]

As shown in the figures, the process of loading a foreground job requires that the USR and KMON be physically moved. Once a foreground job is running, it is possible to communicate with either the background or foreground job via special commands (described in Section 2.7). All of the terminal support functions described in Section 2.6 are available under both the Single-job and F/B Monitors.

In addition to FRUN, other monitor commands can alter the memory map; these are LOAD, UNLOAD, GT ON, and GT OFF. LOAD causes device handlers to be made resident until an UNLOAD command is performed. UNLOAD deletes handlers which have been loaded. GT ON and GT OFF cause terminal service to utilize the VT-ll display hardware. Figure 2-2 illustrates the placement of display modules and device handlers in memory following the GT ON, LOAD, and FRUN commands.
System Communication

![Diagram of memory map](image)

**Figure 2-2**
RT-11 Memory Map (GT40)

RT-11 maintains a free memory list to manage memory. Thus, when a handler is unloaded, the space the handler occupied is returned to the free memory list and is reclaimed by the background.

### 2.4.1 Component Sizes

Following are the approximate sizes (in words) of the components for RT-11, Version 2C (sizes reflect RK).

<table>
<thead>
<tr>
<th></th>
<th>F/B</th>
<th>Single-job</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMON</td>
<td>3575(10)</td>
<td>1703(10)</td>
</tr>
<tr>
<td>USR</td>
<td>2050(10)</td>
<td>2050(10)</td>
</tr>
<tr>
<td>KMON</td>
<td>1800(10)</td>
<td>1540(10)</td>
</tr>
</tbody>
</table>

In the F/B system, the background area must always be large enough to hold KMON and USR (3.9K words). The following list indicates the total space available for the loaded device handlers, the foreground job, and the display handler. Note that the low memory area from 0-477 is never used for executable programs. (These sizes also allow room for the 3.5K RMON).

<table>
<thead>
<tr>
<th>Machine size (words)</th>
<th>Space available (words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16K</td>
<td>8.5K</td>
</tr>
<tr>
<td>24K</td>
<td>16.5K</td>
</tr>
<tr>
<td>28K</td>
<td>20.5K</td>
</tr>
</tbody>
</table>

With the Single-Job Monitor, RMON requires only 1.67K. The following list shows the amount of space available to users with the Single-Job Monitor:
System Communication

<table>
<thead>
<tr>
<th>Machine size (words)</th>
<th>Program space available (words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8K</td>
<td>6K</td>
</tr>
<tr>
<td>16K</td>
<td>14K</td>
</tr>
<tr>
<td>24K</td>
<td>22K</td>
</tr>
<tr>
<td>28K</td>
<td>26K</td>
</tr>
</tbody>
</table>

2.5 ENTERING COMMAND INFORMATION

Once either monitor has been loaded and a system program started, the user must enter the appropriate command information before any operation can be performed.

In most cases, the Command String Interpreter immediately prints an asterisk at the left margin. The user must then type a command string in the general format:

\[ \text{OUTPUT=INPUT/SWITCH} \]

(A few system programs -- EDIT, PATCH, PATCHO -- require that this command information be entered in a slightly different format. Complete instructions are provided in the appropriate chapter.)

In all cases, the format for OUTPUT is:

\[ \text{dev:filnam.ext[n],...dev:filnam.ext[n]} \]

INPUT is:

\[ \text{dev:filnam.ext,...dev:filnam.ext} \]

and SWITCH is:

\[ /s:oval \text{ or } /s!dval \]

where:

\[ \text{dev:} \] in each case is an optional two to three-character name from Table 2-2 whose usage conforms to the NOTE below.

\[ \text{filnam.ext} \] in each case is the name of a file (consisting of one to six alphanumeric characters followed option ally by a dot and a zero to three-character extension). As many as three output and six input files may be allowed.

\[ [n] \] is an optional declaration of the number of blocks (n) desired for an output file. n is a decimal number (\(<65,535\)) enclosed in square brackets immediately following the output filnam.ext to which it applies.

\[ /s:oval \text{ or } /s!dval \] is one or more optional switches whose functions vary according to the program in use (refer to the switch option table in the appropriate chapter). oval is either an octal number or one to three alphanumeric characters (the first of which must be alphabetic) which will be converted to radix-50 (see Section 5.5.4 of the MACRO chapter). dval is a decimal value preceded by an exclamation point.

January 1976 2-10
System Communication

Throughout this manual, the /s:oval construction is used; however, the /sidval format is always valid. Generally, these switches and their associated values, if any, should follow the device and filename to which they apply.

If the same switch is to be repeated several times with different values (e.g., /L:MEB/L:TTM/L:CND to MACRO) the line may be abbreviated as /L:MEB:TTM:CND; octal, RAD50, and decimal values may be mixed.

= if required, is a delimiter that separates the output and input fields. The < sign may be used in place of the = sign. The separator can be omitted entirely if there are no output files.

NOTE

As illustrated in the general format of a command line, the command line consists of an output list, a separator (= or <), and an input list. Omission of a device specification in either the input or output list is handled as follows:

DK: is assumed if the first file in a list has no explicit device. DK (or the device associated with the first file) is default until another device is indicated; that device then becomes default until a new one is used, and so on. If the following command is entered, for example, to MACRO:

*DT1:FIRST. OBJ, LP:=TASK. 1, RK1:TASK. 2, TASK. 3

it is interpreted as though all devices had been indicated as follows:

*DT1:FIRST. OBJ, LP=:DK:TASK. 1, RK1:TASK. 2, RK1:TASK. 3

2.6 KEYBOARD COMMUNICATION (KMON)

Special function keys and keyboard commands allow the user to communicate with the RT-ll monitor and allocate system resources, manipulate memory images, start programs, and use foreground/background services.

The special functions of certain terminal keys used for communication with the Keyboard Monitor are explained in Table 2-4. Note that in the F/B system, the Keyboard Monitor always runs as a background job.

CTRL commands are entered by holding the CTRL key down while typing the appropriate letter.

2-11 January 1976
# System Communication

## Table 2-4

**Special Function Keys**

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL A</td>
<td>Valid when the monitor GT ON command has been typed and the display is in use. The command does not echo on the terminal. It is used after a CTRL S has been typed to effectively page output. Console output is permitted to resume until the screen is completely filled; text previously displayed is scrolled upward off the screen. CTRL A has no special meaning if GT ON is not in effect or if a SET TTY NOPAGE command has been given (see Section 2.7.2.8).</td>
</tr>
<tr>
<td>CTRL B</td>
<td>Under the F/B Monitor echoes B&gt; on the terminal (unless output is already coming from the background job) and causes all keyboard input to be directed to the background job. At least one line of output will be taken from the background job (the foreground job has priority, and control will revert to it if it has output). All typed input will be directed to the background job until control is redirected to the foreground job (via CTRL F). CTRL B has no special meaning when used under a Single-Job Monitor or when a SET TTY NOFB command has been issued (see Section 2.7.2.8).</td>
</tr>
<tr>
<td>CTRL C</td>
<td>CTRL C echoes as C on the terminal and is used to interrupt program execution and return control to the keyboard monitor. If the program to be interrupted is waiting for terminal input, or is using the TT handler for input, typing one CTRL C is sufficient to interrupt execution; in all other cases, two CTRL Cs are necessary. Note that under the F/B Monitor, the job which is currently receiving input will be the job that is stopped (determined by whether a CTRL F or CTRL B was most recently typed). To ensure that the command is directed to the proper job, type CTRL B or CTRL F before typing CTRL C.</td>
</tr>
<tr>
<td>CTRL E</td>
<td>Valid when the monitor GT ON command has been typed and the display is in use. The command does not echo on the terminal, but causes all terminal output to appear on both the display screen and the console terminal simultaneously. A second CTRL E disables console terminal output. CTRL E has no special meaning if GT ON is not in effect.</td>
</tr>
<tr>
<td>CTRL F</td>
<td>Under the F/B Monitor echoes F&gt; on the terminal and instructs that all keyboard input be directed to the foreground job and all output be taken from the foreground job. If no foreground job exists, F? is printed and control is directed to the background job. Otherwise, control remains with the foreground job until redirected to the background job (via CTRL B) or until the foreground job terminates. CTRL F has no special meaning when used under a Single-Job Monitor, or when a SET TTY NOFB command has been used (see Section 2.7.2.8).</td>
</tr>
</tbody>
</table>
Table 2-4 (Cont.)
Special Function Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
</table>
| CTRL O | Echoes \( \text{tO} \) on the terminal and causes suppression of teleprinter output while continuing program execution. Teleprinter output is re-enabled when one of the following occurs:  
  1. A second CTRL O is typed,  
  2. A return to the monitor occurs, or  
  3. The running program issues a \( \text{,RCtrlO} \) directive (see Chapter 9). (RT-ll system programs reset CTRL O to the echoing state each time a new command string is entered.) |
| CTRL Q | Does not echo. Resumes printing characters on the terminal from the point at which printing was previously stopped (via CTRL S). CTRL Q has no special meaning if a SET TTY NOPAGE command has been used (see Section 2.7.2.8). |
| CTRL S | Does not echo. Temporarily suspends output to the terminal until a CTRL Q is typed. If GT ON is in effect, each subsequent CTRL A causes output to proceed until the screen has been refilled once. This feature allows users with high-speed terminals to fill the display screen, stop output with CTRL S, read the screen, and then continue with CTRL Q or CTRL A. (Typing CTRL C in this case also continues output.) Under the F/B Monitor, CTRL S has no special meaning if a SET TTY NOPAGE has been used. |
| CTRL U | Deletes the current input line and echoes as \( \text{tU} \) followed by a carriage return at the terminal. (The current line is defined to be all characters back to, but not including, the most recent line feed, CTRL C or CTRL Z.) |
| CTRL Z | Echoes \( \text{tZ} \) on the terminal and terminates input when used with the terminal device handler (TT). The CTRL Z itself does not appear in the input buffer. If TT is not being used, CTRL Z has no special meaning. |
| RUBOUT | Deletes the last character from the current line and echoes a backslash plus the character deleted. Each succeeding RUBOUT deletes and echoes another character. An enclosing backslash is printed when a key other than RUBOUT is typed. This erasure is done right to left up to the beginning of the current line. |

2.6.1 Foreground/Background Terminal I/O

It is important to note that console input and output under F/B are independent functions; input can be typed to one job while output is printed by another. The user may be in the process of typing input to one job when the other job is ready to print on the terminal. In this case, the job which is ready to print interrupts the user and prints the message on the terminal; input control is not re-directed to this job, however, unless a CTRL B or CTRL F is explicitly typed. If input is typed to one job while the other has output
System Communication

control, echo of the input is suppressed until the job accepting input gains output control; at this point all accumulated input is echoed.

If the foreground job and background job are both ready to print output at the same time, the foreground job has priority. Output from the foreground job prints until a line feed is encountered, at which point output from the background job prints until a line feed is encountered, and so forth.

When the foreground job terminates, control reverts automatically to the background job.

2.6.2 Type-Ahead

The monitor has a type-ahead feature which allows terminal input to be entered while a program is executing. For example:

```
.R PIP
*DT1:TAPE=PR:
DT1:/L
*13-FEB-74
TAPE  78 13-FEB-74
486 FREE BLOCKS
```

While the first command line is executing, the second line (DT1:/L) is entered by the user. This terminal input is stored in a buffer and used when the first operation has completed.

If a single CTRL C is typed while in this mode, it is put into the buffer. The program currently executing exits when a terminal input request needs to be satisfied. A double CTRL C returns control to the monitor immediately.

If type-ahead input exceeds 80 characters, the terminal bell rings and no characters are accepted until part of the type-ahead buffer is used by a program or characters are deleted. No input is lost. Type-ahead is particularly useful in specifying multiple command lines to system programs, as shown in the preceding example. If a job is terminated by typing two CTRL C's, any unprocessed type-ahead is discarded.

NOTE

If type-ahead is used in conjunction with EDIT or BASIC, there is no terminal echo of the characters but they are stored in the buffer until a new command is needed. The characters are echoed only when actually used by the program.

2.7 KEYBOARD COMMANDS

Keyboard commands allow the user to communicate with the monitor. Keyboard commands can be abbreviated; optional characters in a command are delimited (in this section only) by braces. Keyboard commands require at least one space between the command and the first argument. All command lines are terminated by a carriage return.
System Communication

All commands, with the exception of those described in Section 2.7.5, may be used under either the Single-Job or F/B Monitor. The commands described in Section 2.7.5 apply only to the F/B Monitor.

NOTE

Any reference made to "the background job" applies as well to the Single-Job Monitor, since the background job in a F/B system is equivalent to the single-job environment in its normal state.

2.7.1 Commands to Control Terminal I/O (GT ON and GT OFF)

The GT ON and GT OFF commands are used to enable and disable the scroller (VT-11 display hardware). GT ON causes the display screen to replace the console as the terminal output device. Switch options allow the user to control the number of lines to appear on the screen and to position the first line vertically. Output appears on the display in the same format as it would on the console (i.e., output, text, and commands are displayed in the order in which they occur). GT ON is not permitted in an 8K configuration.

The form of the GT ON command is:

```
GT ON{/L:n}//T:n
```

where:

/L:n represents an optional switch setting indicating the number of lines of text to display; the suggested range is:

- 12" screen (GT40, DEClab) 1<=n<=37 octal (31 decimal)
- 17" screen (GT44) 1<=n<=50 octal (40 decimal)

/T:n represents an optional switch setting indicating the top position of the scroll display; the suggested range is:

- 12" screen (GT40, DEClab) 1<=n<=1350 octal (744 decimal)

2-15
System Communication

17" screen
(GT44) 1<=n<=1750 octal (1000 decimal)

If no switches are specified, a test for the screen size is performed and default values are automatically assigned as follows:

12" screen
(GT40, DEClab) /L:37 (31 decimal)
/T:1350 (744 decimal)

17" screen
(GT44) /L:50 (40 decimal)
/T:1750 (1000 decimal)

Line length is always set to 72 for 12" screen and 80 for 17" screen. Once the display has been activated with the GT ON command, CTRL A, CTRL S, CTRL E and CTRL Q can be used to control scrolling behavior. These commands are described in Section 2.6.

NOTE
ODT is one exception to the use of GT ON. This system program has its own terminal handler and cannot make use of the display; output will appear only on the console terminal whenever ODT is running.

The GT OFF command clears the display and resumes output on the teleprinter. The command format is:

GT OFF

If GT ON and GT OFF are used when no display hardware exists or when a foreground job is active, the ?ILL CMD? message is printed.

2.7.2 Commands to Allocate System Resources

DATE

2.7.2.1 DATE Command - The DATE command enters the indicated date to the system. This date is then assigned to newly created files, new device directory entries (which may be listed with PIP), and listing output until a new DATE command is issued.

The form of the command is:

DATE \{dd-mm-yy\}

where dd-mm-yy is the day, month and year to be entered. dd is a decimal number in the range 1-31; mmm is the first three characters of the name of the month, and yy is a decimal number in the range 73-99. If no argument is given, the current date is printed.

January 1976 2-16
System Communication

Examples:

.DATE 21-FEB-74  Enter the date 21-FEB-74 as the current system date.

.DATE 21-FEB-74  Print the current date.

If the date is entered in an incorrect format, the ?DAT? error message is printed.

The form of the command is:

\[ \text{TIME} \{E\} \{hh:mm:ss\} \]

where \( hh:mm:ss \) represents the hour, minute, and second. Time is represented as hours, minutes, and seconds past midnight in 24-hour format (e.g., 1:25:00 P.M. is entered as 13:25:00). If any of the arguments are omitted, 0 is assumed. If no argument is given, the current time of day is output.

Examples:

.TIM 8:15:23  Sets the time of day to 8 hours, 15 minutes and 23 seconds.

.TIM 08:25:27  Approximately 10 minutes later, the TIME command outputs this time.

.TIME 18:5  Sets the time of day to 18:05:00.

Under the F/B Monitor, after the time reaches 24:00, the time and date will be reset when the user next issues a TIME command (or .GTIM pro-

grammed request). Time and date are not reset under the Single-Job Monitor. Month and year are not updated under either monitor.

The clock rate is initially set to 60-cycle. Consult the RT-11 System Generation Manual if conversion to a 50-cycle rate is necessary.
2.7.2.3 INITIALIZE Command - The INITIALIZE command is used to reset several background system tables and do a general "clean-up" of the background area; it has no effect on the foreground job. In particular, this command makes non-resident those handlers which were not loaded (via LOAD), purges the background's I/O channels, disables CTRL O, performs a hard reset, clears locations 40-53, resets the KMON stack pointer, and under the F/B monitor performs an .UNLOCK.

Under the Single-Job Monitor a RESET instruction is done (see Chapter 9). Under the F/B Monitor, I/O is stopped by entering each busy device handler at a special abort entry point.

The form of the command is:

\[ \text{IN\{INITIALIZE\}} \]

The INITIALIZE command can be used prior to running a user program, or when the accumulated results of previously issued GET commands (see Section 2.7.3.1) are to be discarded.

Example:

\[ \text{IN \ R PROG} \]

Initializes system background job

2.7.2.4 ASSIGN Command - The ASSIGN command assigns a user-defined (logical) name as an alternate name for a physical device. This is especially useful when a program refers to a device which is not available on a certain system. Using the ASSIGN command, I/O can be redirected to a device which is available. Only one logical name can be assigned per ASSIGN command, but several ASSIGN commands (14 maximum) can be used to assign different names to the same device. This command is also used to assign FORTRAN logical units to device names.
System Communication

The form of the command is:

```
ASSIGN IGN \{ \{dev\}:udev \}
```

where:

dev is any standard RT-11 (physical) device name (refer to Table 2-2) with the exception of DK and SY.

udev is a 1-3 character alphanumeric (logical) name to be used in a program to represent dev (if more than three characters are given, only the first three are actually used). DK and SY may be used as logical device names.

: is a delimiter character (can be a colon, equal sign, and, if separating physical and logical devices, space).

The placement of the delimiter is very important in the ASSIGN command; it must be placed exactly as shown in the following examples:

```
ASSIGN DT1 INP
```

Physical device DT1 is assigned the logical device name INP. Whenever a reference to INP: is encountered, device DT1: is used.

```
ASSIGN DT3:DK
```

Physical device name DT3 is assigned the default device name DK. Whenever DK is referenced or defaulted to, DT3 is used. (Note that the initial assignment of DK is thus changed.)

```
ASSIGN LP=9
```

FORTRAN logical unit 9 becomes the physical device name LP. All references to unit 9 use the line printer for output.

Assignment of logical names to logical names is not allowed.

If only a logical device name is indicated in the command line, that particular assignment (only) is removed. Thus:

```
ASSIGN :9
```

Deassigns the logical name 9 from its physical device (LP, in the case above).

```
ASSIGN =DK
```

Removes assignment of logical name DK from its physical device (DT3, in the case above).

If neither a physical device name nor a logical device name is indicated, all assignments to all devices are removed.

```
ASSIGN
```

All previous logical device assignments are removed.

2-19

January 1976
System Communication

CLOSE

2.7.2.5 CLOSE Command - The CLOSE command causes all currently open output files in the background job to become permanent files. If a tentative open file is not made permanent, it will eventually be deleted. The CLOSE command is most often used after CTRL C has been typed to abort a background job and to preserve any new files that job had open prior to the CTRL C; it has no effect on a foreground job.

The form of the command is:

CLOSE

The CLOSE command makes temporary directory entries permanent.

Example:

R EDIT
*EWTEXT$$
*IRBCD$$
*^C

CLOSE

LOAD

2.7.2.6 LOAD Command - The LOAD command is used to make a device handler resident for use with background and foreground jobs. Execution is faster when a handler is resident, although memory area for the handler must be allocated. Any device handler to be used by a foreground job must be loaded before it can be used.

The form of the command is:

LOAD \{ \} dev \{ ,dev=B \} \{ ,dev=F,... \}

where:

dev represents any legal RT-ll device name.

= represents a delimiter, denoting device ownership.

B represents the background job.

F represents the foreground job.

The dev=F (and dev=B) construction is valid only under the Foreground/Background system. When used under the Single-Job Monitor, the ?ILL EV7 error message occurs.
System Communication

A device may be owned exclusively by either the foreground or background job. This may be used, for example, to prevent the I/O of two different jobs from being intermixed on the same non-file structured device. For example:

```
LOAD PP=B,PR,LF=F
```

The papertape punch belongs to the background job while the paper tape reader is available for use by either the background or foreground job; the line printer is owned by the foreground job. All three handlers are made resident in memory.

Different units of the same random-access device controller may be owned by different jobs. Thus, for example, DT1 may belong to the background while DT5 may belong to the foreground job. If no ownership is indicated, the device is available for public use.

To change ownership of a device, another LOAD command may be used; it is not necessary to first UNLOAD the device. For example, if RK1 has been assigned to the foreground job as in the example above, the command:

```
LOAD RK1=F
```

reassigns it to the background job.

The system unit of the system device cannot be assigned ownership, and attempts to do so will be ignored. Other units of the same type as the system device, however, can be assigned ownership.

LOAD is valid for use with user-assigned names. For example:

```
ASSIGN RK2:XY
LOAD XY=F
```

If the Single-Job, DECTape-based Monitor is being used, loading the necessary device handlers into memory can significantly improve the throughput of the system, since no handlers need to be loaded dynamically (in other words, they need not be loaded, as required, from the DECTape).

2.7.2.7 UNLOAD Command - The UNLOAD command is used to make handlers that were previously LOADed non-resident, freeing the memory they were using.

2-21
January 1976
System Communication

The form of the command is:

\[ \text{UNL}\{\text{LOAD}\} \text{ dev } \{\text{dev,} ...\} \]

where:

\text{dev} \quad \text{represents any legal RT-11 device name.}

UNLOAD clears ownership for all units of an indicated device type. For example, typing:

\[ \text{UNL RK2} \]

clears all units of RK. (A request to unload the system device handler clears ownership for any assigned units for that device, but the handler remains resident.)

Any memory freed is returned to a free memory list and eventually reclaimed for the background job after the UNLOAD command is given. UNLOAD is not permitted if the foreground job is running. Such an action might cause a handler which is needed by the foreground job to become non-resident.

Example:

\[ \text{UNLOAD LP, PP} \]

The lineprinter and paper tape punch handlers are released and the area which they used is freed.

A special function of this command is to remove a terminated foreground job and reclaim memory, since the space occupied by the foreground job is not automatically returned to the free memory list when it finishes. In this instance, the device name FG is used to specify the foreground job. For example:

\[ \text{UNL FG} \]

FG can be mixed with other device names.

However, if, for example, DT2 has been assigned the name FG and loaded into memory as follows:

\[ \text{ASSIGN DT2;FG} \]
\[ \text{LOAD FG} \]

the command:

\[ \text{UNLOAD FG} \]

causes the foreground job, not DT2, to be unloaded. To unload DT2, this command must be typed:

\[ \text{UNLOAD DT2} \]
2.7.2.8 SET Command - The SET command is used to change device handler characteristics and certain system configuration parameters.

The form of the command is:

```
SET dev:{NO}option=value{NO}option=value,...
```

where:

- `dev:` represents any legal RT-11 physical device name (and in addition, TTY and USR).
- `{NO}option` is the feature or characteristic to be altered.
- `value` is a decimal number required in some cases.

A space may be used in place of or in addition to the colon, equal sign, or comma. Note that the device indicated (with the exception of TTY and USR) must be a physical device name and is not affected by logical device name assignments which may be active. The name of the characteristic or feature to be altered must be legal for the indicated device (see Table 2-5) and may not be abbreviated.

The SET command locates the file SY:dev.SYS and permanently modifies it. No modification is done if the command entered is not completely valid. If a handler has already been loaded when a SET command is issued for it, the modifications will not take effect until the handler is unloaded and a fresh copy called in from the system device.

Table 2-5 lists the system characteristics and parameters which may be altered (those modes designated as "normal" are the modes as set in the distribution copies of the drivers):

<table>
<thead>
<tr>
<th>Device</th>
<th>Option</th>
<th>Alteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>CR</td>
<td>Allows carriage returns to be sent to the printer. The CR option should be set for any FORTRAN program using formatted I/O, to allow the overstriking capability for any line printer, and when using the LS11 or LP05 line printers (since the last line in the buffer may otherwise be lost). <strong>This is the normal mode.</strong></td>
</tr>
<tr>
<td>LP</td>
<td>NOCR</td>
<td>Inhibits sending carriage returns to the line printer. The line printer controller causes a line feed to perform the functions of a carriage return, so using this option produces a significant increase in printing speed on LP11 printers.</td>
</tr>
<tr>
<td>LP</td>
<td>CTRL</td>
<td>Causes all characters, including nonprinting control characters, to be passed to the line printer. This mode may be used for LS11 line printers. (Other line printers will print space for control characters.)</td>
</tr>
</tbody>
</table>

(continued on next page)
## Table 2-5 (Cont.)
### SET Command Options

<table>
<thead>
<tr>
<th>Device</th>
<th>Option</th>
<th>Alteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>LP</td>
<td>NOCTRL</td>
<td>Ignores nonprinting control characters. This is the normal mode.</td>
</tr>
<tr>
<td>LP</td>
<td>FORM0</td>
<td>Causes a form feed to be issued before a request to print block zero. This is the normal mode.</td>
</tr>
<tr>
<td>LP</td>
<td>NOFORM0</td>
<td>Turns off FORM0 mode.</td>
</tr>
<tr>
<td>LP</td>
<td>HANG</td>
<td>Causes the handler to wait for user correction if the line printer is not ready or becomes not ready during printing. This is the normal mode.</td>
</tr>
</tbody>
</table>

New users should note that when expecting output from the line printer and it appears as though the system is not responding or is in an idle state, the line printer should be checked to see if it is on and ready to print.

| LP     | NOHANG   | Generates an immediate error if the line printer is not ready.                                                                                                                                           |
| LP     | LC       | Allows lower case characters to be sent to the printer. This option should be used if the printer has a lower case character set.                                                                      |
| LP     | NOLC     | Causes lower case characters to be translated to upper case before printing. This is the normal mode.                                                                                                  |
| LP     | WIDTH=n  | Sets the line printer width to n, where n is a number between 30 and 255. Any characters printed past column n are ignored. The NO modifier is not permitted.                                               |
| CR     | CODE=n   | Modifies the card reader handler to use either the DEC 026 or the DEC 029 card codes (refer to Appendix H). n must be either 26 or 29. The NO modifier is not permitted.                                         |
| CR     | CRLF     | Causes a carriage return/line feed to be appended to each card image. This is the normal mode.                                                                                                           |
| CR     | NOCRLF   | Transfers each card image without appending a carriage return/line feed.                                                                                                                               |
| CR     | HANG     | Causes the handler to wait for user correction if the reader is not ready at the start of a transfer. This is the normal mode.                                                                     |
| CR     | NOHANG   | Generates an immediate error if the device is not ready at the start of a transfer. Note that the handler will wait regardless of how the option is set if the reader becomes "not ready" during a transfer (i.e., the input hopper is empty, but an end-of-file card has not yet been read). |

January 1976 2-24 (continued on next page)
### Table 2-5 (Cont.)
#### SET Command Options

<table>
<thead>
<tr>
<th>Device</th>
<th>Option</th>
<th>Alteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>IMAGE</td>
<td>Causes each card column to be stored as a 12-bit binary number, one column per word. The CODE option has no effect in IMAGE mode. The format of the 12-bit binary number is:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POP-11 WORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This format allows binary card images to be read and is especially useful if a special encoding of punch combinations is to be used. Mark-sense cards may be read in IMAGE mode.</td>
</tr>
<tr>
<td>CR</td>
<td>NOIMAGE</td>
<td>Allows the normal translation (as specified by the CODE option) to take place; data is packed one column per byte. Invalid punch combinations are translated into the error character, ASCII &quot;&quot; (backslash), which is octal code 134. This is the normal mode.</td>
</tr>
<tr>
<td>CR</td>
<td>TRIM</td>
<td>Causes trailing blanks to be removed from each card read. It is not recommended that TRIM and NOCRLF be used together since card boundaries will be difficult to find. This is the normal mode.</td>
</tr>
<tr>
<td>CR</td>
<td>NOTRIM</td>
<td>Transfers a full 80 characters per card.</td>
</tr>
<tr>
<td>CT</td>
<td>RAW</td>
<td>Causes the cassette handler to perform a read-after-write check for every record written, and retry if an output error occurred. If three retries fail, an output error is detected.</td>
</tr>
<tr>
<td>CT</td>
<td>NORAW</td>
<td>Causes the cassette handler to write every record directly without waiting for it back for verification. This significantly increases transfer rates at the risk of increased error rates. Normal mode is NORAW.</td>
</tr>
</tbody>
</table>

The following options, with the exception of HOLD/NOHOLD and COPY/NOCOPY, are available in the Foreground/Background System only; HOLD/NOHOLD and COPY/NOCOPY are available in both systems. These options are not permanent, and must be reissued whenever the monitor is re-bootstrapped. They can be made permanent by modifying the monitor as described in Chapter 2 of the RT-11 Software Support Manual. (Note that the device specification is TTY, not TT, because the handler itself is not changed.)

| TTY    | COPY    | Enables use of the auto-print mode of the VT50 copier option, if present. The command is a no-op for any terminal other than the VT50, but a "| character may be printed on the terminal. Consult the VT50 Video Terminal Programmer's Manual for more information. |
|        | NOCOPY  | Enables use of the auto-print mode of the VT50 copier option, if present. The command is a no-op for any terminal other than the VT50, but a "| character may be printed on the terminal. This is the normal mode. |

(continued on next page)

January 1976
### Table 2-5 (Cont.)

**SET Command Options**

<table>
<thead>
<tr>
<th>Device</th>
<th>Option</th>
<th>Alteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTY</td>
<td>CRLF</td>
<td>Causes the monitor to issue a carriage return/line feed on the console terminal whenever it attempts to type past the right margin (as set by the WIDTH option). This is the normal mode.</td>
</tr>
<tr>
<td>TTY</td>
<td>NOCRLF</td>
<td>Causes no special action to be taken at the right margin.</td>
</tr>
<tr>
<td>TTY</td>
<td>FB</td>
<td>Causes the monitor to treat CTRL B and CTRL F characters as background and foreground program control characters and does not transmit them to the user program. This is the normal mode.</td>
</tr>
<tr>
<td>TTY</td>
<td>NOFB</td>
<td>Causes CTRL B and CTRL F to have no special meaning.</td>
</tr>
</tbody>
</table>

**NOTE**

SET TTY NOFB is issued to KMON, (which runs as a background job) and disables all communication with the foreground job. To enable communication with the foreground job, issue the command SET TTY FB.

| TTY    | FORM   | Indicates that the console terminal is capable of executing hardware form feeds. |
| TTY    | NOFORM | Causes the monitor to simulate form feeds by typing eight line feeds. This is the normal mode. |
| TTY    | HOLD   | Enables use of the hold screen mode of operation for the VT50 terminal. The command is a no-op for any terminal other than the VT50, but a "[" character may be printed on the terminal. The command is valid for F/B and Single-Job Monitors. Consult the VT50 Video Terminal Programmer's Manual for more information. |
| TTY    | NOHOLD | Disables use of the hold screen mode of operation for the VT50 terminal. The command is a no-op for any terminal other than the VT50, but a "\" character may be printed on the terminal. This is the normal mode. |
| TTY    | PAGE   | Causes the monitor to treat CTRL S and CTRL Q characters as terminal output hold and unhold flags, and does not transmit them to the user program. This is the normal mode. |

(continued on next page)
System Communication

<table>
<thead>
<tr>
<th>Device</th>
<th>Option</th>
<th>Alteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTY</td>
<td>NOPAGE</td>
<td>Causes CTRL S and CTRL Q to have no special meaning.</td>
</tr>
<tr>
<td>TTY</td>
<td>SCOPE</td>
<td>Causes the monitor to echoed RUBOUTs as backspace-space-backspace. This mode should be used when the console is a VT05/VT50 or when GT ON is in effect.</td>
</tr>
<tr>
<td>TTY</td>
<td>NOSCOPE</td>
<td>Causes the monitor to echoed RUBOUTs as backslash followed by the character deleted. This is the normal mode.</td>
</tr>
<tr>
<td>TTY</td>
<td>TAB</td>
<td>Indicates that the console terminal is capable of executing hardware tabs.</td>
</tr>
<tr>
<td>TTY</td>
<td>NOTAB</td>
<td>Causes the monitor to simulate tab stops every eight positions. The normal mode is NOTAB. VT05/VT50 terminals generally have hardware tabs.</td>
</tr>
<tr>
<td>TTY</td>
<td>WIDTH=n</td>
<td>Sets the width of the console terminal to n positions, for the use of the CRLF option. n must be in the range 30-255 (decimal). The width is initially set to 72.</td>
</tr>
</tbody>
</table>

The following variant of the SET command is used to prevent the background job from ever placing the USR in a swapping state (note that USR replaces a device specification in the command line):

```
SET USR {NO}SWAP
```

This is useful when running on a DECtape based system, or when running a foreground job which requires the USR but has no memory allocated into which to read it. When the monitor is bootstrapped, it is in the SWAP condition, i.e., the background may place the USR in a swapping state via a SETTOP.

The Single-Job Monitor behaves as though the following options are set: NOTAB, NOFORM, PAGE, NOCRLF, NOSCOPE, NOHOLD.

2-27
System Communication

2.7.3 Commands to Manipulate Memory Images

GET

2.7.3.1 GET Command - The GET command (valid for use with a background job only) loads the specified memory image file (not ASCII or object) into memory from the indicated device.

The form of the GET command is:

\[
\text{GET} \{T\} \text{dev:filnam.ext}
\]

where:

- `dev:` represents any legal RT-ll device name. If a device is not specified, DK is assumed. Note that devices MT and CT are not block-replaceable devices and therefore cannot be used in a GET command.

- `filnam.ext` represents a valid RT-ll filename and extension. If an extension is not specified, the extension .SAV is assumed.

The GET command is typically used to load a program into memory for modification and/or debugging. The GET command can also be used in conjunction with the Base, Examine, Deposit, and START commands to test patches, and can be used with SAVE to make patches permanent. Multiple GETs can be used to combine programs. Thus:

\[
\begin{align*}
\text{.GET ODT.SAV} & \quad \text{Loads ODT into memory} \\
\text{.GET PROG} & \quad \text{Loads PROG.SAV into memory with ODT} \\
\text{.START (ODTs starting address)} & \quad \text{Starts execution with ODT (see Chapter 8).}
\end{align*}
\]

The GET command cannot be used to load overlay segments of programs; it may only be used to load the root segment (that part which will not be overlaid; refer to Chapter 6, Linker).

Multiple GETs can be used to build a memory image of several programs. If identical locations are required by any of the programs, the later programs overlay the previous ones.

Examples:

\[
\begin{align*}
\text{GET DT3:FILE1.SAV} & \quad \text{Loads the file FILE1.SAV into memory from DECTape unit 3.} \\
\text{GET NAME1} & \quad \text{Loads the file NAME1.SAV from device DK.}
\end{align*}
\]

January 1976 2-28
2.7.3.2 Base Command - The B command sets a relocation base. This relocation base is added to the address specified in subsequent Examine or Deposit commands to obtain the address of the location to be referenced. This command is useful when referencing linked modules with the Examine and Deposit commands. The base address can be set to the address where the module of interest is loaded. The form of the command is:

\[ B \{\text{location}\} \]

where:

- location represents an octal address used as a base address for subsequent Examine and Deposit commands.

**NOTE**

A space must follow the B command even if an address is not specified (the B(space) command is equivalent to B 0).

Any non-octal digit terminates an address. If location is odd, it is rounded down by one to an even address.

The base is cleared whenever user program execution is initiated.

**Examples:**

- \[ .B \Lambda \] Sets base to 0 (\Lambda represents space).
- \[ .B 200 \] Sets base to 200.
- \[ .B 201 \] Sets base to 200.
System Communication

**EXAMINE**

2.7.3.3 Examine Command - The E command prints the contents of the specified location(s) in octal on the console terminal. The form of the Examine command is:

\[ E \text{ location m}(\text{-location n}) \]

where:

- location represents an octal address which is added to the relocation base value (the value set by the B Command) to get the actual address examined. Any non-octal digit terminates an address. An odd address is truncated to become an even address.

If more than one location is specified (location m-location n), the contents of location m through location n inclusive are printed. The second location specified (location n) must not be less than the first location specified, otherwise an error message is printed. If no location is specified, the contents of location 0 are printed. Examination of locations outside the background area is illegal.

Examples:

1. \[ E 1000 \]
   \[ 127401 \]
   Prints contents of location 1000 (added to the base value if other than 0).

2. \[ E 1001-1012 \]
   \[ 127401 \text{ 007624 127400 000000 000000 000000} \]
   Prints the contents of locations 1000 (plus the base value if other than 0) through 1013.

**DEPOSIT**

2.7.3.4 Deposit Command - The Deposit command deposits the specified value(s) starting at the location given.

The form of the command is:

\[ D \text{ location=value1,value2,...,valuen} \]
System Communication

where:

location represents an octal address which is added to the relocation base value to get the actual address where the values are deposited. Any non-octal digit is accepted as a terminator of an address.

value represents the new contents of the location. 0 is assumed if a value is not indicated.

If multiple values are specified (value1,...,valuen), they are deposited beginning at the location specified. The DEPOSIT command accepts word or byte addresses but executes the command as though a word address was specified. An odd address is truncated by one to an even address. All values are stored as word quantities.

Any character that is not an octal digit may be used to separate the locations and values in a DEPOSIT command. However, two (or more) non-octal separators cause 0's to be deposited at the location specified (and those following). For example:

.D 56,,, Deposits 0's in locations 56, 60, and 62.

The user should be aware of situations like the above, which causes system failure since the terminal vector (location 60) is zeroed.

An error results when the address specified references a location outside the background job's area.

Examples:

.D 1000=3705 Deposits 3705 into location 1000
.B 1000 Sets relocation base to 1000
.D 1500=2503 Puts 2503 into location 2500
.B 0 Resets base to 0

2.7.3.5 SAVE Command - The SAVE command writes specified user memory areas to a named file and device in save image format. Memory is written from location 0 to the highest memory address specified by the parameter list or to the program high limit (location 50 in the system communication area).

The SAVE command does not write the overlay segments of programs; it saves only the root segment (refer to Chapter 6, Linker).

The form of the command is:

SAV{E} dev:filnam.ext [parameters]

where:

dev: represents one of the standard RT-11 block-replaceable device names. If no device is specified, DK is assumed.
System Communication

file.ext represents the name to be assigned to the file being saved. If the file name is omitted, an error message is output. If no extension is specified, the extension .SAV is used.

parameters represent memory locations to be saved. RT-11 transfers memory in 256-word blocks beginning on boundaries that are multiples of 256(decimal). If the locations specified make a block of less than 256 words, enough additional locations are transferred to make a 256-word block.

Parameters can be specified in the following format:

areal,area2-arean

where:

areal area2-arean represent an octal number (or numbers separated by dashes). If more than one number is specified, the second number must be greater than the first.

The start address and the Job Status Word are given the default value 0 and the stack is set to 1000. If the user wants to change these or any of the following addresses, he must first use the DEPOSIT command to alter them and then SAVE the correct areas:

<table>
<thead>
<tr>
<th>Area</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start address</td>
<td>40</td>
</tr>
<tr>
<td>Stack</td>
<td>42</td>
</tr>
<tr>
<td>JSW</td>
<td>44</td>
</tr>
<tr>
<td>USR address</td>
<td>46</td>
</tr>
<tr>
<td>High address</td>
<td>50</td>
</tr>
<tr>
<td>Fill characters</td>
<td>56</td>
</tr>
</tbody>
</table>

If the values of the addresses are changed, it is the user's responsibility to reset them to their default values. See Chapter 9 for more information concerning these addresses.

Examples:

```
.SAVE FILE1 10000-11000,14000-14100
Saves locations 10000(8) through 11777(8) (11000 starts the first word of a new block, therefore the whole block, up to 12000, is stored) and 14000(8) through 14777(8) on DK with the name FILE1.SAV.

.SAVE DT1:NAM.NEW 10000
Saves locations 10000 through 10777 on DT1: with the name NAM.NEW.

.D 44:20000
.SAV SY:PRHM 1000-5777
Sets the reenter bit in the JSW and saves locations 1000 through 5777.
```
2.7.4 Commands to Start a Program

2.7.4.1 RUN Command - The RUN command (valid for use with a background job only) loads the specified memory image file into memory and starts execution at the start address specified in location 40. Under the F/B system, 10 words of user stack area are required to start a user program, and the stack address (location 42) must be initialized to some part of memory where these 10 words will not modify it.

The form of the command is:

\[
\text{RUN}\{N\} \text{ dev:filnam.ext}
\]

where:

- \text{dev:} is any standard device name specifying a block-replaceable device. If \text{dev:} is not specified, the device is assumed to be DK. Note that devices MT and CT are not block-replaceable devices and therefore cannot be used in a RUN command.

- \text{filnam.ext} is the file to be executed. If an extension is not specified, the extension .SAV is assumed.

The RUN command is equivalent to a GET command followed by a START command (with no address specified).

\text{NOTE}

If a file containing overlays is to be RUN from a device other than the system device, the handler for that device must be loaded (see Section 2.7.2.6) before the RUN command is issued.

\text{Examples:}

\[
\begin{align*}
\text{.RUN \text{DT1:SRCH.SAV}} & \quad \text{Loads and executes the file SRCH.SAV from DT1.} \\
\text{.RUN \text{PROG}} & \quad \text{Loads PROG.SAV from DK and executes the program.} \\
\text{.GET \text{PROG1}} & \quad \text{Loads PROG1.SAV from device DK without executing it. Then combines PROG1 and PROG2.SAV in memory and begins execution at the starting address for PROG2.} \\
\text{.RUN \text{PROG2}} & \\
\end{align*}
\]
2.7.4.2 **R Command** - This command (valid for use with the background job only) is similar to the RUN command except that the file specified must be on the system device (SY:).

The form of the command is:

\[ R \text{ filnam.ext} \]

No device may be specified. If an extension is not given, the extension .SAV is assumed.

Examples:

\[ R \text{ XYZ.SAV} \] Loads and executes XYZ.SAV from SY.

\[ R \text{ SRC} \] Loads and executes SRC.SAV from SY.

---

2.7.4.3 **START Command** - The START command begins execution of the program currently in memory (i.e., loaded via the GET command) at the specified address. START does not clear or reset memory areas.

The form of the command is:

\[ \text{START} \{\text{address}\} \]

where:

- **address** is an octal number representing any 16-bit address. If the address is omitted, or if 0 is given, the starting address in location 40 will be used.

If the address given does not exist or is not an even address, a trap to location 4 occurs. In this case a monitor error message appears. If no address is given, the program's start address from location 40 is used.
System Communication

Examples:

```
.GET FILE.1
.START 1000

GET FILEA
GET FILEB
.ST
```

Loads FILE.1 into memory and starts execution at location 1000.

Loads FILEA.SAV, then combines FILEA.SAV with FILEB.SAV and starts execution at FILEB's start address.

**REENTER**

2.7.4.4 **REENTER Command** - The REENTER command starts the program at its reentry address (the start address minus two). REENTER does not clear or reset any memory areas and is generally used to avoid reloading the same program for repetitive execution. It can be used to return to a program whose execution was stopped with a CTRL C.

The form of the command is:

```
RE{ENTER}
```

If the reenter bit (bit 13) in the Job Status Word (location 44) is not set, the REENTER command is illegal.

For most system programs, the REENTER command restarts the program at the command level.

If desired, the reentry point in a user program can branch to a routine which initializes the tables and stack, fetches device handlers etc., and then continue normal operation.

Example:

```
.R PIP
.*F
MONTR SYS
[directory prints]
: (C typed)
:
.REENTER
```

CTRL C interrupts the PIP directory listing and transfers control to the monitor level. REENTER returns control to PIP.

2.7.5 **Commands Used Only in a Foreground/Background Environment**

It is important to note that in order to control execution of a foreground job, the commands in this section must be typed to KMON, which is running as the background job. Thus, for example, to SUSPEND
System Communication

the foreground job, the user must be sure he is directing input to KMON as follows:

F> (F8 typed)
B> R PIP
*"C
.SUSPEND

Foreground job is running. Control is redirected to the background job and PIP is called (the foreground is still active). CTRL C stops PIP and starts KMON. The foreground job is suspended. (See Section 2.7.5.2.)

FRUN

2.7.5.1 FRUN Command - The FRUN command is used to initiate foreground jobs. FRUN will only run relocatable files produced with the Linker /R switch (using the Linker supplied with RT-ll, Version 2). Any handlers used by a foreground job must be in memory.

The form of the command is:

FRU{N} dev:file.ext{/N:n}{/S:n}{/P}

where:

dev: represents a block replaceable RT-ll device. If dev: is not specified, DK: is assumed.

file.ext represents the job to be executed. The default extension for a foreground job is .REL.

/N:n or /Nin represents an optional switch used to allocate n words (not bytes) over and above the actual program size. (If running a FORTRAN program, a special formula is used to determine n. Refer to Appendix G for this information.)

/S:n or /Sin represents an optional switch used to allocate n words (not bytes) for stack space. Normally, stack space is set by default to 128 words and is placed in memory below the program. To change the stack size, use /S:n; the stack is still placed in memory under the program. To relocate the stack area, use an .ASECT (see Chapter 5) to define the start of the user stack in location 42. This overrides the /S switch.

/P represents an optional switch (at the end of the FRUN command) for debugging purposes. When the carriage return is typed, FRUN prints the load address of the program, but does not start the
System Communication

program. The foreground job must be explicitly started with the RSUME command (see Section 2.7.5.3). For example:

.FRUN DATA/P
LOADEO AT 125444

If ODT is used with the foreground job, this feature provides the means for determining where the job actually was loaded.

The program is started when the RSUME command is given, allowing the programmer to examine or modify the program before starting it.

If another foreground job is active when the FRUN command is given, an error message is printed. If a terminated foreground job is occupying memory, that region is first reclaimed. Then if the file indicated is found and will fit in memory, the job is installed and started immediately. FRUN destroys the background job's memory image.

Examples:

.FRUN F1 Runs program F1.REL stored on device DK.
.FRUN DT1:F2 Runs F2.REL which is on DT1.

2.7.5.2 SUSPEND Command - The SUSPEND command is used to stop execution of the foreground job.

The form of the command is:

SUS{PEND}

No arguments are required. Foreground I/O transfers in progress will be allowed to complete; however, no new I/O requests will be issued and no completion routines will be entered (see Chapter 9 for a discussion of completion routines). Execution of the job can be resumed only from the keyboard.

Example:

.SUSPEND Suspends execution of the foreground job currently running.
2.7.5.3 RSUME Command - The RSUME command is used to resume execution of the foreground job where it was suspended. Any completion routines which were scheduled while the foreground was suspended are entered at this time.

The form of the command is:

```plaintext
RSUME
```

No arguments are required.

Example:

```plaintext
RSU
```

Resumes execution of the foreground job currently suspended.

2.8 MONITOR ERROR MESSAGES

The following error messages indicate fatal conditions that can occur during system boot:

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?B-I/O ERROR</td>
<td>An I/O error occurred during system boot.</td>
</tr>
<tr>
<td>?B-NO BOOT ON VOLUME</td>
<td>No bootstrap has been written on volume.</td>
</tr>
<tr>
<td>?B-NO MONITR.SYS</td>
<td>No monitor exists on volume being booted.</td>
</tr>
<tr>
<td>?B-NOT ENOUGH CORE</td>
<td>There is not enough memory for the system being booted (e.g., attempting to boot F/B into 8K).</td>
</tr>
</tbody>
</table>

The following error messages are output by the Keyboard Monitor.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ADR?</td>
<td>Address out of range in E or D command.</td>
</tr>
<tr>
<td>?DAT?</td>
<td>The DATE command argument was illegal, or no argument was given and the date has not yet been set.</td>
</tr>
<tr>
<td>?ER RD OVLY?</td>
<td>An I/O error occurred while reading a KMON overlay to process the current command. This is a serious error, indicating that the system file MONITR.SYS is unreadable.</td>
</tr>
<tr>
<td>F?</td>
<td>A CTRL F was typed under the F/B monitor and no foreground job exists.</td>
</tr>
<tr>
<td>?F ACTIVE?</td>
<td>Neither FRUN nor UNLOAD may be used when a foreground job already exists and is active.</td>
</tr>
<tr>
<td>?FIL NOT FND?</td>
<td>File specified in R, RUN, GET, or FRUN command not found.</td>
</tr>
<tr>
<td>?FILE?</td>
<td>No file named where one is expected.</td>
</tr>
</tbody>
</table>

January 1976

2-38
## System Communication

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ILL CMD?</td>
<td>Illegal Keyboard Monitor command or command line too long.</td>
</tr>
<tr>
<td>?ILL DEV?</td>
<td>Illegal or nonexistent device, or an attempt was made to make a device handler resident for use with a foreground job (dev=F) when the Single-Job Monitor was running.</td>
</tr>
<tr>
<td>?NO CLOCK?</td>
<td>No KW11L clock is available for the TIME command.</td>
</tr>
<tr>
<td>?NO FG?</td>
<td>A SUSPEND, RSUME, or UNLOAD FG command was given, but no foreground job was in memory.</td>
</tr>
<tr>
<td>?OVR COR?</td>
<td>Attempt to GET or RUN a file that is too big.</td>
</tr>
<tr>
<td>?PARAMS?</td>
<td>Bad parameters were typed to the SAVE command.</td>
</tr>
<tr>
<td>?REL FIL I/O ER?</td>
<td>Either the program requested is not a REL file or a hardware error was encountered trying to read or write the file.</td>
</tr>
<tr>
<td>?SV FIL I/O ER?</td>
<td>I/O error on .SAV file in SAVE (output) or R, RUN, or GET (input) command. Possible errors include end-of-file, hard error, and channel not open.</td>
</tr>
<tr>
<td>?SY I/O ER?</td>
<td>I/O error on system device (usually reading or writing swap area).</td>
</tr>
<tr>
<td>?TIM?</td>
<td>The TIME command argument was illegal.</td>
</tr>
</tbody>
</table>

The following messages are output by the RT-11 Resident Monitor when an unrecoverable error has occurred. Control passes to the Keyboard Monitor. The program in which the error occurred cannot be restarted with the RE command. To execute the program again, use the R or RUN command.

The format for fatal monitor error messages is:

\[ ?M-text PC \]

where PC is the address+2 of the location where the error occurred.

Note that ?M errors can be inhibited in certain cases by the use of the .SERR macro; see Chapter 9 for details.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?M-BAD FETCH</td>
<td>Either an error occurred while reading in a device handler from SY, or the address at which the handler was to be loaded was illegal.</td>
</tr>
</tbody>
</table>

2-39

January 1976
An error occurred doing I/O in the directory of a device (e.g., .ENTER on a write-locked device).

No more directory segments were available for expansion (occurs during file creation (.ENTER)).

In F/B only, this message may appear in addition to any of the other diagnostics listed in this section. It indicates that the error occurred while the USR was updating a device directory. One or more files on that device may be lost.

A floating-point exception trap occurred, and the user program had no .SFPA exception routine active (see Chapter 9).

Under the F/B Monitor, an address specified in a monitor call was odd or was not within the job's limits.

A channel number was specified which was too large.

An EMT was executed which did not exist; i.e., the function code was out of bounds.

The USR was called from a completion routine. This error does not have a soft return (i.e., .SERR will not inhibit this message; see Chapter 9).

A READ/WRITE operation was tried but no device handler was in memory for it.

A user program with overlays failed to successfully read an overlay.

A hard I/O error occurred while the system was attempting to write a user program to the system swap blocks. This is usually caused by a write-locked system device. Under the Single-Job Monitor, this may cause the system to halt.

An I/O error occurred while trying to read KMON/USR into memory, indicating that the monitor file is situated on the system device in an area that has developed one or more bad blocks. The monitor prints the message and loops trying to read KMON. The message is a warning that the system device is bad.
System Communication

If, after several seconds, it is apparent that attempts to read KMON are failing, halt the processor. It may be impossible to boot the volume because of the bad area in the monitor file. Use another system device to verify the bad blocks and follow the recovery procedures described in section 4.2.12.1 of Chapter 4.

?M-TRAP TO 4
?M-TRAP TO 10

The job has referenced illegal memory or device registers, an illegal instruction was used, stack overflow occurred, a word instruction was executed with an odd address, or a hardware problem caused bus time-out traps through location 4.

If CSI errors occur and input was from the console terminal, an error message is printed on the terminal.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?DEV FUL?</td>
<td>Output file will not fit.</td>
</tr>
<tr>
<td>?FIL NOT FND?</td>
<td>Input file was not found.</td>
</tr>
<tr>
<td>?ILL CMD?</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>?ILL DEV?</td>
<td>Device specified does not exist.</td>
</tr>
</tbody>
</table>

2.8.1 Monitor HALTS

There are two HALT instructions in the RT-11 V02 monitors, one each in F/B and Single-Job. The Single-Job Monitor will halt only if I/O errors occur during swap operations to the system device. If the S/J Monitor halts, look for a write-locked system device.

The F/B Monitor will halt if a trap to location 4 occurs or if I/O occurs while the system is performing critical operations from which it cannot recover. If the F/B Monitor halts, look for use of non-existent devices, traps from interrupt service routines, or user-corrupted queue elements.

The monitor halts can be detected by their address, which is high in memory, above the resident base address (location 54).

When a monitor halt occurs, do not attempt to restart the system by pressing CONTINUE on the processor; the system must be rebooted.
CHAPTER 3

TEXT EDITOR

The Text Editor (EDIT) is used to create and modify ASCII source files so that these files can be used as input to other system programs such as the assembler or BASIC. Controlled by user commands from the keyboard, EDIT reads ASCII files from a storage device, makes specified changes and writes ASCII files to a storage device or lists them on the line printer or console terminal. EDIT allows efficient use of VT-11 display hardware, if this is part of the system configuration.

The Editor considers a file to be divided into logical units called pages. A page of text is generally 50-60 lines long (delimited by form feed characters) and corresponds approximately to a physical page of a program listing. The Editor reads one page of text at a time from the input file into its internal buffers where the page becomes available for editing. Editing commands are then used to:

- Locate text to be changed,
- Execute and verify the changes,
- Output a page of text to the output file,
- List an edited page on the line printer or console terminal.

3.1 CALLING AND USING EDIT

To call EDIT from the system device type:

```
R EDIT
```

and the RETURN key in response to the dot (.) printed by the monitor. EDIT responds with an asterisk (*) indicating it is in command mode and awaiting a user command string.

Type CTRL C to halt the Editor at any time and return control to the monitor. To restart the Editor type .R EDIT or the .REENTER command in response to the monitor's dot. The contents of the buffers are lost when the Editor is restarted.
3.2 MODES OF OPERATION

Under normal usage, the Editor operates in one of two different modes: Command Mode or Text Mode. In Command Mode all input typed on the keyboard is interpreted as commands instructing the Editor to perform some operation. In Text Mode all typed input is interpreted as text to replace, be inserted into, or be appended to the contents of the Text Buffer.

Immediately after being loaded into memory and started, the Editor is in Command Mode. An asterisk is printed at the left margin of the console terminal page indicating that the Editor is waiting for the user to type a command. All commands are terminated by pressing the ALTMODE key twice in succession. Execution of commands proceeds from left to right. Should an error be encountered during execution of a command string, the Editor prints an error message followed by an asterisk at the beginning of a new line indicating that it is still in Command Mode and awaiting a legal command. The command in error (and any succeeding commands) is not executed and must be corrected and retyped.

Text mode is entered whenever the user types a command which must be followed by a text string. These commands insert, replace, exchange, or otherwise manipulate text; after such a command has been typed, all succeeding characters are considered part of the text string until an ALTMODE is typed. The ALTMODE terminates the text string and causes the Editor to reenter Command Mode, at which point all characters are considered commands again.

A special editing mode, called Immediate Mode, can be used whenever the VT-11 display hardware is running. This mode is described in Section 3.7.2.

3.3 SPECIAL KEY COMMANDS

The EDIT key commands are listed in Table 3-1. Control commands are typed by holding down the CTRL key while typing the appropriate character.

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTMODE</td>
<td>Echoes $. A single ALTMODE terminates a text string. A double ALTMODE executes the command string. For example,</td>
</tr>
<tr>
<td></td>
<td>*GMOV R,B$-1D##</td>
</tr>
<tr>
<td>CTRL C</td>
<td>Echoes at the terminal as $C and a carriage return. Terminates execution of EDIT commands, and returns to monitor Command Mode. A double CTRL C is necessary when I/O is in progress. The REENTER command may be used to restart the Editor, but the contents of the text buffers are lost.</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL O</td>
<td>Echoes ‹O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL O resumes output.</td>
</tr>
<tr>
<td>CTRL U</td>
<td>Echoes ‹U and a carriage return. Deletes all the characters on the current terminal input line. (Equivalent to typing RUBOUT back to the beginning of the line.)</td>
</tr>
<tr>
<td>RUBOUT</td>
<td>Deletes character from the current line; echoes a backslash followed by the character deleted. Each succeeding RUBOUT typed by the user deletes and echoes another character. An enclosing backslash is printed when a key other than RUBOUT is typed. This erasure is done right to left up to the last carriage return/line feed combination. RUBOUT may be used in both Command and Text Modes.</td>
</tr>
<tr>
<td>TAB</td>
<td>Spaces to the next tab stop. Tab stops are positioned every eight spaces on the terminal; typing the TAB key causes the carriage to advance to the next tab position.</td>
</tr>
<tr>
<td>CTRL X</td>
<td>Echoes ‹X and a carriage return. CTRL X causes the Editor to ignore the entire command string currently being entered. The Editor prints a &lt;CR&gt;&lt;LF&gt; and an asterisk to indicate that the user may enter another command. For example:</td>
</tr>
<tr>
<td></td>
<td>*ABCDEF</td>
</tr>
<tr>
<td></td>
<td>EFGH*X</td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

A CTRL U would only cause deletion of EFGH; CTRL X erases the entire command.

### 3.4 COMMAND STRUCTURE

EDIT commands fall into six general categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Commands</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input/Output</td>
<td>Edit Backup</td>
<td>3.6.1.3</td>
</tr>
<tr>
<td></td>
<td>Edit Read</td>
<td>3.6.1.1</td>
</tr>
<tr>
<td></td>
<td>Edit Write</td>
<td>3.6.1.2</td>
</tr>
<tr>
<td></td>
<td>End File</td>
<td>3.6.1.9</td>
</tr>
<tr>
<td></td>
<td>Exit</td>
<td>3.6.1.10</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>3.6.1.7</td>
</tr>
<tr>
<td></td>
<td>Next</td>
<td>3.6.1.6</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>3.6.1.4</td>
</tr>
<tr>
<td></td>
<td>Verify</td>
<td>3.6.1.8</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>3.6.1.5</td>
</tr>
<tr>
<td>Pointer location</td>
<td>Advance</td>
<td>3.6.2.3</td>
</tr>
<tr>
<td></td>
<td>Beginning</td>
<td>3.6.2.1</td>
</tr>
<tr>
<td></td>
<td>Jump</td>
<td>3.6.2.2</td>
</tr>
</tbody>
</table>
The general format for the first five categories of EDIT commands is:

```
nCtext$ 
```

or

```
nC$ 
```

where n represents one of the legal arguments listed in Table 3-2, C is a one- or two-letter command, and text is a string of successive ASCII characters.

As a rule, commands are separated from one another by a single ALTMODE; however, if the command requires no text, the separating ALTMODE is not necessary. Commands are terminated by a single ALTMODE; typing a second ALTMODE begins execution. (ALTMODE is used differently when Immediate Mode is in effect; Section 3.7.2 details its use in this case.)

The format of Display Editor commands is somewhat different from the normal editing command format, and is described in Section 3.7.

### 3.4.1 Arguments

An argument is positioned before a command letter and is used either to specify the particular portion of text to be affected by the command or to indicate the number of times the command should be performed. With some commands, this specification is implicit and no arguments are needed; other editing commands require an argument. Table 3-2 lists the formats of arguments which are used by commands of this second type.
## Table 3-2
### Command Arguments

<table>
<thead>
<tr>
<th>Format</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>n stands for any integer in the range -16383 to +16383 and may, except where noted, be preceded by a + or -. If no sign precedes n, it is assumed to be a positive number. Whenever an argument is acceptable in a command, its absence implies an argument of 1 (or -1 if only the - is present).</td>
</tr>
<tr>
<td>0</td>
<td>0 refers to the beginning of the current line.</td>
</tr>
<tr>
<td>/</td>
<td>/ refers to the end of text in the current Text Buffer.</td>
</tr>
<tr>
<td>=</td>
<td>= is used with the J, D and C commands only and represents -n, where n is equal to the length of the last text argument used.</td>
</tr>
</tbody>
</table>

The roles of all arguments are explained more specifically in following sections.

### 3.4.2 Command Strings

All EDIT command strings are terminated by two successive ALTMODE characters. Spaces, carriage returns and line feeds within a command string may be used freely to increase command readability but are ignored unless they appear in a text string. Commands used to insert text can contain text strings that are several lines long. Each line is terminated with a <CR><LF> and the entire command is terminated with a double ALTMODE.

Several commands can be strung together and executed in sequence. For example,

```
*BGMOV PC,R0$-2CR1$5KGCLR @R2##
```

Execution of a command string begins when the double ALTMODE is typed and proceeds from left to right. Except when they are part of a text string, spaces, carriage return, line feed, and single ALTMODES are ignored. For example:

```
*BGMOV R0$=CCLR R1#RV##
```
Text Editor

may be typed as:

```
*E$ GMOV R0$
=CLR R1$
A$ V$
```

with equivalent execution.

3.4.3 The Current Location Pointer

Most EDIT commands function with respect to a movable reference pointer which is normally located between the most recent character operated upon and the next character in the buffer. At any given time during the editing procedure, this pointer can be thought of as representing the current position of the Editor in the text. Most commands use this pointer as an implied argument. Commands are available for moving the pointer anywhere in the text, thereby redefining the current location and allowing greater facility in the use of other commands.

3.4.4 Character- and Line-Oriented Command Properties

Edit commands are line-oriented or character-oriented depending on the arguments they accept. Line-oriented commands operate on entire lines of text. Character-oriented commands operate on individual characters independent of what or where they are.

When using character-oriented commands, a numeric argument specifies the number of characters that are involved in the operation. Positive arguments represent the number of characters in a forward direction (in relation to the pointer), negative arguments the number of characters in a backward direction. Carriage return and line feed characters are treated the same as any other character. For example, assume the pointer is positioned as indicated in the following text (↑ represents the current position of the pointer):

```
MOV $VECT,R2<CR><LF>,
CLR @R2<CR><LF>
```

The EDIT command -2J backs the pointer by two characters.

```
MOV $VECT,R2<CR><LF>
CLR @R2<CR><LF>
```

The command 10J advances the pointer forward by ten characters and places it between the CR and LF characters at the end of the second line.

```
MOV $VECT,R2<CR><LF>
CLR @R2<CR><LF>
```

Finally, to place the pointer after the "C" in the first line, a -14J command is used. The J (Jump) command is explained in Section 3.6.2.2.

```
MOV $VECT,R2<CR><LF>
CLR @R2<CR><LF>
```
When using line-oriented commands, a numeric argument represents the number of lines involved in the operation. The Editor recognizes a line of text as a unit when it detects a <CR><LF> combination in the text. When the user types a carriage return, the Editor automatically inserts a line feed. Positive arguments represent the number of lines forward (in relation to the pointer); this is accomplished by counting carriage return/line feed combinations beginning at the pointer. So, if the pointer is at the beginning of a line, a line-oriented command argument of +1 represents the entire line between the current pointer and the terminating line feed. If the current pointer is in the middle of the line, an argument of +1 represents only the portion of the line between the pointer and the terminating line feed.

For example, assume a buffer of:

MOV PC,R1<CR><LF>
ADD #$DRIV-,R1<CR><LF>
MOV #VECT,R2<CR><LF>
CLR @R2<CR><LF>

The command to advance the pointer one line (1A) causes the following change:

MOV PC,R1<CR><LF>
ADD #$DRIV-,R1<CR><LF>
MOV #VECT,R2<CR><LF>
CLR @R2<CR><LF>

The command 2A moves the pointer over 2 <CR><LF> combinations:

MOV PC,R1<CR><LF>
ADD #$DRIV-,R1<CR><LF>
MOV #VECT,R2<CR><LF>
CLR @R2<CR><LF>

Negative line arguments reference lines in a backward direction (in relation to the pointer). Consequently, if the pointer is at the beginning of the line, a line argument of -1 means "the previous line" (moving backward past the first <CR><LF> and up to but not including the second <CR><LF>); if the printer is in the middle of a line, an argument of -1 means the preceding 1 1/2 lines. Assume the buffer contains:

MOV PC,R1<CR><LF>
ADD #$DRIV-,R1<CR><LF>
MOV #VECT,R2<CR><LF>
CLR @R2<CR><LF>

A command of -1A backs the pointer by 1 1/2 lines.

MOV PC,R1<CR><LF>
ADD #$DRIV-,R1<CR><LF>
MOV #VECT,R2<CR><LF>
CLR @R2<CR><LF>

3-7
Now a command of `1A` backs it by only 1 line.

\[ \text{MOV} \quad \text{PC,RL}<\text{CR}><\text{LF}> \\
\text{ADD} \quad \text{#DRIV=},\text{RL}<\text{CR}><\text{LF}> \\
\text{MOV} \quad \text{#VECT,R2<CR><LF>} \\
\text{CLR} \quad \text{R2<CR><LF>}
\]

3.4.5 Command Repetition

Portions of a command string may be executed more than once by enclosing the desired portion in angle brackets (`<>`) and preceding the left angle bracket with the number of iterations desired. The structure is:

\[ \text{C1$C2$n(C3$C4$)>C5$$} \]

where C1, C2, ..., C5 represent commands and n represents an iteration argument. Commands C1 and C2 are each executed once, then commands C3 and C4 are executed n times. Finally command C5 is executed once and the command line is finished. The iteration argument (n) must be a positive number (1 to 16,383), and if not specified is assumed to be 1. If the number is negative or too large, an error message is printed. Iteration brackets may be nested up to 20 levels. Command lines are checked to make certain the brackets are correctly used and match prior to execution.

Essentially, enclosing a portion of a command string in iteration brackets and preceding it with an iteration argument (n) is equivalent to typing that portion of the string n times. For example:

\[ *\text{BGAA}$3<\text{DIB}$-J>Y$$ \]

is equivalent to typing:

\[ *\text{BGAA}$<\text{DIB}$-J-DIB$-J-DIB$-JY$$ \]

and:

\[ *\text{B3}$2<\text{AD}>Y$$ \]

is equivalent to typing:

\[ *\text{BADADVADADVADV$$} \]

The following bracket structures are examples of legal usage:

\[ <<<\text{<<<}
\]

The following bracket structures are examples of illegal combinations which will cause an error message since the brackets are not properly matched:

\[ >>>>
\]

During command repetition, execution proceeds from left to right until a right bracket is encountered. EDIT then returns to the last left
bracket encountered, decrements the iteration counter and executes the commands within the brackets. When the counter is decremented to 0, EDIT looks for the next iteration count to the left and repeats the same procedures. The overall effect is that EDIT works its way to the innermost brackets and then works its way back again. The most common use for iteration brackets is found in commands such as Unsave, that do not accept repeat counts. For example:

```
*I<U>$$
```

Assume a file called SAMP (stored on device DK) is to be read and the first four occurrences of the instruction MOV #200,R0 on each of the first five pages are to be changed to MOV #244,R4. The following command line is entered:

```
*EBSAMP$5<N4<8GMOV #200,R0#J#3<0G0#C4>>>EX$$
```

The command line contains three sets of iteration loops (A,B,C) and is executed as follows:

Execution initially proceeds from left to right; the file SAMP is opened for input, and the first page is read into memory. The pointer is moved to the beginning of the buffer and a search is initiated for the character string MOV #200,R0. When the string is found, the pointer is positioned at the end of the string, but the =$J command moves the pointer back so that it is positioned immediately preceding the string. At this point, execution has passed through each of the first two sets of iteration loops (A,B) once. The innermost loop (C) is next executed three times, changing the 0s to 4s. Control now moves back to pick up the second iteration of loop B, and again moves from left to right. When loop C has executed three times, control again moves back to loop B. When loop B has executed a total of 4 times, control moves back to the second iteration of loop A, and so forth until all iterations have been satisfied.

### 3.5 MEMORY USAGE

The memory area used by the Editor is divided into four logical buffers as follows:

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro Buffer</td>
<td>High Memory</td>
</tr>
<tr>
<td>Save Buffer</td>
<td></td>
</tr>
<tr>
<td>Free Memory</td>
<td></td>
</tr>
<tr>
<td>Command Input Buffer</td>
<td></td>
</tr>
<tr>
<td>Text Buffer</td>
<td>Low Memory</td>
</tr>
</tbody>
</table>

3-9
Text Editor

The Text Buffer contains the current page of text being edited, and the Command Input Buffer holds the command currently being typed at the terminal. If a command currently being entered by the user is within 10 characters of exceeding the space available in the Command Buffer, the message:

* CB ALMOST FULL *

is printed. If the command can be completed within 10 characters, the user may finish entering the command; otherwise he should type the ALTMODE key twice to execute that portion of the command line already completed. The message is printed each time a character is entered in one of the last 10 spaces.

If the user attempts to enter more than 10 characters the message:

?CB FULL?

is printed and all commands typed within the last 10 characters are ignored. The user again has 10 characters of available space in which to correct the condition.

The Save Buffer contains text stored with the Save (S) command, and the Macro Buffer contains the command string macro entered with the Macro (M) command. (Both commands are explained in Section 3.6.5.)

The Macro and Save Buffers are not allocated space until an M or S command is executed. Once an M or S command is executed, a OM or OU (Unsave) command must be executed to return that space to the free area.

The size of each buffer automatically expands and contracts to accommodate the text being entered; if there is not enough space available to accommodate required expansion of any of the buffers, a "?NO ROOM?" error message is typed.

3.6 EDITING COMMANDS

3.6.1 Input/Output Commands

Input commands are used to create files and read them into the Text Buffer where they become available for editing or listing. Output commands cause text to be listed on the console terminal or line-printer or written out to a storage device. Some commands are specifically designed for either input or output functions, while a few commands serve both purposes.

Once editing is completed and the page currently in the Text Buffer is written to the output file, that page of text is unavailable for further editing until the file is closed and reopened.

3.6.1.1 Edit Read - The Edit Read command opens an existing file for input and prepares it for editing. Only one file can be open for input at a time.
Text Editor

The form of the command is:

\texttt{ERdev:filnam.ext}$

The string argument (\texttt{dev:filnam.ext}) is limited to 19 characters and specifies the file to be opened. If no device is specified, D\texttt{K:} is assumed. If a file is currently open for input, that file is closed; any edits made to the file are preserved.

\texttt{Edit Read} does not input a page of text nor does it affect the contents of the other user buffers (see Section 3.5.)

\texttt{Edit Read} can be used on a file which is already open to close that file for input and reposition \texttt{EDIT} at its beginning. The first \texttt{Read} command following any \texttt{Edit Read} command inputs the first page of the file.

\textbf{Examples:}

\begin{itemize}
\item \texttt{*ERDT1: SAMP. MAC$}$ opens SAMP.MAC on device D\texttt{T1:} for input.
\item \texttt{*ERSOURCE$}$ opens SOURCE on device D\texttt{K:} for input.
\end{itemize}

3.6.1.2 \texttt{Edit Write} - The \texttt{Edit Write} command sets up a file for output of newly created or edited text. However, no text is output and the contents of the user buffers are not affected. Only one file can be open for output at a time. Any current output files are closed.

The form of the command is:

\texttt{EWdev:filnam.ext[n]$}

The string argument (\texttt{dev:filnam.ext[n]}) is limited to 19 characters and is the name to be assigned to the output file being opened. If \texttt{dev:} is not specified, D\texttt{K:} is assumed. \texttt{[n]} is optional and represents the length of the file to be opened. If not specified, one half the largest available space is used; if this is not adequate for the output file size, the \texttt{EF} and \texttt{EX} commands will not close the output file, and all edits will be lost. It is thus recommended that the \texttt{[n]} construction be used whenever there is doubt as to whether enough space is available on the device for the output file.

If a file with the same name already exists on the device, the old file is deleted when an \texttt{EXit}, \texttt{End File} or another \texttt{Edit Write} command is executed.

\textbf{Examples:}

\begin{itemize}
\item \texttt{*EWDK: TEST. MAC$}$ opens the file TEST.MAC on device D\texttt{K:} for output.
\item \texttt{*EWFILE. BAS[11]$}$ opens the file FILE.BAS (allocating 11 blocks) on the device D\texttt{K:} for output.
\end{itemize}

3.6.1.3 \texttt{Edit Backup} - The \texttt{Edit Backup} command is used to open an existing file for editing and at the same time create a backup version of the file. Any currently open file will be closed. No text is read or written with this command.
The form of the command is:

```
EBdev:filnam.ext[n]$ 
```

The device designation, filename and extension are limited to 19 characters. If dev: is not specified, DK: is assumed. [n] is optional and represents the length of the file to be opened; if not specified, one-half the largest available space is used.

The file indicated in the command line must already exist on the device designated since text will be read from this file as input. At the same time, an output file is opened under the same filename and extension. After an EB command has been successfully executed, the original file (used as input) is renamed with the current filename and a .BAK extension; any previous file with this filename and a .BAK extension is deleted. The new output file is closed and assigned the name as specified in the EB command. This renaming of files takes place whenever an Exit, End File, Edit Read, Edit Write or Edit Backup command is executed.

Examples:

```
*EBSY:BAS1.MAC**  Opens BAS1.MAC on SY. When editing is complete, the old BAS1.MAC becomes BAS1.BAK and the new file becomes BAS1.MAC. Any previous version of BAS1.BAK is deleted.

*EBBAS2:BAS[15]**  Opens BAS2.BAS on DK (allocating 15 blocks). When editing is complete, the old BAS2.BAS is labeled BAS2.BAK and the new file becomes BAS2.BAS. Any previous version of BAS2.BAK is deleted.
```

In EB, ER and EW commands, leading spaces between the command and the filename are illegal (the filename is considered to be a text string). All dev:file.ext specifications for EB, ER and EW commands conform to the RT-11 conventions for file naming and are identical to filenames entered in command strings used with other system programs.

3.6.1.4 Read - The Read command (R) causes a page of text to be read from the input file (previously specified in an ER or EB command) and appended to the current contents, if any, of the Text Buffer.

The form of the command is:

```
R 
```

No arguments are used with the R command and the pointer is not moved. Text is input until one of the following conditions is met:

1. A form feed character, signifying the end of the page, is encountered. At this point, the form feed will be the last character in the buffer; or
2. The Text Buffer is within 500 characters of being full. (When this condition occurs, Read inputs up to the next <CR><LF> combination, then returns to Command Mode. An asterisk is printed as though the Read were complete, but text will not have been fully input); or

3. An end-of-file condition is detected, (the *EOF* message is printed when all text in the file has been read into memory and no more input is available).

The maximum number of characters which can be brought into memory with an R command is approximately 6,000 for an 8K system. Each additional 4K of memory allows approximately 8,000 additional characters to be input. An error message is printed if the Read exceeds the memory available or if no input is available.

3.6.1.5 Write - The Write command (W) moves lines of text from the Text Buffer to the output file (as specified in the EW or EB command). The format of the command is:

  nW  Write all characters beginning at the pointer and ending at the nth <CR><LF> to the output file.
  -nW Write all characters beginning on the -nth line and terminating at the pointer to the output file.
  0W  Write the text from the beginning of the current line to the pointer.
  /W  Write the text from the pointer to the end of the buffer.

The pointer is not moved and the contents of the buffer are not affected. If the buffer is empty when the Write is executed, no characters are output.

Examples:

  +5W###  Writes the next 5 lines of text starting at the pointer, to the current output file.

  *-2W###  Writes the previous 2 lines of text, ending at the pointer, to the current output file.

  +B/W###  Writes the entire Text Buffer to the current output file.
Text Editor

3.6.1.6 Next - The Next command acts as both an input and output command since it performs both functions. First it writes the current Text Buffer to the output file, then clears the buffer, and finally reads in the next page of the input file. The Next command can be repeated n times by specifying an argument before the command. The command format is:

nN

Next accepts only positive arguments (n) and leaves the pointer at the beginning of the buffer. If fewer than n pages are available in the input file, all available pages are input to the buffer, output to the current file, and deleted from the buffer; the pointer is left positioned at the beginning of an empty buffer, and an error message is printed. This command is equivalent to a combination of the Beginning, Write, Delete and Read commands (B/W/DR). Next can be used to space forward, in page increments, through the input file.

Example:

*2N$S$ Writes the contents of the current Text Buffer to the output file. Read and write the next page of text. Clear the buffer and then read in another page.

3.6.1.7 List - The List command prints the specified number of lines on the console terminal. The format of the command is:

nL Print all characters beginning at the pointer and ending with the nth <CR><LF>.

-nL Print all characters beginning with the first character on the -nth line and terminating at the pointer.

0L Print from the beginning of the current line up to the pointer.

/L Print from the pointer to the end of the buffer.

The pointer is not moved after the command is executed.

Examples:

*-2L$S$ Prints all characters starting at the second preceding line and ending at the pointer.

*4L$S$ Prints all characters beginning at the pointer and terminating at the 4th <CR><LF>.

Assuming the pointer location is:

MOV B 5(R1),@R2
ADD $ R1,(R2)+

3-14
Text Editor

The command:

\[*-1L\]* $$

Prints the previous 1 l/2 lines up to the pointer:

\[ \text{MOV} \ \text{B} \ 5(R1), @R2 \]
\[ \text{ADD} \]

3.6.1.8 Verify - The Verify command prints the current text line (the line containing the pointer) on the terminal. The position of the pointer within the line has no effect and the pointer does not move. The command format is:

\[ \text{V} \]

No arguments are used. The V command is equivalent to a OLL (List) command.

Example:

\[*V* $$
\[ \text{ADD} \]
\[ \text{R1}, (R2)+ \]

The command causes the current line of text to be printed.

3.6.1.9 End File - The End File command closes the current output file. This command does no input/output operations and does not move the pointer. The buffer contents are not affected. The output file is closed, containing only the text previously output.

The form of the command is:

\[ \text{EF} \]

No arguments are used. Note that an implied EF command is included in EW and EB commands.

3.6.1.10 EXit - The EXIT command is used to terminate editing, copy the text buffer and the remainder of the input file to the output file, close input and output files, and return control to the monitor. It performs consecutive Next commands until the end of the input file is reached, then closes both the input and output files.

The command format is:

\[ \text{EX} \]

No arguments are used. Essentially, Exit is used to copy the remainder of the input file into the output file and return to the monitor. Exit is legal only when there is an output file open. If an output file is not open and it is desired to terminate the editing session, return to the monitor with CTRL C.
NOTE

An EF or EX command is necessary in order to make an output file permanent. If CTRL C is used to return to the monitor without a prior execution of an EF command, the current output file is not saved. (It can however, be made permanent using the monitor CLOSE command; see Section 2.7.2.5.)

An example of the contrasting uses of the EF and EX commands follows. Assume an input file, SAMPLE, contains several pages of text. The user wishes to make the first and second pages of the file into separate files called SAM1 and SAM2, respectively; the remaining pages of text will then make up the file SAMPLE. This can be done using these commands:

*EWSAM1**
*ERSAMPLE**
*RNEF**
*EWSAM2**
*NEF**
*EWSAMPLE#EX**

The user might note that the EF commands are not necessary in this example since the EW command closes a currently open output file before opening another.

3.6.2 Pointer Relocation Commands

Pointer relocation commands allow the current location pointer to be moved within the Text Buffer.

3.6.2.1 Beginning - The Beginning command moves the current location pointer to the beginning of the Text Buffer.

The command format is:

    B

There are no arguments.

For example, assume the buffer contains:

    MOV B 5(R1),@R2
    ADD R1,(R2)+
    CLR @R2
    MOV B 6(R1),@R2
Text Editor

The B command:

*B$$

moves the pointer to the beginning of the Text Buffer:

MOV B 5(R1),@R2
ADD R1,(R2)+
CLR @R2
MOV B 6(R1),@R2

3.6.2.2 Jump - The Jump command moves the pointer over the specified number of characters in the Text Buffer.

The form of the command is:

(+ or -) nJ Move the pointer (backward or forward) n characters.
0J Move the pointer to the beginning of the current line (equivalent to 0A).
/J Move the pointer to the end of the Text Buffer (equivalent to /A).
=J Move the pointer backward n characters, where n equals the length of the last text argument used.

Negative arguments move the pointer toward the beginning of the buffer, positive arguments toward the end. Jump treats carriage return, line feed, and form feed characters the same as any other character, counting one buffer position for each.

Examples:

*3J$$ Moves the pointer ahead three characters.
*-4J$$ Moves the pointer back four characters.
*B$jABC$=J$$ Move the pointer so that it immediately precedes the first occurrence of 'ABC' in the buffer.

3.6.2.3 Advance - The Advance command is similar to the Jump command except that it moves the pointer a specified number of lines (rather than single characters) and leaves it positioned at the beginning of the line.

The form of the command is:

nA Advance the pointer forward n lines and position it at the beginning of the nth line.
Text Editor

-nA  Move the pointer backward past \text{n} \textbackslash<CR><LF> combinations and position it at the beginning of the \text{-nth} line.

0A  Advance the pointer to the beginning of the current line (equivalent to 0J).

/AIDS  Advance the pointer to the end of the Text Buffer (equivalent to /J).

Examples:

*3A##

Moves the pointer ahead three lines.

Assuming the buffer contains:

CLR  @R2

The command:

*8A##

Moves the pointer to:

,CLR  @R2

3.6.3 Search Commands

Search commands are used to locate specific characters or strings of characters within the Text Buffer.

3.6.3.1 Get - The Get command starts at the pointer and searches the current Text Buffer for the \text{n}th occurrence of a specified text string. If the search is successful, the pointer is left immediately following the \text{n}th occurrence of the text string. If the search fails, an error message is printed and the pointer is left at the end of the Text Buffer. The format of the command is:

\text{nGtext}$

The argument (n) must be positive and is assumed to be 1 if not otherwise specified. The text string may be any length and immediately follows the G command. The search is made on the portion of the text between the pointer and the end of the buffer.

Example:

Assuming the buffer contains:

\text{MOV}  \text{PC,R1}
\text{ADD}  \#$DR1V-,R1
\text{MOV}  \#VECT,R2
\text{CLR}  @R2
\text{MOVB}  5(R1),@R2
\text{ADD}  R1,(R2)+
\text{CLR}  @R2
\text{MOVB}  6(R1),@R2
The command:

```
*GRADD$
```

positions the pointer at:

```
ADD  $DRIV-,R1
```

The command:

```
*30@R2$
```

positions the pointer at:

```
ADD  R1,(R2)+
CLR  @R2+
```

After search commands, the pointer is left immediately following the text object. Using a search command in combination with =J will place the pointer before the text object, as follows:

```
*GETEST$=J$
```

This command combination places the pointer before 'TEST'.

3.6.3.2 Find - The Find command starts at the current pointer and searches the entire input file for the nth occurrence of the text string. If the nth occurrence of the text string is not found in the current buffer, a Next command is automatically performed and the search is continued on the new text in the buffer. When the search is successful, the pointer is left immediately following the nth occurrence of the text string. If the search fails (i.e., the end-of-file is detected for the input file and the nth occurrence of the text string has not been found), an error message is printed and the pointer is left at the beginning of an empty Text Buffer.

The form of the command is:

```
nFtext$
```

The argument (n) must be positive and is assumed to be 1 if not otherwise specified.

By deliberately specifying a nonexistent search string, the user can close out his file; that is, he can copy all remaining text from the input file to the output file.

Find is a combination of the Get and Next commands.

Example:

```
*2FM0VB 6(R1),@R2$
```

Searches the entire input file for the second occurrence of the text string MOVB 6(R1),@R2. Each unsuccessfully searched buffer is written to the output file.
Text Editor

3.6.3.3 Position - The Position command searches the input file for the nth occurrence of the text string. If the desired text string is not found in the current buffer, the buffer is cleared and a new page is read from the input file. The format of the command is:

nPtext$

The argument (n) must be positive, and is assumed to be 1 if not otherwise specified. When a P command is executed the current contents of the buffer are searched from the location of the pointer to the end of the buffer. If the search is unsuccessful, the buffer is cleared and a new page of text is read and the cycle is continued.

If the search is successful, the pointer is positioned after the nth occurrence of the text. If it is not, the pointer is left at the beginning of an empty Text Buffer.

The Position command is a combination of the Get, Delete and Read commands; it is most useful as a means of placing the location pointer in the input file. For example, if the aim of the editing session is to create a new file from the second half of the input file, a Position search will save time.

The difference between the Find and Position commands is that Find writes the contents of the searched buffer to the output file while Position deletes the contents of the buffer after it is searched.

Example:

*PADD R1,<R2>+##

Searches the entire input file for the specified string ignoring the unsuccessfully searched buffers.

3.6.4 Text Modification Commands

The following commands are used to insert, relocate, and delete text in the Text Buffer.

3.6.4.1 Insert - The Insert command causes the Editor to enter Text Mode and allows text to be inserted immediately following the pointer. Text is inserted until an ALTMODE is typed and the pointer is positioned immediately after the last character of the insert. The command format is:

Itext$

No arguments are used with the Insert command, and the text string is limited only by the size of the Text Buffer and the space available. All characters except ALTMODE are legal in the text string. ALTMODE terminates the text string.

NOTE

Forgetting to type the I command will cause the text entered to be executed as commands.
EDIT automatically protects against overflowing the Text Buffer during an Insert. If the I command is the first command in a multiple command line, EDIT ensures that there will be enough space for the Insert to be executed at least once. If repetition of the command exceeds the available memory, an error message is printed.

Example:

```
*IMOV   #BUFF,R2
MOV    #LINE,R1
MOVEB  -1(R2),R0##
*
```

Inserts the specified text at the current location of the pointer and leaves the pointer positioned after R0.

3.6.4.2 Delete - The Delete command removes a specified number of characters from the Text Buffer. Characters are deleted starting at the pointer; upon completion, the pointer is positioned at the first character following the deleted text.

The form of the command is:

```
( + or - ) nD
```

Delete n characters (forward or backward from the pointer).

```
0D
```

Delete from beginning of current line to the pointer (equivalent to 0K).

```
/D
```

Delete from pointer to end of Text Buffer (equivalent to /K).

```
=D
```

Delete -n characters, where n equals the length of the last text argument used.

Examples:

```
*\-2D##
```

Deletes the two characters immediately preceding the pointer.

```
*E#FMOV R1=#D#
```

Deletes the text string 'MOV R1'. (=D used in combination with a search command will delete the indicated text string).

Assuming a buffer of:

```
ADD    R1,(R2)+
CLR    @R2
```

the command:

```
*0D##
```

leaves the buffer with:

```
ADD    R1,(R2)+
@R2
```

3-21
3.6.4.3 Kill - The Kill command removes n lines from the Text Buffer. Lines are deleted starting at the location pointer; upon completion of the command, the pointer is positioned at the beginning of the line following the deleted text. The command format is:

\[ nK \]  
Delete lines beginning at the pointer and ending at the nth \(<CR><LF>\).

\[-nK\]  
Delete lines beginning with the first character in the -nth line and ending at the pointer.

\[ 0K \]  
Delete from the beginning of the current line to the pointer (equivalent to 0D).

\[/K\]  
Delete from the pointer to the end of the Text Buffer (equivalent to /D).

Example:

\[ *2K\$\$ \]  
Delete lines starting at the current location pointer and ending at the 2nd \(<CR><LF>\).

Assuming a buffer of:

\[
\begin{align*}
\text{ADD} & \quad \text{RL,(R2)+} \\
\text{CLR+} & \quad \text{RL,R2} \\
\text{MOV} & \quad \text{G(R1),R2}
\end{align*}
\]

the command:

\[ */K\$\$ \]

alters the contents of the buffer to:

\[
\begin{align*}
\text{ADD} & \quad \text{RL,(R2)+} \\
\text{CLR+}
\end{align*}
\]

Kill and Delete commands perform the same function, except that Kill is line-oriented and Delete is character-oriented.

3.6.4.4 Change - The Change command replaces n characters, starting at the pointer, with the specified text string and leaves the pointer positioned immediately following the changed text.

The form of the command is:

\[ (+ \text{ or } -) \text{nCtext}\$ \]  
Replace n characters (forward or backward from the pointer) with the specified text.

\[ 0Ctext\$ \]  
Replace the characters from the beginning of the line up to the pointer with the specified text (equivalent to 0X).

\[ /Ctext\$ \]  
Replace the characters from the pointer to the end of the buffer with the specified text (equivalent to /X).
Text Editor

=Ctext$ Replace n characters with the indicated text string, where n represents the length of the last text argument used.

The size of the text is limited only by the size of the Text Buffer and the space available. All characters are legal except ALTMODE which terminates the text string.

If the C command is to be executed more than once (i.e., it is enclosed in angle brackets) and if there is enough space available so that the command can be entered, it will be executed at least once (provided it appears first in the command string). If repetition of the command exceeds the available memory, an error message is printed. The Change command is identical to executing a Delete command followed by an Insert (nDItext$).

Examples:

*5C#VECT## Replaces the five characters to the right of the pointer with #VECT.

Assuming a buffer of:

CLR @R2
MOV, 5(R1),@R2

The command:

*0CADDB##

leaves the buffer with:

CLR @R2
ADDB+ 5(R1),@R2

=C can be used in conjunction with a search command to replace a specific text string as follows:

*GFIFTY:=$CFIVE:$/ Find the occurrence of the text string FIFTY: and replace it with the text string FIVE:.

3.6.4.5 Exchange - The Exchange command exchanges n lines, beginning at the pointer, with the indicated text string and leaves the pointer positioned after the changed text.

The form of the command is:

nXtext$ Exchange all characters beginning at the pointer and ending at the nth <CR><LF> with the indicated text.

-nXtext$ Exchange all characters beginning with the first character on the -nth line and ending at the pointer with the indicated text.

0Xtext$ Exchange the current line from the beginning to the pointer with the specified text (equivalent to 0C).
Text Editor

/Xtext$  Exchange the lines from the pointer to the end of the buffer with the specified text (equivalent to /C).

All characters are legal in the text string except ALTMODE which terminates the text.

The Exchange command is identical to a Kill command followed by an Insert (nKItex$t$), and accepts all legal line-oriented arguments.

If the X command is enclosed within angle brackets so that it will be executed more than once, and if there is enough memory space available so that the X command can be entered, it will be executed at least once (provided it is first in the command string). If repetition of the command exceeds the available memory, an error message is printed.

Example:

```
*2XADD  R1,(R2)+
CLR @R2
$$
*
```

Exchanges the two lines to the right of the pointer location with the text string.

3.6.5 Utility Commands

3.6.5.1 Save - The Save command starts at the pointer and copies the specified number of lines into the Save Buffer (described previously in Section 3.5).

The form of the command is:

```
nS
```

The argument (n) must be positive. The pointer position does not change and the contents of the Text Buffer are not altered. Each time a Save is executed, the previous contents of the Save Buffer, if any, are destroyed. If the Save command causes an overflow of the Save Buffer, an error message is printed.

Example:

Assume the Text Buffer contains the following assembly language subroutine:
Text Editor

;SUBROUTINE MSGTYP
;WHEN CALLED, EXPECTS RO TO POINT TO AN
;ASCII MESSAGE THAT ENDS IN A ZERO BYTE,
;TYPES THAT MESSAGE ON THE USER TERMINAL

.ASECT
.=1000
MSGTYP:
  TSTB (%0) ;DONE?
  BEQ MDONE ;YES-RETURN
MLOOP:
  TSTB @177564 ;NO-IS TERMINAL READY?
  BPL MLOOP ;NO-WAIT
  MOVB (%0)+,@@177566 ;YES PRINT CHARACTER
  BR MSGTYP ;LOOP
  RTS %7 ;RETURN

The command:

  *145**

stores the entire subroutine in the Save Buffer; it may then be
inserted in a program wherever needed by using the U command.

3.6.5.2 Unsave - The Unsave command inserts the entire contents of
the Save Buffer into the Text Buffer at the pointer location and
leaves the pointer positioned following the inserted text.

The form of the command is:

U Insert in the Text Buffer the contents of the Save
Buffer.  

OU Clear the Save Buffer and reclaim the area for text.

Zero is the only legal argument to the U command.

The contents of the Save Buffer are not destroyed by the Unsave
command (only by the OU command) and may be Unsaved as many times as
desired.

If there is no text in the Save Buffer and the U command is given, the
?*NO TEXT*? error message is printed. If the Unsave command causes
an overflow of the Text Buffer, the ?*NO ROOM*? error message is
displayed.

3.6.5.3 Macro - The Macro command inserts a command string into the
EDIT Macro Buffer. The Macro command is of the form:

M/command string/ Store the command string in the Macro
Buffer.

or

OM Clear the Macro Buffer and reclaim the
area for text.

/ represents the delimiter character. The delimiter is always the
first character following the M command, and may be any character
which does not appear in the Macro command string itself.

3-25
Starting with the character following the delimiter, EDIT places the Macro command string characters into its internal Macro Buffer until the delimiter is encountered again. At this point, EDIT returns to Command Mode. The Macro command does not execute the Macro string; it merely stores the command string so that it can be executed later by the Execute Macro (EM) command. Macro does not affect the contents of the Text or Save Buffers.

All characters except the delimiter are legal Macro command string characters, including single ALTMODEs to terminate text commands. All commands, except the M and EM commands, are legal in a command string macro.

In addition to the OM command, typing the M command immediately followed by two identical characters (assumed to be delimiters) and two ALTMODE characters also clears the Macro Buffer.

Examples:

*M//##
Clears the Macro Buffer

*M/GR0S-C1S/##
Stores a Macro to change R0 to R1.

NOTE

Be careful to choose infrequently used characters as macro delimiters; use of frequently used characters can lead to inadvertent errors. For example,

*M GMOV R0$=CADD R1S ##
?*NO FILE*

In this case, it was intended that the macro be GMOV R0$=CADD R1$ but since the delimiter character (the character following the M) is a space, the space following MOV is used as the second delimiter, terminating the macro. EDIT then returns an error when the R0$= becomes an illegal command structure.

3.6.5.4 Execute Macro - The Execute Macro command executes the command string specified in the last Macro command.

The form of the command is:

nEM

The argument (n) must be positive. The macro is executed n times and returns control to the next command in the original command string.
Text Editor

Examples:

*M/BGR0*-C1$/##
*B1000EM$$
?*SRCH FAIL IN MACRO*?
*

Executes the MACRO stored in the previous example. An error message is returned when the end of buffer is reached. (This macro effectively changes all occurrences of R0 in the Text Buffer to R1.)

*IMOV PC,R1$2EMICLR @R2$$
*

In a new program, inserts MOV PC,R1 then executes the command in the Macro Buffer twice before inserting CLR @R2.

3.6.5.5 Edit Version - The Edit Version command displays the version number of the Editor in use on the console terminal.

The form of the command is:

EV$

Example:

*EV$$
V02-01
*

3.6.5.6 Upper- and Lower-Case Commands - Users who have any upper/lower-case terminal as part of their hardware configuration may take advantage of the upper- and lower-case capability of this terminal. Two editing commands, EL and EU, permit this.

When the Editor is first called (R EDIT), upper-case mode is assumed; all characters typed are automatically translated to upper case. To allow processing of both upper- and lower-case characters, the Edit Lower command is entered:

*EL$$
*i Text and commands can be entered in UPPER and lower case.$$ *

The Editor now accepts and echoes upper- and lower-case characters received from the keyboard, and outputs text on the teleprinter in upper- and lower-case.

To return to upper-case mode, the Edit Upper command is used:

*EU$$

Control also reverts to upper-case mode upon exit from the Editor (via EF, EX, or CRTL C).
Text Editor

Note that when an EL command has been issued, Edit commands can be entered in either upper- or lower-case. Thus, the following two commands are equivalent:

\*GTEXT\* = Cnew text\*V\*V\*
\*GTEXT\* = Cnew text\*V\*V\*

The Editor automatically translates (internally) all commands to upper-case independent of EL or EU.

3.7 THE DISPLAY EDITOR

In addition to all functions and commands mentioned thus far, the Editor has additional capabilities to allow efficient use of VT-11 display hardware which may be part of the system configuration (GT40, GT44, DECLAB 11/40).

The most apparent feature is the ability to use the display screen rather than the console terminal as a window into the Text Buffer for printout of all textual input and output. When all the features of the display Editor are in use, a 12" screen displays text as shown in Figure 3-1:

![Image of display editor format with annotations]

Figure 3-1
Display Editor Format
Text Editor

The major advantage is that the user can now see immediately where the pointer is. The pointer appears between characters on the screen as a bright blinking L-shaped cursor and can be detected easily and quickly. Note that if the pointer is placed between a carriage return and line feed, it appears in an inverted position at the beginning of the next line.

In addition to displaying the current line (the line containing the cursor), the 10 lines of text preceding the current line and the 9 lines following it are also in view. Each time a command string is executed (via a double ALTMODE) this portion of the screen is refreshed so that it reflects the results of the commands just performed.

The lower section of the screen contains 4 lines of editing commands. The command line currently being entered is last, preceded by the three most recent command lines. This section is separated from the text portion of the screen by a horizontal line of dashes. As new command lines are entered, previous command lines are scrolled upward off the command section so that only four command lines are ever in view.

A 17" screen displays 30 lines of text and 8 command lines.

3.7.1 Using the Display Editor

The display features of the Editor are automatically invoked whenever the system scroller is in use and the user types:

R EDIT

However, if the system does not contain VT-11 display hardware, the display features are not enabled.

Providing that the system does contain VT-11 display hardware and that the user wishes to employ the screen during the editing session, he may activate it in one of two ways (all editing commands and functions previously discussed in this chapter are valid for use):

1. If the scroller is in use (i.e., the GT ON monitor command has been typed prior to calling the Editor), EDIT recognizes this and automatically continues using the screen for display of text and commands. However, it rearranges the scroller so that a "window" into the Text Buffer appears in the top two/thirds of the screen, while the bottom third is used to display command lines. This arrangement is shown in Figure 3-1.

The Edit Console command can be used to return the scroller to its normal mode so that text and commands appear as described in Chapter 2, Section 2.7.1 (i.e., using the full screen for display of command lines, and eliminating the window). The form of the command is:

EC
Text Editor

For example:

*BAEC2L** The second and third lines of the current buffer are listed on the screen; there is no window into the Text Buffer at this point.

Subsequent EC commands are ignored if the window into the Text Buffer is not being displayed.

To recall the window, the Edit Display command is used:

ED

The screen is again arranged as shown in Figure 3-1.

2. Assume the scroller is not in use (i.e., the GT ON command has not been typed, or the monitor GT OFF command has been typed prior to calling the Editor). When the user calls EDIT, an asterisk appears on the console terminal as described in Section 3.1. Using the ED command at this time provides the window into the Text Buffer; however, commands continue to be echoed to the console terminal.

When ED is used in this case, it must be the first command issued; otherwise, it becomes an illegal command (since the memory used by the display buffer and code, amounting to over 600 words, is reclaimed as working space). The display cannot be used again until a fresh copy of EDIT is loaded.

While the display of the text window is active, ED commands are ignored.

Typing the EC command clears the screen and returns all output to the console terminal.

NOTE

Under the Single-Job Monitor only, after the editing session is over, it is recommended that the screen be cleared by either typing the EC command, or returning to the monitor and using the monitor INITIALIZE command. Failure to do this may cause unpredictable results.

3.7.2 Setting the Editor to Immediate Mode

An additional mode is available in EDIT to provide an easier and faster degree of interaction during the editing session. This mode is called Immediate Mode and combines the most-used functions of the Text and Command Modes—namely, to reposition the pointer and to delete and insert characters.

Immediate Mode may be used only when the VT-11 display hardware is active and the Editor is running; it is entered by typing two ALTMODES (only) in response to the Command Mode asterisk:

*$$
Text Editor

The Editor responds by echoing an exclamation point on the screen. The exclamation character remains on the screen as long as control remains in Immediate Mode.

Once Immediate Mode has been entered, only the commands in Table 3-3 are used. None of these commands echoes, but the text appearing on the screen is constantly refreshed and updated during the editing process. Note that no EDIT commands other than those in Table 3-3 may be used while control remains in Immediate Mode.

To return control to the display Editor's normal Command Mode at any time while in Immediate Mode, type a single ALTMODE. The Editor responds with an asterisk and the user may proceed using all normal Editing commands. (Immediate Mode commands typed at this time will be accepted as Command Mode input characters.) To return control to the monitor while in Immediate Mode, type CTRL C.

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL N</td>
<td>Advance the pointer (cursor) to the beginning of the next line (equivalent to A).</td>
</tr>
<tr>
<td>CTRL G</td>
<td>Move the pointer (cursor) to the beginning of the previous line (equivalent to -A).</td>
</tr>
<tr>
<td>CTRL D</td>
<td>Move the pointer (cursor) forward by one character (equivalent to J).</td>
</tr>
<tr>
<td>CTRL V</td>
<td>Move the pointer (cursor) back by one character (equivalent to -J).</td>
</tr>
<tr>
<td>RUBOUT</td>
<td>Delete the character immediately preceding the pointer (cursor) (equivalent to -D).</td>
</tr>
<tr>
<td>CTRL C</td>
<td>Return control to the monitor.</td>
</tr>
<tr>
<td>ALTMODE (one only) (two)</td>
<td>Return control to Command Mode. Direct control to Immediate Mode.</td>
</tr>
<tr>
<td>Any other character than those above</td>
<td>Insert the character as text positioned immediately before the pointer (cursor) (equivalent to I).</td>
</tr>
</tbody>
</table>

Table 3-3
Immediate Mode Commands
3.8 EDIT EXAMPLE

The following example illustrates the use of some of the EDIT commands to change a program stored on the device DK. Sections of the terminal output are coded by letter and corresponding explanations follow the example.

```
.R EDIT
  \*ERDK:TEST1.MAC**
  \*EWDK:TEST2.MAC**
  \*R**
  \*/L**
  ;TEST PROGRAM
  START:  MOV #1000,%6  ;INITIALIZE STACK
          MOV #MSG,%0  ;POINT R0 TO MESSAGE
          JSR %7(MSG)  ;PRINT IT
          HALT  ;STOP
  MSG:  .ASCII/IT WORKS/
          .BYTE 15
          .BYTE 12
          .BYTE 0
  .B 1J 5D**
  .GPROGRAM**
  .BL**
  ;PROGRAM I TO TEST SUBROUTINE MSGTYP. TYPES
  ;"THE TEST PROGRAM WORKS"
  ;ON THE TERMINAL
  .F.ASCII/**
  .GTHE TEST PROGRAM WORKS**
  .P.BYTE"X"
  .G.BYTE 0$$
  .I
  \*/L**
  ;PROGRAM TO TEST SUBROUTINE MSGTYP. TYPES
  ;"THE TEST PROGRAM WORKS"
  ;ON THE TERMINAL
  \*/E**
  \*/L**
  ;START:  MOV #1000,%6  ;INITIALIZE STACK
          MOV #MSG,%0  ;POINT R0 TO MESSAGE
          JSR %7(MSG)  ;PRINT IT
          HALT  ;STOP
  MSG:  .ASCII/ THE TEST PROGRAM WORKS/
          .BYTE 15
          .BYTE 12
          .BYTE 0
  \*/EX**

```

3-32
A The EDIT program is called and prints an *. The input file is TEST1.MAC; the output file is TEST2.MAC and the first page of input is read.

B The buffer contents are listed.

C Be sure the pointer is at the beginning of the buffer. Advance pointer one character (past the ;) and delete the "TEST ".

D Position pointer after PROGRAM and verify the position by listing up to the pointer.

E Insert text. RUBOUT used to correct typing error.

F Search for .ASCII/ and change "IT WORKS" to "THE TEST PROGRAM WORKS".

G CTRL X typed to cancel P command. Search for ".BYTE 0" and verify location of pointer with V command.

H Insert text. Return pointer to beginning of buffer and list entire contents of buffer.

I Close input and output files after copying the current text buffer as well as the rest of input file into output file. EDIT returns control to the monitor.

3.9 EDIT ERROR MESSAGES

The Editor prints an error message whenever one of the error conditions listed next occurs. Prior to executing any commands, the Editor first scans the entire command string for errors in command format (illegal arguments, illegal combinations of commands, etc.). If an error of this type is found, an error message of the form:

?ERROR MSG?

is printed and no commands are executed. The user must retype the command.

If the command string is syntactically correct, execution is started. Execution errors are still possible, however (buffer overflow, I/O errors, etc.), and if such an error occurs, a message of the form:

?*ERROR MSG*?

is printed. In this case, all commands preceding the one in error are executed, while the command in error and those following are not executed. Most errors will generally be of the syntax type and can be corrected before execution.
When an error occurs during execution of a Macro, the message format is:

?message IN MACRO?

or

?*message IN MACRO*?

depending on when it is detected.

<table>
<thead>
<tr>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>CB ALMOST FULL</em></td>
<td>The command currently being entered is within 10 characters of exceeding the space available in the Command Buffer.</td>
</tr>
<tr>
<td>?CB FULL?</td>
<td>Command exceeds the space allowed for a command string in the Command Buffer.</td>
</tr>
<tr>
<td>?<em>DIR FULL</em>?</td>
<td>No room in device directory for output file.</td>
</tr>
<tr>
<td>?<em>EOF</em>?</td>
<td>Attempted a Read, Next or file searching command and no data was available.</td>
</tr>
<tr>
<td>?<em>FILE FULL</em>?</td>
<td>Available space for an output file is full. Type a CTRL C and the CLOSE monitor command to save the data already written.</td>
</tr>
<tr>
<td>?<em>FILE NOT FND</em>?</td>
<td>Attempted to open a nonexisting file for editing.</td>
</tr>
<tr>
<td>?<em>HDW ERR</em>?</td>
<td>A hardware error occurred during I/O. May be caused by WRITE LOCKed device.</td>
</tr>
<tr>
<td>?ILL ARG?</td>
<td>The argument specified is illegal for the command used. A negative argument was specified where a positive one was expected or argument exceeds the range + or - 16,383.</td>
</tr>
<tr>
<td>?ILL CMD?</td>
<td>EDIT does not recognize the command specified; ED was not the first command issued when used to activate the display hardware.</td>
</tr>
<tr>
<td>?<em>ILL DEV</em>?</td>
<td>Attempted to open a file on an `illegal device, or attempted to use display hardware when none was available (it may be in use by the other job).</td>
</tr>
<tr>
<td>?ILL MAC?</td>
<td>Delimiters were improperly used, or an attempt was made to enter an M command during execution of a Macro or an EM command while an EM was in progress.</td>
</tr>
<tr>
<td>Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>?<em>ILL NAME</em>?</td>
<td>File name specified in EB, EW, or ER is illegal.</td>
</tr>
<tr>
<td>?<em>NO FILE</em>?</td>
<td>Attempted to read or write when no file is open.</td>
</tr>
<tr>
<td>?<em>NO ROOM</em>?</td>
<td>Attempted to Insert, Save, Unsave, Read, Next, Change or Exchange when there was not enough room in the appropriate buffer. Delete unwanted buffers to create more room or write text to the output file.</td>
</tr>
<tr>
<td>?<em>NO TEXT</em>?</td>
<td>Attempted to call in text from the Save Buffer when there was no text available.</td>
</tr>
<tr>
<td>?<em>SRCH FAIL</em>?</td>
<td>The text string specified in a Get, Find or Position command was not found in the available data.</td>
</tr>
<tr>
<td>?&quot;&lt;&gt;&quot;ERR?</td>
<td>Iteration brackets are nested too deeply or used illegally or brackets are not matched.</td>
</tr>
</tbody>
</table>
CHAPTER 4

PERIPHERAL INTERCHANGE PROGRAM (PIP)

The Peripheral Interchange Program (PIP) is the file transfer and maintenance utility for RT-11. PIP is used to transfer files between any of the RT-11 devices (listed in Table 2-2), merge and delete files from these devices, and list, zero, and compress device directories.

4.1 CALLING AND USING PIP

To call PIP from the system device type:

    R PIP

in response to the dot printed by the Keyboard Monitor. The Command String Interpreter prints an asterisk at the left margin of the terminal and waits to receive a line of filenames and command switches. PIP accepts up to six input filenames and three output filenames; command switches are generally placed at the end of the command string but may follow any filename in the string. There is no limit to the number of switches which may be indicated in a command line, as long as only one operation (insertion, deletion, etc.) is represented.

Since PIP performs file transfers for all RT-11 data formats (ASCII, object, and image) there are no assumed extensions for either input or output files; all extensions, where present, must be explicitly specified.

Following completion of a PIP operation, the Command String Interpreter prints an asterisk at the left margin of the teleprinter and waits for another PIP command line. Typing CTRL C at any time returns control to the Keyboard Monitor. To restart PIP, type R PIP or the REENTER command in response to the monitor's dot.

4.1.1 Using the "Wild Card" Construction

PIP follows the standard file specification syntax explained in Section 2.5 (Chapter 2) with one exception; the asterisk character can be used in a command string to represent filenames or extensions. The asterisk (called the "wild card") in a file specification means "all". For instance, "*.MAC" means all files with the extension .MAC.
Peripheral Interchange Program

regardless of filename. "FORTN.*" means all files with the filename FORTN regardless of extension. "**.*" means all files, regardless of name or extension.

The wild card character is legal in the following cases only (switches are explained in the next section):

1. Input file specification for the copy and multiple copy operations (i.e., no switch, /I, /B, and /A).
2. File specification for the delete operation (/D).
3. Input and output file specifications for the rename operation (/R).
4. Input and output file specifications for the multiple copy operation (/X).
5. Input file specifications for the directory list operations (/L, /E, /F).

Operations on files implied by the wild card asterisk are performed in the order in which the files appear in the directory. System files with the extension .SYS and files with bad blocks and the extension .BAD are ignored when the wild card character is used unless the /Y switch is specified.

Examples:

**.BAK/D
Causes all files with the extension .BAK (regardless of their filenames) to be deleted from the device DK.

**.TST=*.BAK/R
Renames all files with a .BAK extension (regardless of filenames) so that these files now have a .TST extension (maintaining the same filenames).

**RK1=*.X/Y=*,*
Transfers all files, including system files, (regardless of filename or extension) from device DK to device RK1.

**.MAC=*.OBJ/L
Lists all files with .MAC and .OBJ extensions.

4.2 PIP SWITCHES

The various operations which can be performed by PIP are summarized in Table 4-1. If no switch is specified, PIP assumes the operation is a file transfer in image (/I) mode. Detailed explanations of the switches follow the table.
Peripheral Interchange Program

Table 4-1
PIP Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Section</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/A</td>
<td>4.2.2</td>
<td>Copies file(s) in ASCII mode; ignores nulls and rubouts; converts to 7-bit ASCII; CTRL Z (32 octal) treated as logical end-of-file on input.</td>
</tr>
<tr>
<td>/B</td>
<td>4.2.2</td>
<td>Copies files in formatted binary mode.</td>
</tr>
<tr>
<td>/C</td>
<td>4.2.2</td>
<td>May be used in conjunction with another switch to cause only files with current date (as designated using the monitor DATE command) to be included in the specified operation.</td>
</tr>
<tr>
<td>/D</td>
<td>4.2.4</td>
<td>Deletes file(s) from specified device.</td>
</tr>
<tr>
<td>/E</td>
<td>4.2.6</td>
<td>Lists the device directory including unused spaces and their sizes. An empty space on a cassette or magtape directory represents a deleted file. Sequence numbers are listed for cassettes.</td>
</tr>
<tr>
<td>/F</td>
<td>4.2.6</td>
<td>Prints a short directory (filenames only) of the specified device.</td>
</tr>
<tr>
<td>/G</td>
<td>4.2.2</td>
<td>Ignores any input errors which occur during a file transfer and continues copying.</td>
</tr>
<tr>
<td>/I or no switch</td>
<td>4.2.2</td>
<td>Copies file(s) in image mode (byte by byte). This is the default switch.</td>
</tr>
<tr>
<td>/K</td>
<td>4.2.12</td>
<td>Scans the specified device and types the absolute block numbers (in octal) of any bad blocks on the device.</td>
</tr>
<tr>
<td>/L</td>
<td>4.2.6</td>
<td>Lists the directory of the specified device, including the number of files, their dates, and the number of blocks used by each file. Sequence numbers are listed for cassettes.</td>
</tr>
<tr>
<td>/M:n</td>
<td>4.2.1</td>
<td>Used when I/O transfers involve either cassette or magtape. n represents the numeric position of the file to be accessed in relation to the physical position of the cassette or magtape on the drive. If n is positive, the tape spaces forward from its current position until either the filename or the nth file is found; if n is negative, the tape is rewound first, and then it spaces forward until either the filename or the nth file is found. If n is 0 (or not indicated) the tape is rewound and searched for the filename. For wild card operations, specification of /M with a positive argument will prevent the tape from rewinding between each file involved in the operation.</td>
</tr>
<tr>
<td>/N:n</td>
<td>4.2.7</td>
<td>Used with /Z to specify the number of directory segments (n) to allocate to the directory.</td>
</tr>
<tr>
<td>/O</td>
<td>4.2.10</td>
<td>Bootstraps the specified device (DT0, RKn, RF, DPn, DSN, DXn only).</td>
</tr>
</tbody>
</table>

(continued on next page)
### Peripheral Interchange Program

#### Table 4-1 (Cont.)

<table>
<thead>
<tr>
<th>Switch</th>
<th>Section</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Q</td>
<td>4.2.2</td>
<td>When used in conjunction with another PIP operation, causes PIP to type each filename which is eligible for a wild card operation and to ask for a confirmation of its inclusion in the operation. Typing a &quot;y&quot; causes the named file to be included in the operation; typing anything else excludes the file. The command line is not processed until the user has confirmed each file in the operation.</td>
</tr>
<tr>
<td>/R</td>
<td>4.2.5</td>
<td>Renames the specified file.</td>
</tr>
<tr>
<td>/S</td>
<td>4.2.8</td>
<td>Compresses the files on the specified directory device so that free blocks are combined into one area.</td>
</tr>
<tr>
<td>/T</td>
<td>4.2.4</td>
<td>Extends number of blocks allocated for a file.</td>
</tr>
<tr>
<td>/U</td>
<td>4.2.9</td>
<td>Copies the bootstrap from the specified file into absolute blocks 0 and 2 of the specified device.</td>
</tr>
<tr>
<td>/V</td>
<td>4.2.11</td>
<td>Types the version number of the PIP program being used.</td>
</tr>
<tr>
<td>/W</td>
<td>4.2.6</td>
<td>Includes the absolute starting block and any extra directory words in the directory listing for each file on the device (numbers in octal). Used with /F, /L, or /E.</td>
</tr>
<tr>
<td>/X</td>
<td>4.2.3</td>
<td>Copies files individually (without concatenation).</td>
</tr>
<tr>
<td>/Y</td>
<td>4.2.2</td>
<td>Causes system files and .BAD files to be operated on by the command specified. Attempted modifications or deletions of .SYS or .BAD files without /Y are not done and cause the message ?NO SYS ACTION? to be printed.</td>
</tr>
<tr>
<td>/Z:n</td>
<td>4.2.7</td>
<td>Zeroes (initializes) the directory of the specified device; n is used to allocate extra words per directory entry. When used with /N, the number of directory segments for entries may be specified. When used with cassette, /Z writes a sentinel file at the beginning of the tape; with magtape, /Z writes a volume label followed by a dummy file followed by double tape marks indicating logical end-of-tape.</td>
</tr>
</tbody>
</table>

#### 4.2.1 Operations Involving Magtape or Cassette

PIP operations involving cassette and magtape devices are handled somewhat differently than other FT-11 devices, because of the sequential nature of these devices. The last file on a cassette or magtape (the logical end-of-tape) is specially formatted so that it marks the end of current data and indicates where new data may begin (double end-of-file for magtape, sentinel file or physical end-of-tape for cassette). Therefore, operations which designate specific block lengths (such as /T and /N) are meaningless, and unused spaces on the tape (resulting from file deletions) cannot be filled.
Peripheral Interchange Program

PIP operations which are legal using cassette and magtape (including the bootable magtape on which the system may have been distributed) include the following: /A, /B, /D, /E, /F, /G, /I, /L, /M, /O, /V, /W, /X, /Y, and /Z. Usually the device (CT or MT) is rewound each time an operation is performed. Since there is no inclusive directory at the beginning of the tape the only way to access a file is to search the tape from the beginning until it is found. However, the /M:n switch is available for situations where it is not necessary or desirable to rewind the tape before each operation. If the argument (n) is positive, the operation indicated will not rewind the tape first, but will space forward until it finds either the nth file, the filename indicated in the command line, or the logical end-of-tape, whichever occurs first. If the argument is negative, the cassette or magtape will be rewound first and then spaced forward until the filename (or nth file, or logical end-of-tape) is found. Thus:

/M:1
means suppress rewind, begin operation at current position.

/M:-1
means rewind tape and access the first file on it.

Remember that when /M:n is used, n is interpreted as an octal number. /M:n must be used if it is intended that n represent a decimal number.

For example, assume the directory of a cassette on unit 1 is:

```
17-JUL-74
FILE .1 0 5-MAY-74
FILE .2 0 5-MAY-74
FILE .3 1 13-MAY-74
FILE .4 1 28-JUN-74
FILE .5 0 17-JUL-74
5 FILES, 2 BLOCKS
*  
```

and the last PIP operation involved FILE.4, leaving the cassette positioned at the end of FILE.4. To access FILE.2, the next operation (for example, deleting FILE.2) could use the /M construction:

```
*CT1:DUM/M:-2/D
```

In this case, the cassette rewinds first, then spaces forward from its current position to the second file in sequence and deletes it. (In a delete operation, the dummy filename is necessary; otherwise, a non-file structured delete is performed and the tape is zeroed. See Section 4.2.4).

Another useful application of the /M switch involves a case where a number of files are to be created on a magtape or cassette. Using the construction:

```
*MT:*.*/X=FILE.1 FILE.2.../M:1000
```

prevents a rewind from occurring before each new file is created on the tape. Normal operation (when creating a new file on magtape or cassette) is to rewind, then search the tape for the logical end. If a file with the same name as the one being created is encountered, it is deleted and the new file is opened at the logical end of the tape. The /M:1000 command first causes the tape to space forward until it reaches the logical end-of-tape, (assuming less than 1000 (octal) files on the tape), at which point the next file is entered, and so on. If the tape were already positioned at the end of the tape, an
Peripheral Interchange Program

/M:1 would suffice to cause the new file to be written there. Note that creation of a new file with the /M switch can result in several files with the same name on the same tape; those files occurring before the tape position are not searched for duplication prior to the creation of the new file.

RT-11 magtapes sometimes contain a dummy file at the beginning of the tape, which is written when the tape is initialized with the /Z switch. This file shows up in extended directories (/E) as an <UNUSED> entry in the first file position. Deleted files on magtape or cassette do not show up in /F or /L directory listings, but must always be considered when the /M:n switch is used. Care must always be taken to use a /E directory when counting file position prior to using that position as an /M:n argument; <UNUSED> files must be counted as files on the tape.

For example:

```
.R PIP
*MT0:/E
11-SEP-74
< UNUSED > 0
  < UNUSED > 0
A . MAC 40 11-SEP-74
B . MAC 15 11-SEP-74
< UNUSED > 2
D . MAC 2 11-SEP-74
3 FILES, 57 BLOCKS

*MT0:/L
11-SEP-74
A . MAC 40 11-SEP-74
B . MAC 15 11-SEP-74
D . MAC 2 11-SEP-74
3 FILES, 57 BLOCKS
```

Extended directory: shows absolute file positions.

Normal directory: does not accurately display file positions.

If the user wished to access file A,MAC on the magtape in the example above, /M:2 must be used (/M:1 would access the first empty file). Likewise, B,MAC is accessed with /M:3. Rewind can also be suppressed for cassette and magtape as input devices by specifying a very large number in conjunction with wild card transfers from magtape or cassette.

```
**.*=MT0:**.*M:2000/X
```

This transfers all files from MT0: to DK: without rewinding between each file. The argument 2000 is an arbitrarily large number; any number larger than the actual number of files on the tape will suffice.

The most common method for spacing to the end of the tape is:

```
*DUMMY=MT0:DUMMY/M:2000
?FIL NOT FND?
```

where DUMMY is a file name which does not exist on the tape. Note that an error message is printed when the end of the tape is reached.

July 1975 4-6
Peripheral Interchange Program

Directory listings of magtapes include the length of each file in 256(decimal) word blocks. In cassette directories, however, sequence numbers rather than block numbers are printed. Sequence numbers indicate the sequential ordering of a file in cases where it has been continued on more than one cassette. In the example cassette directory listing (at the beginning of this section), the numbers in the middle column represent sequence numbers; both FILE.3 and FILE.4 are the second segments of continued files. All files on cassette are initially assigned a sequence number of 0 (meaning this is the first segment of the cassette file, not that the file has no length). The sequence number is automatically updated whenever the file must be continued as a result of a full cassette.

During I/O transfer operations involving cassette, if the cassette is full before the transfer has finished, the message:

CTn: PUSH REWIND OR MOUNT NEW VOLUME

is printed; n represents the number of the drive (0 or 1) on which the current cassette is mounted. If the cassette rewind button is subsequently pushed, an error message is typed (IN or OUT ERR) and the tape is rewound.

To continue an output operation, mount a new cassette (which has been properly formatted as described in Section 4.2.7) on the same drive. The new cassette is rewound automatically and a file is opened on it under the same name and extension; the sequence number in its directory is updated to reflect the continuation, and the transfer continues.

If the message occurs during an input operation, mount the cassette containing the continued portion of the file on the drive; the cassette is rewound first. PIP then looks for a file with the same name and extension and the proper sequence number and continues the input operation. The message is repeated if the next segment is not found.

For example:

*CT0: FILE.AGA=DT1:ASC.MAC,DK:BALOR.MAC/A
CT0: PUSH REWIND OR MOUNT NEW VOLUME

This copies in ASCII mode the file ASC.MAC from DECTape 1 and BALOR.MAC from device DK and combines them under the name FILE.AGA on CT0. The cassette runs out of room and requests that a new one be mounted. The operation continues automatically when the second cassette has been mounted.

A directory of the second cassette in the above operation is next requested; note that the sequence number of FILE.AGA is 1, signifying it is the second part of a continued file.

*CT0: /L
23-MAY-74
TRA .BIN 0 16-FEB-74
FILE .AGA 1 23-MAY-74
2 FILES, 1 BLOCKS
*

(The number of blocks in a cassette directory simply represents the total of sequence numbers in the directory.)

Any cassette mounted in response to a continuation message MUST have been previously initialized at some time as described in Section 4.2.7.

January 1976
Peripheral Interchange Program

If a full cassette is mounted or an attempt is made to access some file on it that does not exist, the continuation message recurs. The operation may be continued by mounting another cassette.

Note that if an attempt is made to access a file which has a non-zero sequence number (during some operation which is not a continuation of an operation), the file will not be found.

To copy multiple files to a cassette using a wild card command, use the following:

```*CTn:*.*=DEV:*.*/X/M:1      (rewind is inhibited)```

Continue to mount new cassettes in response to the PUSH REWIND OR MOUNT NEW VOLUME message. Do not abort the process at any time (using two CTRL Cs) since continuation files may not be completed and no sentinel file will be written on the cassette.

To read multiple files from a cassette, use the following:

```*DEV:*.*=CTn:*.*/X/M:1000   (rewind is inhibited)```

Whenever a continued volume is detected, the PUSH REWIND OR MOUNT NEW VOLUME message will appear, until the entire file has been copied (assuming that each sequential cassette is mounted in response to each occurrence of the message). Whenever PIP has copied the final section of a continued file, it will return to command level. To copy the remaining files on that cassette, reissue the command:

```*DEV:*.*=CTn:*.*/X/M:1000```

Repeat the process as often as necessary to copy all files. Do not abort the process at any time (using two CTRL Cs) since continuation files may not be completed.

If the end of a tape is reached during a magtape I/O operation, an IN or OUT ERR message is printed. In the case of an output operation, the magtape backspaces and deletes the partial file by writing logical end of tape over the file's header label. The operation must then be repeated using another magtape.

If CTRL C is typed during any output operation to cassette or magtape, an end-of-tape or sentinel file is not written on the tape first. Consequently, no future enters may occur to the tape unless one of two recovery procedures is followed:

1. Transfer all good files from the bad tape to another tape and zero the bad tape in the following manner:

   ```*dev1:*.*/X=dev0:file1,file2,...,filen/M:1000
   *dev0:/Z
   dev0:/Z ARE YOU SURE ?```

   This causes a logical end-of-tape to be written onto the bad tape and makes it again available for use.
Peripheral Interchange Program

2. Determine the sequential number of the file which was interrupted and use the /M construction to enter a replacement file (either a new file or a dummy file). Assuming the bad file is the 4th file on the tape, use a command line of this construction:

*dev0:file.new=file.dum/M:-4

A logical end-of-tape now exists on the tape, making it available for use.

Since magtapes and cassettes are not random access devices, each unit can have only one file accessed at a time. Avoid PIP command strings which specify the same unit number for both input and output, since a loss of information can occur. For example:

*CT0:FILE1.MAC=CT0:FILE2.MAC
FIL NOT FND?
*

The result of this operation is to delete FILE1.MAC before the error message is printed, and the tape label structure may be destroyed.

Recovery procedures for errors caused by bad tapes are described in RT-11 Software Support Manual.
This page intentionally blank.
Peripheral Interchange Program

4.2.2 Copy Operations

A command line without a switch causes files to be copied onto the destination device in image mode (byte by byte). This operation is used to transfer memory image (save format) files and any files other than ASCII or formatted binary. For example:

*ABC<XYZ  Makes a copy of the file named XYZ on device DK and assigns the name ABC. (Both files exist on device DK following the operation).

*SY:BACK.BIN=PR:/I  Copies a tape from the papertape reader to the system device in image mode and assigns it the name BACK.BIN.

The /A switch is used to copy file(s) in ASCII mode as follows:

*DT1:F1<F2/A  Copies F2 from device DK onto device DT1 in ASCII mode and assigns the name F1.

Nulls and rubouts are ignored in an ASCII mode file transfer. CTRL Z (32 octal) is treated as logical end-of-file if encountered in the input file.

The /B switch is used to transfer formatted binary files. The formatted binary copy switch should be used for .OBJ files produced by the assembler or FORTRAN and for .LDA files produced by the Linker. For example:

*DK:FILE.OBJ<PR:/B  Transfers a formatted binary file from the papertape reader to device DK and assigns the name FILE.OBJ.

When performing formatted binary transfers, PIP verifies checksums and prints the message ?CHK SUM? if a checksum error occurs.

If neither /A nor /B is used in a copy operation that involves a paper tape device, the size of the output file in the operation depends upon the memory size of the system. The transfer mode defaults to image mode and PIP attempts to do a single read to fill its input buffer. When a read from the paper tape reader encounters end-of-tape, no count of words transferred can be returned; PIP assumes its input buffer is full and copies it to the output device. The output file size thus depends upon the input buffer size, which is determined by the memory size of the system. The output file will have several blocks of zeroes after the end of the paper tape image. If copying to the punch, large amounts of blank tape will be punched after the input tape image is output. The extra length is harmless, but can be avoided by use of /A or /B. Image mode files (for example, .SAV files) cannot reliably be transferred to or from paper tape.

To combine more than one file into a single file, use the following format:

*DK:AA<DT1:BB,CC,DD/I  Transfers files BB, CC and DD to device DK as one file and assigns this file the name AA.
Peripheral Interchange Program

*DT3:MERGE=DT2:FILE2,FILE3/A
Merges ASCII files FILE2 and FILE3 on DT2 into one ASCII file named MERGE on device DT3.

Errors which occur during the copy operation (such as a parity error) cause PIP to output an error message and return for another command string.

The /G switch is used to copy files but ignore all input errors. For example:

*ABC<DT1:TOP/G
Copies file TOP in image mode from device DT1 to device DK and assigns the name ABC. Any errors during the copy operation are ignored.
This page intentionally blank.
Peripheral Interchange Program

*DT2:COMB<DT1:F1,F2/A/G
Copies files F1 and F2 in ASCII mode from device DT1 to device DT2 as one file with the name COMB. Ignores input errors.

The wild card construction may be used for input file specifications during copy operations. Be sure to use the /Y switch if system files (.SYS) are to be copied. For example:

*DT1:PROG1<*,MAC
Copies, in image mode, all files with a .MAC extension from device DT1 to device DT1 and combines them under the name PROG1.

**,=*DT3:*,/*/Y/X
Copies to device DT3, in image mode, all files (including .SYS files) from device DT3; ignores any input errors.

If only files with the current date are to be copied (using the wild card construction), the /C switch must also be used in the command line. For example:

*DT2:NN3=ITEM1.*/C,ITEM2/A
Copies, in ASCII mode, all files having the filename ITEM1 and the current date, (the date entered using the monitor DATE command) and copies ITEM2 (regardless of its date) from device DTK to device DT2 and combines them under the name NN3.

*DT3:*,***.*/C/X
Copies all files with the current date from DTK to DT3. Note that commands of this nature are an efficient way to backup all new files after a session at the computer.

The /Q switch is used in conjunction with another PIP operation and the wild card construction to list all files and allow the user the opportunity to confirm individually which of these files should be processed during the wild card expansion. Typing a "Y" causes the named file to be processed; typing anything else excludes the file. For example:

**,OBJ<DT1:*.*,OBJ/Q/X
FIRST .OBJ?Y
GETR .OBJ?Y
RORD .OBJ?Y
CARJ .OBJ?Y
Copies the files FIRST.OBJ and CARJ.OBJ to the disk in image mode from DECTape 1 and ignores the others.

The file allocation scheme for RT-11 normally allows half the entire largest available space or the second largest space, or a maximum size (a constant which may be patched in the RT-11 monitor; see the RT-11 System Generation Manual), whichever is largest, for a new file. The user can, using the [n] construction explained in Chapter 2, force RT-11 to allow the entire largest possible space by setting n=177777. If n is set equal to any other value (other than 0 which is default and gives the normal allocation described first above), that size will be allocated for the file.
Peripheral Interchange Program

Therefore, assume that the directory for a given device shows a free area of 200 blocks and that PIP returns an OUT ER? message when a transfer is attempted to that device with a file which is longer than 100 blocks but less than 200 blocks. Transfers in this situation can be accomplished in either of two ways:

1. Use the [n] construction on the output file to specify the desired length (refer to Chapter 2, Section 2.5 for an explanation of the [n] construction).

2. Use the /X switch during the transfer to force PIP to allocate the correct number of blocks for the output file. This procedure will operate correctly if the input device is DECTape or disk.

For example, assume that file A is 150 blocks long and that a directory listing shows that there is a 200 block (unused) space on DT1:

```
.R PIP
*DT1:A=A
?OUT ER?

*DT1:A[150]=A

or

*DT1:A=A/X
```

File longer than 100 blocks.

Either command causes a correct transfer.

4.2.3 Multiple Copy Operations

The /X switch allows the transfer of several files at a time onto the destination device as individual files. The /A, /G, /C, /Q, /B and /Y switches can be used with /X. If /X is not indicated, all output files but the first will be ignored.

Examples:

```
*FILE1,FILE2,FILE3<DT1:FILEA,FILEB,FILEC/X

Copies, in image mode, FILEA, FILEB and FILEC from device DT1 to device DK as separate files called FILE1, FILE2 and FILE3, respectively.

*DT2:F1.*=F2. */X
?NO SYS ACTION?

Copies, in image mode, all files named F2 (except files with .SYS or .BAD extensions) from device DK to device DT2. Each file is assigned the filename F1 but retains its original extension.

*DT1:*,*=DT2:*,*/X
?NO SYS ACTION?

Copies, in image mode, all files on device DT2 to device DT1 (except files with .SYS or .BAD extensions); the files are copied separately and retain the same names and extensions.

*DT1:FILE1,FILE2<FILE3. */R/G/X

This command line assumes there are two files with the filename FILE3 (and any extension excluding .SYS or .BAD extensions) and copies these files in
Peripheral Interchange Program

ASCII mode to device DT0. The files are transferred in the order they are found in the directory; the first file found is copied and assigned the name FILE1, and the second is assigned FILE2. If there is a third, it is ignored and a fourth causes an ?OUT FIL? error.

*DT0:* SYS=* SYS/X/Y

Copies all system files from device DK to device DT0.

File transfers performed via normal operations place the new file in the largest available area on the disk. The /X switch, however, places the copied files in the first free place large enough to accommodate it. Therefore, the /X switch should be used whenever possible (i.e., when no concatenation is desired) as an aid to reducing disk fragmentation.

*A=B

and

*A=B/X

perform the same operation; however, using the second construction whenever possible increases the system disk-usage efficiency.

For example, assume the directory of DT1 is:

9-MAY-74
MONITR SYS 32 5-MAY-74
< UNUSED > 2
PR SYS 2 5-MAY-74
< UNUSED > 528
2 FILES, 34 BLOCKS
530 FREE BLOCKS

To copy the file PP.SYS (2 blocks long) from DK to DT1, the command:

*DT1:PP.SYS=PP.SYS/Y

can be entered, and the new directory is:

9-MAY-74
MONITR SYS 32 5-MAY-74
< UNUSED > 2
PR SYS 2 5-MAY-74
PP SYS 2 9-MAY-74
< UNUSED > 526
3 FILES, 36 BLOCKS
528 FREE BLOCKS

If the command:

*DT1:PP.SYS=PP.SYS/Y/X

had been entered, the new directory would appear:
Peripheral Interchange Program

9-MAY-74
MONITOR.SYS 32 5-MAY-74
PP.SYS 2 9-MAY-74
PR.SYS 2 5-MAY-74
<UNUSED> 528
3 FILES, 36 BLOCKS
528 FREE BLOCKS

4.2.4 The Extend and Delete Operations

The /T switch is used to increase the number of blocks allocated for the specified file. The file associated with the /T switch must be followed by a numeric argument of the form [n] where n is a decimal number indicating the number of blocks to be allocated to the file at the completion of the extend operation.

The format of the /T switch is:

    dev:filnam.ext[n]=/T

A file can be extended in this manner only if it is followed by an unused area of sufficient size (on whichever device it is located) to accommodate the additional length of the extended file. It may be necessary to create this space by moving other files on the device using the /X switch.

Specifying the /T switch in conjunction with a file that does not currently exist creates a file of the designated length.

Error messages are printed if the /T command makes the specified file smaller (?EXT NEG?) or if there is insufficient space following the file (?ROOM?).

Examples:

*ABC[200]=/T  Assigns 200 blocks to file ABC on device DK.

*DT1:XYZ[100]K=/T Assigns 100 blocks to the file named XYZ on device DT1.

The /D switch is used to delete one or more files from the specified device. The wild card character (*) can be used in conjunction with this command.

Only six files can be specified in a delete operation if each file to be deleted is individually named (i.e., if the wild card character is not used).

A cassette or magtape may be initialized by indicating the /D switch and omitting any filenames. For example:

*MT:/D
*CT:/D

Both devices are zeroed. This is not the case with the other RT-11 devices, where omission of a filename causes no action to occur.
Peripheral Interchange Program

When a file is deleted on block-replaceable devices, the information is not destroyed. The file name is merely removed from the directory. If a file has been deleted but not overwritten, it can be recovered with the /T switch by specifying a command of the form:

`filena.ext[n]=/T`

where `filena.ext` is the name desired and `n` is the length of the deleted file. For example:

```
*DT1:/E
   4-JUN-74
   A   MAC    18   3-JUN-74
   B   MAC    17   3-JUN-74
   C   MAC    19   3-JUN-74
   < UNUsed > 510
   3 FILES, 54 BLOCKS
   510 FREE BLOCKS

*DT1: B.MAC/D

*DT1:/E
   4-JUN-74
   A   MAC    18   3-JUN-74
   < UNUsed > 17
   C   MAC    19   3-JUN-74
   < UNUsed > 510
   2 FILES, 37 BLOCKS
   527 FREE BLOCKS
```

File B.MAC could now be recovered by:

```
*DT1:B.MAC[17]=/T
```

The /T switch looks for the first unused area large enough to accommodate the requested file length. If the file to be recovered is in the first area large enough to accommodate the size specified, the preceding command is sufficient. If not, all larger unused spaces preceding the desired file must be given dummy names before the recovery can be made.

For instance, assume the previous example with the exception that A.MAC has a 33 block unused file before it, so that the directory looks like:

```
*DT1:/E
   4-JUN-74
   < UNUsed > 33
   A   MAC    18   3-JUN-74
   < UNUsed > 17
   C   MAC    19   3-JUN-74
   < UNUsed > 477
   2 FILES, 37 BLOCKS
   527 FREE BLOCKS
```

A recovery of B.MAC would require:

```
*DT1: DUMMY[33]=/T
*DT1:B.MAC[17]=/T
```
Peripheral Interchange Program

If the 33 block unused area was not named prior to .MAC, the first 17 blocks of the 33 block area would become .MAC. Note that magtape and cassette files cannot be recovered once deleted.

Examples:

*FILE1.SAV/D Deletes FILE1.SAV from device DK.

*DT1::* */D Deletes all files from device DT1 except those with a .SYS or .BAD extension. If there is a file with a .SYS or .BAD extension, the message ?NO SYS ACTION? is printed to remind the user that these files have not been deleted.

**.MAC/D Deletes all files with a .MAC extension from device DK.

*DT1:B1,DT2:R1,DT3:AA/D Deletes the files specified from the associated devices.

*RK1::* */Y Deletes all files from device RK1.

4.2.5 The Rename Operation

The /R switch is used (in a manner similar to the multiple copy command described in Section 4.2.3) to rename a file given as input with the associated name given in the output specification. There must be an equal number of input and output files and they must reside on the same device, or an error message will be printed. The /Y switch must be used in conjunction with /R if .SYS files are to be renamed.

The Rename command is particularly useful when a file on disk or DECTape contains bad blocks. By renaming the file with a .BAD extension, the file permanently resides in that area of the device so that no other attempts to use the bad area will occur. Once a file is given a .BAD extension it cannot be moved during a compress operation. .BAD files are not renamed in wild card operations unless /Y is used.

Examples:

*DT1:F1,X1<DT1:F0,X0/R Renames F0 to F1 and X0 to X1 on device DT1.

*FILE1.*<FILE2. */R Renames all files on device DK with the name FILE2 (except files with .SYS or .BAD extension) to FILE1, retaining the original extensions.

/R cannot be used with magtape or cassette.

4.2.6 Directory List Operations

The /L switch lists the directory of the specified device. The listing contains the current date, all files with their associated creation dates, total free blocks on the device if disk or DECTape, the number of files listed, and number of blocks used by the files.
Peripheral Interchange Program

(sequence number for cassette). File lengths, number of blocks and number of files are indicated as decimal values. If no output device is specified, the directory is output to the terminal (TT:).

Examples:

```
*DT1:/L
1-AUG-74
MONITR.SYS  32  5-MAY-74
PP .SYS     2   9-MAY-74
PR .SYS     2   5-MAY-74
F2 .REL     15
MERGE       2
COMB         2
6 FILES, 55 BLOCKS
509 FREE BLOCKS
```

Outputs complete directory of device DT1 to the terminal.

```
*DIRECT=DT3:/L

** .MAC/L
1-AUG-74
VTMAC .MAC  7  22-JUL-74
FILE2 .MAC  1
2 FILES, 8 BLOCKS
328 FREE BLOCKS
```

Outputs complete directory of device DT3 to a file, DIRECT, on the device DK.

```
*CT1:/L
10-SEP-74
PAT1 .FOR  0 10-SEP-74
PAT2 .FOR  0 10-SEP-74
IMUL .OBJ  0 10-SEP-74
SORT .FTN  0 10-SEP-74
4 FILES, 0 BLOCKS
```

Lists all files on cassette drive 1. For cassette only, the third column represents the sequence number. In this example, the first segment of each file is on this cassette. (See Section 4.2.1.)

The /E switch lists the entire directory including the unused areas and their sizes in blocks (decimal); an empty space appears in cassette and magtape directories to designate a deleted file.

Examples:

```
/*E
9-SEP-74
BATCH .HLP  2  23-AUG-74
CHESS .SAV  20 23-AUG-74
PAT1 .FOR 10  23-AUG-74
IRAD50 .MAC 8  23-AUG-74
```

Outputs to the terminal a complete directory of the device DK including the size of unused areas.

January 1976  4-16
Peripheral Interchange Program

< UNUSED > 2
TRIG .OBJ 2 6-SEP-74

STP .OBJ 2 6-SEP-74
BAC .OBJ 2 6-SEP-74
< UNUSED > 20

LIBK1 .OBJ 137 6-SEP-74
DIRECT 1 9-SEP-74
< UNUSED > 230
254 FILES, 4280 BLOCKS
498 FREE BLOCKS

*LP:=CT1:/E
11-SEP-74
A .MAC 0 11-SEP-74
A .MAC 0 11-SEP-74
B .MAC 0 11-SEP-74
3 FILES, 0 BLOCKS

The /F switch lists only filenames, omitting the file lengths and associated dates.

Examples:

*DT0:/F
TRACE .MAC
CARGO .REL
BMAP .OBJ
AAA

*LP:=CT1:/F

Outputs to the line printer a complete directory of cassette drive 1. 0's represent segment numbers.

The /F switch lists only filenames, omitting the file lengths and associated dates.

Examples:

*DT0:/F
TRACE .MAC
CARGO .REL
BMAP .OBJ
AAA

*LP:=CT1:/F

Outputs to the line printer a complete directory of the device DT0 to the terminal.

Outputs a filename directory of the device CT1 to the line printer.

The /L, /E and /F commands have no effect on the files of the specified device. If the /W switch is used in conjunction with the /L or /E switches, the absolute starting block of the file and extra words (in octal) will be included in the listing (for all but cassette and magtape). For example:

*RK1:/L/W
16-SEP-74
DSQRT .OBJ 1 16-SEP-74 16 0
MAIN .OBJ 1 16-SEP-74 17 0
BASICR .OBJ 11 16-SEP-74 20 0
OTSV2 .OBJ 3 16-SEP-74 33 0

The first three columns indicate the filename and extension, block length, and date. The fourth column shows the absolute starting block (in octal), and the fifth column shows the contents of each extra word per directory entry (in octal). (This is allocated using the /L:n switch; see Section 4.2.7.)
Peripheral Interchange Program

Using the /L, /E, or /F switch in conjunction with a device and filename causes the filename, and optionally the date and file length, to be output rather than a directory of the entire device. For example:

*F1.SAV/L

causes:

4-JUN-74
F1   .SAV   18  4-JUN-74
3710 FREE BLOCKS
*

to be output, providing the file exists on device DK.

Directories are made up of segments which are two blocks long. Full directory listings with multiple segments contain blank lines as segment boundaries.

4.2.7 The Directory Initialization Operation

The /Z switch clears and initializes the directory of an RT-11 directory-structured device and writes logical end-of-file to a cassette or magtape device. The /Z operation must always be the first operation performed on a new (that is, previously unused) device. The form of the switch is:

/Z:n

where n is an optional octal number to increase the size of each directory entry on a directory-structured device. If n is not specified, each entry is 7 words long (for filename and file length information) and 70 entries can be made in a directory segment. When extra words are allocated, the number of entries per directory segment decreases. The formula for determining the number of entries per directory segment is:

507/((# of extra words)+7)

For example, if the switch /Z:1 is used, 63 entries can be made per segment.

More information concerning the format of directory entries is supplied in Chapter 3 of the RT-11 Software Support Manual.

When /Z is used, PIP responds as follows:

device/Z ARE YOU SURE ?

For example:

*DT1:/Z
DT1:/Z ARE YOU SURE ?

Answer Y and a carriage return to perform the initialization. An answer beginning with a character other than Y is considered to be no.

Example:

*DT1:/Z
DT1:/Z ARE YOU SURE ?Y<CR>
*

Zeroes the directory on device DT1 and allocates no extra words for the directory.

January 1976 4-18
Peripheral Interchange Program

The /N switch is used with /Z to specify the number of directory segments for entries in the directory. The form of the switch is:

/N:n

where n is an octal number less than or equal to 37. Initially RT-11 allocates four directory segments, each two blocks (512 words) long. Refer to Chapter 3 of the RT-11 Software Support Manual for more information.

Example:

*RK1:/Z:2/N:6

Zeroes the directory on device RK1, allocates two extra words per directory entry and allocates six directory segments.

4.2.8 The Compress Operation

The /S switch is used to compress the directory and files on the specified device, condensing all the free (unused) blocks into one area. Input errors are reported on the console terminal unless the /G switch is used; output errors are always reported. In either case, the compress continues. /S can also be used to copy DECTapes and disks. When DT, DP, or RK devices are copied, /S serves to both initialize the volume and to copy directory and files. When DX disks are copied, however, the output diskette must first be initialized using /Z to write the appropriate volume identification. (It is important to note that the /S switch destroys any previous directory on the output device. The new directory on the output device has the same number of segments as the directory on the input device.) /S does not copy the bootstrap onto the volume.

To increase the number of directory blocks in a two-volume compress (that is, from one volume to another rather than from one volume to itself), use the /N:n switch in conjunction with the /S switch (any attempts to decrease the directory size are ignored).

/S does not move files with the .BAD extension. This feature provides protection against reusing bad blocks which may occur on a disk. Files containing bad blocks can be renamed with the .BAD extension and are then left in place when a /S is executed.

If a compress operation is performed on the system device, the message:

?REBOOT?

is printed to indicate that it may be necessary to reboot the system. If .SYS files were not moved during the compress operation, it is not necessary to reboot the system.

NOTE

Rebooting the system in response to the ?REBOOT? warning message should ONLY be done AFTER the operation which generated the message is complete. ?REBOOT? does not signify that the system should be
Peripheral Interchange Program

rebooted immediately; the user should wait for the "*" signifying that PIP is ready for another command before rebooting.

If the command attempts to compress a large device to a smaller one, an error results and the directory of the smaller device is zeroed. If a device is being compressed in place, input and output errors are reported on the terminal and the operation continues to completion.

Examples:

*SY:/S
?REBOOT?

*DT1:A<DT2:/S

Compresses the files on the system device SY:

Transfers and compresses the files from device DT2 to device DT1. Device DT2 is not changed. The filename A is a dummy specification required by the Command String Interpreter.

/S cannot be used when a foreground job is present; a ?FG PRESENT? error message results if this is attempted.

4.2.9 The Bootstrap Copy Operation

The bootstrap copy switch (/U) copies the bootstrap portion of the specified file into absolute blocks 0 and 2 of the specified device.

Examples:

*DK:A<DK:MONITR.SYS/U

Writes the bootstrap file MONITR.SYS in blocks 0 and 2 of the device DK. A is a dummy filename.

*DT:MONITR.SYS/X/Y=RK:DTMNSJ.SYS

*DTA=RK:DTMNSJ.SYS/U

Writes the Single-Job DECTape Monitor to device DT0 and then writes the bootstrap into blocks 0 and 2 (the bootstrap is written from disk rather than DECTape because disk is faster).

4.2.10 The Boot Operation

The boot switch reboots the system, reinitializing monitor tables and returning the system to the monitor level. The boot switch performs the same operation as a hardware bootstrap.

Example:

*DK:/O

Reboots the device DK.
Peripheral Interchange Program

If a boot switch is specified on an illegal device, the message:

?BAD BOOT?

is printed. Legal devices are DT0, RK0-RK7, RF, SY, DK, DP0-DP7,
DX0-DX1, and DS0-DS7. Note that /O is illegal if a foreground job
is present; the ?FG PRESENT? error message results. The user must
abort the foreground job and unload it before using /O.

4.2.11 The Version Switch

The Version switch (/V) outputs a version number message (representing
the version of PIP in use) to the terminal using the form:

    PIP V02-XX

The rest of the command line, if any, is ignored.

4.2.12 Bad Block Scan (/K)

The bad block switch (/K) scans the specified device and types the
absolute block numbers of those blocks on the device which return
hardware errors. The block numbers typed are octal; the first block
on a device is 0(8). Note that if no errors occur, nothing will be
output. A complete scan of a disk pack takes several minutes.

Example:

    *RK2:/K
    BLOCK 149 IS BAD

    *RK1:/K
    *

    Scan disk drive 2 for bad blocks.
    Scan drive 0. No blocks are bad.

4.2.12.1 Recovery from Bad Blocks

As a disk ages, the recording surface wears. Eventually unrecoverable
I/O errors occur during attempts to read or write a bad disk block.
PIP protects against usage of bad disk areas by ignoring files with a
.BAD extension (unless the /Y switch is used). Once a bad block is
uncovered in an I/O operation, it can be located using the /K switch
and a .BAD file can be created which encompasses the bad block.

When a hardware I/O error is detected, the recovery procedure is as
follows:

1. Use the PIP /K switch to scan the device and print on the
terminal the absolute block numbers (in octal) of the bad
blocks. For example:

    R PIP
    *RK1:/K
    BLOCK 7723 IS BAD
    *

4-21

January 1976
Peripheral Interchange Program

2. Obtain an extended directory with the /W switch, showing the starting block numbers of all the files on the disk.

3. If a bad block occurs in a file with valuable information, copy the file to another file using the /G switch. In most cases, only 1 bit (character) of the file is affected.

4. If the file is small, it can then be renamed with a .BAD extension to prevent further use of that disk area.

5. If the file is large or the bad block occurs in an empty area, a 1-block .BAD file can be created using the /T switch as follows:

   a. Delete the bad file (if any).

   b. If the bad block is at block n of the free area, create a file of length n-1 with the /T switch. Remember that there must be no spaces larger than n-1 blocks before the desired one (refer to Section 4.2.4). Also note that the block numbers printed in the /K and /W operations are octal, while the argument to the /T operation is decimal.

   c. Create a 1-block .BAD file with the /T switch to cover the bad block.

   d. Delete any temporary files created during the operation.

For example, assume the extended directory is:

```
* * 
NEWSC,BAT  8 11*SEP=74  6203
RTTEMP,BAT 27 11*SEP=74  6213
PIP  .MAC  150 12*SEP=74  6246
< UNUSED > 154
VERIFY  .SAV  3  6726
< UNUSED > 300
PIP  .OBJ  15 12*SEP=74  7405
MKPIP  .CTL  1 12*SEP=74  7424
MKV2RK  .CTL  4 12*SEP=74  7425
VTLIB  .OBJ  16 12*SEP=74  7431
< UNUSED > 150
A  4 12*SEP=74  7671
PIP   .LST  300 3*SEP=74  7675  Block 7723 (octal) of PIP.LST is bad.
* *
```

and a bad block is detected at block 7723 (octal) of the file PIP.LST. To recover, make a copy, ignoring the error, and delete the bad file:

*RK1:PIPR.LST=RK1:PIPR.LST/G
*RK1:PIPR.LST/D

The directory now reads:

```
* * 
NEWSC,BAT  8 11*SEP=74  6203
RTTEMP,BAT 27 11*SEP=74  6213
PIP  .MAC  150 12*SEP=74  6246
* *
```
Peripheral Interchange Program

< UNUSED > 154
VERIFY_SAV 3 6726
PIPA_LST 300 18=SEP=74 6731
PIP_OBJ 15 12=SEP=74 7405
MKPIP_CTL 1 12=SEP=74 7424
MKV2RK_CTL 4 12=SEP=74 7425
VTLIB_OBJ 10 12=SEP=74 7431
< UNUSED > 150
A 4 12=SEP=74 7671

An unused area following A contains block 7723 (octal), which is bad. Continuing in PIP:

*RK1:TEMP.002[154]=/T
*RK1:TEMP.003[150]=/T
*RK1:TEMP.004[22]=/T

This fills the unused areas with temporary files. Specifying TEMP.004 with a length of 22 blocks makes the file just long enough to precede the bad block (i.e., 7675 (octal) and 22 (decimal) equal 7723, which would be the starting block number of the next file created). The directory now contains:

NEWSRC.BAT 8 11=SEP=74 6203
RTTEMP.BAT 27 11=SEP=74 6213
PIP_MAC 150 12=SEP=74 6248
TEMP.002 154 16=SEP=74 6474
VERIFY_SAV 3 6726
PIPA_LST 300 18=SEP=74 6731
PIP_OBJ 15 12=SEP=74 7405
MKPIP_CTL 1 12=SEP=74 7424
MKV2RK_CTL 4 12=SEP=74 7425
VTLIB_OBJ 10 12=SEP=74 7431
TEMP.003 150 18=SEP=74 7443
A 4 12=SEP=74 7671

Continuing with PIP:

*RK1:FILE.BAD[1]/Y/T Create a bad file.

The directory now contains:

NEWSRC.BAT 8 11=SEP=74 6203
RTTEMP.BAT 27 11=SEP=74 6213
PIP_MAC 150 12=SEP=74 6248
TEMP.002 154 16=SEP=74 6474
VERIFY_SAV 3 6726
PIPA_LST 300 18=SEP=74 6731
PIP_OBJ 15 12=SEP=74 7405
MKPIP_CTL 1 12=SEP=74 7424
MKV2RK_CTL 4 12=SEP=74 7425
VTLIB_OBJ 10 12=SEP=74 7431
TEMP.003 150 18=SEP=74 7443
A 4 12=SEP=74 7671
Peripheral Interchange Program

TEMP  004  22 18-SEP-74  7675
FILE  BAD  1 18-SEP-74  7723

Bad block is here.

Next delete all temporary files and rename PIPA.LST to PIP.LST. The final directory now contains:

\[
\begin{align*}
\text{NEWSRC} & .BAT & 8 & 11-SEP-74 & 6203 \\
\text{RTTEMP} & .BAT & 27 & 11-SEP-74 & 6213 \\
\text{PIP} & .MAC & 150 & 12-SEP-74 & 6246 \\
< \text{UNUSED} > & 154 \\
\text{VERIFY} & .SAV & 3 & & 6726 \\
\text{PIP} & .LST & 300 & 18-SEP-74 & 6731 \\
\text{PIP} & .OBJ & 15 & 12-SEP-74 & 7405 \\
\text{MKPIP} & .CTL & 1 & 12-SEP-74 & 7424 \\
\text{MKV2R} & .CTL & 4 & 12-SEP-74 & 7425 \\
\text{VTLIB} & .OBJ & 10 & 12-SEP-74 & 7431 \\
< \text{UNUSED} > & 150 \\
\text{A} & & 4 & 12-SEP-74 & 7671 \\
< \text{UNUSED} > & 22 \\
\text{FILE} & .BAD & 1 & 18-SEP-74 & 7723
\end{align*}
\]

Disks with many bad blocks can often be reused by reformatting them. First copy all desired files, since reformatting destroys all information contained on a volume.

4.3 PIP ERROR MESSAGES

The following error messages are output on the terminal when PIP is used incorrectly:

<table>
<thead>
<tr>
<th>Errors</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?BAD BOOT?</td>
<td>A boot switch was specified on an illegal device.</td>
</tr>
<tr>
<td>?BOOT COPY?</td>
<td>An error occurred during an attempt to write bootstrap with /U switch.</td>
</tr>
<tr>
<td>?CHK SUM?</td>
<td>A checksum error occurred in a formatted binary transfer.</td>
</tr>
<tr>
<td>?COR OVR?</td>
<td>Memory overflow--too many devices and/or file specifications (usually <em>.</em> operations) and no room for buffers.</td>
</tr>
<tr>
<td>?ER RD DIR?</td>
<td>Unrecoverable error reading directory. Check volume for off-line or write-locked condition and try the operation again.</td>
</tr>
<tr>
<td>?EXT NEG?</td>
<td>A /T command attempted to make file smaller.</td>
</tr>
<tr>
<td>?FG PRESENT?</td>
<td>An attempt was made to use /O or /S while a foreground job was still in memory. Unload it if it is no longer desired.</td>
</tr>
<tr>
<td>?FIL NOT FND?</td>
<td>File not found during a delete, copy, or rename operation, or no input files with the expected name or extension were found during a <em>.</em> expansion.</td>
</tr>
</tbody>
</table>
Peripheral Interchange Program

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ILL CMD?</td>
<td>The command specified was not syntactically correct; a device name is missing which should be specified, a switch argument is too large, a filename is specified where one is inappropriate, or a nonfile-structured device is specified for a file-structured operation.</td>
</tr>
<tr>
<td>?ILL DEV?</td>
<td>Illegal or nonexistent device.</td>
</tr>
<tr>
<td>?ILL DIR?</td>
<td>The device did not contain a properly initialized directory structure (EOT file on magtape and cassette; empty file directory on other devices). Use /Z.</td>
</tr>
<tr>
<td>?ILL REN?</td>
<td>Illegal rename operation. Usually caused by different device names on the input and output sides of the command string.</td>
</tr>
<tr>
<td>?ILL SWT?</td>
<td>Illegal switch or switch combination.</td>
</tr>
<tr>
<td>?IN ER?</td>
<td>Unrecoverable error reading file. Try again (this error is ignored during /G operation).</td>
</tr>
<tr>
<td>?OUT ER?</td>
<td>Unrecoverable error writing file. Perhaps a hardware or checksum error; try recopying file. Also may be caused by an attempt to compress a larger device to a smaller one or by not enough room when creating a file. The system takes the largest space available and divides it in half before attempting to insert the file. Try the [n] construction or /X switch.</td>
</tr>
<tr>
<td>?OUT FIL?</td>
<td>Illegal output file specification or missing output file.</td>
</tr>
<tr>
<td>?ROOM?</td>
<td>Insufficient space following file specified with a /T switch.</td>
</tr>
</tbody>
</table>

The following warning messages are output by PIP:

**CTn: PUSH REWIND OR MOUNT NEW VOLUME**

A new cassette must be mounted on drive n to allow continuation of an I/O operation. The operation is continued automatically as soon as the new cassette is mounted.

**?NO .SYS/.BAD ACTION?**

The /Y switch was not included with a command specified on a .SYS or .BAD file. The command is executed for all but the .SYS and .BAD files. A *.* transfer is most likely to cause this message.

**?REBOOT?**

.SYS files have been transferred, renamed, compressed or deleted from the system device. It may be necessary to reboot the system.

**NOTE**

The message is typed immediately after execution of the relevant command has begun, but the actual reboot operation must not be performed until PIP returns with the prompting asterisk for the next command. If the system is halted and rebooted before the prompting asterisk returns, disk information may be lost.

4-25 January 1976
Peripheral Interchange Program

If any of the .SYS files in use by the current system (MONITR.SYS and handler files) have been physically moved on the system device, it is necessary to reboot the system immediately. If not, this message can be ignored. If the cause of the message was a /S operation, the system need be rebooted only if there was an empty space before any of the .SYS files or if the /N:n switch was used to increase the number of directory segments. The need to reboot can be permanently avoided by placing all .SYS files at the beginning of the system device, then avoiding their involvements in PIP operations by not using the /Y switch.

dev:/Z ARE YOU SURE?

Confirmation must be given by the user before a device can be zeroed.
CHAPTER 5
MACRO ASSEMBLER

MACRO is a 2-pass macro assembler requiring an RT-ll system configuration (or background partition) of 12K or more. Macros are instructions in a source or command language which are equivalent to a specified sequence of machine instructions or commands. Users with minimum memory configurations must use ASEMGL and EXPAND and should read this chapter and Chapters 10 and 11 before assembling any programs. (The macro features not supported by ASEMGL are indicated in this chapter; many of the features not available in ASEMGL are supported by EXPAND.)

Some notable features of MACRO are:

1. Program control of assembly functions
2. Device and file name specifications for input and output files
3. Error listing on command output device
4. Alphabetized, formatted symbol table listing
5. Relocatable object modules
6. Global symbols declaration for linking among object modules
7. Conditional assembly directives
8. Program sectioning directives
9. User defined macros
10. Comprehensive set of system macros
11. Extensive listing control, including cross reference listing

Operating instructions for the MACRO assembler appear in Section 5.7.
MACRO Assembler

5.1 SOURCE PROGRAM FORMAT

A source program is composed of a sequence of source lines; each source line contains a single assembly language statement followed by a statement terminator. A terminator may be either a line feed character (which increments the line count by 1) or a form feed character (which resets the line count and increments the page count by 1).

NOTE

EDIT automatically appends a line feed to every carriage return encountered in a source program. For listing format, MACRO automatically inserts a carriage return before any line feed or form feed not already preceded by one.

An assembly language line can contain up to 132(decimal) characters (exclusive of the statement terminator). Beyond this limit, excess characters are ignored and generate an error flag.

5.1.1 Statement Format

A statement can contain up to four fields which are identified by order of appearance and by specified terminating characters. The general format of a MACRO assembly language statement is:

```
label:   operator operand(s) ;comments
```

The label and comment fields are optional. The operator and operand fields are interdependent; either may be omitted depending upon the contents of the other.

The assembler interprets and processes these statements one by one, generating one or more binary instructions or data words or performing an assembly process. A statement contains one of these fields and may contain all four types. Blank lines are legal.

Some statements have one operand, for example:

```
CLR   R0
```

while others have two:

```
MOV   #344,R2
```

An assembly language statement must be complete on one source line. No continuation lines are allowed. (If a continuation is attempted with a line feed, the assembler interprets this as the statement terminator.)

MACRO source statements may be formatted with EDIT so that use of the TAB character causes the statement fields to be aligned. For example:
MACRO Assembler

<table>
<thead>
<tr>
<th>Label</th>
<th>Operator</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK:</td>
<td>BIT</td>
<td>#1, RO</td>
<td>; IS NUMBER ODD?</td>
</tr>
<tr>
<td></td>
<td>BBQ</td>
<td>EVEN</td>
<td>; NO, IT'S EVEN</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>#1, ODDFLAGS</td>
<td>; ELSE SET FLAG</td>
</tr>
<tr>
<td>EVEN:</td>
<td>RTS</td>
<td>PC</td>
<td>; RETURN</td>
</tr>
</tbody>
</table>

5.1.1.1 Label Field - A label is a user-defined symbol that is unique within the first six characters and is assigned the value of the current location counter and entered into the user-defined symbol table. The value of the label may be either absolute (fixed in memory independently of the position of the program) or relocatable (not fixed in memory), depending on whether the location counter value (see Section 5.2.6) is currently absolute or relocatable.

A label is a symbolic means of referring to a specific location within a program. If present, a label always occurs first in a statement and must be terminated by a colon. For example, if the current location is 100(octal), the statement:

```
ABCD: MOV A, B
```

assigns the value 100(octal) to the label ABCD. Subsequent reference to ABCD references location 100(octal). In this example if the location counter was declared relocatable within the section, the final value of ABCD would be 100(octal) plus a value assigned by LINK when it relocates the code, called the relocation constant. (The final value of ABCD would therefore not be known until link-time. This is discussed later in this chapter and in Chapter 6.)

More than one label may appear within a single label field, in which case each label in the field is assigned the same value. For example, if the current location counter is 100(octal), the multiple labels in the statement:

```
ABC: ERREX: MASK: MOV A, B
```

cause each of the three labels—ABC, ERREX, and MASK—to be equated to the value 100(octal).

A symbol used as a label may not be redefined within the user program. An attempt to redefine a label results in an error flag in the assembly listing.

5.1.1.2 Operator Field - An operator field follows the label field in a statement and may contain a macro call, an instruction mnemonic, or an assembler directive. The operator may be preceded by zero, one or more labels and may be followed by one or more operands and/or a comment. Leading and trailing spaces and tabs are ignored.

When the operator is a macro call, the assembler inserts the appropriate code to expand the macro. When the operator is an instruction mnemonic, it specifies the instruction to be generated and the action to be performed on any operand(s) which follow. When the operator is an assembler directive, it specifies a certain function or action to be performed during assembly.
MACRO Assembler

An operator is legally terminated by a space, tab, or any non-alphanumeric character (symbol component).

Consider the following examples:

```
MOV A,B  (space terminates the operator MOV)
MOV#A,B  (# terminates the operator MOV)
```

When the statement line does not contain an operand or comment, the operator is terminated by a carriage return followed by a line feed or form feed character.

A blank operator field is interpreted as a .WORD assembler directive (See Section 5.5.3.2).

5.1.1.3 Operand Field - An operand is that part of a statement which is manipulated by the operator. Operands may be expressions, numbers, or symbolic or macro arguments (within the context of the operation). When multiple operands appear within a statement, each is separated from the next by one of the following characters: comma, tab, space, or paired angle brackets around one or more operands (see Section 5.2.1.1). Multiple delimiters separating operands are not legal (with the exception of spaces and tabs---any combination of spaces and/or tabs represents a single delimiter). An operand may be preceded by an operator, a label or another operand and followed by a comment.

The operand field is terminated by a semicolon when followed by a comment, or by a statement terminator when the operand completes the statement. For example:

```
LABEL: MOV A,B ;COMMENT
```

The space between MOV and A terminates the operator field and begins the operand field; a comma separates the operands A and B; a semicolon terminates the operand field and begins the comment field.

5.1.1.4 Comment Field - The comment field is optional and may contain any ASCII characters except null, rubout, carriage return, line feed, vertical tab or form feed. All other characters, even special characters with defined usage, are ignored by the assembler when appearing in the comment field.

The comment field may be preceded by one, any, none or all of the other three field types. Comments must begin with the semicolon character and end with a statement terminator.
MACRO Assembler

Comments do not affect assembly processing or program execution, but are useful in source listings for later analysis, debugging, or documentation purposes.

5.1.2 Format Control

Horizontal or line formatting of the source program is controlled by the space and tab characters. These characters have no effect on the assembly process unless they are embedded within a symbol, number, or ASCII text; or unless they are used as the operator field terminator. Thus, these characters can be used to provide an orderly source program. A statement can be written:

```
LABEL: MOV(SP)+,TAG; POP VALUE OFF STACK
```
or, using formatting characters, it can be written:

```
LABEL: MOV (SP)+,TAG; POP VALUE OFF STACK
```

which is easier to read in the context of a source program listing.

Vertical formatting, i.e., page size, is controlled by the form feed character. A page of n lines is created by inserting a form feed (CTRL FORM) after the nth line. (See also Section 5.5.1.6 for a description of page formatting with respect to macros and Section 5.5.1.2 for a description of assembly listing output.)

5.2 SYMBOLS AND EXPRESSIONS

This section describes the various components of legal MACRO expressions: the assembler character set, symbol construction, numbers, operators, terms and expressions.

5.2.1 Character Set

The following characters are legal in MACRO source programs:

1. The letters A through Z. Both upper- and lower-case letters are acceptable, although, upon input, lower-case letters are converted to upper-case letters. Lower-case letters can only be output by sending their ASCII values to the output device. This conversion is not true for .ASCII, .ASCIZ, ' (single quote) or " (double quote) statements if .ENABLE LC is in effect.

2. The digits 0 through 9.

3. The characters . (period or dot) and $ (dollar sign) which are reserved for use in system program symbols (with the exception of local symbols; see Section 5.2.5).

4. The following special characters:
MACRO Assembler

<table>
<thead>
<tr>
<th>Character</th>
<th>Designation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>carriage return</td>
<td>line feed</td>
<td>formatting character</td>
</tr>
<tr>
<td>form feed</td>
<td></td>
<td>source statement terminators</td>
</tr>
<tr>
<td>vertical tab</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>colon</td>
<td>label terminator</td>
</tr>
<tr>
<td>=</td>
<td>equal sign</td>
<td>direct assignment indicator</td>
</tr>
<tr>
<td>%</td>
<td>percent sign</td>
<td>register term indicator</td>
</tr>
<tr>
<td>tab</td>
<td>space</td>
<td>item or field terminator</td>
</tr>
<tr>
<td>#</td>
<td>number sign</td>
<td>immediate expression indicator</td>
</tr>
<tr>
<td>@</td>
<td>at sign</td>
<td>deferred addressing indicator</td>
</tr>
<tr>
<td>(</td>
<td>left parenthesis</td>
<td>initial register indicator</td>
</tr>
<tr>
<td>)</td>
<td>right parenthesis</td>
<td>terminal register indicator</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
<td>operand field separator</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
<td>comment field indicator</td>
</tr>
<tr>
<td>&lt;</td>
<td>left angle bracket</td>
<td>initial argument or expression indicator</td>
</tr>
<tr>
<td>&gt;</td>
<td>right angle bracket</td>
<td>terminal argument or expression indicator</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
<td>arithmetic addition operator or auto increment indicator</td>
</tr>
<tr>
<td>-</td>
<td>minus sign</td>
<td>arithmetic subtraction operator or auto decrement indicator</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
<td>arithmetic multiplication operator</td>
</tr>
<tr>
<td>/</td>
<td>slash</td>
<td>arithmetic division operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>ampersand</td>
<td>logical AND operator</td>
</tr>
<tr>
<td>!</td>
<td>exclamation</td>
<td>logical inclusive OR operator</td>
</tr>
<tr>
<td>'</td>
<td>double quote</td>
<td>double ASCII character indicator</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
<td>single ASCII character indicator</td>
</tr>
<tr>
<td>↑</td>
<td>uparrow</td>
<td>universal unary operator, argument indicator</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
<td>macro numeric argument indicator (not available in ASEMBL)</td>
</tr>
</tbody>
</table>

5.2.1.1 Separating and Delimiting Characters - Reference is made in the remainder of the chapter to legal separating characters and macro argument delimiters. These terms are defined in Table 5-1 and following.

Table 5-1
Legal Separating Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Definition</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>one or more spaces and/or tabs</td>
<td>A space is a legal separator only for argument operands. Spaces within expressions are ignored.</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
<td>A comma is a legal separator for both expressions and argument operands.</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>paired angle brackets</td>
<td>Paired angle brackets are used to enclose an argument,</td>
</tr>
</tbody>
</table>

(Continued on next page)
Table 5-1 (cont.)
Legal Separating Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Definition</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>...\</td>
<td>Up arrow construction where the up arrow character is followed by an argument bracketed by any paired printing characters.</td>
<td>particularly when that argument contains separating characters. Paired angle brackets may be used anywhere in a program to enclose an expression for treatment as a term. (The angle bracket construction should be used when the argument contains unary operators.)</td>
</tr>
</tbody>
</table>

Macro arguments may appear in several forms to allow for special cases. The rules to observe when separating arguments are:

1. If an argument string contains only non-separating characters (those not defined in Table 5-1) and no spaces, then it may appear in the argument list separated, if necessary, from the other arguments by commas.

2. If an argument string contains separating characters or spaces, but does not contain the characters < or > (left or right angle brackets), then the argument may appear enclosed in paired angle brackets (e.g., <argument string>). The paired angle brackets are removed before the argument string is used. Successive pairs of angle brackets may be used to enclose an argument; only the outermost pair is removed.

3. If an argument string contains separating characters or spaces (possibly including the left or right angle bracket characters), then it may appear in the following form: \argument string\ where the backslashes may be replaced by any character not appearing in the argument string. The uparrow and backslashes (or other character) are removed before the argument string is substituted into the text.

Note that regardless of the method used to specify an argument, it must be separated from any other arguments by commas.

5.2.1.2 Illegal Characters - A character can be illegal in one of two ways:

1. A character which is not recognized as an element of the MACRO character set is always an illegal character and causes immediate termination of the current line at that point, plus the output of an error flag in the assembly listing. For example:

   ```
   LABEL=*A:  MOV A,B
   ```

   Since the backarrow is not a recognized character, the entire line is treated as a:

   ```
   .WORD LABEL
   ```

   statement and is flagged in the listing.
MACRO Assembler

2. A legal MACRO character may be illegal in context. Such a character generates a Q error on the assembly listing.

5.2.1.3 Operator Characters - Under MACRO, legal unary operators (operators applying to only one operand) are as follows:

<table>
<thead>
<tr>
<th>Unary Operator</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>plus sign</td>
<td>+A (positive value of A, equivalent to A)</td>
</tr>
<tr>
<td>-</td>
<td>minus sign</td>
<td>-A (negative, 2's complement, value of A)</td>
</tr>
<tr>
<td>†</td>
<td>uparrow, universal</td>
<td>†F3.0 (interprets 3.0 as a 1-word floating-point number)</td>
</tr>
<tr>
<td></td>
<td>unary operator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(this usage is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>described in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>greater detail</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in Sections</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.5.4.2 and 5.5.6.2.</td>
<td></td>
</tr>
<tr>
<td>†</td>
<td>C24</td>
<td>†C24 (interprets the one's complement of the binary representation of 24(8))</td>
</tr>
<tr>
<td></td>
<td>D127</td>
<td>†D127 (interprets 127 as a decimal number)</td>
</tr>
<tr>
<td></td>
<td>034</td>
<td>†034 (interprets 34 as an octal number)</td>
</tr>
<tr>
<td></td>
<td>Bl1000111</td>
<td>†Bl1000111 (interprets 11000111 as a binary value)</td>
</tr>
</tbody>
</table>

The unary operators described above can be used adjacent to each other in a term. For example:

†C†012
-†05

Legal binary operators under MACRO are as follows:

<table>
<thead>
<tr>
<th>Binary Operator</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>A+B</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>A-B</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>A*B (16-bit product returned)</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>A/B (16-bit quotient returned)</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical AND</td>
<td>A&amp;B</td>
</tr>
<tr>
<td>l</td>
<td>logical inclusive OR</td>
<td>AIB</td>
</tr>
</tbody>
</table>

All binary operators have the same priority. Division and multiplication are signed operations. Items can be grouped for evaluation within an expression by enclosure in angle brackets. Terms in angle brackets are evaluated first, and remaining operations are performed left to right. For example:

WORD 1+2*3 115 OCTAL
WURD 1<2*3> 115 7 OCTAL
MACRO Assembler

5.2.2 Symbols

There are three types of symbols: permanent, user-defined and macro. MACRO maintains three types of symbol tables: the Permanent Symbol Table (PST), the User Symbol Table (UST) and the Macro Symbol Table (MST). The PST contains all the permanent symbols and is part of the MACRO Assembler load module. The UST and MST are constructed as the source program is assembled; user-defined symbols are added to the table as they are encountered.

5.2.2.1 Permanent Symbols - Permanent symbols consist of the instruction mnemonics (Appendix C) and assembler directives and macro directives (sections 5.5 and 5.6, Appendix C). These symbols are a permanent part of the assembler and need not be defined before being used in the source program.

5.2.2.2 User-Defined and Macro Symbols - User-defined symbols are those used as labels or defined by direct assignment (Section 5.2.3). These symbols are added to the User Symbol Table as they are encountered during the first pass of the assembly. Macro symbols are those symbols used as macro names in the operator field (Section 5.6.1). These symbols are added to the Macro Symbol Table as they are encountered during the assembly.

User-defined and macro symbols can be composed of alphanumerical characters, dollar signs, and periods only; any other character is illegal.

The $ and . characters are reserved for system software symbols (for example, the system macro symbol .READ); it is recommended that $ and . not be inserted in user-defined or macro symbols.

The following rules apply to the creation of user-defined and macro symbols:

1. The first character must not be a number (except in the case of local symbols, see Section 5.2.5).
2. Each symbol must be unique within the first six characters.
3. A symbol can be written with more than six legal characters, but the seventh and subsequent characters are only checked for legality, and are not otherwise recognized by the assembler.
4. Spaces, tabs, and illegal characters must not be embedded within a symbol.

The value of a symbol depends upon its use in the program. A symbol in the operator field may be any one of the three symbol types. To determine the value of the symbol, the assembler searches the three symbol tables in the following order:

1. Macro Symbol Table
2. Permanent Symbol Table
3. User-Defined Symbol Table
MACRO Assembler

A symbol found in the operand field is sought in the:

1. User-Defined Symbol Table
2. Permanent Symbol Table

in that order. The assembler never expects to find a macro name in an operand field.

These search orders allow redefinition of Permanent Symbol Table entries as user-defined or macro symbols. The same name can be assigned to both a macro and a label.

User-defined symbols are either internal or external (global). All user-defined symbols are internal unless explicitly defined as being global with the .GLOBAL directive (see Section 5.5.10).

Global symbols provide links between object modules. A global symbol defined as a label is generally called an entry point (to a section of code). Such symbols are referenced from other object modules to transfer control throughout the load module (which may be composed of a number of object modules).

Since MACRO provides program sectioning capabilities (Section 5.5.9), two types of internal symbols must be considered:

1. Symbols that belong to the current program section, and
2. Symbols that belong to other program sections.

In both cases, the symbol must be defined within the current assembly; the significance of the distinction is critical in evaluating expressions involving type (2) above (see Section 5.2.9).

5.2.3 Direct Assignment

A direct assignment statement associates a symbol with a value. When a direct assignment statement defines a symbol for the first time, that symbol is entered into the user symbol table and the specified value is associated with it. A symbol may be redefined by assigning a new value to a previously defined symbol. The latest assigned value replaces any previous value assigned to a symbol.

The general format for a direct assignment statement is:

\[
\text{symbol} = \text{expression}
\]

Symbols take on the relocatable or absolute attribute of their defining expression. However, if the defining expression is global, the symbol is not global unless explicitly defined as such in a .GLOBAL directive. For example:

\[
\begin{align*}
A &= 1 & & \text{THE SYMBOL A IS EQUATED TO THE VALUE 1} \\
B &= A = 1 & & \text{THE SYMBOL B IS EQUATED TO THE VALUE OF THE EXPRESSION} \\
C1 & = D &= 3 & & \text{THE SYMBOL D IS EQUATED TO 3}
\end{align*}
\]
MACRO Assembler

E1:    MOV     #1,ABLE

LABELS C AND E ARE EQUATED TO THE
LOCATION OF THE MOV COMMAND

The following conventions apply to direct assignment statements:

1. An equal sign (=) must separate the symbol from the
   expression defining the symbol value.

2. A direct assignment statement is usually placed in the
   operator field and may be preceded by a label and followed by
   a comment.

NOTE

If the program jumps to or references
the label of a direct assignment
statement, it is actually referencing
the following instruction statement.
For example:

    .F,+1000
    C1     D=3
    E1     MOV #0,ABLE

    ...

    JMP C

This code causes a jump to the label E.

3. Only one symbol can be defined by any one direct assignment
   statement.

4. Only one level of forward referencing is allowed. That is,
   the following arrangement is illegal:

      X = Y
      Y = Z
      Z = 1

X and Y are both undefined throughout pass 1, X is undefined
throughout pass 2 and causes an error flag in the assembly
listing.

5.2.4 Register Symbols

The eight general registers of the PDP-11 are numbered 0 through 7 and
can be expressed in the source program as:

%0
%1
...
...
%7
MACRO Assembler

The digit indicating the specific register can be replaced by any legal term which can be evaluated during the first assembly pass.

It is recommended that the programmer create and use symbolic names for all register references. A register symbol may be defined in a direct assignment statement among the first statements in the program. A register symbol cannot be defined after the statement which uses it. The defining expression of a register symbol must be absolute. For example:

```
  R0=%0
  R1=%1
  R2=%2
  R3=%3
  R4=%4
  R5=%5
  SP=%6
  PC=%7
```

The symbolic names assigned to the registers in the example above are the conventional names used in all PDP-11 system programs. Since these names are fairly mnemonic, it is suggested the user follow this convention. Registers 6 and 7 are given special names because of their special functions, while registers 0 through 5 are given similar names to denote their status as general purpose registers.

All register symbols must be defined before they are referenced. A forward reference to a register symbol causes phase errors in an assembly.

The % character can be used with any term or expression to specify a register. (A register expression less than 0 or greater than 7 is flagged with an R error code.) For example:

```
  CLR %3+1
```

is equivalent to:

```
  CLR %4
```

and clears the contents of register 4, while:

```
  CLR 4
```

clears the contents of memory address 4.

In certain cases a register can be referenced without the use of a register symbol or register expression; these cases are recognized through the context of the statement. An example is shown below:

```
  J&R 5, SUBR  
  IFIRST OPERAND FIELD MUST ALWAYS  
  IBE A REGISTER
```

5.2.5 Local Symbols

Local symbols are specially formatted symbols used as labels within a given range.
MACRO Assembler

Local symbols provide a convenient means of generating labels to be referenced by branch instructions. Use of local symbols reduces the possibility of multiply-defined symbols within a user program and separates entry point symbols from local references. Local symbols, then, are not referenced from other object modules or even from outside their local symbol block.

Local symbols are of the form n$, where n is a decimal integer from 1 to 127, inclusive, and can only be used on word boundaries. Local symbols include:

- 1$
- 27$
- 59$
- 104$

Within a local symbol block, local symbols can be defined and referenced. However, a local symbol cannot be referenced outside the block in which it is defined. There is no conflict with labels of the same name in other local symbol blocks.

Local symbols 64$ through 127$ can be generated automatically as a feature of the macro processor (see Section 5.6.3.5 for further details). When using local symbols the user is advised to first use the range from 1$ to 63$.

A local symbol block is delimited in one of the following ways:

1. The range of a single local symbol block can consist of those statements between two normally constructed symbolic labels. (Note that a statement of the form:

   \[
   \text{LABEL=.}
   \]

   is a direct assignment, does not create a label in the strict sense, and does not delimit a local range.)

2. The range of a local symbol block is terminated upon encountering a .CSECT directive.

3. The range of a single local symbol block can be delimited with .ENABL LSB and the first symbolic label or .CSECT directive following the .DSABL LSB directives. The default for LSB is off.

For examples of local symbols and local symbol blocks, see Figure 5-1.

The maximum offset of a local symbol from the base of its local symbol block is 128 decimal words. Symbols beyond this range are flagged with an A error code.
MACRO Assembler

5.2.6 Assembly Location Counter

The period (.) is the symbol for the assembly location counter. When used in the operand field of an instruction, it represents the address of the first word of the instruction. When used in the operand field of an assembler directive, it represents the address of the current byte or word. For example:

```
A1  MOV  #.,R0  ;, REFERS TO LOCATION A,
    ;I.E., THE ADDRESS OF THE
    ;MOV INSTRUCTION
```

(# is explained in Section 5.4.9).

At the beginning of each assembly pass, the assembler clears the location counter. Normally, consecutive memory locations are assigned to each byte of object data generated. However, the location where the object data is stored may be changed by a direct assignment statement altering the location counter:

```
. = expression
```

The expression defining the location counter must not contain forward references or symbols that vary from one pass to another. If an expression is assigned to the current location counter in a relocatable CSECT, an error flag is generated. (The construction `.+expression must be used.)

Similar to other symbols, the location counter symbol has a mode associated with it, either absolute or relocatable; the mode cannot be external. The existing mode of the location counter cannot be changed by using a defining expression of a different mode.

5-14
<table>
<thead>
<tr>
<th>Line</th>
<th>Octal Expansion</th>
<th>Source Code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>.MCALL .REGDEF,,V2,,</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>.REGDEF</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>.V2,,</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>.SETTL</td>
<td>SECTOR INITIALIZATION</td>
</tr>
<tr>
<td>6</td>
<td>000000</td>
<td>.CSECT IMPURE</td>
<td>PURE STORAGE AREA</td>
</tr>
<tr>
<td>7</td>
<td>000000</td>
<td>.CSECT IMPAS</td>
<td>CLEARED EACH PASS</td>
</tr>
<tr>
<td>8</td>
<td>000000</td>
<td>.CSECT IMPLIN</td>
<td>CLEARED EACH LINE</td>
</tr>
<tr>
<td>9</td>
<td>000000</td>
<td>.CSECT XCTPRG</td>
<td>PROGRAM</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>INITIALIZATION</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>MOV #IMPURE,R0</td>
<td>CLEAR IMPURE AREA</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>(R0)+</td>
<td>CLEAR IMPURE PART</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>CMP #IMPTOP,R0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5-1
Assembly Source Listing of MACRO Code Showing Local Symbol Blocks
MACRO Assembler

The mode of the location counter symbol can be changed by the use of the .AECT or .CSECT directive as explained in Section 5.5.9.

Examples:

```
.AECT
=500

FIRSTI MOV +10,COUNT
THE LABEL FIRST HAS THE VALUE
1500(8)
+10 EQUALS 510(8), THE
CONTENTS OF THE LOCATION
1510(8) WILL BE DEPOSITED
IN LOCATION COUNT.

COUNT: .WURD 0

.=520

THE ASSEMBLY LOCATION COUNTER
NOW HAS A VALUE OF
ABSOLUTE 520(8).

SECOND: MOV ,,INDEX
THE LABEL SECOND HAS THE
VALUE 520(8)
THE CONTENTS OF LOCATION
1520(8), THAT IS, THE BINARY
CODE FOR THE INSTRUCTION
ITSELF WILL BE DEPOSITED IN
LOCATION INDEX.

INDEX: .WURD 0

.CSECT

=+20

THIRD: .WURD 0
THE LABEL THIRD HAS THE
VALUE OF RELOCATABLE 20.
```

Storage area may be reserved by advancing the location counter. For example, if the current value of the location counter is 1000, the direct assignment statement:

```
=+100
```

reserves 100 (octal) bytes of storage space in the program. The next instruction is stored at 1100. (The .BLKW and .BLKB directives can also be used to reserve blocks of storage; see Section 5.5.5.3.)
MACRO Assembler

5.2.7 Numbers

The MACRO Assembler assumes all numbers in the source program are to be interpreted in octal radix unless otherwise specified. The assumed radix can be altered with the .RADIX directive or individual numbers can be treated as being of decimal, binary, or octal radix (see Section 5.5.4.2).

Octal numbers consist of the digits 0 through 7 only. A number not specified as a decimal number and containing an 8 or 9 is flagged with an N error code and treated as a decimal number.

Negative numbers are preceded by a minus sign (the assembler translates them into two's complement form). Positive numbers may be preceded by a plus sign, although this is not required.

A number which is too large to fit into 16 bits (177777<n) is truncated from the left and flagged with a T error code in the assembly listing.

Numbers are always considered absolute quantities (that is, not relocatable).

The single-word floating-point numbers which can be generated with the FF operator (see Section 5.5.6.2) are stored in the following format:

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGN BIT</td>
<td>8-BIT EXPONENT</td>
<td>7-BIT MANTISSA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Refer to PDP-11/45 Processor Handbook for details of the floating-point format.

5.2.8 Terms

A term is a component of an expression. A term may be one of the following:

1. A number whose 16-bit value is used.

2. A symbol that is interpreted according to the following hierarchy:
   a. a period that causes the value of the current location counter to be used
   b. a permanent symbol whose basic value is used and whose arguments (if any) are ignored
   c. user defined symbols
   d. an undefined symbol that is assigned a value of zero and inserted in the user-defined symbol table

5-17
MACRO Assembler

3. An ASCII conversion using either an apostrophe followed by a single ASCII character or a double quote followed by two ASCII characters, which results in a word containing the 7-bit ASCII value of the character(s). (This construction is explained in greater detail in Section 5.5.3.3.)

4. An expression enclosed in angle brackets. Any quantity enclosed in angle brackets is evaluated before the remainder of the expression in which it is found. Angle brackets are used to alter the left to right evaluation of expressions (for example, to differentiate between $\text{A*B+C}$ and $\text{A*(B+C)}$) or to apply a unary operator to an entire expression (e.g., $\langle\langle\text{A+B}\rangle\rangle$).

5.2.9 Expressions

Expressions are combinations of terms that are joined together by binary operators and that reduce to a 16-bit value. The operands of a .BYTE directive are evaluated as word expressions before truncation to the low-order eight bits. Prior to truncation, the high-order byte must be zero or all ones (when the byte value is negative, the sign bit is propagated). The evaluation of an expression includes the evaluation of the mode of the resultant expression—that is, absolute, relocatable or external. Expression modes are defined further below.

Expressions are evaluated left to right with no operator hierarchy rules except that unary operators take precedence over binary operators. A term preceded by a unary operator can be considered as containing that unary operator. (Terms are evaluated, where necessary, before their use in expressions.) Multiple unary operators are valid and are treated as follows:

$$-++A$$

is equivalent to:

$$-\langle+(-A)\rangle$$

The value of an external expression is the value of the absolute part of the expression; e.g., EXTA+A has a value of A. This is modified by the Linker to become EXTA.

Expressions, when evaluated, are either absolute, relocatable, or external. For the programmer writing position independent code, the distinction is important.

1. An expression is absolute if its value is fixed. An expression whose terms are numbers and ASCII conversions has an absolute value. A relocatable expression minus a relocatable term, where both items belong to the same program section, is also absolute.

2. An expression is relocatable if its value is fixed relative to a base address but will have an offset value added when linked. Expressions whose terms contain labels defined in relocatable sections and the assembly location counter (in relocatable sections) have a relocatable value.

3. An expression is external (or global) if its value is only partially defined during assembly and is completed at link
time. An expression whose terms contain a global symbol not defined in the current program is an external expression. External expressions have relocatable values at execution time if the global symbol is defined as being relocatable or absolute if the global symbol is defined as absolute.

An example of the three expression types follows:

```
.AS盆地
  .100
  ABSSYM*,
  .CSECT MAIN
  .GLOBAL EXTVAL
  BEGSYM, .BLK4 .ASCII /ABCD/
  ENDSYM*,
  SIZE = ENDSYM-BEGSYM
  RELEXP = ENDSYM-BEGSYM*
  EXTEXP .WORD EXTVAL+4
  CHARA*4

THE VALUE OF ABSSYM IS NOT RELOCATABLE, BECAUSE WE ARE IN AN ASPECT

START RELOCATABLE PROGRAM SECTION

EXTVAL IS DEFINED ELSEWHERE, ITS VALUE WILL NOT BE KNOWN UNTIL LINK TIME

THE VALUES OF BEGSYM AND ENDSYM ARE RELOCATABLE, BECAUSE THE ADDRESS AT WHICH "MAIN" WILL BE LOADED IS NOT DETERMINED UNTIL LINK TIME

HOWEVER, THE VALUE OF SIZE IS KNOWN (IT IS 12.) AT ASSEMBLY TIME AND IS ABSOLUTE

RELEXP (*,12.) IS RELOCATABLE

THE EXPRESSION "EXTVAL+4" IS EXTERNAL (OR GLOBAL) BECAUSE EXTVAL IS DEFINED IN ANOTHER PROGRAM UNIT.

THE VALUE OF CHARA IS ABSOLUTE
```

5.3 RELOCATION AND LINKING

The output of the MACRO Assembler is an object module which must be processed by LINK before loading and execution (refer to Chapter 6 for details). The Linker essentially fixes (i.e., makes absolute) the values of external or relocatable symbols and turns the object module into a load module.

To enable the Linker to fix the value of an expression, the assembler issues certain directives to the Linker together with required parameters. In the case of relocatable expressions, the Linker adds the base of the associated relocatable section (the location in memory of relocatable 0) to the value of the relocatable expression provided

5-19
MACRO Assembler

by the assembler. In the case of an external expression, the value of the external term in the expression is determined by the Linker (since the external symbol must be defined in one of the other object modules which are being linked together) and adds it to the value of the external expression provided by the assembler.

All words that are to be modified (as described in the previous paragraph) are marked with an apostrophe in the assembly listing. A G in the listing indicates that the value is external, or that a global is added to that value. Thus, the binary text output looks as follows:

```
005065  CLR  EXTERNAL(R5)  /VALUE OF EXTERNAL SYMBOL
0000006
005065  CLR  EXTERNAL+6(R5)  /ASSEMBLED ZERO WILL BE
0000006  MODIFIED BY THE LINKER,
005065  CLR  RELOCATABLE(R5)  /THE ABSOLUTE PORTION OF THE
000040  /EXPRESSION (000006) IS ADDED
       /BY THE LINKER TO THE VALUE OF
       /THE EXTERNAL SYMBOL
       /ASSUMING WE ARE IN A
       /RELOCATABLE SECTION
       /AND THE VALUE OF RELOCATABLE
       /IS RELOCATABLE 40
```

5.4 ADDRESSING MODES

The program counter (PC, register 7 of the eight general registers) always contains the address of the next word to be fetched; i.e., the address of the next instruction to be executed, or the second or third word of the current instruction.

In order to understand how the address modes operate and how they assemble, the action of the program counter must be understood. The key rule is:

Whenever the processor implicitly uses the program counter to fetch a word from memory, the program counter is automatically incremented by two after the fetch.

That is, when an instruction is fetched, the PC is incremented by two so that it is pointing to the next word in memory; if an instruction uses indexing (Sections 5.4.7, 5.4.9 and 5.4.11) the processor uses the program counter to fetch the base from memory. Hence, using the rule above, the PC increments by two, and now points to the next word.

The following conventions are used in this section:

1. Let E be any expression as defined in Section 5.2.

2. Let R be a register expression. This is any expression containing a term preceded by a % character or a symbol previously equated to such a term.

5-20
MACRO Assembler

Examples:

\[ R0 := %0 \quad \text{GENERAL REGISTER 0} \]
\[ R1 := R0 + 1 \quad \text{GENERAL REGISTER 1} \]
\[ R2 := 1 + X1 \quad \text{GENERAL REGISTER 2} \]

3. Let \( ER \) be a register expression or an expression in the range 0 to 7 inclusive.

4. Let \( A \) be any general address specification which produces a 6-bit mode address field as described in Sections 3.1 and 3.2 of the PDP-11 PROCESSOR HANDBOOK (both 11/20 and 11/45 versions).

The addressing specifications, \( A \), can be explained in terms of \( E \), \( R \), and \( ER \) as defined above. Each is illustrated with the single operand instruction \( CLR \) or double operand instruction \( MOV \).

5.4.1 Register Mode

The register contains the operand.

Format for \( A \): \( R \)

Examples:

\[ R0 := %0 \quad \text{DEFINE R0 AS REGISTER 0} \]
\[ CLR \quad R0 \quad \text{CLEAR REGISTER 0} \]

5.4.2 Register Deferred Mode

The register contains the address of the operand.

Format for \( A \): \( @R \) or \( (ER) \)

Examples:

\[ CLR \quad @R1 \quad \text{BOTH INSTRUCTIONS CLEAR} \]
\[ CLR \quad (R1) \quad \text{THE WORD AT THE ADDRESS} \]
\[ \quad \text{CONTAINED IN REGISTER 1} \]

5.4.3 Autoincrement Mode

The contents of the register are incremented immediately after being used as the address of the operand. (See NOTE below.)
MACRO Assembler

Format for A: (ER)+

Examples:

- CLM (R0)+  
  EACH INSTRUCTION CLEARS THE WORD AT THE ADDRESS

- CLM (R0)+3  
  CONTENTS IN THE SPECIFIED REGISTER AND INCREMENTS THAT REGISTER'S CONTENTS BY TWO.

- CLMB (R4)+  
  CLEARS THE BYTE AT THE ADDRESS SPECIFIED BY THE CONTENTS OF R4 AND INCREASES R4 BY ONE.

NOTE

Both JMP and JSR instructions using non-decrement autoincrement mode, autoincrement the register before its use on the PDP-11/20 and 11/05 (but not on the PDP-11/40 or 11/45). In double operand instructions of the addressing form %R,(R)+ or %R,-(R) where the source and destination registers are the same, the source operand is evaluated as the autoincremented or autodecremented value, but the destination register, at the time it is used, still contains the originally intended effective address.

In the following two examples, as executed on the PDP-11/20, R0 originally contains 100.

- MOV R0,(R0)+  
  THE QUANTITY 102 IS MOVED TO LOCATION 100

- MOV R0,-(R0)  
  THE QUANTITY 76 IS MOVED TO LOCATION 76

The use of these forms should be avoided as they are not compatible with the PDP-11/05, 11/40 and 11/45.

A Z error code is printed with each instruction which is not compatible among all members of the PDP-11 family. This is merely a warning code.

5.4.4 Autoincrement Deferred Mode

The register contains the pointer to the address of the operand. The contents of the register are incremented after being used.
MACRO Assembler

Format for A: 0(ER)+

Example: CLR *(R3)+ ICONTENTS OF REGISTER 3 POINT INTO ADDRESS OF WORD TO BE I CLEARED, AND REGISTER 3 IS I THEN INCREMENTED BY TWO

5.4.5 Autodecrement Mode

The contents of the register are decremented before being used as the address of the operand (see NOTE under autoincrement mode).

Format for A: -(ER)

Examples: CLR *(R0) IDECREMENT CONTENTS OF CLR *(R0+3) 10, 3, AND 2 BY TWO CLR *(R2) IBEFORE USING AS ADDRESSES IOF WORDS TO BE CLEARED.

5.4.6 Autodecrement Deferred Mode

The contents of the register are decremented before being used as the pointer to the address of the operand.

Format for A: 0-(ER)

Example: CLR **(R2) IDECREMENT CONTENTS OF IREGISTER 2 BY TWO BEFORE IUSING AS A POINTER ITO ADDRESS OF WORD TO BE ICLEARED.

5.4.7 Index Mode

The value of an expression E is stored as the second or third word of the instruction. The effective address is calculated as the value of E plus the contents of register ER. The value E is called the base.

Format for A: E(ER)

Examples: CLR X+2(R1) IEFFECTIVE ADDRESS IS X+2 PLUS ITHE CONTENTS OF REGISTER 1 CLR ***(R3) IEFFECTIVE ADDRESS IS ***(R3) ITHE CONTENTS OF REGISTER 3.

5.4.8 Index Deferred Mode

An expression plus the contents of a register gives the pointer to the address of the operand.
MACRO Assembler

Format for A: @E(ER)

Example: CLR #14(R4) IF REGISTER 4 HOLDS 100 AND
ILOC 114 HOLDS 2000,
1LOCATION 2000 IS CLEARED.

5.4.9 Immediate Mode

The immediate mode allows the operand itself to be stored as the
second or third word of the instruction. It is assembled as an
autoincrement of register 7, the PC.

Format for A: #E

Examples: MOV #100,R0 ;MOVE AN OCTAL 100 TO
IREGISTER 0
MOV #x,R0 ;MOVE THE VALUE OF THE SYMBOL X TO
IREGISTER 0

The operation of this mode can be explained by the following example.
The statement MOV #100,R3 assembles as two words. These are:

012703
000100

Just before this instruction is fetched and executed, the PC points to
the first word of the instruction. The processor fetches the first
word and increments the PC by two. The source operand mode is 27
(autoincrement the PC). Thus, the PC is used as a pointer to fetch the
operand (the second word of the instruction) before being incremented
by two, to point to the next instruction.

5.4.10 Absolute Mode

Absolute mode is the equivalent of immediate mode deferred. @@E
specifies an absolute address which is stored in the second or third
word of the instruction. Absolute mode is assembled as an
autoincrement deferred of register 7, the PC.

Format for A: @@E

Examples: MOV @@100,R0 ;MOVE THE VALUE OF CONTENTS
1OF LOCATION 100 TO
IREGISTER 0,
CLR @@x ;CLEAR THE CONTENTS OF THE
ILOCATION WHOSE ADDRESS IS X.

5.4.11 Relative Mode

Relative mode is the normal mode for memory references.

5-24
MACRO Assembler

Format for A: E

Examples: CLR 100 } CLEAR LOCATION 100
MOV X,Y } MOV THE CONTENTS OF LOCATION X INTO LOCATION Y.

Relative mode is assembled as index mode, using register 7, the PC, as the index register. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand (as in index mode), but the number which, when added to the PC, becomes the address of the operand. Thus, the base is X-PC, which is called an offset. The operation is explained as follows:

If the statement MOV 100,R3 is assembled at absolute location 20, the assembled code is:

Location 20: 016703
Location 22: 000054

The processor fetches the MOV instruction and adds two to the PC so that it points to location 22. The source operand mode is 67, that is, indexed by the PC. To pick up the base, the processor fetches the word pointed to by the PC and adds two to the PC. The PC now points to location 24. To calculate the address of the source operand, the base is added to the designated register, that is, BASE+PC=54+24=100, the operand address.

Since the assembler considers "." as the address of the first word of the instruction, an equivalent index mode statement would be:

MOV 100-.4(PC),R3

This mode is called relative because the operand address is calculated relative to the current PC. The base is the distance or offset (in bytes) between the operand and the current PC. If the operator and its operand are moved in memory so that the distance between the operator and data remains constant, the instruction will operate correctly anywhere in memory.

5.4.12 Relative Deferred Mode

Relative deferred mode is similar to relative mode, except that the expression, E, is used as the pointer to the address of the operand.

Format for A: E

Example: MOV #X,R6 } MOVE THE CONTENTS OF THE LOCATION WHOSE ADDRESS IS IN #X INTO REGISTER 6

5.4.13 Table of Mode Forms and Codes

Each instruction assembles into at least one word. Operands of the first six forms listed below do not increase the length of an instruction. Each operand in one of the other modes, however, increases the instruction length by one word.

5-25
MACRO Assembler

<table>
<thead>
<tr>
<th>Form</th>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0n</td>
<td>Register mode</td>
</tr>
<tr>
<td>@R or (ER)</td>
<td>1n</td>
<td>Register deferred mode</td>
</tr>
<tr>
<td>(ER)+</td>
<td>2n</td>
<td>Autoincrement mode</td>
</tr>
<tr>
<td>@(ER)+</td>
<td>3n</td>
<td>Autoincrement deferred mode</td>
</tr>
<tr>
<td>-(ER)</td>
<td>4n</td>
<td>Autodecrement mode</td>
</tr>
<tr>
<td>@-(ER)</td>
<td>5n</td>
<td>Autodecrement deferred mode</td>
</tr>
</tbody>
</table>

n represents the register number.

Any of the following forms adds one word to the instruction length:

<table>
<thead>
<tr>
<th>Form</th>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E (ER)</td>
<td>6n</td>
<td>Index mode</td>
</tr>
<tr>
<td>@E(ER)</td>
<td>7n</td>
<td>Index deferred mode</td>
</tr>
<tr>
<td>#E</td>
<td>27</td>
<td>Immediate mode</td>
</tr>
<tr>
<td>@@E</td>
<td>37</td>
<td>Absolute memory reference mode</td>
</tr>
<tr>
<td>E</td>
<td>67</td>
<td>Relative mode</td>
</tr>
<tr>
<td>@E</td>
<td>77</td>
<td>Relative deferred reference mode</td>
</tr>
</tbody>
</table>

n represents the register number. Note that in the last four forms, register 7 (the PC) is referenced.

NOTE

An alternate form for @R is (ER). However, the form @(ER) is equivalent to @0(ER).

The form @$E differs from the form E in that the second or third word of the instruction contains the absolute address of the operand rather than the relative distance between the operand and the PC. Thus, the instruction CLR @$100 clears absolute location 100 even if the instruction is moved from the point at which it was assembled. See the description of the .ENABL AMA function in Section 5.5.2, which directs the assembly of all relative mode addresses as absolute mode addresses.

5.4.14 Branch Instruction Addressing

The branch instructions are 1-word instructions. The high byte contains the op code and the low byte contains an 8-bit signed offset which specifies the branch address relative to the PC. Upon execution of a branch instruction, the hardware calculates the branch address as follows:

1. Extend the sign of the offset through bits 8-15.
2. Multiply the result by 2. This creates a word offset rather than a byte offset.

5-26
MACRO Assembler

3. Add the result to the PC to form the final branch address.

The assembler performs the reverse operation to form the byte offset from the specified address. Remember that when the offset is added to the PC, the PC is pointing to the word following the branch instruction; hence the term \(-2\) in the calculation.

\[
\text{Byte offset} = (E-\text{PC})/2 \text{ truncated to eight bits.}
\]

Since \(\text{PC} = .+2\), we have:

\[
\text{Byte offset} = (E-.2)/2 \text{ truncated to eight bits.}
\]

**NOTE**

It is illegal to branch to a location specified as an external symbol, or to a relocatable symbol from within an absolute section, or to an absolute symbol or a relocatable symbol or another program section from within a relocatable section.

5.4.15 EMT and TRAP Addressing

The EMT and TRAP instructions do not use the low-order byte of the word. This allows information to be transferred to the trap handlers in the low-order byte. If EMT or TRAP is followed by an expression, the value is put into the low-order byte of the word. However, if the expression is too big (>377(8)) it is truncated to eight bits and a T error flag is generated.

5.5 ASSEMBLER DIRECTIVES

Directives are statements which cause the assembler to perform certain processing operations.

Assembler directives can be preceded by a label, subject to restrictions associated with specific directives, and followed by a comment. An assembler directive occupies the operator field of a MACRO source line. Only one directive can be placed on any one line. Zero, one, or more operands can occupy the operand field; legal operands differ with each directive and may be either symbols, expressions, or arguments.

5.5.1 Listing Control Directives

5.5.1.1 .LIST and .NLIST - Listing options can be specified in the text of a MACRO program through the .LIST and .NLIST directives. These are of the form:

```
_LIST arg
_NLIST arg
```
MACRO Assembler

where arg represents one or more optional arguments. When used without arguments, the listing directives alter the listing level count. The listing level count causes the listing to be suppressed when it is negative. The count is initialized to zero, incremented for each .LIST and decremented for each .NLIST. For example:

```
.LIST ME
.MACRO LTEST .LIST TEST
;A=THIS LINE SHOULD LIST
;NLIST
;B=THIS LINE SHOULD NOT LIST
;NLIST
;C=THIS LINE SHOULD NOT LIST
;LIST
;D=THIS LINE SHOULD NOT LIST (LEVEL NOT BACK TO ZERO)
;LIST
;E=THIS LINE SHOULD LIST (LEVEL BACK TO ZERO)
;ENDM
LTEST
;CALL THE MACRO
;A=THIS LINE SHOULD LIST
;E=THIS LINE SHOULD LIST (LEVEL BACK TO ZERO)
```

The primary purpose of the level count is to allow macro expansions to be selectively listed and yet exit with the level returned to the status current during the macro call.

The use of arguments with the listing directives does not affect the level count; however, .LIST and .NLIST can be used to override the current listing control. For example:

```
.MACRO XX

.LIST .LIST NEXT LINE
X=
;NLIST IDO NOT LIST REMAINDER
;OF MACRO EXPANSION

;ENDM
;NLIST ME IDO NOT LIST MACRO EXPANSIONS
X=
```

Allowable arguments for use with the listing directives are as follows (these arguments can be used singly or in combination):

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>list</td>
<td>Controls the listing of source line sequence numbers.</td>
</tr>
<tr>
<td>LOC</td>
<td>list</td>
<td>Controls the listing of the location counter (this field would not normally be suppressed).</td>
</tr>
<tr>
<td>BIN</td>
<td>list</td>
<td>Controls the listing of generated binary code (supersedes BEX).</td>
</tr>
</tbody>
</table>

5-28
MACRO Assembler

BEX list Controls listing of binary extensions; that is, prevents listing those locations and binary contents beyond the first line of an expansion. This is a subset of the BIN argument.

SRC list Controls the listing of the source code.

COM list Controls the listing of comments. This is a subset of the SRC argument and can be used to reduce listing time and/or space where comments are unnecessary.

MD list Controls listing of macro definitions and repeat range expansions (has no effect in ASEMBL).

MC list Controls listing of macro calls and repeat range expansions (has no effect in ASEMBL).

ME no list Controls listing of macro expansions (supersedes MEB; has no effect in ASEMBL).

MEB no list Controls listing of macro expansion binary code. A .LIST MEB causes only those macro expansion statements producing binary code to be listed. This is a subset of the ME argument (has no effect in ASEMBL).

CND list Controls the listing of unsatisfied conditions and all .IF and .ENDC statements. This argument permits conditional assemblies to be listed without including unsatisfied code.

LD no list Controls listing of all listing directives having no arguments (those used to alter the listing level count).

TOC list Controls listing of table of contents on pass 1 of the assembly (see Section 5.5.1.4 describing the .SBTTL directive). The full assembly listing is printed during pass 2 of the assembly.

TTM Terminal mode Controls listing output format (has no effect in ASEMBL). The TTM argument (the default case) causes output lines to be truncated to 72 characters. Binary code is printed with the binary extensions below the first binary word. The alternative (,NLIST TTM) to Terminal mode is line printer mode, which is shown in Figure 5-2.

SYM list Controls the listing of the symbol table for the assembly.

5-29
MACRO Assembler

An example of an assembly listing as sent to a 132-column line printer is shown in Figure 5-2. Notice that binary extensions for statements generating more than one word are spread horizontally on the source line. An example of an assembly listing as sent to an 80-column line printer is shown in Figure 5-3 (this is the same format as a terminal listing). Notice that binary extensions for statements generating more than one word are printed on subsequent lines.

Figure 5-4 illustrates a symbol table listing. With the exception of local symbols and macro names, all user-defined symbols are listed in the symbol table. The characters following the symbols listed have special meanings as follows:

- the symbol is assigned in a direct assignment statement
- the symbol is a register symbol
- the symbol is relocatable
- the symbol is global

The final value of the symbol is expressed in octal. If the symbol is undefined six asterisks are printed in place of the octal number.

CSECT numbers are listed if the symbol is in a named CSECT. All CSECTs are listed at the end of the table with their lengths and corresponding number.
Figure 5-2
Example of MACRO Line Printer Listing
(132-column Line Printer)
RTEXEC RT-11 MACRO VM02-09  5-SEP-74  22:30:23  PAGE 21
GET PHYSICAL SOURCE LINE

1                .SBITL GET PHYSICAL SOURCE LINE
2                WINST=EMT+240
3                GETPLI:
4  001756          104240
5  001756          WREAW SRC
6  001760          025000
7  001770          032737
                    800000
                    000012
8  001776          001424
9  002000          013700
                    02362
10  02004          005200
11  02006          02027
                    000010
12  02012          101017
13  02014          005037
                    000026
14  02020          013737
                    023106
                    002362
15  02026          010037
                    02362
16  02032          052700
                    104240
17  02036          010017
18  02040          .WAIT 0
19  02042          13403
20  02044          012700
                    177777
21  02050          251
22  02052          012700
                    000001
23  02056          RETURN

Figure 5-3
Example of Page Heading from MACRO 80-Column Line Printer
(same format as Terminal Listing)
MACRO Assembler

5.5.1.2 Page Headings — The MACRO Assembler outputs each page in the format shown in Figure 5-3. On the first line of each listing page the assembler prints (from right to left):

1. title taken from .TITLE directive (most recent one encountered)
2. assembler version identification
3. the date and time of day if entered
4. page number

The second line of each listing page contains the subtitle text specified in the last encountered .SBTTL directive.

5.5.1.3 .TITLE — The .TITLE directive is used to print a heading in the output listing and to assign a name to the object module. The heading printed on the first line of each page of the listing is taken from the first 31 characters of the argument in the .TITLE directive. The first six characters (symbol name) of this same line are also used as the name of the object module. These six characters must be Radix-50 characters (any characters beyond the first six are ignored). Non-Radix-50 characters are not acceptable.

For example:

```
.TITLE PROG TO PERFORM DAILY ACCOUNTING
```

causes PROG TO PERFORM DAILY ACCOUNTING to be printed in the heading for each page and causes the object module of the assembled program to be PROG (this name is distinguished from the filename of the object module specified in the command string to the assembler).

If there is no TITLE statement, the default name assigned to the first object module is:

```
.MAIN,
```

The first tab or space following the .TITLE directive is not considered part of the object module name or header text, although subsequent tabs and spaces are significant.

If there is more than one .TITLE directive, the last .TITLE directive in the program conveys the name of the object module.

5.5.1.4 .SBTTL — The .SBTTL directive is used to provide the elements for a printed table of contents of the assembly listing. The text following the directive is printed as the second line of each of the following assembly listing pages until the next occurrence of a .SBTTL directive.

For example:

```
.SBTTL CONDITIONAL ASSEMBLIES
```
MACRO Assembler

The text:

CONDITIONAL ASSEMBLES

is printed as the second line of each of the following assembly listing pages.

During pass 1 of the assembly process, MACRO automatically prints a table of contents for the listing containing the line sequence number and text of each .SBTTL directive in the program. Such a table of contents is inhibited by specifying the .NLIST TOC directive within the source.

An example of a table of contents is shown in Figure 5-5.

```
*MAIN, RT=11 MACRO VM02-09 5-SEP-74 22:30:23
TABLE OF CONTENTS

1= 29         RT=11 MACRO PARAMETER FILE
1= 37         COMMON PARAMETER FILE
2=  1         ASSEMBLY OPTIONS
3=  1         VARIABLE PARAMETERS
4=  1         GLOBALS
5=  1         SECTOR INITIALIZATION
7=  1         SUBROUTINE CALL DEFINITIONS
10=  1        MISCELLANEOUS MACRO DEFINITIONS
11=  2        MCI0CH - I/O CHANNEL ASSIGNMENTS
12=  2        ***EXEC***
13=  1        PROGRAM START
14=  1        INIT OUTPUT FILES
15=  1        SWITCH HANDLERS
16=  1        END-OF-PASS ROUTINES
17=  1        SWITCH AND DATE DATA AREAS
18=  1        INIT OUTPUT FILES (CONTINUED)
19=  1        FINISH ASSEMBLY AND RESTART
20=  1        MEMORY MANAGEMENT
21=  1        GET PHYSICAL SOURCE LINE
22=  1        SYSTEM MACRO HANDLERS
23=  1        WRITE ROUTINES
24=  1        READ ROUTINE
25=  1        COMMON I/O ROUTINES
26=  1        MESSAGES
27=  1        I/O TABLES
29=  1        FINIS
```

Figure 5-5
Assembly Listing Table of Contents

Table of Contents text is taken from the text of each .SBTTL directive. The associated numbers are the page and line numbers of the .SBTTL directives.

5-35
MACRO Assembler

5.5.1.5 .IDENT - The .IDENT directive is not used or supported by the RT-11 system, but is handled by MACRO for compatibility with other systems. .IDENT provides a means of labeling the object module produced as a result of a MACRO assembly. In addition to the name assigned to the object module with the .TITLE directive, a character string (up to six characters, treated like a .RAD50 string) can be specified between paired delimiters. For example:

. IDENT /V005A/

The character string:

V005A

is converted to Radix-50 notation and output to the global symbol directory of the object module.

When more than one .IDENT directive is found in a given program, the last .IDENT found determines the symbol which is passed as part of the object module identification.

5.5.1.6 Page Ejection (.PAGE Directive) - There are several means of obtaining a page eject in a MACRO assembly listing:

1. After a line count of 58 lines, MACRO automatically performs a page eject to skip over page perforations on line printer paper and to formulate terminal output into pages.

2. A form feed character used as a line terminator (or as the only character on a line) causes a page eject. Used within a macro definition a form feed character causes a page eject. A page eject is not performed when the macro is invoked.

3. More commonly, the .PAGE directive is used within the source code to perform a page eject at that point. The format of this directive is:

.PAGE

This directive takes no arguments and causes a skip to the top of the next page.

Used within a macro definition, the .PAGE is ignored, but the page eject is performed at each invocation of that macro.

5.5.2 Functions: .ENABL and .DSABL Directives

Several functions are provided by MACRO through the .ENABL and .DSABL directives. These directives use 3-character symbolic arguments to designate the desired function and are of the forms:

.ENABL arg
.DSABL arg

where arg is one of the legal symbolic arguments defined below.
MACRO Assembler

The following list describes the symbolic arguments and their associated functions in the MACRO language:

<table>
<thead>
<tr>
<th>Symbolic Argument</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Enabling of this function (has no effect in ASEMBL) produces absolute binary output; (i.e., for input to the Paper Tape Software System absolute binary loader using a .BIN extension instead of .OBJ). The default case is .DSABL ABS.</td>
</tr>
<tr>
<td>AMA</td>
<td>Enabling of this function directs the assembly of all relative addresses (address mode 67) as absolute addresses (address mode 37). This switch is useful during the debugging phase of program development.</td>
</tr>
<tr>
<td>CDR</td>
<td>The statement .ENABL CDR (has no effect in ASEMBL) causes source columns 73 and greater to be treated as comments. This accommodates sequence numbers in card columns 72-80.</td>
</tr>
<tr>
<td>FPT</td>
<td>Enabling of this function (has no effect in ASEMBL) causes floating point truncation, rather than rounding as is otherwise performed. .DSABL FPT returns to floating point rounding mode.</td>
</tr>
<tr>
<td>LC</td>
<td>Enabling of this function causes the assembler to accept lower case ASCII input instead of converting it to upper case (has no effect in ASEMBL).</td>
</tr>
<tr>
<td>LSB</td>
<td>Enable or disable a local symbol block (has no effect in ASEMBL). While a local symbol block is normally entered by encountering a new symbolic label or .CSECT directive, .ENABL LSB forces a local symbol block which is not terminated until a label or .CSECT directive following the .DSABL LSB statement is encountered. The default case is .DSABL LSB.</td>
</tr>
<tr>
<td>PNC</td>
<td>The statement .DSABL PNC (has no effect in ASEMBL) inhibits binary output until an .ENABL PNC is encountered. The default case is .ENABL PNC.</td>
</tr>
</tbody>
</table>

An incorrect argument causes the directive containing it to be flagged as an error.

5.5.3 Data Storage Directives

A wide range of data and data types can be generated with the following directives and assembly characters:
MACRO Assembler

`.BYTE`
`.WORD`
`
.ASCII`
`.ASCIZ`
`.RAD50`
1B
1D
1O

These facilities are explained in the following sections.

5.5.3.1 `.BYTE` - The `.BYTE` directive is used to generate successive bytes of data. The directive is of the form:

```
.BYTE exp

; WHICH STORES THE OCTAL
; EQUIVALENT OF THE EXPRESSION
; EXP IN THE NEXT BYTE
.
.BYTE exp1,exp2

; WHICH STORES THE OCTAL
; EQUIVALENTS OF THE LIST OF
; EXPRESSIONS IN SUCCESSIVE BYTES.
```

A legal expression must have an absolute value (or contain a reference to an external symbol) and must result in eight bits or less of data. The 16-bit value of the expression must have a high-order byte (which is truncated) that is either all zeros or all ones. Each operand expression is stored in a byte of the object program. Multiple operands are separated by commas and stored in successive bytes. For example:

```
SAM=5
Hat=410
.BYTE "048,SAM"

1068 (OCTAL EQUIVALENT OF 48
DECIMAL) IS STORED IN LOCATION
1411 - 005 IS STORED IN
1LOCATION 411 
```

If the high-order byte of the expression equates to a value other than 0 or -1, it is truncated to the low-order eight bits and flagged with a T error code. If the expression is relocatable, an A-type warning flag is given.

At link time it is likely that relocation will result in an expression of more than eight bits, in which case, the Linker prints an error message. For example:

```
.BYTE 23
STORES OCTAL 23 IN NEXT BYTE

.BYTE B
IRELOCATABLE VALUE CAUSES AN "A"
ERROR FLAG
```
MACRO Assembler

Here, X has an absolute value,

```
.GLOBAL X
X+3
.BYTE X    ;STORES 3 IN NEXT BYTE
```

and can be linked later with another program:

```
.GLOBAL X
.BYTE X
```

If an operand following the .BYTE directive is null, it is interpreted as a zero. For example (assume assembly begins at relocatable 0):

```
*+420
.BYTE ,,    ;ZEROS ARE STORED IN BYTES
1420, 421, AND 422,
```

5.5.3.2 .WORD - The .WORD directive is used to generate successive words of data. The directive is of the form:

```
.WORD exp   ;WHICH STORES THE OCTAL
             ;EQUIVALENT OF THE EXPRESSION
             ;EXP IN THE NEXT WORD

.WORD exp1,exp2... ;WHICH STORES THE OCTAL
                    ;EQUIVALENTS OF THE LIST OF
                    ;EXPRESSIONS IN SUCCESSIVE
                    ;WORDS
```

where a legal expression must result in 16 bits or less of data. Each operand expression is stored in a word of the object program.

Multiple operands are separated by commas and stored in successive words. For example:

```
.SAL=0
*.+500
.WORD 177535,+4,SAL  ;STORES 177535, 506, AND 0
                    ;IN WORDS 500, 502, AND 504.
```

If an expression equates to a value of more than 16 bits, it is truncated and flagged with a T error code.

If an operand following the .WORD directive is null, it is interpreted as zero. For example:

```
*.+500
.WORD ,5,      ;STORES 0,5,0 IN LOCATIONS
    1500, 502, AND 504
```

A blank operator field (any operator not recognized as a macro call, op-code, directive or semicolon) is interpreted as an implicit .WORD directive. Use of this convention is discouraged. The first term of the first expression in the operand field must not be an instruction mnemonic or assembler directive unless preceded by a + or - operator.

5-39
MACRO Assembler

For example:

```
*442
LABEL: *MOV, LABEL
THE OP-CODE FOR MOV, WHICH IS
1010000, IS STORED IN LOCATION
442. 440 IS STORED IN
LOCATION 442.
```

Note that the default .WORD directive occurs whenever there is a leading arithmetic or logical operator, or whenever a leading symbol is encountered which is not recognized as a macro call, an instruction mnemonic or assembler directive. Therefore, if an instruction mnemonic, macro call, or assembler directive is misspelled, the .WORD directive is assumed and errors will result. Assume that MOV is spelled incorrectly as MOR:

```
MOR A,B
```

Two error codes result: A and U. Two words are then generated, one for MOR A and one for B.

5.5.3.3 ASCII Conversion of One or Two Characters — The ' and " characters are used to generate text characters within the source text. A single apostrophe followed by a character results in a term in which the 7-bit ASCII representation of the character is placed in the low-order byte and zero is placed in the high-order byte. For example:

```
MOV #"A,R0
```

results in the following 16 bits being moved into R0:

```
0000000001000001
```

The ' character is never followed by a carriage return, null, RUBOUT, line feed, or form feed. (For another use of the ' character, see Section 5.6.3.6.)

```
STMNT1
GETSYM
BEQ 45
CMPB #CHRPTNT,#1
ICOLON DELIMITS LABEL FIELD
BEQ LABEL
CMPB #CHRPTNT,#6
IEQUAL DELIMITS
BEQ ASGMT
IASSIGNMENT PARAMETER
```

A double quote followed by two characters results in a term in which the 7-bit ASCII representations of the two characters are placed. For example:

```
MOV #"AB,R0
```

results in the following binary word being moved into R0:

```
0100001001000001
```

Note that the first character is placed in the low-order byte and the second character in the high-order byte.

5-40
MACRO Assembler

The " character is never followed by a carriage return, null, rubout, line feed, or form feed. For example:

```
;DEVICE NAME TABLE
DEVNAM: WORD "RF" JRF DISK
        WORD "RK" JRK DISK
DEVNKB: WORD "TT" JT TERMINAL KEYBOARD
        WORD "DT" JDCTAPE
        WORD "LP" JL LINE PRINTER
        WORD "PR" JP PAPER TAPE READER
        WORD "PP" JP PAPER TAPE PUNCH
        WORD @ TABLE'S END
```

5.5.3.4 .ASCII - The .ASCII directive translates character strings into their 7-bit ASCII equivalents for use in the source program. The format of the .ASCII directive is:

```
.ASCI /character string/
```

where: character string

is a string of any acceptable printing ASCII characters including spaces. The string may not include null characters, rubout, return, line feed, vertical tab, or form feed. Nonprinting characters can be expressed in digits of the current radix and delimited by angle brackets. (Any legal, defined expression is allowed between angle brackets.)

```
/ /
```

are delimiting characters and may be any printing characters other than ; < and = characters and any character within the string.

As an example:

```
A1 .ASCII /HELLO/ JSTORES ASCII REPRESENTATION OF
     /TIME LETTERS H E L L O IN
     /CONSECUTIVE BYTES
```

The order of the characters as they are stored in memory is illustrated below.
MACRO Assembler

1001 1002
1003 E H 1002
1005 L L 1004
1007 O 1006

;ASCII /ABC/<15>/DEF/
;STORES 1101,102,103,15,12,104,105,106
;IN CONSECUTIVE BYTES

;ASCII /<AB>/
;STORES 74,101,102,76 IN
;CONSECUTIVE BYTES

The ; and = characters are not illegal delimiting characters, but are preempted by their significance as a comment indicator and assignment operator, respectively. For other than the first group, semicolons are treated as beginning a comment field. For example:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ASCII ;ABC;/DEF/</td>
<td>A B C D E F</td>
<td>Acceptable, but not recommended procedure.</td>
</tr>
<tr>
<td>.ASCII /ABC;/DEF;</td>
<td>A B C</td>
<td>;DEF; is treated as a comment and ignored.</td>
</tr>
<tr>
<td>.ASCII /ABC/=DEF=</td>
<td>A B C D E F</td>
<td>Acceptable, but not recommended procedure.</td>
</tr>
<tr>
<td>.ASCII =DEF=</td>
<td></td>
<td>The assignment .ASCII=DEF is performed and an error generated upon encountering the second =.</td>
</tr>
</tbody>
</table>

5.5.3.5 .ASCIIZ - The .ASCIIZ directive is equivalent to the .ASCII directive with a zero byte automatically inserted as the final character of the string. For example:

When a list or text string has been created with a .ASCIIZ directive, a search for the null character can determine the end of the list as follows:

CR=15
LF=12
.
.
.
.
MOV X:HELLO,P1

5-42
MACRO Assembler

```
MOV    #LINBUF, R2
X1     MOVB  (R1)+, (R2)+  ; MOVE A CHARACTER OF THE
BNE X  ; MESSAGE STRING INTO THE
        ; OUTPUT BUFFER
        ; BRANCH BACK IF BYTE
        ; NOT EQUAL TO 0

HELLO1 .ASCIZ <CR><LF>/MACRO=11 V001A/<CR><LF>
        ; INTRO MESSAGE
```

5.5.3.6 .RAD50 - The .RAD50 directive allows the user the capability to handle symbols in Radix-50 coded form (this form is sometimes referred to as MOD40 and is used in PDP-11 system programs). Radix-50 form allows three characters to be packed into sixteen bits; therefore, any 6-character symbol can be held in two words. The form of the directive is:

```
.RAD50 /string/
```

where:

- `/` string delimiters can be any printing characters other than the `=`, `<`, and `;` characters.

- `string` is a list of the characters to be converted (three characters per word) and may consist of the characters A through Z, 0 through 9, dollar ($), dot (.) and space (.). If there are fewer than three characters (or if the last set is fewer than three characters) they are considered to be left justified and trailing spaces are assumed. Illegal nonprinting characters are replaced with a ? character and cause an I error flag to be set. Illegal printing characters set the Q error flag.

The trailing delimiter may be a carriage return, semicolon, or matching delimiter. (A warning code is printed if it is not a matching delimiter, however.) For example:

```
**** A
20 00040 003223 .RAD50 /ABC
21
22 00042 003220 .RAD50 /AB/
23 00044 000000 .RAD50 //
24 00046 003223 .RAD50 /ABCD/
00050 014400 .RAD50 /D (SPACE) (SPACE) INTO SECOND WORD
```

Each character is translated into its Radix-50 equivalent as indicated:
MACRO Assembler

<table>
<thead>
<tr>
<th>Character</th>
<th>Radix-50 Equivalent (octal)</th>
<th>ASCII (octal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(space)</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>A-Z</td>
<td>1-32</td>
<td>101-132</td>
</tr>
<tr>
<td>$</td>
<td>33</td>
<td>44</td>
</tr>
<tr>
<td>.</td>
<td>34</td>
<td>56</td>
</tr>
<tr>
<td>undefined</td>
<td>35</td>
<td>undefined</td>
</tr>
<tr>
<td>0-9</td>
<td>36-47</td>
<td>60-71</td>
</tr>
</tbody>
</table>

Note that another character could be defined for code 35, which is currently unused.

The Radix-50 equivalents for three characters (C1,C2,C3) are combined in one 16-bit word as follows:

Radix-50 value = ((C1*50)+C2)*50+C3

For example:

Radix-50 value of ABC is ((1*50)+2)*50+3 or 3223

See Appendix E for a table to quickly determine Radix-50 equivalents.

Use of angle brackets is encouraged in the .ASCII, .ASCIZ, and .RAD50 statements whenever leaving the text string to insert special codes. For example:

```plaintext
.ASCII <101> /EQUIVALENT TO /ASCII/A/  
.RAD50 /AB/<35>/STORES 3255 IN NEXT WORD.  
CHR1=1  
CHR2=2  
CHR3=3  
.  
.  
.RAD50 <CHR1><CHR2><CHR3> /EQUIVALENT TO RAD50/ABC/ 
```

5.5.4 Radix Control

5.5.4.1 .RADIX

Numbers used in a MACRO source program are initially considered to be octal numbers. However, the programmer has the option of declaring the following radices:

2, 4, 8, 10

This is done via the .RADIX directive of the form:

```
.RADIX n
```

where n is one of the acceptable radices.

5-44
MACRO Assembler

The argument to the .RADIX directive is always interpreted in decimal radix. Following any radix directive, that radix is the assumed base for any number specified until the following .RADIX directive.

The default radix at the start of each program, and the argument assumed if none is specified, is 8 (octal). For example:

```
.RADIX 10
```

```
BEGINS SECTION OF CODE WITH
DECIMAL RADIX
```

```
.*
```

```
.REVERTS TO OCTAL RADIX
```

In general it is recommended that macro definitions not contain or rely on radix settings from the .RADIX directive. The temporary radix control characters should be used within a macro definition. (#D, #O, and #B are described in the following section.) A given radix is valid throughout a program until changed. Where a possible conflict exists within a macro definition or in possible future uses of that code module, it is suggested that the user specify values using the temporary radix controls.

5.5.4.2 Temporary Radix Control: #D, #O, and #B - Once the user has specified a radix for a section of code, or has determined to use the default octal radix, he may discover a number of cases where an alternate radix is more convenient (particularly within macro definitions). For example, the creation of a mask word might best be done in the binary radix.

MACRO has three unary operators to provide a single interpretation in a given radix within another radix as follows:

- #Dx (x is treated as being in decimal radix)
- #Ox (x is treated as being in octal radix)
- #Bx (x is treated as being in binary radix)

For example:

```
#D123
#O 47
#B 00001101
#O(A+3)
```

Notice that while the uparrow and radix specification characters may not be separated, the radix operator can be physically separated from the number by spaces or tabs for formatting purposes. Where a term or expression is to be interpreted in another radix, it should be enclosed in angle brackets.

These numeric quantities may be used any place where a numeric value is legal.

A temporary radix change from octal to decimal may be made by specifying a decimal radix number with a "decimal point". For example:
MACRO Assembler

100. (144(8))
1376. (2540(8))
128. (200(8))

5.5.5 Location Counter Control

The four directives that control movement of the location counter are .EVEN and .ODD which move the counter a maximum of one byte, and .BLKB and .BLKW which allow the user to specify blocks of a given number of bytes or words to be skipped in the assembly.

5.5.5.1 .EVEN - The .EVEN directive ensures that the assembly location counter contains an even memory address by adding one if the current address is odd. If the assembly location counter is even, no action is taken. Any operands following a .EVEN directive are ignored.

The .EVEN directive is used as follows:

```
.*ASCIZ /THIS IS A TEST/
.*EVEN
.*ASSURES NEXT STATEMENT
.*begins on a word boundary
.*word xyz
```

5.5.5.2 .ODD - The .ODD directive ensures that the assembly location counter is odd by adding one if it is even. For example:

```
/*code to move data from an input line
 *into a buffer

n=5
*ibuffer has 5 words

*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
*.
MACRO Assembler

In this case, .ODD is used to place the buffer byte count in the byte preceding the buffer, as follows:

<table>
<thead>
<tr>
<th>COUNT</th>
<th>BUFF-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BUFF</td>
</tr>
</tbody>
</table>

5.5.5.3 .BLKB and .BLKW - Blocks of storage can be reserved using the .BLKB and .BLKW directives. .BLKB is used to reserve byte blocks and .BLKW reserves word blocks. The two directives are of the form:

```
.BLKB   exp
.BLKW   exp
```

where exp is the number of bytes or words to reserve. If no argument is present, 1 is the assumed default value. Any legal expression which is completely defined at assembly time and produces an absolute number is legal. For example:

```
1
2
3
4  000000
5  000000
6  SYMBOL: .BLKW
7  000002
8  000006
9  000006
10  00007
11  00010
12  00012
13  00020
14  00024
15  00025
16  00025
17  00025
18  00030
19  000001
```

The .BLKB directive has the same effect as:

```
+=.exp
```

but is easier to interpret in the context of source code.

5.5.6 Numeric Control

Several directives are available to provide software complements to the floating-point hardware on the PDP-11.

5-47
MACRO Assembler

A floating-point number is represented by a string of decimal digits. The string (which can be a single digit in length) may optionally contain a decimal point, and may be followed by an optional exponent indicator in the form of the letter E and a signed decimal exponent. The list of number representations below contains seven distinct, valid representations of the same floating-point number:

3
3.0
3.0E0
3E0
.3E1
300E-2

As can be quickly inferred, the list could be extended indefinitely (e.g., 3000E-3, .03E2, etc.). A leading plus sign is ignored (e.g., +3.0 is considered to be 3.0). Leading minus signs complement the sign bit. No other operators are allowed (e.g., 3.0+N is illegal).

Floating-point number representations are valid only in the contexts described in the remainder of this section.

Floating-point numbers are normally rounded. That is, when a floating-point number exceeds the limits of the field in which it is to be stored, the high-order excess bit is added to the low-order retained bit. For example, if the number were to be stored in a 2-word field, but more than 32 bits were needed for its value, the highest bit carried out of the field would be added to the least significant position. In order to enable floating-point truncation, the .ENABL FPT directive is used and .DSABL FPT is used to return to floating-point rounding.

5.5.6.1 .FLT2 and .FLT4 - Like the .WORD directive, the two floating-point storage directives cause their arguments to be stored in-line with the source program (have no effect in ASEMBL). These two directives are of the form:

```
.FLT2    arg1,arg2,...
.FLT4    arg1,arg2,....
```

where arg1, arg2, etc. represent one or more floating point numbers separated by commas.

.FLT2 causes two words of storage to be generated for each argument while .FLT4 generates four words of storage.

5-48
MACRO Assembler

The following code shows the use of the .FLT4 directive:

```
1
2
3
4 000000 037314 ATOF8I .FLT4 1,E=1 110°=1
   000002 146314
   000004 146314
   000006 146315
5 000010 036443 .FLT4 1,E=2 110°=2
   000012 153412
   000014 036560
   000016 121727
6 000020 034721 .FLT4 1,E=4 110°=4
   000022 133427
   000024 054342
   000026 014545
7 000030 031453 .FLT4 1,E=8 110°=8
   000032 141617
   000034 010604
   000036 000717
8 000040 022746 .FLT4 1,E=16 110°=16
   000042 112624
   000044 137364
   000046 046741
9 000050 095517 .FLT4 1,E=32 110°=32
   000052 130436
   000054 126505
   000056 034825
```

5.5.6.2 Temporary Numeric Control: tF and tC - Like the temporary radix control operators, operators are available to specify either a 1-word floating-point number (tF— not available in ASSEMBLY) or the one's complement of a 1-word number (tC). For example:

```
FL3.7:  tF3.7
```

creates a 1-word floating-point number at location FL3.7 containing the value 3.7 as follows:

```
<table>
<thead>
<tr>
<th>SIGN BIT</th>
<th>15</th>
<th>14</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

This 1-word floating-point number is the first word of the 2- or 4-word floating-point number format shown in the PDP-11 Processor Handbook, and the statement:

```
CMP151:  tC151
```

stores the one's complement of 151 in the current radix (assume current radix is octal) as follows:
MACRO Assembler

Since these control operators are unary operators, their arguments may be integer constants or symbols, and the operators may be expressed successively. For example:

\[ \uparrow C \uparrow D25 \text{ or } \uparrow C31 \text{ or } 177746 \]

The term created by the unary operator and its argument is then a term which can be used by itself or in an expression. For example:

\[ \uparrow C2+6 \]

is equivalent to:

\[ \langle \uparrow C2 \rangle + 6 \text{ or } 177775+6 \text{ or } 000003 \]

For this reason, the use of angle brackets is advised. Expressions used as terms or arguments of a unary operator must be explicitly grouped.

An example of the importance of ordering with respect to unary operators is shown below:

\[
\begin{array}{l}
\uparrow F1.0 = 040200 \\
\uparrow F-1.0 = 140200 \\
-\uparrow F1.0 = 137600 \\
-\uparrow F-1.0 = 037600
\end{array}
\]

The argument to the \( \uparrow F \) operator must not be an expression and should be of the same format as arguments to the .FLT2 and .FLT4 directives.

5.5.7 Terminating Directives

5.5.7.1 .END - The .END directive indicates the physical end of the source program. The .END directive is of the form:

\[ .END \ exp \]

where exp is an optional argument which, if present, indicates the program entry point, i.e., the transfer address.
MACRO Assembler

When the load module is loaded, program execution begins at the transfer address indicated by the .END directive. In a runtime system (the load module output of the Linker) a .END exp statement should terminate the first object module and .END statements should terminate any other object modules.

5.5.7.2 .EOT - Under the RT-ll System, the .EOT directive is ignored. The physical end file allows several physically separate tapes to be assembled as one program.

5.5.8 Program Boundaries Directive: .LIMIT

The .LIMIT directive reserves two words into which the Linker puts the low and high addresses of the load module's relocatable code (the load module is the result of the link). The low address (inserted into the first word) is the address of the first byte of code. The high address is the address of the first free byte following the relocated code. These addresses are always even since all relocatable sections are loaded at even addresses. (If a relocatable section consists of an odd number of bytes, the Linker adds one to the size to make it even.)

5.5.9 Program Section Directives

The assembler provides for 255(10) program sections: an absolute section declared by .ASECT, an unnamed relocatable program section declared by .CSECT, and 253(10) named relocatable program sections declared by .CSECT symbol, where symbol is any legal symbolic name. These directives allow the user to:

1. Create his program (object module) in sections:

   The assembler maintains separate location counters for each section. This allows the user to write statements which are not physically contiguous but will be loaded contiguously. The following examples will clarify this:
MACRO Assembler

.CSEGTC
.START THE UNRELOCATABLE SECTION
.AI 0
.ASSEMBLED AT RELOCATABLE 0,
.BI 0
.RELOCATABLE 2 AND
.CI 0
.RELOCATABLE 4
.ST
.CLRA
.ASSEMBLE CODE AT
.CLRA
.RELOCATABLE ADDRESS
.CLRC
.16 THROUGH 21
.ASECT
.START THE ABSOLUTE SECTION
.*
.ASSEMBLE CODE AT
.WORD .+2,HALT
.ABSOLUTE 4 THROUGH 7
.CSEGTC
.REMOVE THE UNRELOCATABLE SECTION
.INC A
.ASSEMBLE CODE AT
.BR ST
.RELOCATABLE 22 THROUGH 27
.END

The first appearance of .CSEGTC or .ASECT assumes the location counter is at relocatable or absolute zero, respectively. The scope of each directive extends until a directive to the contrary is given. Further occurrences of the same .CSEGTC or .ASECT resume assembling where the section was left off.

.CSEGTC COM1
.DECLARE SECTION COM1
.AI 0
.ASSEMBLED AT RELOCATABLE 0
.BI 0
.ASSEMBLED AT RELOCATABLE 2
.CI 0
.ASSEMBLED AT RELOCATABLE 4
.XI 0
.DECLARE SECTION COM2
.YI 0
.ASSEMBLED AT RELOCATABLE 0
.REASSEMBLED AT RELOCATABLE 2
.D1 0
.REASSEMBLED AT RELOCATABLE 6
.END

The assembler automatically begins assembling at relocatable zero of the unnamed .CSEGTC if not instructed otherwise; that is, the first statement of an assembly is an implied .CSEGTC.

All labels in an absolute section are absolute; all labels in a relocatable section are relocatable. The location counter symbol, ",", is relocatable or absolute when referenced in a relocatable or absolute section, respectively. Undefined internal symbols are assigned the value of relocatable or absolute zero in a relocatable or absolute section, respectively. Any labels appearing on a .ASECT or .CSEGTC statement are assigned the value of the location counter before the .ASECT or .CSEGTC takes effect. Thus, if the first statement of a program is:

.AI .ASECT

then A is assigned to relocatable zero and is associated with the unnamed relocatable section (because the assembler implicitly begins assembly in the unnamed relocatable section).

Since it is not known at assembly time where the program sections are to be loaded, all references between sections in a single assembly are translated by the assembler to references relative to the base of that section. The assembler provides the Linker with the necessary information to resolve the linkage. Note that this is not necessary when
MACRO Assembler

making a reference to an absolute section (the assembler knows all load addresses of an absolute section).

Examples:

```
.ASECT
.*1000
A1 CLR X
     JASSEMBLED AS CLR BASE OF UNNAMED
     RELOCATABLE SECTION +12
JMP Y
     JASSEMBLED AS JMP BASE OF UNNAMED
     RELOCATABLE SECTION +10

.CSECT
MOV R0,R1
JMP A
     JASSEMBLED AS JMP 1000

Y1 HALT
X1 0
.END
```

In the above example the references to X and Y were translated into references relative to the base of the unnamed relocatable section.

2. Share code and/or data between object modules (separate assemblies):

Named relocatable program sections operate as FORTRAN labeled COMMON; that is, sections of the same name from different assemblies are all loaded at the same location by LINK. The unnamed relocatable section is the exception to this as all unnamed relocatable sections are loaded in unique areas by LINK.

Note that there is no conflict between internal symbolic names and program section names; that is, it is legal to use the same symbolic name for both purposes. In fact, considering FORTRAN again, this is necessary to accommodate the FORTRAN statement:

```
COMMON /X/A,B,C,X
```

where the symbol X represents the base of this program section and also the fourth element of this program section.

Program section names should not duplicate .GLOBL names. In FORTRAN language, COMMON block names and SUBROUTINE names should not be the same.

The .ASECT and .CSECT program section directives are provided in MACRO to allow the user to specify an unnamed absolute or relocatable section. These directives are formatted as follows:

```
.ASECT
.CSECT
.CSECT symbol
```
MACRO Assembler

The single absolute section can be declared with an:

.ASECT
directive. No name can be associated with the absolute section specified by means of the .ASECT directive. The single unnamed relocatable program section can be declared with a:

.CSECT
directive.

All named relocatable sections are loaded in unique areas by LINK. Up to 253(10) named relocatable program sections can be declared with:

.CSECT symbol
directives, where symbol is any legal symbolic name.

The assembler automatically begins assembling at relocatable zero of the unnamed .CSECT if not instructed otherwise; that is, the first statement of an assembly is an implied .CSECT.

5.5.10 Symbol Control: .GLOBL

If a program is created in segments which are assembled separately, global symbols are used to allow reference to one symbol by the different segments.

A global symbol must be declared in a .GLOBL directive. The form of the .GLOBL directive is:

 .GLOBL sym1,sym2,...

where:

sym1,sym2, etc. are legal symbolic names, separated by commas, tabs, or spaces where more than one symbol is specified.

Symbols appearing in a .GLOBL directive are either defined within the current program or are external symbols, in which case they are defined in another program which is to be linked with the current program, by LINK, prior to execution.

A .GLOBL directive line may contain a label in the label field and comments in the comment field.

```
JDEFINE A SUBROUTINE WITH 2 ENTRY POINTS WHICH JCALLS AN EXTERNAL SUBROUTINE

.CSECT .JEDECLARE THE CONTROL SECTION
.GLOBL A,B,C .JEDECLARE A,B,C AS GLOBALS
A1 MOV @(R5)+,R0 JENTRY A IS DEFINED
MOV #X,R1
X1 JSR PC,C JCALL EXTERNAL SUBROUTINE C
RTS R5 JEXIT
B1 MOV @(R5)+,R1 JDEFINE ENTRY B
CLR R1
BR X
```

January 1976 5-54
MACRO Assembler

In the previous example, A and B are entry symbols (entry points), C is an external symbol and X is an internal symbol.

A global symbol is defined only when it appears in a .GLOBL directive. A symbol is not considered a global symbol if it is assigned the value of a global expression in a direct assignment statement.

References to external symbols can appear in the operand field of an instruction or assembler directive in the form of a direct reference, i.e.:

```
CLR EXT
.WORD EXT
CLR #EXT
```

or a direct reference plus or minus a constant, i.e.:

```
D+6
CLR EXT+6
.WORD EXT+2
CLR #EXT+2
```

A global symbol defined within the program can be used in the evaluation of a direct assignment statement, but an external symbol cannot. Since MACRO determines at the end of pass 1 whether a given global symbol is defined within the program or is expected to be external, a construction such as the following will cause errors at link time:

```
.GLOBL FREE
.GLOBL LIMITS
FREE=LIMITS+2 ;FREE WILL NOT BE
              ;DEFINED UNTIL PASS 2

LIMITS:.LIMIT
```

FREE will be flagged as an undefined global at link time. To allow correct linking, define FREE after LIMITS:

5.5.11 Conditional Assembly Directives

Conditional assembly directives provide the programmer with the capability to conditionally include or ignore blocks of source code in the assembly process. This technique is used extensively to allow several variations of a program to be generated from the source program.

The general form of a conditional block is as follows:

```
.IF cond,argument(s) ;START CONDITIONAL BLOCK
     ;STATEMENTS IN RANGE OF
     ;CONDITIONAL
     ;BLOCK
.ENDC ;END CONDITIONAL BLOCK
```

where: cond is a condition which must be met if the block is to be included in the assembly. These conditions are defined below.
MACRO Assembler

argument(s) are a function of the condition to be tested. If more than one argument is specified, they must be separated by commas.

range is the body of code which is included in the assembly or ignored depending upon whether the condition is met.

The following are the allowable conditions:

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Positive</th>
<th>Complement</th>
<th>Arguments</th>
<th>Assemble Block If</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>NE</td>
<td></td>
<td>expression</td>
<td>expression=0 (or ≠ 0)</td>
</tr>
<tr>
<td>GT</td>
<td>LE</td>
<td></td>
<td>expression</td>
<td>expression&gt;0 (or &gt; 0)</td>
</tr>
<tr>
<td>LT</td>
<td>GE</td>
<td></td>
<td>expression</td>
<td>expression&lt;0 (or &lt; 0)</td>
</tr>
<tr>
<td>DF</td>
<td>NDF</td>
<td></td>
<td>symbolic argument</td>
<td>symbol is defined (or undefined)</td>
</tr>
<tr>
<td>B</td>
<td>NB</td>
<td></td>
<td>macro-type argument</td>
<td>argument is blank (or nonblank)</td>
</tr>
<tr>
<td>IDN</td>
<td>DIF</td>
<td></td>
<td>two macro-type arguments separated by a comma</td>
<td>arguments identical (or different)</td>
</tr>
<tr>
<td>Z</td>
<td>NZ</td>
<td></td>
<td>expression</td>
<td>same as EQ/NE</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td>expression</td>
<td>same as GT/LE</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td>expression</td>
<td>same as LT/GE</td>
</tr>
</tbody>
</table>

IF DIF and IF IDN are not available in ASEMBL.

NOTE

A macro-type argument is enclosed in angle brackets or within an up-arrow construction (as described in Section 5.2.1.1). For example:

```
<A,B,C>
```

t/124/

For example:

```
ALPHA=1
 IF EQ,ALPHA+1 JASSEMBLE IF ALPHA+1=0
```

Within the conditions DF and NDF the following two operators are allowed to group symbolic arguments:

& logical AND operator

| logical inclusive OR operator

5-56
MACRO Assembler

For example:

```
IFDEF SYM1 & SYM2  ; ASSEMBLE IF BOTH SYM1
                ;  AND SYM2 ARE DEFINED
```

5.5.11.1 Subconditionals - Subconditionals may be placed within conditional blocks to indicate:

1. Assembly of an alternate body of code when the condition of the block indicates that the code within the block is not to be assembled,

2. Assembly of a non-contiguous body of code within the conditional block depending upon the result of the conditional test to enter the block,

3. Unconditional assembly of a body of code within a conditional block.

There are three subconditional directives, as follows:

<table>
<thead>
<tr>
<th>Subconditional</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>.IFF</td>
<td>The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is false.</td>
</tr>
<tr>
<td>.IFT</td>
<td>The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is true.</td>
</tr>
<tr>
<td>.IFTF</td>
<td>The code following this statement up to the next subconditional or the end of the conditional block is included in the program regardless of the value of the condition tested upon entering the conditional block.</td>
</tr>
</tbody>
</table>

The implied argument of the subconditionals is the value of the condition upon entering the conditional block. Subconditionals are used within outer level conditional blocks. Subconditionals are ignored within nested, unsatisfied conditional blocks.

For example:
MACRO Assembler

```
..IF  DF,SYM  ;ASSEMBLE BLOCK IF SYM IS DEFINED
..IFF  DF,SYM  ;ASSEMBLE THE FOLLOWING CODE ONLY IF
           ;SYM IS UNEFIND  
  
..IFT  DF,SYM  ;ASSEMBLE THE FOLLOWING CODE ONLY IF
           ;SYM IS DEFINED  
  
..IFTF  DF,SYM  ;ASSEMBLE THE FOLLOWING CODE
           ;UNCONDITIONALLY  
  
  .ENDC

..IF  DF,X  ;ASSEMBLY TESTS FALSE
..IF  DF,Y  ;TESTS FALSE
..IFF   ;NESTED CONDITIONAL
         ;IGNORED WITHIN NESTED, UNSATISFIED
         ;CONDITIONAL BLOCK
         
..IFT   ;NOT SEEN
          
  .ENDC  .ENDC
```

However,

```
..IF  DF,X  ;TESTS TRUE
..IF  DF,Y  ;TESTS FALSE
..IFF   ;IS ASSEMBLED
         ;OUTER CONDITIONAL SATISFIED,
         
..IFT   ;NOT ASSEMBLED
          
  .ENDC  .ENDC
```

5.5.11.2 Immediate Conditionals – An immediate conditional directive is a means of writing a 1-line conditional block. In this form, no .ENDC statement is required and the condition is completely expressed on the line containing the conditional directive. Immediate conditions are of the form:

```
..IIF cond, arg, statement
```

where: cond is one of the legal conditions defined for conditional blocks in Section 5.5.11.
MACRO Assembler

arg is the argument associated with the conditional specified, that is, either an expression, symbol, or macro-type argument, as described in Section 5.5.11.

statement is the statement to be executed if the condition is met.

For example:

.IIF DF,FOO,BEQ ALPHA

This statement generates the code:

BEQ ALPHA

if the symbol FOO is defined.

A label must not be placed in the label field of the .IIF statement. Any necessary labels may be placed on the previous line, as in the following example:

LABEL: .IIF DF,FPP BEQ ALPHA

or included as part of the conditional statement:

.IIF DF,FOO LABEL: BEQ ALPHA

5.5.11.3 PAL-llR and PAL-llS Conditional Assembly Directives - In order to maintain compatibility with programs developed under PAL-llR and PAL-llS, the following conditionals remain permissible under MACRO. It is advisable that future programs be developed using the format for MACRO conditional assembly directives.

<table>
<thead>
<tr>
<th>Directive</th>
<th>Arguments</th>
<th>Assemble Block if</th>
</tr>
</thead>
<tbody>
<tr>
<td>.IFZ or .IFEQ</td>
<td>expression</td>
<td>expression=0</td>
</tr>
<tr>
<td>.IFNZ or .IFNE</td>
<td>expression</td>
<td>expression #0</td>
</tr>
<tr>
<td>.IFL or .IFLT</td>
<td>expression</td>
<td>expression&lt;0</td>
</tr>
<tr>
<td>.IFGE</td>
<td>expression</td>
<td>expression&gt;0</td>
</tr>
<tr>
<td>.IFLE</td>
<td>expression</td>
<td>expression&gt;=0</td>
</tr>
<tr>
<td>.IFDF</td>
<td>logical expression</td>
<td>expression is true(defined)</td>
</tr>
<tr>
<td>.IFND</td>
<td>logical expression</td>
<td>expression is false(undefined)</td>
</tr>
</tbody>
</table>

The rules governing the usage of these directives are now the same as for the MACRO conditional assembly directives previously described. Conditional assembly blocks must end with the .ENDC directive and are limited to a nesting depth of 16(10) levels (instead of the 127(10) levels allowed under PAL-llR).
MACRO Assembler

5.6 MACRO DIRECTIVES

5.6.1 Macro Definition

It is often convenient in assembly language programming to generate a recurring coding sequence with a single statement. In order to do this, the desired coding sequence is first defined with dummy arguments as a macro. Once a macro has been defined, a single statement calling the macro by name with a list of real arguments (replacing the corresponding dummy arguments in the definition) generates the correct sequence or expansion.

5.6.1.1 .MACRO - The first statement of a macro definition must be a .MACRO directive (not available in ASEMBL). The .MACRO directive is of the form:

```
.MACRO name, dummy argument list
```

where:

- `name` is the name of the macro. This name is any legal symbol. The name chosen may be used as a label elsewhere in the program.

- `dummy argument list` represents any legal separator (generally a comma or space).

A comment may follow the dummy argument list in a statement containing a .MACRO directive. For example:

```
.MACRO ABS A,B
```

A label must not appear on a .MACRO statement. Labels are sometimes used on macro calls, but serve no function when attached to .MACRO statements.

5.6.1.2 .ENDM - The final statement of every macro definition must be an .ENDM directive (not available in ASEMBL) of the form:

```
.ENDM name
```

where `name` is an optional argument and is the name of the macro being terminated by the statement.

For example:
MACRO Assembler

.ENDM (terminates the current macro definition)
.ENDM ABS (terminates the definition of the macro ABS)

If specified, the symbolic name in the .ENDM statement must correspond to that in the matching .MACRO statement. Otherwise the statement is flagged and processing continues. Specification of the macro name in the .ENDM statement permits the assembler to detect missing .ENDM statements or improperly nested macro definitions.

The .ENDM statement may contain a comment field, but must not contain a label.

An example of a macro definition is shown below:

.MACRO TYPMSG MESSGE /TYPE A MESSAGE
JSR RS,TYPMSG
.WORD MESSGE
.ENDM

5.6.1.3 .MEXIT - In order to implement alternate exit points from a macro (particularly nested macros), the .MEXIT directive is provided. .MEXIT (not available in ASEMBL) terminates the current macro as though an .ENDM directive were encountered. Use of .MEXIT bypasses the complications of conditional nesting and alternate paths. For example:

.MACRO ALTR N,A,B

.IF EQ,N /START CONDITIONAL BLOCK

.MEXIT /EXIT FROM MACRO DURING CONDITIONAL BLOCK

.ENDC /END CONDITIONAL BLOCK

.ENDM /NORMAL END OF MACRO

In an assembly where N=0, the .MEXIT directive terminates the macro expansion.

Where macros are nested, a .MEXIT causes an exit to the next higher level. A .MEXIT encountered outside a macro definition is flagged as an error.

5.6.1.4 Macro Definition Formatting - A form feed character used as a line terminator in a macro source statement (or as the only character on a line), causes a page eject when the source program is listed. Used within a macro definition, a form feed character also causes a page eject. A page eject is not performed, however, when the macro is invoked.
MACRO Assembler

Used within a macro definition, the .PAGE directive is ignored, but a page eject is performed at invocation of that macro.

5.6.2 Macro Calls

A macro must be defined prior to its first reference. Macro calls are of the general form:

```
label: name, real arguments
where:  
label  represents an optional statement label.
name    represents the name of the macro specified in the .MACRO directive preceding the macro definition.
,       represents any legal separator (comma, space, or tab). No separator is necessary where there are no real arguments. (Refer to Section 5.2.1.1.)
real arguments
```

Where a macro name is the same as a user label, the appearance of the symbol in the operation field designates a macro call, and the occurrence of the symbol in the operand field designates a label reference. For example:

```
ABS1 MOV #R0,R1 ;ABS IS USED AS A LABEL
 .
BR ABS ;ABS IS CONSIDERED A LABEL
 .
ABS #4,ENT,LAR ;CALL MACRO WITH 3 ARGUMENTS
```

Arguments to the macro call are treated as character strings whose usage is determined by the macro definition.

5.6.3 Arguments to Macro Calls and Definitions

Arguments within a macro definition or macro call are separated from other arguments by any of the separating characters described in Section 5.2.1.1. For example:

```
.MACRO REN A,B,C ;MACRO DEFINITION
 .
 .
REN ALPHA,BETA,<C1,C2> ;MACRO CALL
```

5-62
MACRO Assembler

Arguments which contain separating characters are enclosed in paired angle brackets. An up-arrow construction is provided to allow angle brackets to be passed as arguments.

For example:

```
REN <MOV X,Y>,#44,WEV
```

This call would cause the entire statement:

```
MOV X,Y
```

to replace all occurrences of the symbol A in the macro definition. Real arguments within a macro call are considered to be character strings and are treated as a single entity until their use in the macro expansion.

The up-arrow construction could have been used in the above macro call as follows:

```
REN */MOV X,Y/,#44,WEV
```

which is equivalent to:

```
REN <MOV X,Y>,#44,WEV
```

Since spaces are ignored preceding an argument, they can be used to increase legibility of bracketed constructions.

5.6.3.1 Macro Nesting - Macro nesting (nested macro calls), where the expansion of one macro includes a call to another macro, causes one set of angle brackets to be removed from an argument with each nesting level. The depth of nesting allowed is dependent upon the amount of memory space used by the program. To pass an argument containing legal argument delimiters to nested macros, the argument should be enclosed in one set of angle brackets for each level of nesting, as shown below:

```
R0=0
R1=1
X=10

.MACRO LEVEL1 DUM1,DUM2
LEVEL2 DUM1
LEVEL2 DUM2
.ENDM

.MACRO LEVEL2 DUM3
DUM3
ADD #10,R0
MOV R0,(R1)+
.ENDM
```

A call to the LEVEL1 macro:

```
LEVEL1 <<MOV X,R0>>,<<CLR R0>>
```

causes the following expansion:
MACRO Assembler

    MOV   X,R0
    ADD   #10,R0
    MOV   R0,(R1)+
    CLR   R0
    ADD   #10,R0
    MOV   R0,(R1)+

Where macro definitions are nested (that is, a macro definition is entirely contained within the definition of another macro) the inner definition is not defined as a callable macro until the outer macro has been called and expanded. For example:

    .MACRO LV1 A,B
    :
    :
    .MACRO LV2 A
    :
    :
    ENDM
    ENDM

The LV2 macro cannot be called by name until after the first call to the LV1 macro. Likewise, any macro defined within the LV2 macro definition cannot be referenced directly until LV2 has been called.

5.6.3.2 Special Characters - Arguments may include special characters without enclosing the argument in a bracket construction if that argument does not contain spaces, tabs, semicolons, or commas. For example:

    .MACRO PUSH ARG
    MOV   ARG,=(SP)
    ENDM
    :
    :
    PUSH   X+3(%2)

generates the following code:

    MOV   X+3(%2),=(SP)

5.6.3.3 Numeric Arguments Passed as Symbols - When passing macro arguments, a useful capability is to pass a symbol which can be treated by the macro as a numeric string. An argument preceded by the unary operator backslash (\) is treated as a number in the current radix. (\ is not available in ASEML.) The ASCII characters representing the number are inserted in the macro expansion; their function is defined in context (see Section 5.6.3.6 for an explanation of single-quote usage). For example:
MACRO Assembler

A'B1

.MACRO CNT A,B
  .WORD
  .ENDM
C=0
.MACRO INC A,B
  CNT A,B
  B=B+1
  .ENDM
.
  INC X,C

The macro call would expand to:

X01

.WORD

A subsequent identical call to the same macro would generate:

X11

.WORD

and so on for later calls. The two macros are necessary because the
dummy value of B cannot be updated in the CNT macro. In the CNT
macro, the number passed is treated as a string argument. (Where the
value of the real argument is 0, a single 0 character is passed to the
macro expansion.)

The number being passed can also be used to make source listings
somewhat clearer. For example, versions of programs created through
conditional assembly of a single source can identify themselves as
follows:

.MACRO IOT SYM
  .IDENT /SYM/
  .ENDM IOT

.MACRO OUT ARG
  IOT 005A*ARG
  .ENDM
.
.
  OUT \ID

/ASSUME THAT THE SYMBOL ID TAKES
/ON A UNIQUE 2 DIGIT VALUE FOR
/Each POSSIBLE CONDITIONAL
/ASSEMBLY OF THE PROGRAM

WHERE 005A IS THE UPDATE VERSION
OF THE PROGRAM AND ARG INDICATES
THE CONDITIONAL ASSEMBLY VERSION

The above macro call expands to:

.IDENT /005AXX/

where XX is the conditional value of ID.

Two macros are necessary since the text delimiting characters in the
.IDENT statement would inhibit the concatenation of a dummy argument.
MACRO Assembler

5.6.3.4 Number of Arguments - If more arguments appear in the macro call than in the macro definition, the excess arguments are ignored. If fewer arguments appear in the macro call than in the definition, missing arguments are assumed to be null (consist of no characters). The conditional directives .IF B and .IF NB can be used within the macro to detect null arguments.

A macro can be defined with no arguments.

5.6.3.5 Automatically Created Symbols Within User-Defined Macros - MACRO can be made to create symbols of the form n$ where n is a decimal integer number such that 64≤n≤127. Created symbols are always local symbols between 64$ and 127$. Such local symbols are created by the assembler in numerical order, i.e.:

64$
65$
$
$
$
126$
127$

Created symbols are particularly useful where a label is required in the expanded macro. Such a label must otherwise be explicitly stated as an argument with each macro call or the same label is generated with each expansion (resulting in a multiply-defined label). Unless a label is referenced from outside the macro, there is no reason for the programmer to be concerned with that label.

The range of these local symbols extends between two explicit labels. Each new explicit label causes a new local symbol block to be initialized.

The macro processor creates a local symbol on each call of a macro whose definition contains a dummy argument preceded by the ? character. For example:

```
.Macro ALPHA A,B
TST A
BEQ B
ADD #5,A
B:
.EMDN
```

Local symbols are generated only where the real argument of the macro call is either null or missing. If a real argument is specified in the macro call, the generation of a local symbol is inhibited and normal replacement is performed. Consider the following expansions of the macro ALPHA above.
MACRO Assembler

Generate a local symbol for missing argument:

\[
\begin{align*}
\text{ALPHA} & \quad \%1 \\
\text{TST} & \quad \%1 \\
\text{BEO} & \quad 643 \\
\text{ADD} & \quad \#5,\%1 \\
\end{align*}
\]

6451

Do not generate a local symbol:

\[
\begin{align*}
\text{ALPHA} & \quad \%2,\text{XYZ} \\
\text{TST} & \quad \%2 \\
\text{BEO} & \quad \text{XYZ} \\
\text{ADD} & \quad \#5,\%2 \\
\end{align*}
\]

XYZ1

These assembler-generated symbols are restricted to the first sixteen (decimal) arguments of a macro definition.

5.6.3.6 Concatenation - The apostrophe or single quote character ('') operates as a legal separating character in macro definitions. An ' character which precedes and/or follows a dummy argument in a macro definition is removed, and the substitution of the real argument occurs at that point. For example:

\[
\begin{align*}
\text{MACRO DEF } & \quad A,B,C \\
A'B': & \quad \text{ASCIZ } '/C/' \\
& \quad \text{WORD } 'A'''B' \\
& \quad \text{ENDM}
\end{align*}
\]

When this macro is called:

\[
\text{DEF } X,Y,<\text{MACRO}=11>
\]

it expands as follows:

\[
\begin{align*}
XY: & \quad \text{ASCIZ } '/\text{MACRO}=11/' \\
& \quad \text{WORD } 'X''Y'
\end{align*}
\]

In the macro definition, the scan terminates upon finding the first ' character. Since A is a dummy argument, the ' is removed. The scan resumes with B, notes B as another dummy argument and concatenates the two dummy arguments. The third dummy argument is noted as going into the operand of the .ASCIZ directive. On the next line (this example is for purely illustrative purposes) the argument to .WORD is seen as follows: The scan begins with a ' character. Since it is neither preceded nor followed by a dummy argument, the ' character remains in the macro definition. The scan then encounters the second ' character which is followed by a dummy argument and is discarded. The scan of the argument A terminates upon encountering the second ' which is also discarded since it follows a dummy argument. The next ' character is neither preceded nor followed by a dummy argument and remains in the macro expansion. The last ' character is followed by another dummy argument and is discarded. (Note that the five ' characters were necessary to generate two ' characters in the macro expansion.)

Within nested macro definitions, multiple single quotes can be used, with one quote removed at each level of macro nesting.
MACRO Assembler

5.6.4 .NARG, .NCHR, and .NTYPE

These three directives allow the user to obtain the number of arguments in a macro call (.NARG), the number of characters in an argument (.NCHR), or the addressing mode of an argument (.NTYPE). (They are not available in ASEMBL.) Use of these directives permits selective modifications of a macro depending upon the nature of the arguments passed.

The .NARG directive enables the macro being expanded to determine the number of arguments supplied in the macro call and is of the form:

```
label:   .NARG symbol
```

where:  label is an optional statement label

symbol  is any legal symbol which is equated to the number of arguments in the macro call currently being expanded. The symbol can be used by itself or in expressions.

This directive can occur only within a macro definition. An example of the use of .NARG follows.

```
.MACRO ARG5  A1,A2,A3,A4
.NARG NUM
.IF EQ NUM=1
  .IF A2, A3, A4 NOT SPECIFIED
.WORD A1
.IFF
.WORD A1
.ASCII /A2/
.WORD A3
.ASCII /A4/
.ENDC
.ENDM

ARGS ALPHA
.WORD ALPHA  generated

ARGS ALPHA,BETA,GAMMA,DELTA
.WORD ALPHA
.ASCII /BETA/  generated
.WORD GAMMA
.ASCII /DELTA/
```

The .NCHR directive enables a program to determine the number of characters in a character string, and is of the form:

```
label:   .NCHR symbol, <character string>
```

where:  label is an optional statement label

symbol  is any legal symbol which is equated to the number of characters in the specified character string. The symbol is separated from the character string argument by any legal separator.
MACRO Assembler

A string is a string of printing characters which should only be enclosed in angle brackets if it contains a legal separator. A semicolon also terminates the character string.

This directive can occur anywhere in a MACRO program. For example:

```
.MACRO CHARS ...
  NCHR NUM,A
  ASCII /A/
  IF EQ NUM&1 IF THE STRING CONTAINS AN EVEN NUMBER OF CHARACTERS
  WORD +1
  IFF
  BYTE +2
ENDC
.ENDM
```

For example, using the above definition, the code:

```
CHARS ALPHA
```

expands to:

```
ASCII /ALPHA/
BYTE +2
```

and

```
CHARS BETA
```

expands to:

```
ASCII /BETA/
WORD +1
```

The .NTYPE directive enables the macro being expanded to determine the addressing mode and register of any argument, and is of the form:

```
label: .NTYPE symbol, arg
```

where:

- `label` is an optional statement label
- `symbol` is any legal symbol, the low order 6-bits of which is equated to the 6-bit addressing mode of the argument. The symbol is separated from the argument by a legal separator. This symbol can be used by itself or in expressions.
- `arg` is any legal macro argument (dummy argument) as defined in Section 5.6.3.

This directive can occur only within a macro definition. An example of .NTYPE usage in a macro definition is shown below:
MACRO Assembler

.MACRO SAVE ARG
.NOTYPE SYM,ARG
.IF EU,SYM&70
  MOV ARG,TEMP
  JREGISTER MODE
.IFF
  MOV #ARG,TEMP
  JNONREGISTER MODE
.ENDC
.ENDM

Using this definition, the code:

SAVE #Z

expands to:

MOV #R2,TEMP

and

SAVE ALPHA

expands to:

MOV #ALPHA,TEMP

5.6.5 .ERROR and .PRINT

The .ERROR directive (not available in ASEMBl) is used to output messages to the listing file during assembly pass 2. A common use is to provide diagnostic announcements of a rejected or erroneous macro call. The form of the .ERROR directive is as follows:

label: .ERROR expr;text

where:
label is an optional statement label

expr is an optional legal expression whose value is output to the listing file when the .ERROR directive is encountered. Where expr is not specified, the text only is output to the listing file.

; denotes the beginning of the text string to be output.

text is the string to be output to the listing file. The text string is terminated by a line terminator.

Upon encountering a .ERROR directive anywhere in a MACRO program, the assembler outputs a single line containing:

1. the sequence number of the .ERROR directive line,
2. the current value of the location counter,
3. the value of the expression if one is specified, and,
MACRO Assembler

4. the text string specified.

For example, assume the following error macro occurs:

```
.MACRO STORE SRC,DEST
 .NTYPE A,DEST
 .IF EQ,<A&7>=6 .IF STACK POINTER USED
 .ERROR A\UNACCEPTABLE MACRO ARGUMENT
 .IFF
 .MOV SRC,DEST
 .ENDC
 .ENDM

STORE R3,#4(SP)
```

and the following line is output:

```
***** P
00000 000076 .ERROR A\UNACCEPTABLE MACRO ARGUMENT
```

This message is used to indicate an inability of the subject macro to cope with the argument DEST which is detected as being indexed deferred addressing mode (mode 70) with the stack pointer (%6) used as the index register. The line is flagged on the assembly listing with a P error code.

The .PRINT directive is identical to .ERROR except that it is not flagged with a P error code. (.PRINT is not available in ASEMBL.)

5.6.6 Indefinite Repeat Block: .IRP and .IRPC

An indefinite repeat block (not available in ASEMBL) is a structure very similar to a macro definition. An indefinite repeat is essentially a macro definition which has only one dummy argument and is expanded once for every real argument supplied. An indefinite repeat block is coded in-line with its expansion rather than being referenced by name as a macro is referenced. An indefinite repeat block is of the form:

```
label: .IRP arg,(real arguments)
 .
 .
 (range of the indefinite repeat)
 .
 .
 .ENDM
```

where:

- `label` is an optional statement label. A label may not appear on any .IRP statement within another macro definition, repeat range or indefinite repeat range, or on any .ENDM statement.

- `arg` is a dummy argument which is successively replaced with the real arguments in the .IRP statement.
MACRO Assembler

<real arguments>

is a list of arguments to be used in the expansion of the indefinite repeat range and enclosed in angle brackets. Each real argument is a string of zero or more characters or a list of real arguments (enclosed in angle brackets). The real arguments are separated by commas.

range is the code to be repeated once for each real argument in the list. The range may contain macro definitions, repeat blocks, or other indefinite repeat blocks. Note that only created symbols should be used as labels within an indefinite repeat range.

An indefinite repeat block can occur either within or outside macro definitions, repeat ranges, or indefinite repeat ranges. The rules for creating an indefinite repeat block are the same as for the creation of a macro definition (for example, the _MEXIT statement is allowed in an indefinite repeat block). Indefinite repeat arguments follow the same rules as macro arguments. A second type of indefinite repeat block is available which handles character substitution rather than argument substitution. The _IRPC directive is used as follows:

label: _IRPC arg,string

.(range of indefinite repeat)

.ENDM

On each iteration of the indefinite repeat range, the dummy argument (arg) assumes the value of each successive character in the string. Terminators for the string are: space, comma, tab, carriage return, line feed, and semicolon.

Figure 5-6 is an example of _IRP and _IRPC usage.

IRPTST RT-11 MACRO VM02-09 11104149 PAGE 1

1
2
3
4
5
6
7
8
9

000000
400000
012700
000056

.TITLE IRPTST
.LIST MD,MC,ME
.R0=%0
MOV #TABLE,R0
MACRO Assembler

10 .IRP X,A,B,C,D,E,F
11
12 MOV X,(R0)+
13 .ENDM

00004 016720 MOV A,(R0)+
000032

00010 016720 MOV B,(R0)+
000030

00014 016720 MOV C,(R0)+
000026

00020 016720 MOV D,(R0)+
000024

00024 016720 MOV E,(R0)+
000022

00030 016720 MOV F,(R0)+
000020

14 .IRPC X,ABCDEFGHIJKLMNOPQRSTUVWXYZ
15 .ASCII /X/
16 .ENDM

00034 101 .ASCII /A/
00035 102 .ASCII /B/
00036 103 .ASCII /C/
00037 104 .ASCII /D/
00040 105 .ASCII /E/
00041 106 .ASCII /F/

17
18 00042 041101 AI .WORD "AB"
19 00044 041502 BI .WORD "BC"
20 00046 042103 CI .WORD "CD"
21 00050 042504 DI .WORD "DE"
22 00054 043105 EI .WORD "EF"
23 00054 043506 FI .WORD "FG"
24 00056 "TABLE" .BLKW 6
25 00061 .END

Figure 5-6
.IRP and .IRPC Example

5.6.7 Repeat Block: .REPT

Occasionally it is useful to duplicate a block of code a number of times in line with other source code. (.REPT is not available in ASEMBL.) This is performed by creating a repeat block of the form:

label: .REPT expr

.

.(range of repeat block)

.

5-73
MACRO Assembler

.
.
.ENDM ;OR .ENDR

where: label is an optional statement label. The .ENDR or .ENDM directive may not have a label. A .REPT statement occurring within another repeat block, indefinite repeat block, or macro definition may not have a label associated with it.

expr is any legal expression controlling the number of times the block of code is assembled. Where expr<0, the range of the repeat block is not assembled.

range is the code to be repeated expr number of times. The range may contain macro definitions, indefinite repeat blocks, or other repeat blocks. Note that no statements within a repeat range can have a label.

The last statement in a repeat block can be an .ENDM or .ENDR statement. The .ENDR statement is provided for compatibility with previous assemblers.

The .MEXIT statement is also legal within the range of a repeat block.

5.6.8 Macro Libraries: .MCALL

All macro definitions must occur prior to their referencing within the user program. MACRO provides a selection mechanism for the programmer to indicate in advance those system macro definitions required by his program.

The .MCALL directive is used to specify the names of all system macro definitions not defined in the current program but required by the program (not available in ASEMBL). The .MCALL directive must appear before the first occurrence of a macro call for an externally defined macro. The .MCALL directive is of the form:

.MCALL arg1,arg2,...

where arg1,arg2, etc. are the names of the macro definitions required in the current program.

When this directive is encountered, MACRO searches the system library file, SYSMAC.SML, to find the requested definition(s). MACRO searches for SYSMAC.SML on the system device (SY:).

See Appendix D for a listing of the system macro file (SYSMAC.SML) stored on the system device.

5.7 CALLING AND USING MACRO

The MACRO Assembler assembles one or more ASCII source files containing MACRO statements into a single relocatable binary object file. Assembler output consists of this binary object file and an
MACRO Assembler

optional assembly listing followed by the symbol table listing. CREF (Cross Reference) listings may also be specified as part of the assembly output by means of switch options.

MACRO is executed using the RT-ll Monitor R command as follows:

```
.R MACRO
```

The assembler responds by typing an asterisk (*) to indicate readiness to accept command string input. In response to the * printed by the assembler, the user types the output file specification(s), followed by an equal sign or left angle bracket, followed by the input file specification(s) in a command line as follows:

```
*dev:obj,dev:list/s:arg=dev:sourcel,...,dev:sourcen/s:arg
```

where: dev: is any legal RT-ll device for output; must be file-structured for input

obj is the binary object file

list is the assembly listing file containing the assembly listing and symbol table

sourcel, ... , sourcen are the ASCII source files containing the macro source program(s); a maximum of six source files is allowed

/s:arg represents a switch and argument as explained in Section 5.7.1

A null specification in either of the output file fields signifies that the associated output file is not desired.

One or more switches can be indicated with the appropriate file specification to provide MACRO with information about that file.

The default case for each file specification is noted below:

<table>
<thead>
<tr>
<th>file</th>
<th>device</th>
<th>filename</th>
<th>extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>DK:</td>
<td>-</td>
<td>.OBJ</td>
</tr>
<tr>
<td>listing</td>
<td>device used</td>
<td>-</td>
<td>.LST</td>
</tr>
<tr>
<td></td>
<td>for object</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sourcel</td>
<td>DK:</td>
<td>-</td>
<td>.MAC</td>
</tr>
<tr>
<td>source2</td>
<td>device used</td>
<td>-</td>
<td>.MAC</td>
</tr>
<tr>
<td></td>
<td>for last</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>source file specified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sourcen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system</td>
<td>system device</td>
<td>SYSMAC</td>
<td>.SML</td>
</tr>
<tr>
<td>macro</td>
<td>SY:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Type CTRL C to halt MACRO at any time and return control to the monitor. To restart the assembler type R MACRO or the REENTER command in response to the monitor's dot.
MACRO Assembler

NOTE

If \( \dagger C \) was typed while a CREF listing was being produced, the REENTER command may not be accepted. In this case, type \( \dagger R \) MACRO to restart the assembler.

5.7.1 Switches

There are three types of switch options: listing control switches, function switches, and CREF specification switches. The listing control switches (/L, /N) provide capabilities similar to those described in detail in section 5.5.1.1. The function control switches (/D, /E) provide function control as described in Section 5.5.2; arguments for these switches are summarized in Section 5.7.1.2. CREF control switches allow the user to obtain a detailed cross-referenced listing of his assembled file, and are described in detail in Section 5.7.1.3. Multiple arguments may be specified for a particular switch, if desired, by separating each switch value from the next by a colon.

For example:

\[ /N:TTM:CND \]

These switches turn off teleprinter mode and suppress printing of unsatisfied conditionals (as described in the next section). Also, the switches are not restricted to appearing near a particular file in the command string; \( /N:TTM \), for example, is legal in all of the following places:

\[ *:LP:/N:TTM=source \]
\[ *:LP=source/N:TTM \]
\[ */N:TTM,LP=source \]

and they are all equivalent in function.

5.7.1.1 Listing Control Switches - A listing control switch (/L for list or /N for nolist) is indicated in a command line as follows:

\[ *\text{dev}:\text{obj}.\text{ext},\text{dev}:\text{list}.\text{ext}/s:\text{arg}=\text{dev}:\text{source}.\text{ext} \]

where \( s:\text{arg} \) represents /L or /N; the remainder of the command line abbreviations are as described in Section 5.7.

The /N switch with no argument causes only the symbol table, table of contents and error listings to be produced. The /L switch with no arguments causes .LIST and .NLIST directives that appear in the source program but have no arguments to be ignored. A summary of the arguments which are valid for the listing control switches follows (refer to Section 5.5.1.1 for details):

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Controls listing of</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>list</td>
<td>Source line sequence numbers</td>
</tr>
<tr>
<td>LOC</td>
<td>list</td>
<td>Location counter</td>
</tr>
<tr>
<td>BIN</td>
<td>list</td>
<td>Generated binary code</td>
</tr>
<tr>
<td>BEX</td>
<td>list</td>
<td>Binary extensions</td>
</tr>
</tbody>
</table>

January 1976

5-76
MACRO Assembler

SRC list Source code
COM list Comments
MD list Macro definitions, repeat range expansions
MC list Macro calls, repeat range expansions
ME nolist Macro expansions
MEB nolist Macro expansion binary code
CND list Unsatisfied conditionals, .IF and .ENDC statements
LD nolist Listing directives with no arguments
TOC list Table of Contents
TTM terminal mode Listing output format
SYM list Symbol table

For example, a command line in the following format allows binary code to be listed throughout the assembly using the 132-column line printer format:

*\LP:/L:MEB/N:TTM=FILE

5.7.1.2 Function Switches - The function control switches (/D for disable and /E for enable) are used to enable or disable certain functions in source input files and are indicated in the command line as follows:

*dev:obj.ext, dev:list.ext=dev:source/s:arg

/s:arg here represents either /D:arg or /E:arg. A summary of the arguments which are valid for use with the function control switches follows (refer to Section 5.5.2 for details):

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Enables or disables</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>disable</td>
<td>Absolute binary output</td>
</tr>
<tr>
<td>AMA</td>
<td>disable</td>
<td>Assembly of all absolute addresses as relative addresses</td>
</tr>
<tr>
<td>CDR</td>
<td>disable</td>
<td>Source columns 73 and greater to be treated as comments</td>
</tr>
<tr>
<td>FPT</td>
<td>disable</td>
<td>Floating point truncation</td>
</tr>
<tr>
<td>LC</td>
<td>disable</td>
<td>Accepts lower case ASCII input</td>
</tr>
<tr>
<td>LSB</td>
<td>disable</td>
<td>Local symbol block</td>
</tr>
<tr>
<td>PNC</td>
<td>enable</td>
<td>Binary output</td>
</tr>
</tbody>
</table>

For example, the following commands assemble a file allowing all 80 columns of each card to be used as input (note that since MACRO is a two-pass assembler, the cards cannot be read directly from the card reader; input from any nonfile-structured device must first be transferred to a file-structured device before assembly):

.R PIP
*C HD.MAC=CR:/A
*^C

.R MACRO
+.LP:CARDS.MAC/E;CDR

Use of either the function control or listing control switches and arguments at assembly-time will override any corresponding listing or function control directives and arguments in the source program. For example, assume the following appears in the source program:

.NLIST MEB
.
.

MACRO References
.

.LIST MEB

5-77
January 1976
MACRO Assembler

The "MEB" printing will be disabled for the block indicated; however, if /L:MEB is indicated in the assembly command line, both the .NLIST MEB and the .LIST MEB will be ignored and the "MEB" printing will be enabled everywhere in the program.

5.7.1.3 Cross Reference Table Generation (CREF) - A cross reference table of all or a subset of all symbols used in the source program and the statements where they were defined or used can be obtained automatically following an assembly by specifying /C:arg with the assembly listing file specification (and any listing or function control specifications) as follows:

*dev:obj.ext, dev:list.ext/s:arg/C:arg=dev:source.ext

/s:arg represents /L:arg, /N:arg, /E:arg, or /D:arg. (If the listing device is magtape or cassette, and a CREF listing is desired, the handler must first be loaded, using the monitor LOAD command.)

There are six sections to a complete cross reference listing:

1. Cross reference of program symbols (i.e., labels used in the program and symbols used on the left of the "=" operator).

2. Cross reference of register-equate symbols (those symbols which are defined in the program by a "SYMBOL=%N", 0<%N<7, construct. (Normally this consists of the symbols R0, R1, R2, R3, R4, R5, SP, and PC.)

3. Cross reference of MACRO symbols (names of macros as defined by a .MACRO directive, or as specified in a .MCALL directive).

4. Cross reference of permanent symbols (all operation mnemonics and assembler directives).

5. Cross reference of control sections (those names specified as the operand of a .CSECT directive, plus the blank .CSECT and the absolute section ".ABS." which are always defined by MACRO).

6. Cross reference of errors (all errors flagged on the listing are grouped by error type).

Any or all of the above sections may be included in the cross reference listing as desired. The associated switch options and their arguments are listed below:

<table>
<thead>
<tr>
<th>Switch Argument</th>
<th>Section Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>/C:S</td>
<td>User-defined symbols</td>
</tr>
<tr>
<td>/C:R</td>
<td>Register symbols</td>
</tr>
<tr>
<td>/C:M</td>
<td>Macro symbolic names</td>
</tr>
<tr>
<td>/C:P</td>
<td>Permanent symbols</td>
</tr>
<tr>
<td></td>
<td>(instructions, directives)</td>
</tr>
<tr>
<td>/C:C</td>
<td>Control sections (.CSECT</td>
</tr>
<tr>
<td></td>
<td>symbolic names)</td>
</tr>
<tr>
<td>/C:E</td>
<td>Error codes</td>
</tr>
<tr>
<td>/C&lt;no arg&gt;</td>
<td>Equivalent to /C:S:M:E</td>
</tr>
</tbody>
</table>

January 1976 5-78
MACRO Assembler

The specification of a /C switch in a command string causes a temporary file, "DK:CREF.TMP", to be generated. If device DK: is write-locked or contains insufficient free space for the temporary file, the user may allocate the temporary file on another device. To do so, a third output file specification is given in the MACRO command string; this file is then used instead of DK:CREF.TMP, and is purged after use. For example, a command string of this type:

```
*,LP:,RK2:TEMP.TMP=SOURCExC
```

causes "RK2:TEMP.TMP" to be used as the temporary file.

Figure 5-7 illustrates assembled source code and Figure 5-8 contains the CREF output. The command line used to produce these listings was:

```
*,LP:/C:S:M:R:P:C:E/N:BEX=EXAMPL
```

An explanation of the CREF output follows the figures.
EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE 1

1
2
000000 R0 = %0
3
000001 R1 = %1
4
000002 R2 = %2
5
000003 R3 = %3
6
000004 R4 = %4
7
000005 R5 = %5
8
000006 SP = %6
9
000007 PC = %7
10
11
000012 LF = 012
12
13
.*CALL \TTYIN, \EXIT
14
15
.*MACRO CALL NAME
16
JSR PC, NAME
17
.*ENDM
18
19
.*GLOBAL SUBR1, SUBR2
20
21
22
000000
23 000000 012702 START: MOV #BUFFER,R2
24
000004 1S1 \TTYIN
25
00010 110022 MOVB R0,(R2)+
26
00012 120027 CMPB R0, #LF
27
00016 001372 SNE 1%
28
00020 105022 CLRB (R2)+
29
00022 012703 MOV #BUFFER,R3
30
00026 CALL SUBR1
31
00032 103762 RCS START
32
00034 CALL SUBR2
33
00040 010067 MOV R0, ANSWER
34
00044 \EXIT
35
36 00046 \BLKW \BLKB 72.
37
38
39

Figure 5-7
MACRO Source Code
EXAMPLE OF CROSS-REFERENCE LST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE 1+
SYMBOL TABLE

ANSWER 000046R 002 BUFFER 000050R 002 LF = 000012
PC  =%000007 R0  =%000000 R1  =%000001
R2  =%000002 R3  =%000003 R4  =%000004
R5  =%000005 SP  =%000006 START 000000R 002
SUBR1 = ***** G SUBR2 = ***** G
'* ABS, 000000 000
        000000 001
PROG 000160 002
ERRORS DETECTED: 0
FREE CORE: 16248 WORDS

;LP1/CIS/MIRIPICIE/NIBEX=EXAMPL

Figure 5-7 (Cont.)
MACRO Source Code

EXAMPLE OF CROSS-REFERENCE LST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE 5-1
CROSS REFERENCE TABLE (CREF VM01-02)

.  1-24
ANSWER 1-33#  1-36#
BUFFER 1-23  1-29  1-37#
LF  1-17#  1-26
START 1-24#  1-31  1-39
SUBR1 1-20#  1-30
SUBR2 1-20#  1-32

Figure 5-8
CREF Listing Output
MACRO Assembler

EXAMPLE OF CROSS-REFERENCE LIST RT-11 MACRO VM02-09 5-SEP-74 22:11:59 PAGE R-1

CROSS REFERENCE TABLE (REF V01-02)

<table>
<thead>
<tr>
<th>PC</th>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*EXIT
1-14#
1-34
call
1-16#
1-50
1-32

Figure 5-8 (Cont.)
CREP Listing Output

5-82
Example of cross-reference list RT-11 macro VM02-09 5-SEP-74 22:11:59 page P-1

CROSS REFERENCE TABLE (CREF V01-02)

```
BLKR  1-37
BLKW  1-36
CSECT 1-22
END   1-39
GLOBAL 1-20
IF    1-28
MACRO 1-16
MCALL 1-14
TITLE 1-1
ACS   1-24  1-31
RNF   1-27
CLRB  1-29
CMPB  1-26
EMT   1-24  1-34
JSR   1-30  1-32
MOV   1-23  1-29  1-33
MOVA  1-25
```

Example of cross-reference list RT-11 macro VM02-09 5-SEP-74 22:11:59 page C-1
CROSS REFERENCE TABLE (CREF V01-02)

```
ABS*  0-0
PRNG  1-22
```

Figure 5-8 (Cont.)
CREF Listing Output
MACRO Assembler

Cross reference tables, if requested, are generated at the end of a MACRO assembly listing. Each table begins on a new page (the tables in Figure 5-8 have been consolidated due to space considerations). Symbols, control sections, and error codes are listed at the left margin of the page; corresponding references are indicated next to them across the page from left to right. A reference is of the form p-l, where p is the page on which the symbol, control section, or error code appears, and l is the line number within the page. A number sign (#) appears next to a reference wherever a symbol has been defined. An asterisk appears next to a reference wherever a destructive reference has been made to the symbol (i.e., the contents of the location defined by that symbol have been altered at that point).

The CREF output requested in the preceding figures included user defined symbols, macro symbolic names, control sections, error codes, register symbols, and permanent symbols. Since no errors were generated in this assembly, no CREF output for error codes was produced.

5.8 MACRO ERROR MESSAGES

MACRO error messages enclosed in question marks are output on the terminal. The single-letter error codes are printed in the assembly listing.

In terminal mode these error codes are printed following a field of six asterisk characters and on the line preceding the source line containing the error. For example:

```
****** A
26 00236 000002' ,WORD REL1+REL2
```

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Addressing error. An address within the instruction is incorrect. Also may indicate a relocation error. The addition of two relocatable symbols is flagged as an A error. May also indicate that a local symbol is being defined more than 128 words from the beginning of a local symbol block.</td>
</tr>
<tr>
<td>B</td>
<td>Bounding error. Instructions or word data would be assembled at an odd address in memory. The location counter is updated by +1.</td>
</tr>
<tr>
<td>D</td>
<td>Multiply-defined symbol referenced. Reference was made to a label (not a local label) that is defined more than once.</td>
</tr>
<tr>
<td>E</td>
<td>End directive not found. (A .END is generated.)</td>
</tr>
<tr>
<td>I</td>
<td>Illegal character detected. Illegal characters which are also non-printing are replaced by a ? on the listing. The character is then ignored.</td>
</tr>
<tr>
<td>L</td>
<td>Line buffer overflow, i.e., input line greater than 132 characters. Extra characters on a line are ignored in terminal mode.</td>
</tr>
</tbody>
</table>

January 1976

5-84
MACRO Assembler

M Multiple definition of a label. A label was encountered which was equivalent (in the first six characters) to a previously encountered label.

N Number containing 8 or 9 has decimal point missing.


P Phase error. A label's definition or value varies from one pass to another or a local symbol occurred twice within a local symbol block.

Q Questionable syntax. There are missing arguments or the instruction scan was not completed or a carriage return was not immediately followed by a line feed or form feed.

R Register-type error. An invalid use of or reference to a register has been made.

T Truncation error. A number generated more than 16 bits of significance or an expression generated more than 8 bits of significance during the use of the .BYTE directive.

U Undefined symbol. An undefined symbol was encountered during the evaluation of an expression. Relative to the expression, the undefined symbol is assigned a value of zero.

Z Instruction which is not compatible among all members of the PDP-11 family.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>?BAD SWITCH?</td>
<td>The switch specified was not recognized by the program.</td>
</tr>
<tr>
<td>?INSUFFICIENT CORE?</td>
<td>There are too many symbols in the program being assembled. Try dividing</td>
</tr>
<tr>
<td></td>
<td>program into separately-assembled subprograms.</td>
</tr>
<tr>
<td>?I/O ERROR ON CHANNEL n?</td>
<td>A hardware error occurred while attempting to read from or write to the</td>
</tr>
<tr>
<td></td>
<td>device on the channel specified in the message. (Channel numbers</td>
</tr>
<tr>
<td></td>
<td>(0&lt;=n&lt;10 octal) are assigned to files in the manner described in Section</td>
</tr>
<tr>
<td></td>
<td>9.4.7, Chapter 9.)</td>
</tr>
<tr>
<td></td>
<td>Note that the CREF temporary file is on channel 2 even if it was not</td>
</tr>
<tr>
<td></td>
<td>specified in the command string (i.e., if the default file</td>
</tr>
<tr>
<td></td>
<td>DK;CREF_TMP is used).</td>
</tr>
</tbody>
</table>

5-85 January 1976
MACRO Assembler

**?NO INPUT FILE?**  No input file was specified and there must be at least one input file.

**?OUTPUT DEVICE FULL?**  No room to continue writing output. Try to compress device with PIP.

**TOO MANY OUTPUT FILES**  Too many output files were specified.

All CREF error messages begin with C- to distinguish them from MACRO error messages. When a CREF error occurs, the error message is printed on the console terminal and CREF chains back to MACRO; MACRO prints an asterisk, at which time another command line may be entered.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>?C-CHAIN-ONLY-CUSP?</td>
<td>An attempt was made either to &quot;R CREF&quot; or to &quot;START&quot; a copy of CREF which was in memory. CREF can only be &quot;chained&quot; to.</td>
</tr>
<tr>
<td>?C-CREF FILE ERROR?</td>
<td>An output error occurred while accessing &quot;DK:CREF.TMP&quot;, the temporary file passed to CREF.</td>
</tr>
<tr>
<td>?C-DEVICE?</td>
<td>An invalid device was specified to CREF.</td>
</tr>
<tr>
<td>?C-LST FILE ERROR?</td>
<td>An output error occurred while attempting to write the cross-reference table to the listing file.</td>
</tr>
</tbody>
</table>
CHAPTER 6
LINKER

6.1 INTRODUCTION

The RT-11 Linker converts object modules produced by either one of the RT-11 assemblers or FORTRAN IV into a format suitable for loading and execution. This allows the user to separately assemble a main program and each of its subroutines without assigning an absolute load address at assembly time. The object modules of the main program and subroutines are processed by the Linker to:

1. Relocate each object module and assign absolute addresses
2. Link the modules by correlating global symbols defined in one module and referenced in another module
3. Create the initial control block for the linked program
4. Create an overlay structure if specified and include the necessary run-time overlay handlers and tables
5. Search user specified libraries to locate unresolved globals
6. Optionally produce a load map showing the layout of the load module

The RT-11 Linker requires two or three passes over the input modules. During the first pass it constructs the global symbol table, including all control section names and global symbols in the input modules. If library files are to be linked with input modules, an intermediate pass is needed to force the modules resolved from the library file into the root segment (that part of the program which is never overlaid). During the final pass, the Linker reads the object modules, performs most of the functions listed above, and produces a load module (.LDA for use with the Absolute Loader, save image (.SAV) for a Single-job system or for the background job of an P/B System, and relocatable (.REL) format for the foreground job of an P/B System).

The Linker runs in a minimal RT-11 system of 8K; any additional memory is used to facilitate efficient linking and to extend the symbol table. Input is accepted from any random-access device on the system; there must be at least one random-access device (disk or DECTape) for save image or relocatable format output.

6-1 January 1976
Linker

6.2 CALLING AND USING THE LINKER

To call the Linker, type the command:

    R LINK

and the RETURN key in response to the Keyboard monitor's dot. The Linker prints an asterisk and awaits a command string.

Type CTRL C to halt the Linker at any time and return control to the monitor. To restart the Linker, type R LINK or the REENTER command in response to the monitor's dot. The Linker outputs an extra line feed character when it is restarted with REENTER or after an error in the first command line. When the Linker is finished linking, control returns to the CSI automatically. An extra line feed character precedes the asterisk printed by the CSI.

6.2.1 Command String

The first command string entered in response to the Linker's asterisk has the following format:

    *dev:binout,dev:mapout=dev:obj1,dev:obj2,.../sl/s2/s3

where:

    dev:  is a random-access device for all files except dev:mapout, which can be any legal output device. If dev: is not specified, DK is assumed. If the output is to be LDA format (that is, the /L switch was used), the output file need not be on a random-access device.

    binout  is the name to be assigned to the Linker's save image, LDA format, or REL format output file. This file is optional; if not specified, no binary output is produced. (Save image is the assumed output format unless the /L or /R switches are used.)

    mapout  is the optional load map file.

    obj1,...  are files of one or more object modules to be input to the Linker (these may be library files).

    /sl/s2/s3  are switches as explained in Table 6-1 and Section 6.8.

If the /C switch is given, subsequent command lines may be entered as:

    *objm,objn,.../sl/s2

The /C switch is necessary only if the command string will not fit on one line or if the overlay structure is used. If an error occurs in a continued command line (e.g., ?FILE NOT FND?), only the line in error need be retyped.

January 1976 6-2
Linker

If an output file is not specified, the Linker assumes that the associated output is not desired. For example, if the load module and load map are not specified, only error messages (if any) are printed by the Linker.

The default values for each specification are:

<table>
<thead>
<tr>
<th>Device</th>
<th>Filename</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Module DK:</td>
<td>none</td>
<td>SAV, REL(/R), LDA(/L)</td>
</tr>
<tr>
<td>Map Output</td>
<td>none</td>
<td>MAP</td>
</tr>
<tr>
<td>Object Module DK: or same as previous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>object module</td>
<td>none</td>
<td>OBJ</td>
</tr>
</tbody>
</table>

If a syntax error is made in a command string, an error message is printed. A new command string can then be typed following the asterisk.

If a nonexistent file is specified a fatal error occurs; control is returned to the command string interpreter, an asterisk is printed and a new command string may be entered.

6.2.2 Switches

The switches associated with the Linker are listed in Table 6-1. The letter representing each switch is always preceded by the slash character. Switches must appear on the line indicated if the command is continued on more than one line. They may be positioned anywhere on the line. (A more detailed explanation of each switch is provided in Section 6.8.)

Table 6-1
Linker Switches

<table>
<thead>
<tr>
<th>Switch Name</th>
<th>Command Line</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/A</td>
<td>1st</td>
<td>Alphabetizes the entries in the load map.</td>
</tr>
<tr>
<td>/B:n</td>
<td>1st</td>
<td>Bottom address of program is indicated as n (illegal for foreground links).</td>
</tr>
<tr>
<td>/C</td>
<td>any</td>
<td>Continues input specification on another command line. Used also with /O.</td>
</tr>
<tr>
<td>/F</td>
<td>1st</td>
<td>Instructs the Linker to use the default FORTRAN library, FORLIB.OBJ; note that FORLIB does not have to be specified in the command line.</td>
</tr>
<tr>
<td>/I</td>
<td>1st</td>
<td>Includes the global symbols to be searched from the library.</td>
</tr>
<tr>
<td>/L</td>
<td>1st</td>
<td>Produces an output file in LDA format (illegal for foreground links).</td>
</tr>
</tbody>
</table>

(Continued on next page)
Linker

Table 6-1 (Cont.)
Linker Switches

<table>
<thead>
<tr>
<th>Switch Name</th>
<th>Command Line</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/M or /Min</td>
<td>1st</td>
<td>Stack address is to be specified at the terminal keyboard or via n.</td>
</tr>
<tr>
<td>/On</td>
<td>any but the 1st</td>
<td>Indicates that the program will be an overlay structure; n specifies the overlay region to which the module is assigned.</td>
</tr>
<tr>
<td>/R</td>
<td>1st</td>
<td>Produces output in REL format; only files in REL format will run in the foreground (REL format files may not be run under a Single-Job system)</td>
</tr>
<tr>
<td>/S</td>
<td>1st</td>
<td>Allows the maximum amount of space in memory to be available for the Linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.)</td>
</tr>
<tr>
<td>/T or /Tin</td>
<td>1st</td>
<td>Transfer address is to be specified at terminal keyboard or via n.</td>
</tr>
</tbody>
</table>

6.3 ABSOLUTE AND RELOCATABLE PROGRAM SECTIONS

A program produced by one of the RT-11 assemblers or FORTRAN IV can consist of an absolute program section, declared by the .ASECT assembler directive, and relocatable program sections declared by the .CSECT assembler directive. A .CSECT directive is assumed at the beginning of the source program. The instructions and data in relocatable sections are normally assigned locations beginning at 1000 (octal) or 0 for a foreground link. The assignment of addresses can be influenced by command string switches and the size of the absolute section (.ASECT, if present). Each control section is assigned a memory address; the Linker then appropriately modifies all instructions and/or data as necessary to account for the relocation of the control sections.

NOTE

Foreground programs cannot use .ASECTs beyond 1000 (octal); as a general practice, they should be avoided under a Foreground/Background system.

The RT-11 Linker handles the absolute section as well as the named and unnamed control sections. The unnamed control section is internal to each object module. That is, every object module can have an unnamed control section but the Linker treats each control section independently. Each is assigned an absolute address such that it occupies an exclusive area of memory. Named control sections, on the other hand, are treated globally; if different object modules have control sections with the same name, they are all assigned the same absolute load address and the size of the area reserved for loading of the section is the size of the largest. Thus, named control sections allow for the sharing of data and/or instructions among object modules. This is the same as the handling and function of COMMON in FORTRAN IV. The names assigned to control sections are global and can be referenced as any other global symbol.
NOTE

If relocatable code is to be linked for the foreground, no location may be filled more than once (using location counter arithmetic); any such location may be improperly relocated during the FRUN and may cause program or system failure.

For example, the following code illustrates a program using location counter arithmetic that is illegal if linked for the foreground. Note that the code at line 15 starts at 0 due to location counter modification. To correct this program for foreground linking, remove all .=.=. instructions.

*MAIN. RT=11 MACRO VM02=12 PAGE 1

1 000000* .CSECT TEST
2 000000* .GLOBAL A,B,C
3 000000* .MCALL ..V2...,REGDEF
4 000000* ..V2...
5 000000* .REGDEF
6 000000 000000G .WORD A
7 000002 016701 STARTI MOV A+6,R1
     000006G
8 000006 060127 ADD R1,(PC)+
9 000010 000000G .WORD B
10 000002* *=-10
11 000002 000004G .WORD A+4
12 000002* *=-2
13 000002 000002G .WORD B+2
14 000000* *=-4
15 000000 000006G .WORD C+6
16 000002* .END START

6-4.1 January 1976
This page intentionally blank.
6.4 GLOBAL SYMBOLS

Global symbols provide the link, or communication, between object modules. Global symbols are created with the .GLOBL assembler directive (see Chapter 5). If the global symbol is defined in an object module (as a label or by direct assignment), it is called an entry symbol and other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined (as an entry symbol) in some other object module.

As the Linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and/or data which reference the global symbols. Undefined globals are printed on the console terminal after pass 1 (or pass 2 if a library file is also linked).

6.5 INPUT AND OUTPUT

Linker input and output is in the form of modules; one or more input modules (object files produced by either assembler or FORTRAN IV) are used to produce a single output (load) module.

6.5.1 Object Modules

Object files, consisting of one or more object modules, are the input to the Linker (the Linker ignores files which are not object modules). Object modules are created by the RT-11 assemblers or FORTRAN IV. The Linker reads each object module at least twice (three times if library files are linked). During the first pass each object module is read to construct a global symbol table and to assign absolute values to the control section names and global symbols. If library files are linked, a second pass is needed to resolve the undefined globals from the library files and force their associated object modules into the root; on the final pass, the Linker reads the object modules, links and relocates the modules and outputs the load module.

6.5.2 Load Module

The primary output of the Linker is a load module which may be loaded and run under RT-11. The load module is output as a save image file (SAV) for use under a Single-Job system or the background job. Under an F/B System, the /R switch must be used to produce a REL (reloctable) format foreground load module if the job is to run in the foreground. An absolute load module (LDA) is produced if the module is to be loaded by the Absolute Loader.

The load module for a save image file is arranged as follows:

<table>
<thead>
<tr>
<th>Root Segment</th>
<th>Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Segments (optional)</td>
</tr>
</tbody>
</table>

For a REL image file, the load modules are arranged as:

6-5
Linker

<table>
<thead>
<tr>
<th>Root Segment</th>
<th>Overlay Segments (optional)</th>
<th>Resident REL Blocks</th>
<th>Overlay REL Blocks (optional)</th>
</tr>
</thead>
</table>

The first 256-word block of the root segment (main program) contains the memory usage map and the locations used by the Linker to pass program control parameters. The memory usage map outlines the blocks of memory used by the load module and is located in locations 360 to 377.

The control parameters are located in locations 40-50 and contain the following information when the module is loaded:

<table>
<thead>
<tr>
<th>Address</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>40:</td>
<td>Start Address of program</td>
</tr>
<tr>
<td>42:</td>
<td>Initial setting of R6 (stack pointer)</td>
</tr>
<tr>
<td>44:</td>
<td>Job Status Word</td>
</tr>
<tr>
<td>46:</td>
<td>USR Swap Address (0 implies normal location)</td>
</tr>
<tr>
<td>50:</td>
<td>Highest Memory Address in user's program</td>
</tr>
</tbody>
</table>

For a foreground link the following additional parameters contain information:

<table>
<thead>
<tr>
<th>Address</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>34,36:</td>
<td>TRAP Vector</td>
</tr>
<tr>
<td>52:</td>
<td>Size of Resident (words)</td>
</tr>
<tr>
<td>54:</td>
<td>Sum of the Resident and largest Overlay Region (words)</td>
</tr>
<tr>
<td>56:</td>
<td>F/B Identification</td>
</tr>
<tr>
<td>60:</td>
<td>Address of Resident REL Block</td>
</tr>
</tbody>
</table>

Memory locations 0-476 (comprising the interrupt vectors and system communication area) may be assigned initial values by using an .ASECT assembler statement and will appear in block 0 of the load module, but there are restrictions on the use of .ASECTs in this region. The Linker does not permit an .ASECT of location 54 or of locations 360-377 (the memory usage map is passed in those locations). In addition, for foreground links, an .ASECT of words 40-60 is not permitted (additional parameters are passed to FRUN in those locations).

Any location which is not restricted may be set with an .ASECT, but caution should be used in changing the system communication area. Restricted areas, such as the region 360-377, must be initialized by the program itself. There are no restrictions on .ASECTs if the output format is LDA.

Locations in the region 0-476 which are initialized by an .ASECT in a program may never be loaded when the program is executed. There are two reasons for this. For background tasks (or the Single-Job system) the R, RUN, and GET commands will not load an address protected by the monitor's memory protection map. The addresses normally protected include such important areas as the system device and console device vectors, but protection may be extended dynamically (e.g., by a foreground task issuing a .PROTECT call). For foreground tasks, the FRUN command will load only locations 34-50 (34 is the TRAP instruction vector) and all other .ASECTs are ignored. The procedure for loading these locations is to do so at run-time using MOV instructions.

January 1976 6-6
6.5.3 Load Map

If requested, a load map is produced following the completion of the initial pass(es) of the Linker. This map, shown in Figure 6-1, diagrams the layout of memory for the load module.

Each CSECT included in the linking process is listed in the load map. The entry for a CSECT includes the name and low address of the section and its size (in bytes). The remaining columns contain the entry points (or globals) found in the section and their addresses.

The map begins with the name of the load module and the date of creation. The modules located in the root segment of the load module are listed next, followed by those modules which were assigned to overlays in order by their region number (see Section 6.6). Any undefined global symbols are then listed. The map ends with the transfer address (start address) and high limit of relocatable code.

**NOTE**

The load map will not reflect the absolute addresses for a REL file created to be run as a foreground job; the base relocation address specified at FRUN time must be added to obtain the absolute addresses.

For example, assume the FRUN command is used to run the program RETLST:

.\FRUN RETLST/P
LOADED AT 137150

When linked, the following load map is produced:

<table>
<thead>
<tr>
<th>RT-11 LINK</th>
<th>XO3-16</th>
<th>LOAD MAP</th>
<th>XO3-SEP-74</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION</td>
<td>ADDR</td>
<td>SIZE</td>
<td>ENTRY</td>
</tr>
<tr>
<td>.ABS</td>
<td>000000</td>
<td>000350</td>
<td>STKSIZ</td>
</tr>
<tr>
<td>TSTYTI</td>
<td>000350</td>
<td>000326</td>
<td>$FVEC</td>
</tr>
<tr>
<td>$GTB</td>
<td>000676</td>
<td>001116</td>
<td>SCRPOS</td>
</tr>
<tr>
<td>$DBINT</td>
<td>000676</td>
<td>000710</td>
<td>SNR</td>
</tr>
<tr>
<td>$STACK</td>
<td>000764</td>
<td>001106</td>
<td>$DPC</td>
</tr>
<tr>
<td>$LINK</td>
<td>001146</td>
<td>001150</td>
<td>$LCNT</td>
</tr>
<tr>
<td>$DBVEC</td>
<td>001154</td>
<td>001202</td>
<td>$DFILE</td>
</tr>
<tr>
<td>$SYS</td>
<td>001244</td>
<td>001246</td>
<td>$LINK</td>
</tr>
<tr>
<td>$BYPAS</td>
<td>001256</td>
<td>001260</td>
<td>$LSRA</td>
</tr>
<tr>
<td>$CUSER</td>
<td>001302</td>
<td>001316</td>
<td>$XT</td>
</tr>
<tr>
<td>$TY</td>
<td>001330</td>
<td>001404</td>
<td>$LPBUF</td>
</tr>
<tr>
<td>$NRBUF</td>
<td>001446</td>
<td>001562</td>
<td>$XSTOR</td>
</tr>
<tr>
<td>$YSTOR</td>
<td>001574</td>
<td>001670</td>
<td>$VSTIN</td>
</tr>
<tr>
<td>$VSTP</td>
<td>001722</td>
<td>001756</td>
<td>$PEXIT</td>
</tr>
</tbody>
</table>

6-7
To determine the address of TSTVT1, 137150 must be added to 000350; thus 137520 is the absolute address of TSTVT1. The transfer address is 137150 plus 350, or 137520.

6.5.4 Library Files

The RT-11 Linker has the capability of automatically searching libraries. Libraries are composed of library files—specially formatted files produced by the Librarian program (Chapter 7) which contain one or more object modules. The object modules provide routines and functions to aid the user in meeting specific programming needs. (For example, FORTRAN has a special library containing all necessary computational functions—TAN, ATAN, etc.) By using the Librarian, libraries can be created and updated so that routines which are used more than once, or routines which are used by more than one program, may be easily accessed. Selected modules from the appropriate library file are linked as needed with the user program to produce one load module. Libraries are further described in Section 6.7 and in Chapter 7.

NOTE

Library files that have been combined under PIP are illegal as input to both the Linker and the Librarian.
<table>
<thead>
<tr>
<th>RT-11 LINK</th>
<th>V03=01</th>
<th>LOAD MAP</th>
<th>19-SEP-74</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION</td>
<td>ADDR</td>
<td>SIZE</td>
<td>ENTRY ADDR</td>
</tr>
<tr>
<td></td>
<td>000000</td>
<td>000100</td>
<td>000000</td>
</tr>
<tr>
<td></td>
<td>002000</td>
<td>00220</td>
<td>00204</td>
</tr>
<tr>
<td></td>
<td>002400</td>
<td>002604</td>
<td>002604</td>
</tr>
<tr>
<td></td>
<td>005000</td>
<td>000160</td>
<td>005004</td>
</tr>
<tr>
<td></td>
<td>005200</td>
<td>005264</td>
<td>005264</td>
</tr>
<tr>
<td></td>
<td>005500</td>
<td>005536</td>
<td>005536</td>
</tr>
<tr>
<td></td>
<td>005700</td>
<td>005770</td>
<td>005770</td>
</tr>
<tr>
<td></td>
<td>006000</td>
<td>006062</td>
<td>006062</td>
</tr>
<tr>
<td></td>
<td>006100</td>
<td>006154</td>
<td>006154</td>
</tr>
<tr>
<td></td>
<td>006200</td>
<td>006256</td>
<td>006256</td>
</tr>
<tr>
<td></td>
<td>006300</td>
<td>006364</td>
<td>006364</td>
</tr>
<tr>
<td></td>
<td>006400</td>
<td>006472</td>
<td>006472</td>
</tr>
<tr>
<td></td>
<td>006500</td>
<td>006536</td>
<td>006536</td>
</tr>
<tr>
<td></td>
<td>006600</td>
<td>006602</td>
<td>006602</td>
</tr>
<tr>
<td></td>
<td>006700</td>
<td>006712</td>
<td>006712</td>
</tr>
</tbody>
</table>

TRANSFER ADDRESS = 001000
HIGH LIMIT = 016154

Figure 6-1
Linker Load Map for Background Job
6.6 USING OVERLAYS

The RT-11 program overlay facility enables the user to have virtually unlimited memory space for an assembly language or FORTRAN program. A program using the overlay facility can be much larger than would normally fit in the available memory space, since portions of the program (called overlay segments) reside on a backup storage device (disk or DECTape).

The RT-11 overlay scheme is a strict multi-region arrangement; it is not tree-structured. Figure 6-2 diagrams this scheme. The overlay system which the user constructs from his completed program is composed of a root segment, memory-resident overlay regions, and the overlay segments stored on the backup storage device. The root segment is a required part of every overlay program and contains all transfer addresses; it must therefore never be overlaid. An overlay region corresponds to a run-time area of memory that is shared by two or more subroutines (called co-resident subroutines); there is a distinct memory area for each overlay region. Overlay segments are portions of the save image or REL format file from which the user's program is run; these are brought into memory as needed.

\[
\begin{align*}
A & = A/C \quad = \text{Root} \\
B/01 & = \text{Segment 1} \\
C/01 & = \text{Segment 2} \\
D/02 & = \text{Segment 3} \\
E/02 & = \text{Segment 4} \\
\end{align*}
\]

= Region 1

= Region 2

Figure 6-2
Overlay Scheme

Overlay regions are specified to the Linker via the /O switch as described in Section 6.8.8. The size of the overlay region is calculated by the Linker to be the size of the largest group of subroutines that can occupy the region at one time. The Linker creates the overlay regions and edits the program to produce the desired overlays at run-time.

Figure 6-3 shows a diagram of memory for a program which has an overlay structure and Figure 6-4 is a listing of the run-time overlay handler.
Memory Diagram Showing BASIC Link With Overlay Regions

SSTTL THE RUN-TIME OVERLAY HANDLER

THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
LINKER WHENEVER OVERLAYS ARE REQUESTED BY THE USER,
56.8 MICROSECONDS (APPROX) IS ADDED TO EACH REFERENCE OF
A RESIDENT OVERLAY SEGMENT.

THE RUN-TIME OVERLAY HANDLER IS CALLED BY A DUMMY
SUBROUTINE OF THE FOLLOWING FORM:

JSR RS,30VKH ICALL TO COMMON CODE
.WORD <OVERLAY #> IN OF DESIRED SEGMENT
.WORD <ENTRY ADDR> IACTUAL CORE ADDR

ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT
PORTION OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO
AN OVERLAY SEGMENT. ALL REFERENCES TO THE ENTRY POINT ARE
MODIFIED BY THE LINKER TO INSTEAD BE REFERENCES TO THE APPROPRIATE DUMMY ROUTINE. EACH OVERLAY SEGMENT IS CALLED INTO
CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE. AN OVERLAY
Linker

; SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
; OF CORE MEMORY. ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN
; OVERLAY REGION.

; RESTRICTIONS
; SINCE REFERENCES TO OVERLAY SEGMENTS ARE AUTOMATICALLY TRANS-
; LATED BY THE LINKER INTO REFERENCES TO DUMMY SUBROUTINES,
; THE PROGRAMMER MUST NOT ATTEMPT TO REFERENCE DATA IN AN OVER-
; LAY BY USING GLOBAL SYMBOLS.

; SOVTAB=1000+SOVRME=SOVRH
; SOVRME MUV R0,*(SP)
; MUV R1,*(SP)
; MUV R2,*(SP)

; SOVRMBE
; MOV (R0)+,R0
; MOV (R1)+,R1
; ADD $SOVTAB,R1,R1
; MOV (R1)+,R2
; CMP R0,R2
; BEQ $SENTER
; JYES; BRANCH TO IT
; JREAD 17,R2,(R1)+,(R1)+; JREAD FROM OVERLAY FILE
; BCS $SEHR

; $SENTER
; MOV (SP)+,R2
; MOV (SP)+,R1
; MOV (SP)+,R0
; MOV @R0,R5
; RTS R5
; JENTER OVERLAY ROUTINE AND
; JRESTORE USER'S REGS

; $FIRSTI
; MOV #12500,SOVRHB
; JRESTORE SWITCH INSTR
; MOV (PC)+,R1
; JSTART ADDR FOR CLEAR OPERATION

; SHRQTI
; MOV #8,SOVRHB
; JHIGH ADDR OF ROOT SEGMENT
; MOV (PC)+,R2
; JCOUNT

; SHVLTI
; MOV #8,SOVRHB
; JHIGH LIMIT OF OVERLAYS
; CLR (R1)+
; JCLEAR ALL OVERLAY REGIONS

; ISI
; CMP R1,R2
; BLO 18
; JNO; SOVRHB
; JAND RETURN TO CALL IN PROGRESS
; JEMT 379
; JGNERATE ALWAYS FATAL ERROR
; JAND DISREGARD SOFT ERROR

; SOVRME

; OVERLAY SEGMENT TABLE FOLLOWS:
; SOVTAB: 3WORD <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>
; THREE WORDS PER ENTRY, ONE ENTRY PER OVERLAY SEGMENT.

; ALSO, TERE IS ONE WORD PREFIXED TO EACH OVERLAY REGION
; THAT IDENTIFIES THE SEGMENT CURRENTLY RESIDENT IN THAT REGION.
; THIS WORD IS AN INDEX INTO THE SOVTAB TABLE.

Figure 6-4
The Run-Time Overlay Handler

6-12
Linker

There is no special code or function call needed to use overlays but the following rules must be observed when referencing parts of the user program which might be overlaid.

1. Calls or branches to overlay segments must be made directly to entry points in the segment. Entry points are locations tagged with a global symbol (refer to Chapter 5, Section 5.5.10). For example, if ENTER is a global tag in an overlay segment:

   JMP ENTER       is legal, but
   JMP ENTER+6     is illegal.

2. Entries in overlay segments can be used only for transfer of control and not for referencing data within an overlay section (e.g., MOV ENTER,R4 is illegal if ENTER is in an overlay segment, but MOV #ENTER,R7 is legal because it is used for transfer of control). A violation of this rule cannot be detected by the assembler or Linker so no error is issued; however, it can cause the program to use incorrect data.

3. When calls are made to overlays, the entire return path must be in memory. This will happen if these rules are followed:

   Calls (with expected return) may be made from an overlay segment only to entries in the same segment, the root segment, or an overlay segment with a greater region number.

   Calls to entries in the same region as the call must be entirely within the same segment, not another segment in the same region.

   Jumps (with no expected return) can be made from an overlay segment to any entry in the program. However, jumps should not reference an overlay region whose number is lower than the region from which the last unreturned call was made (e.g., if a call was made from region 3, then no jumps should reference regions 1, 2 or 3 until the call has returned).

   Subroutines in the root segment may be called from overlay segments; in turn, they may call entries from the same overlay segment which called them, or from the root segment, or from another overlay segment with a greater region number. Such subroutines are considered part of the overlay segment which called them.

4. A .CSECT name cannot be used to pass control to an overlay. It will not cause the appropriate segment to be loaded into memory (e.g., JSR PC,OVSEC is illegal if OVSEC is used as a .CSECT name in an overlay). As stated in 1 above, a global symbol must be used to pass control from one segment to the next.

5. Channel 17(octal) cannot be used by the user program because overlays are read on that channel.

6. Object modules acquired from a library file cannot be placed into overlays.
7. Library files may not be specified on the same command line as an overlay.

8. Overlay regions must be specified in ascending order and are read-only. Unlike USR swapping, an overlay segment does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are re-initialized to their original values in the SAV or REL file if that segment has been overlaid by another segment. Any variables or tables whose values must be maintained across overlays should be placed in the root segment.

9. .ASECTs of any size in an overlay foreground link are illegal; the error message ?ILL ASEC? is printed and the line is aborted.

The following information should be noted when writing FORTRAN overlays.

1. When dividing a FORTRAN program into a root segment and overlay regions (and subsequently dividing each overlay region into overlay segments), routine placement should be carefully considered. The user should always remember that it is illegal to call a routine located in a different overlay segment in the same overlay region, or an overlay region with a lower numeric value (as specified by the Linker overlay switch, /O:n) from the calling routine. The user should divide each overlay region into overlay segments which never need to be resident simultaneously (i.e., if segments A and B are assigned to region X, they cannot call each other because they occupy the same locations in memory).

2. The FORTRAN main program unit must be placed in the root segment.

3. In an overlay environment, subroutine calls and function subprogram references may refer only to one of the following:
   - A FORTRAN library routine (e.g., ASSIGN, DCOS)
   - A FORTRAN or assembly language routine contained in the root segment
   - A FORTRAN or assembly language routine contained in the same overlay segment as the calling routine
   - A FORTRAN or assembly language routine contained in a segment whose region number is greater than that of the calling routine

4. In an overlay environment, COMMON blocks must be placed so that they are resident when referenced. Blank COMMON is always resident since it is always placed in the root segment. All named COMMON must be placed either in the root segment, or into the segment whose region number is lowest of all segments which reference the COMMON block. A named COMMON block cannot be referenced by two segments in the same region unless the COMMON block appears in a segment of a lower region number. The Linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN main program or by a subprogram that is located in

6-14
the root segment. Otherwise the Linker places a COMMON block in the first segment encountered in the Linker command string that references that COMMON block.

5. All COMMON blocks which are initialized (by use of DATA statements) must be so initialized in the segment in which they are placed.


The .ASECT never takes part in overlaying in any way (i.e., if part of an .ASECT is destroyed by overlay operations, it is not restored by the overlay handler).

The aforementioned sets of rules apply only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP-11 (as described in the PDP-11 Processor Handbook and in Chapter 5).

It should be noted that the condition codes set by a user program are not preserved across overlay segment boundaries.

The Linker provides overlay services by including a small resident overlay handler (Figure 6-4) in the same file with the user program to be used at program run-time. This overlay handler plus some tables are inserted into the user's program beginning at the bottom address computed by the Linker. The Linker moves the user's program up in memory by an appropriate amount to make room for the overlay handler and tables, if necessary.

6.7 USING LIBRARIES

Libraries are specified in a command string in the same fashion as normal modules; they may be included anywhere in the command string, with the exception of overlay lines. If a global symbol is undefined at the time the library is encountered in the input stream and a module is included in the library which includes that global definition, that module is pulled from the library and linked into the load image. Only the modules needed to resolve references are pulled from the library; unused modules are not linked.

NOTE

Modules in one library may call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls SQR from the FORTRAN library. To correctly resolve all globals, the order of ALIB and the FORTRAN library should appear in the command line as:

```bash
$Z=B',ALIB/F
```

or

```bash
$Z=B,ALIB,FORLIB
```

6-15

January 1976
Linker

Module B is the root. It calls X from ALIB and brings X into the root. X in turn calls SQRT which is brought from FORLIB into the root.

FORTRAN libraries cannot precede their root segment in a command line as this creates a bad transfer address. For example:

\*X=ROOT/F
\*X=ROOT,FORLIB

are legal, but:

\*X=FORLIB,ROOT

is not. Unpredictable results will occur.

6.7.1 User Library Searches

Object modules from the named user libraries built by the Librarian are relocated selectively and linked by the Linker. The RT-11 Linker searches a specified library file during the library pass as follows (refer to Figure 6-5 for a flowchart representation of this process):

1. If there are any undefined globals in the Linker's table when a library is encountered in the command string, proceed to step 2; otherwise skip this library (go to step 5).

2. Read the library directory.

3. If any of the undefined globals can be defined by a module in this library, include the relevant module into the linked output file; otherwise, go to step 5.

4. If any undefined globals remain in the Linker's table and they have not been looked for in the library, return to step 2; otherwise go to step 5.

5. Close the library file.

6. Go to the next element in the command string.

This search method allows modules to appear in any order in the library. Any number of libraries may be specified in a link, and they may be positioned anywhere, with the exception of overlay segments and the restrictions noted in Section 6.7.
Figure 6-5
Library Searches

NOTE

For faster Linker performance, the user should specify all object files before library files, and all user library files before the system library files. For example:
Linker

*A:* A + B + USELIB/F

where A and B are object modules, USELIB is a user-created library file, and /F denotes the default FORTRAN library, FORLIB.

Libraries are input to the Linker as any other input file. Assume the following command string to the Linker:

*TASK01.SAV,LF=MAIN.OBJ,MEASUR.OBJ

This causes program MAIN.OBJ to be read from DK: as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The load module, TASK01.SAV is put on DK: and a load map goes to the line printer.

6.8 SWITCH DESCRIPTION

The switches summarized in Table 6-1 are described in detail below.

6.8.1 Alphabetize Switch

The /A switch requests the Linker to list linked modules in alphabetical order as follows: .CSECTS, module names, and entry points within modules. The load map is normally arranged in order by module address as shown in Figure 6-1. Figure 6-6 is an example of an alphabetized load map for a background job.

6.8.2 Bottom Address Switch

The /B switch specifies the lowest address to be used by the relocatable code in the load module. When /B is not specified, the Linker positions the load module so that the lowest address is location 1000 (octal), or 0 for a foreground link. If the .ASECT length is greater than 1000, the length of .ASECT is used.

The form of the bottom switch is:

/B:n

n is a six-digit unsigned octal number which defines the bottom address of the program being linked. An error message results if n is not specified as part of the /B command.

If more than one /B switch is specified during the creation of a load module, the first /B switch specification is used.

NOTE

The bottom value must be an unsigned even octal number. If the value is odd, an error message is generated.

6-18
<table>
<thead>
<tr>
<th>SECTION</th>
<th>ADDR</th>
<th>SIZE</th>
<th>ENTRY</th>
<th>ADDR</th>
<th>ENTRY</th>
<th>ADDR</th>
<th>ENTRY</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>. ABS</td>
<td>00000</td>
<td>00100</td>
<td>$LRECL</td>
<td>000210</td>
<td>$NLCHN</td>
<td>000006</td>
<td>$USHRW</td>
<td>00000</td>
</tr>
<tr>
<td></td>
<td>001000</td>
<td>000220</td>
<td>$STRACE</td>
<td>004737</td>
<td>$V005A</td>
<td>000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>001220</td>
<td>001364</td>
<td>$QTI</td>
<td>001246</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>002504</td>
<td>002300</td>
<td>$OCOS</td>
<td>004164</td>
<td>$ECOS</td>
<td>004156</td>
<td>$FCOS</td>
<td>004152</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$GCOS</td>
<td>004144</td>
<td>$ICIS</td>
<td>002612</td>
<td>$ICOS</td>
<td>003220</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$CGIS</td>
<td>002604</td>
<td>$OCOS</td>
<td>003712</td>
<td>$RCIS</td>
<td>005066</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$IFEST</td>
<td>002772</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>005104</td>
<td>000160</td>
<td>$ISNS</td>
<td>005104</td>
<td>$LSNS</td>
<td>005124</td>
<td>$SISNTW</td>
<td>005110</td>
</tr>
<tr>
<td></td>
<td>005264</td>
<td>000102</td>
<td>$MOISIA</td>
<td>005310</td>
<td>$MOISIM</td>
<td>005304</td>
<td>$MOISIS</td>
<td>005300</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$MOISMA</td>
<td>005324</td>
<td>$MOISMM</td>
<td>005320</td>
<td>$MOISMS</td>
<td>005314</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$MOISSA</td>
<td>005327</td>
<td>$MOISSM</td>
<td>005320</td>
<td>$MOISSS</td>
<td>005264</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$MOISNA</td>
<td>005340</td>
<td>$MOISMN</td>
<td>005334</td>
<td>$MOISNS</td>
<td>005330</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$MOISSA</td>
<td>005360</td>
<td>$MOISMN</td>
<td>005352</td>
<td>$MOISIS</td>
<td>005544</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$MOISIA</td>
<td>005380</td>
<td>$MOISMS</td>
<td>005264</td>
<td>$RELS</td>
<td>005300</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$MOISIE</td>
<td>005406</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>005406</td>
<td>000046</td>
<td>$EOLS</td>
<td>005406</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>005454</td>
<td>000042</td>
<td>$TVOS</td>
<td>005470</td>
<td>$TVFS</td>
<td>005462</td>
<td>$TVIS</td>
<td>005512</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$TVLS</td>
<td>005454</td>
<td>$TVPS</td>
<td>005504</td>
<td>$TVQS</td>
<td>005476</td>
</tr>
<tr>
<td></td>
<td>005536</td>
<td>000036</td>
<td>$CAIS</td>
<td>005536</td>
<td>$CALS</td>
<td>005544</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>005574</td>
<td>000174</td>
<td>$QRT</td>
<td>005574</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>005776</td>
<td>000026</td>
<td>$MOFSRA</td>
<td>006066</td>
<td>$MOFSRM</td>
<td>005776</td>
<td>$MOFSRP</td>
<td>006012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$MOFRS</td>
<td>005770</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>006016</td>
<td>000044</td>
<td>$REGS</td>
<td>006036</td>
<td>$REGS</td>
<td>006046</td>
<td>$BGT$</td>
<td>006044</td>
</tr>
<tr>
<td></td>
<td>006062</td>
<td>000072</td>
<td>$EXIT</td>
<td>006074</td>
<td>$FCOS</td>
<td>006062</td>
<td>$STPS</td>
<td>006074</td>
</tr>
<tr>
<td></td>
<td>006154</td>
<td>000092</td>
<td>$SACTS</td>
<td>006154</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>006256</td>
<td>002637</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERRTB</td>
<td>006156</td>
<td>000100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERRS</td>
<td>006256</td>
<td>002637</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TRANSFER ADDRESS = 001000
HIGH LIMIT = 016154

Figure 6-6
Alphabetized Load Map for a Background Job
Linker

/B is illegal with foreground links. (0 is assumed.)

Example:

*OUTPUT, LP:=INPUT/B:500

Causes the input file to be linked starting at location 500 (octal).

6.8.3 Continue Switch

The Continue switch (/C) is used to allow additional lines of command string input. The /C switch is typed at the end of the current line and may be repeated on subsequent command lines as often as necessary to specify all input modules for which memory is available. If memory is exceeded, an error message is output. A /C switch is not entered on the last line of input.

Example:

*OUTPUT, LP:=INPUT/C
*

Input is to be continued on the next line; the Linker prints an asterisk.

6.8.4 Default FORTRAN Library Switch

By indicating the /F switch in the command line, the FORTRAN library, FORLIB.OBJ on the default device (SY:), is linked with the other object modules specified; the user does not need to specify FORLIB. For example:

*FILE, LP:=AB.OBJ/F

The object module AB.OBJ and the FORTRAN library FORLIB are linked together to form a load module called FILE.SAV. (Note that the FORLIB default is SY:FORLIB.OBJ, not DK:FORLIB.OBJ.)

6.8.5 Include Switch

The /I switch allows subsequent entry at the keyboard of global symbols to be taken from any library and included in the linking process. When the /I switch is specified, the Linker prints:

LIBRARY SEARCH:

Reply with the list of global symbols to be included in the load module; type a carriage return to enter each symbol in the list. A carriage return alone terminates the list of symbols. This provides a method for forcing modules (which are not called by other modules) to be loaded from the library.

Example:

*OUTPUT, LP:=INPUT/XLIB/I

LIBRARY SEARCH:

A <CR>
GETSYM <CR>
CHAR <CR>
CHFLG <CR>
<CR>

User enters A, GETSYM, etc. which are to be included in the linking process. Each symbol is entered by typing a carriage return; the list is terminated by an additional carriage return.

Linker prints LIBRARY SEARCH:
Linker

6.8.6 LDA Format Switch

The LDA format switch (/L) causes the output file to be in LDA format instead of save image format. The LDA format file can be output to any device, including devices that are not block-replaceable such as paper tape or cassette, and is useful for files which are to be loaded with the Absolute Loader. The default extension .LDA is assigned when the /L switch is used.

The /L switch cannot be used in conjunction with the overlay switch (/O) or in foreground links (/R).

Example:

```
* D L O T , L P : = I N , I N 2 / L
```

Links disk files IN and IN2; outputs an LDA format file OUT.LDA to the system device and a load map to the line printer.

6.8.7 Modify Stack Address

The stack address, location 42, is the address which contains the user's stack pointer. The /M switch allows terminal keyboard specification of the user's stack address.

The form of the switch is:

```
/M:n
```

n is an even unsigned 6-digit octal number which defines the stack address. If n is not specified, the Linker prints the message:

```
STACK ADDRESS =
```

In this case, specify the global symbol whose value is the stack address. A number cannot be specified, and if a nonexistent symbol is specified, an error message is printed and the stack address is set to the system default (1000 for save files, 0 for REL).

Direct assignment (via .ASECT) of the stack address within the program takes precedence over assignment with the /M switch.

Example:

```
*OUTPUT=INPUT/M
```

```
STACK ADDRESS = BEG
```

6.8.8 Overlay Switch

The Overlay switch (/O) is used to segment the load module so that the entire program is not memory resident at one time (overlay feature). This allows programs larger than the available memory to be executed. The switch has the form:

```
/O:n
```
where \( n \) is an unsigned octal number (up to six digits in length) specifying the overlay region to which the module is assigned. The /O switch must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line as this is the root segment. Therefore, the /C continuation switch must be used.

Co-resident overlay routines (a group of subroutines which occupy the overlay region and segment at the same time) are specified as follows:

```plaintext
*OBJA,OBJB,OBJC/O:n/C
*OBJD,OBJE/O:m/C
...
```

All modules mentioned until the next /O switch will be co-resident overlay routines. If at a later time the /O switch is given with the same value previously used (same overlay region), then the corresponding overlay area is opened for a new group of subroutines. The new group of subroutines will occupy the same locations in memory as the first group, but not at the same time. For example, if subroutines in object modules R and S are to be in memory together, but are never needed at the same time as T, then the following commands to the Linker make R and S occupy the same memory as T (but at different times):

```plaintext
*MAIN+LP:=ROOT/C
*R/S/O:1/C
*T/O:1
```

The above could also be written as:

```plaintext
*MAIN+LP:=ROOT/C
*R/O:1/C
*S/C
*T/O:1
```

Example:

```plaintext
*OUTPUT+LP:=INPUT/C  Establishes two overlay
*OBJA/O:11/C
*OBJB/O:12
```

Overlays must be specified in order of increasing region number. For example:

```plaintext
.R LINK
*K=A/C
*B/O:1/C
*C/O:1/C
*D/O:1/C
*E,F/O:12/C
G/O:13
```

The following overlay specification is illegal since the overlay regions are given in a random numerical order (an error message is printed in each case):
6.8.9 REL Format Switch

The REL format switch (/R) causes the output file to be in REL format for use as a foreground job under the F/B Monitor. REL format files must be used in a foreground job (they may not be used under a Single-Job system). The /R switch assigns the default extension .REL to the output file.

Example:

```
*DT2;FILE0,LP:=FILE1,NEXT/R
```

Disk files FILE0 and NEXT are linked and output to DT2 as FILE0.REL; a load map is output to the line printer.

The /B and /L switches cannot be used with /R since a foreground REL job has no bottom address and is always relocated by PRUN. A ?BAD SWITCH? error message is generated if this is attempted.

6.8.10 Symbol Table Switch

Use of the symbol table switch in the command line instructs the Linker to allow the largest possible memory area for its symbol table at the expense of making the link process slower. With the /S switch, library directories are not made resident in memory, but are left on disk. For example:

```
*OUTF,LP:=INPUT.OBJ,LIBR1.OBJ,LIBR2.OBJ/S
```

The directories of the library files LIBR1 and LIBR2 are not brought into memory, resulting in more room in the symbol table but longer link time.

If the /S switch is not used and the memory available to the Linker is approximately 10K or larger, the library directory is brought into memory (providing there is room); the directory is kept there until the library has been completely processed, thus reducing the size of the Linker's symbol table. If there is not enough room in memory for the directory (as is the case in an 8K system), the Linker determines this and leaves the directory on disk regardless of whether the /S switch was used or not.

The /S switch should be used only if an attempt to link a program failed because of symbol table overflow. Often, use of /S will allow the program to link.
6.8.11 Transfer Address Switch

The transfer address is the address at which a program is to be started when executed via an R, RUN, or FRUN command. The Transfer Address switch (/T) allows terminal keyboard specification of the start address of the load module to be executed. This switch has the form:

/T: n

where n is a six-digit unsigned octal number which defines the transfer address. If n is not specified, the message:

TRANSFER ADDRESS =

is printed. In this case, specify the global symbol whose value is the transfer address of the load module, followed by a carriage return. A number cannot be specified in answer to this message. When a nonexistent symbol is specified, an error message is printed and the transfer address is set to 1 (so that the program cannot be executed).

If the transfer address specified is odd, the program does not start after loading and control returns to the monitor.

Direct assignment (.AECT) of the transfer address within the program takes precedence over assignment with the /T switch. The transfer address assigned with a /T has precedence over that assigned with a .END assembly directive.

Example:

*PROG=PROG1,PROG2,ODT/T  The files PROG1.OBJ,PROG2.OBJ
TRANSFER ADDRESS =  and ODT.OBJ are linked together and started at ODT's
0:ODT  transfer address.

6.9 LINKER ERROR HANDLING AND MESSAGES

The following error messages can be output by the Linker. The messages enclosed in question marks are output to the terminal; the other messages are only warnings and are included in the load map. If a load map is not requested in the command string, all messages are output to the terminal.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDITIVE REF OF xxxxxx AT SEGMENT # yyyyyy</td>
<td>Rule 1 of overlay rules explained in Section 6.6 has been violated. xxxxxx represents the entry point; yyyyyy represents the segment number.</td>
</tr>
<tr>
<td>*/B NO VALUE?</td>
<td>The */B switch requires an unsigned even octal number as an argument.</td>
</tr>
<tr>
<td>*/B ODD VALUE?</td>
<td>The argument to the */B switch was not an unsigned even octal number.</td>
</tr>
<tr>
<td>?BAD GSD?</td>
<td>There is an error in the global symbol directory (GSD). The file is probably not a</td>
</tr>
</tbody>
</table>

January 1976 6-24
Linker

legal object module. This error message occurs on pass 1 of the Linker.

BAD OVERLAY AT SEG # YYYY
Overlay tries to store text outside its region; check for a .AECT in overlay. YYYY represents the segment number.

?BAD RLD?
There is an invalid relocation directory (RLD) command in the input file; the file is probably not a legal object module. The message occurs on pass 2 of the Linker.

?BAD SWITCH?
LINK did not recognize a switch specified on the first command line. On a subsequent command line, a bad switch causes this warning message but does not restart the Linker.

?BAD x SWITCH IGNORED?
LINK did not recognize a switch (x) specified in the command line. The switch is ignored and linking continues.

BYTE RELOCATION ERROR AT XXXXXX
Linker attempted to relocate and link byte quantities but failed. XXXXXX represents the address at which the error occurred. Failure is defined as the high byte of the relocated value (or the linked value) not being all zero. In such a case, the value is truncated to 8 bits and the Linker continues processing (for save image and LDA files only; byte relocation is completely illegal for REL files).

?CORE?
There is not enough memory to accommodate the command or the resultant load module.

?ERROR ERROR?
An error occurred while the Linker was in the process of recovering from a previous system or user error.

?ERROR IN FETCH?
The device is not available.

?FILE NOT FND?
Input file was not found.

?FORLIB NOT FND?
The user indicated via the /F switch that the FORTRAN library, FORLIB, was to be linked with the other object modules in the command line, and the Linker could not find FORLIB.OBJ on the system device.

?HARD I/O ERROR?
A hardware error occurred; try the operation again.

?ILL AECT?
The user has attempted to place an .AECT above 1000 in a foreground link or to place an .AECT into an overlay foreground link.
There was a hardware problem with the device specified for LDA output or the device was full.

An odd value has been specified for the stack address. Control returns to the Linker and another command line may be indicated.

There was a hardware problem with the device specified for map output or the device is full.

The symbol, xxxxxx, was defined more than once.

No input files were specified.

Overlays have been specified in the wrong order (see overlay restrictions); the overlay switch is ignored.

The output device was full.

The Linker encountered a problem writing the REL file; try the operation again.

The Linker encountered a problem writing the save image file; try the operation again.

The stack address specified by the /M switch was either undefined or in an overlay. The stack address is set to the system default.

There were too many global symbols used in the program. Retry the link using the /S switch. If the error still occurs, the link cannot take place in the available memory.

The Linker allows specification of only two output files.

The transfer address was not defined or was in an overlay.

A load map was requested and undefined globals existed.

The globals listed (xxxxxxx) were undefined. If a load map is requested, this condition also causes the warning message, UNDEF GLBLS, to be printed on the terminal.

January 1976
CHAPTER 7
LIBRARIAN

The RT-11 system provides the user with the capability of maintaining libraries which may be composed of functions and routines of his choice. Each library is a file containing a library header, library directory (or entry point table), and one or more object modules. The object modules in a library file may be routines which are repeatedly used in a program, routines which are used by more than one program, or routines which are related and simply gathered together for ease in usage—the contents of the library file are determined by the user's needs. An example of a typical library file is the FORTRAN library, FORLIB.OBJ. This library is provided with the FORTRAN package and contains all the mathematical functions needed for normal usage.

Object modules in a library file are accessed from another program via calls to their entry points; the object modules are linked with the program which uses them by the Linker (Chapter 6) to produce a single load module.

The RT-11 Librarian (LIBR) allows the user to create, update, modify, list, and maintain library files.

7.1 CALLING AND USING LIBR

The RT-11 Librarian is called from the system device by entering the command:

   R LIBR

in response to the dot printed by the Keyboard Monitor. The Command String Interpreter prints an asterisk at the left margin on the console terminal when it is ready to accept a command line.

Type CTRL C to halt the Librarian at any time and return control to the monitor. To restart the Librarian, type R LIBR or the REENTER command in response to the monitor's dot.
7.2 USER SWITCH COMMANDS AND FUNCTIONS

The user maintains library files through the use of switch commands. Functions which can be performed include object module deletion, insertion and replacement, library file creation, and listing of a library file's contents.

7.2.1 Command Syntax

LIBR accepts command strings in the following general format:

*dev:lib,dev:list=dev:input/s1/s2/s3

where:

dev: represents a legal RT-ll device specification
lib represents the library file to be created or updated
list represents a listing file for the library's contents
input represents the filenames of the input object modules
/s1,... represents one or more of the switches listed in Table 7-1

Devices and filenames are specified by the user in the standard RT-ll command string syntax, with default extensions assigned as follows:

<table>
<thead>
<tr>
<th>File</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>list file:</td>
<td>.LLD</td>
</tr>
<tr>
<td>library file:</td>
<td>.OBJ</td>
</tr>
<tr>
<td>input files:</td>
<td>.OBJ</td>
</tr>
</tbody>
</table>

If no device is specified, the default device (DK:) is assumed.

Each input file is made up of one or more object modules, and is stored on a given device under a specific filename and extension. Once an object module has been inserted into a library file, the module is no longer referenced by the name of the file of which it was a part, but by its individual module name. (This module name has been assigned by the assembler either via a .TITLE statement in the assembly source program, or, if no .TITLE statement is present, with the default name .MAIN.; see Chapter 5.) Thus, for example, the input file FORT.OBJ may exist on DT2: and may contain an object module called ABC. Once the module is inserted into a library file, reference is made only to ABC (not FORT.OBJ).

7.2.2 LIBR Switch Commands

Table 7-1 summarizes the switches available for use under RT-ll LIBR. Switches are explained in detail following the table.
<table>
<thead>
<tr>
<th>Switch</th>
<th>Position In Command String</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/C</td>
<td>Any line but last</td>
<td>Command continuation; the command is too long for the current line and is continued on the next line</td>
</tr>
<tr>
<td>/D</td>
<td>1st line only</td>
<td>Delete; delete modules from a library file</td>
</tr>
<tr>
<td>/G</td>
<td>1st line only</td>
<td>Global deletion; delete entry points from the library directory</td>
</tr>
<tr>
<td>/R</td>
<td>1st line only</td>
<td>Replace; replace modules in a library file</td>
</tr>
<tr>
<td>/U</td>
<td>1st line only</td>
<td>Update; insert and replace modules in a library file</td>
</tr>
</tbody>
</table>

There is no switch to indicate module insertion. The function of inserting a module into a library file is assumed in the absence of other switches.

7.2.2.1 Command Continuation Switch - The Command Continuation switch is necessary whenever there is not enough room to enter a command string on one line and additional lines are needed. The /C switch is typed at the end of the current line and may be repeated at the end of subsequent command lines as often as necessary as long as memory is available; if memory is exceeded, an error message is output. A /C switch is not entered on the last line of input.

Command Format:

*dev:lib,dev:list=dev:input1,dev:input2,...,/C
*dev:inputn

where:

dev: represents a device specification
lib represents the filename of the library to be created or updated
list represents the filename of a listing file containing the library file's contents
input represents the filenames of the input modules to be inserted into the library
/C represents the Continuation switch, indicating that the command is to be continued on the following line
Librarian

Examples:

*ALIB,LIBLST=DT1:MAIN,TEST,FXN/C
*DT1:TRACK

In this example, a library file is created on the default device (DK1) under the filename ALIB.OBJ; a listing of the library file's contents is created as LIBLST.LLD also on the default device; the filenames of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DT1.

*BLIB=MAIN/C
*RK1:TEST/C
*RK0:FXN/C
*DT1:TRACK

A library file is created on the default device, (DK1) under the name BLIB. No listing is produced. Input files are MAIN from the default device, TEST from RK1, FXN from RK0, and TRACK from DT1.

Another way of writing this command line is:

*BLIB=MAIN,RK1:TEST,RK0:FXN/C
*DT1:TRACK

7.2.2.2 Creating a Library File - A library file is created whenever a filename is indicated on the output side of a command line which does not represent a list file.

Command Format:

*dev:lib=dev:input1,...,dev:inputn

where:

dev: represents a device specification
lib represents the filename of the library to be created
input represents the filenames of the input modules to be inserted into the new library

Example:

*NEWLIB=FIRST,SECOND

A new library called NEWLIB.OBJ is created on the default device (DK1). The modules which will make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device.

Assume this command line is next entered:

*NEWLIB,LIST=THIRD,FOURTH

The already existing library file NEWLIB is destroyed when the new library file is created. A listing of the library file's contents is created under the filename LIST, and the object modules in the files THIRD and FOURTH are inserted into the library file NEWLIB.
7.2.2.3 Inserting Modules Into a Library - The Insert function is assumed whenever an input file does not have an associated switch; the modules in the file are inserted into the library file named on the output side of the command string. Any number of input files are allowed. If an attempt is made to insert a file which contains an entry point or .CSECT having the same name as an entry point or .CSECT already existing in the library file, a warning message is printed. However, the library file is updated, ignoring the entry point or .CSECT in error, and control returns to the CSI; the user may enter another command string.

Although the user may insert object modules which exist under the same name (as assigned by the .TITLE statement) this practice is not recommended because of the difficulty involved when replacing or updating these modules (refer to Sections 7.2.2.4 and 7.2.2.7).

NOTE

The library operations of module insertion, replacement, deletion, merge, and update are actually performed in conjunction with the library file creation operation. Therefore, the library file to which the operation is directed must be indicated on both the input and output sides of the command line, since effectively a "new" output library file is created each time the operation is performed. The library file must be specified first in the input field.

Command Format:

*dev:lib=dev:lib,dev:input1,...,dev:inputn

where:

- dev: represents a device specification
- lib represents the filename of an existing library file
- input represents the filenames of the modules to be inserted into the library file

Example:

*XAX=DAX,D71;FA,FB,FC

The modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on D71 are inserted into a library file named DAX.OBJ on the default device. The library header and Entry Point Table of the library file are updated accordingly (see Section 7.4).

7.2.2.4 Replace Switch - The Replace function is used to replace modules in a library file. All modules contained in the file(s) indicated as input will replace existing modules of the same names in the library file specified as output.

7-5 January 1976
An error message is printed and no modules are replaced if an old module does not exist under the same name as an input module, or if the user specifies the /R switch on a library file. /R must follow each input filename containing modules for replacement.

**Command Format:**

```
*dev:lib=dev:lib,input1/R,...,dev:inputn/R
```

where:

- **dev:** represents a device specification
- **lib:** represents the filename of an existing library file
- **input:** represents the names of the files containing modules to be replaced
- **/R:** represents the Replace switch

**Examples:**

```
*TFIL=TFIL,INA,INB/R,INC
```

This command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added. All files are stored on the default device DK:

```
*XFIL=TFIL,INA,INB/R,INC
```

The same operation occurs here as in the preceding example, except that this updated library file is assigned the new name XFIL.

### 7.2.2.5 Delete Switch

The Delete switch deletes modules and all their associated entry points from the library.

**Command Format:**

```
*dev:lib=dev:lib/D
```

where:

- **dev:** represents the device on which the library file exists
- **lib:** represents the filename of an existing library file
- **/D:** represents the Delete switch; may be positioned anywhere on the input side of the command line

When the /D switch is used, the Librarian prints:

**MOD NAME:**
The user should respond with the name of the module to be deleted followed by a carriage return; he may continue until all modules to be deleted have been entered. Typing only a carriage return (either on a line by itself or immediately after the MOD NAME: message) terminates input and initiates execution of the command line.

Examples:

*DT3:TRAP=DT3:TRAP/D

MOD NAME:
SGN <CR>
TAN <CR>
<CR>

The modules SGN.OBJ and TAN.OBJ are deleted from the library file TRAP.OBJ on DT3:

*LIBFIL=LIBFIL/D,ABC/R,DEF

MOD NAME:
FIRST <CR>
<CR>

The module FIRST.OBJ is deleted from the library (LIBFIL); the module ABC.OBJ replaces an old module of the same name in the library, and the modules in the file DEF.OBJ are inserted into the library.

*LIBFIL=LIBFIL/D

MOD NAME:
X<CR>
X<CR>
<CR>

Two modules of the same name are deleted from the library file LIBFIL (module names are assigned with the .TITLE statement as described in Section 7.2.1).

7.2.2.6 Delete Global Switch - The Delete Global switch gives the user the ability to delete a specific entry point from a library file's Entry Point Table.

Command Format:

*dev:lib=dev:lib/G

where:

dev: represents the device on which the library file exists

lib represents the filename of an existing library file
Librarian

/G represents the Delete Global switch; may be positioned anywhere on the input side of the command line.

When the /G switch is used, the Librarian prints:

ENTRY POINT:

The user should respond with the name of the entry point to be deleted followed by a carriage return; he may continue until all entry points to be deleted have been entered. Typing only a carriage return (either on a line by itself or immediately after the ENTRY POINT message) terminates input and initiates execution of the command line.

Example:

*ROLL=ROLL/G

ENTRY POINT:
NAMEA <CR>
NAMEB <CR>
<CR>

This command instructs LIBR to delete the entry points NAMEA and NAMEB from the entry point table found in the library file ROLL.OBJ on DK:

Since entry points are only deleted from the Entry Point Table (and not from the library itself) whenever a library file is updated, all entry points that were previously deleted are restored unless the /G switch is again used to delete them. This feature allows the user to recover from inadvertently deleting the wrong entry point.
7.2.2.7 Update Switch - The Update switch allows the user to update a library file by combining the insert and replace functions. If the object modules included in an input file in the command line already exist in the library file, they are replaced; if not, they are inserted. (No error messages are printed when using the Update function as might occur under the Insert and Replace functions.) /U must follow each input file containing modules to be updated.

Command Format:

*dev:lib=dev:lib,dev:input1/U,...,dev:inputn/U

where:

dev: represents a device specification
lib represents the filename of an existing library file
input represents the names of files containing object modules to be updated.
/U represents the Update switch

Examples:

*BALIB=BALIB+FOLT/U,TAL+BART/U

This command line instructs LIBR to update the library file BALIB.OBJ on the default device. First the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under their names, the modules are inserted. Then the modules from the file TAL.OBJ are inserted; an error message is printed if the name of the module in TAL.OBJ already exists.

*XLIB=XLIB/D,Z/U/G

MOD NAME:
X <CR>
X <CR>
<CR>

ENTRY POINT:
SEC <CR>
SEC1 <CR>
<CR>

There are two object modules of the same name (X) in both Z and XLIB; these are first deleted from XLIB. This ensures that both modules X in file Z are correctly placed into the library. Entry points SEC and SEC1 are also deleted from the Entry Point Table, but automatically return when the library (XLIB) is updated.

7.2.2.8 Listing the Directory of a Library File - The user may specify that a listing of the contents of a library file be output by indicating both the library file and a list file in the command line. Since a library file is not being created or updated, it is not necessary to indicate the filename on the output side of the command line; however a comma must be used to designate a null output library file.
Command Formats:

*.,LP:=dev:lib
or
*.,dev:list=dev:lib

where:

dev: represents a device specification
lib represents the file name of an existing library file
LP: indicates the listing is to be sent directly to the line printer
list represents a list file of the library file's contents

Examples:

*.,DT2:LIST=LIBFIL

This command line outputs to DECtape 2 as LIST.LLD a listing of the contents of the library file LIBFIL.OBJ on the default device.

*.,LP:=FLIB

This command outputs on the line printer a listing of all modules in the library file FLIB.OBJ stored on the default device. Assuming this library is composed of modules STOP, WAIT, and IMUL, is 2 blocks long, was created on September 6, 1974, and the listing was requested on September 6, 1974, the directory format appears as follows:

```
RT=11 LIBRARIA N X82=85
FLIB 6=SEP=74 2 BLOCKS

MODULE ENTRY/CSECT ENTRY/CSECT ENTRY/CSECT
STOP STPS
WAIT SWAIT
IMUL MUISIS MUISIS MUISPS
       MUISSS SMLI
```

7.2.2.9 Merging Library Files - Two or more library files may be merged under one filename by indicating all the library files to be merged in a single command line. The individual library files are not deleted following the merge.

Command Format:

*dev:lib=dev:input1,...,dev:inputn

where:

dev: represents a device specification
lib represents the name of the library file which will contain all the merged files (if a library file
already exists under this name, it must also be indicated in the input side of the command line in order to be included in the merge)

input represents the library files to be merged together

Thus, the command:

```plaintext
*MAIN=MAIN,TRIG,STRP,BAC
```

combines library files MAIN.OBJ, TRIG.OBJ, STRP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device DK:

```plaintext
*FORT=A,B,C
```

This command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the filename FORT.OBJ.

**NOTE**

Library files that have been combined under PIP are illegal as input to both the Librarian and the Linker.

### 7.3 COMBINING LIBRARY SWITCH FUNCTIONS

Two or more library functions may be requested in the same command line. The Librarian performs functions in the following order:

1. /C
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

**Example:**

```plaintext
*FILE,LP=FILE/D,MODX,MODY/R
```

```
MOD NAME:
XYZ <CR>
A <CR>
<CR>
```

Functions in this example are performed in order, as follows:

1. Delete modules XYZ.OBJ and A.OBJ from the library file FILE.OBJ
2. Replace any duplicate of the module in the file MODY.OBJ
3. Insert the modules in the file MODX.OBJ
4. List the contents of FILE.OBJ on the line printer
7.4 FORMAT OF LIBRARY FILES

A library file is a contiguous file consisting of a header, an Entry Point Table (library directory) and one or more library object modules, as illustrated in Figure 7-1:

```
| LIBRARY HEADER |
| ENTRY POINT TABLE |
| OBJECT MODULES |
| LIBRARY END TRAILER BLOCK |
```

Figure 7-1
General Library File Format

The following paragraphs describe in detail each component of a library file.

7.4.1 Library Header

The header section of a library file contains 17 (decimal) words which describe the current status of the file (refer to Figure 7-2). This includes information relating to the version of the Librarian in use, the date and time of file creation or update, the relative starting address of the Entry Point Table (EPT), the number of EPT entries available and in use, and the placing of the next module to be inserted into the library file. The contents of the library header are updated as the library file is modified, so that LIBR can always quickly and easily access the information it needs to perform its functions. Figure 7-2 illustrates the header format.
7.4.2 Entry Point Table (Library Directory)

The Entry Point Table is located immediately after the library header. It is composed of four-word entries which include the names, addresses, and entry points of all object modules in the library file. The first two words of an entry in the EPT contain the Radix 50 name by which an entry point, CSECT, or module is referenced. The third word provides a pointer to the object module where an entry point is defined. The fourth word contains the total number of CSECTs in the object module (information needed by the Linker), and the relative byte within the block pointing to the object module’s starting point, as shown in Figure 7-3.

![Figure 7-2](image)

**Figure 7-2**
Library Header Format

![Figure 7-3](image)

**Figure 7-3**
Format of Entry Point Table
7.4.3 Object Modules

Object modules follow the Entry Point Table. An object module consists of three main types of data blocks: a global symbol directory, text blocks, and a relocation directory. The information contained in these data blocks is used by the Linker during creation of a load module.

7.4.4 Library End Trailer

Following all object modules in a library file is a specially coded library end trailer which signifies the end of the file. This trailer is illustrated in Figure 7-4.

<table>
<thead>
<tr>
<th></th>
<th>FORMATTED BINARY HEADER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<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 7-4
Library End Trailer

7.5 LIBR ERROR MESSAGES

The following error messages are printed following incorrect use of LIBR; if any errors result during library processing, the user must reenter the command.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD LIBR?</td>
<td>The user has attempted to build a library file containing no directory entries or he has given an illegally constructed library file to the Librarian as input.</td>
</tr>
<tr>
<td>BAD OBJ?</td>
<td>A bad object module was detected during input.</td>
</tr>
<tr>
<td>CSECT ERROR?</td>
<td>The user has extended beyond the allowable .CSECT space for an object module to be placed in the library (i.e., the object module contains greater than 127(decimal) .CSECTs).</td>
</tr>
<tr>
<td>Librarian Message</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
</tr>
<tr>
<td>?DEV FULL?</td>
<td>The device is full; LIBR is unable to create or update the indicated library file. The CSI prints an asterisk and waits for the user to enter another command line.</td>
</tr>
<tr>
<td>?FIL NOT FND?</td>
<td>One of the input files indicated in the command line was not found. The CSI prints an asterisk; the command may be reentered.</td>
</tr>
<tr>
<td>?ILL CMD?</td>
<td>An illegal command was used in the command line. The CSI prints an asterisk; the command may be reentered.</td>
</tr>
<tr>
<td>xxxxxx ILL DEL</td>
<td>An attempt was made to delete from the library's directory a module or an entry point that does not exist; xxxxxx represents the module or entry point name. The name is ignored and processing continues.</td>
</tr>
<tr>
<td>?ILL DEV?</td>
<td>An illegal device was specified in the command line. The CSI prints an asterisk; the command may be reentered.</td>
</tr>
<tr>
<td>xxxxxx ILL INS</td>
<td>An attempt was made to insert a module into a library which contains the same entry point as an existing module. xxxxxx represents the entry point name. The entry point is ignored but the module is still inserted into the library.</td>
</tr>
<tr>
<td>xxxxxx ILL REPL</td>
<td>An attempt was made to replace in the library file a module which does not already exist. xxxxxx represents the module name. The module is ignored and the library is built without it.</td>
</tr>
<tr>
<td>?IN ERR?</td>
<td>An unrecoverable hardware/software error has occurred while processing an input file. The CSI prints an asterisk and waits for another command to be entered.</td>
</tr>
<tr>
<td>?LIBR FIL ILL REPL?</td>
<td>The user has specified that a library file be replaced by another library file. Only object modules can be replaced.</td>
</tr>
<tr>
<td>?NO CORE?</td>
<td>Available free memory has been used up. The current command is aborted and the CSI prints an asterisk; a new command may be entered.</td>
</tr>
<tr>
<td>?OUT ERR?</td>
<td>An unrecoverable hardware/software error has occurred while processing an output file. This may indicate that there is not enough space left on the device to create the library file, even though there may be enough directory entries. The CSI prints an asterisk and waits for the user to enter another command.</td>
</tr>
</tbody>
</table>
CHAPTER 8
ON-LINE DEBUGGING TECHNIQUE

RT-11 On-line Debugging Technique (ODT) is a system program that aids in debugging assembled and linked object programs. From the keyboard, the user interacts with ODT and the object program to:

1. Print the contents of any location for examination or alteration.

2. Run all or any portion of an object program using the breakpoint feature.

3. Search the object program for specific bit patterns.

4. Search the object program for words which reference a specific word.

5. Calculate offsets for relative addresses.

6. Fill a single word, block of words, byte or block of bytes with a designated value.

The assembly listing of the program to be debugged should be readily available when ODT is being used. Minor corrections to the program can be made on-line during the debugging session, and the program may then be run under control of ODT to verify any changes made. Major corrections, however (such as a missing subroutine), should be noted on the assembly listing and incorporated in a subsequent updated program assembly.

8.1 CALLING AND USING ODT

ODT is supplied as a relocatable object module. It can be linked with the user program (using the RT-11 Linker) for an absolute area in memory and loaded with the user program. When linked with the user program, ODT should reside in low memory, starting at 1000, to accommodate its stack.

Once loaded in memory with the user program, ODT has three legal start or restart addresses. The lowest (0,ODT) is used for normal entry, retaining the current breakpoints. The next (O,ODT+2) is a restart address which clears all breakpoints and re-initializes ODT saving the general registers and clearing the relocation registers. The last address (O,ODT+4) is used to reenter ODT. A reenter saves the Processor Status and general registers and removes the breakpoint
On-Line Debugging Technique

instructions from the user program. ODT prints the Bad Entry (BE) error message. Breakpoints which were set are reset on the next ;G command. (;P is illegal after a BE message.) The ;G and ;P commands are used to run a program and are explained in Section 8.3.7.

The absolute address used is the address of the entry point O.ODT shown in the Linker load map. O.ODT is always the lowest address of ODT+172, i.e., O.ODT is relative location 172 in ODT.

NOTE

If linked with an overlay structured file, ODT should reside in the root segment so it is always in memory. A breakpoint inserted in an overlay will be destroyed if it is overlaid during program execution.

If ODT is being used in a Foreground/Background environment with another job running, ODT's priority bit must be set to 0 as follows:

```
*$P/000000 0 <CR>
```

This puts ODT into the wait state at level 0, not 7. If this is not done, all interrupts (including clock) will be locked out while ODT is waiting for terminal input.

Examples:

1. ODT Linked with the User Program:

   ```
   GET USER.SAV
   START 1172
   ODT V01-01
   *
   User program previously linked to ODT is brought into memory.
   Value (1172) of entry point O.ODT (determined from Linker load map)
   is used to start ODT.
   ```

2. Loading ODT with the User Program:

   ```
   GET ODT.SAV
   GET USER.SAV
   START 40172
   ODT V01-02
   *
   ODT is loaded into memory.
   User program is loaded into memory.
   Assuming ODT has been linked for a bottom address of 40000, ODT starts.
   ```

3. Restarting ODT Clearing Breakpoints:

   ```
   START 1174
   *
   Assuming ODT was originally linked for a bottom address of 1000,
   this command (0.ODT+2) re-initializes ODT and clears any
   previous breakpoints.
   ```

January 1976

8-2
On-Line Debugging Technique

4. Reentering ODT:

```
.START 1176
BE001212
*Assuming ODT was linked for a bottom address of 1000, the value of .ODT 1172+4 is used as the start address.
```

5. Using ODT with Foreground/Background Jobs:

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the Single-Job Monitor, ODT can be linked with the program as described in Example 1 above.

To debug a program in the foreground area, it is recommended that ODT be run in the background while the program to be debugged is in the foreground. The sequence of commands to do this is:

```
.FRUN PROG/P
LOADED AT xxxxxx
RUN ODT

ODT V01-01
*xxxxxx; OR
*SF/000000 0<CR>
*8:G
*RSU
```

Load the foreground program. The first address of the job is printed (xxxxxx)

Run ODT in the background and set a relocation register to the start of the job. SF is the format register. It should be cleared to enable proper address print out. 0;G starts the Keyboard Monitor again, and .RSU starts the foreground job.

The copy of ODT used must be linked low enough so that it will fit in memory along with the foreground job.

NOTE

Since ODT uses its own terminal handler, it cannot be used with the display hardware. If GT ON has been typed, ODT will ignore it and direct I/O only to the console terminal.

8.1.1 Return to Monitor, CTRL C

If ODT is awaiting a command, a CTRL C from the keyboard calls the RT-ll Keyboard Monitor. The monitor responds with a ^C on the terminal and awaits a Keyboard Monitor command. (The monitor REENTER command may be used to reenter ODT only if the user program has set the reenter bit. Otherwise ODT is reentered at address O.ODT+4 as shown above.)
On-Line Debugging Technique

8.1.2 Terminate Search, CTRL U

If typed during a search printout, a CTRL U terminates the search and ODT prints an asterisk.

8.2 RELOCATION

When the assembler produces a binary object module, the base address of the module is taken to be location 000000, and the addresses of all program locations as shown in the assembly listing are indicated relative to this base address. After the module is linked by the Linker, many values within the program, and all the addresses of locations in the program, will be incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, RT-ll ODT provides an automatic relocation facility.

The basis of the relocation facility lies in eight relocation registers, numbered 0 through 7, which may be set to the values of the relocation biases at different times during debugging. Relocation biases should be obtained by consulting the memory map produced by the Linker. Once set, a relocation register is used by ODT to relate relocatable code to relocated code. For more information on the exact nature of the relocation process, consult Chapter 6, the RT-ll Linker.

8.2.1 Relocatable Expressions

A relocatable expression is evaluated by ODT as a 16-bit (6-digit octal) number and may be typed in any one of the three forms presented in Table 8-1. In this table, the symbol n stands for an integer in the range 0 to 7 inclusive, and the symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If more than six digits are typed, ODT takes the last six digits, truncated to the low-order 16 bits. k may be preceded by a minus sign, in which case its value is the two's complement of the number typed. For example:

<table>
<thead>
<tr>
<th>k (number typed)</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000001</td>
</tr>
<tr>
<td>-1</td>
<td>177777</td>
</tr>
<tr>
<td>400</td>
<td>0000400</td>
</tr>
<tr>
<td>-177730</td>
<td>000050</td>
</tr>
<tr>
<td>1234567</td>
<td>034567</td>
</tr>
</tbody>
</table>
## On-Line Debugging Technique

### Table 8-1

<table>
<thead>
<tr>
<th>r</th>
<th>Value of r</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) k</td>
<td>The value of r is simply the value of k.</td>
</tr>
<tr>
<td>B) n,k</td>
<td>The value of r is the value of k plus the contents of relocation register n. If the n part of this expression is greater than 7, ODT uses only the last octal digit of n.</td>
</tr>
<tr>
<td>C) C or C, k or n, C or C, C</td>
<td>Whenever the letter C is typed, ODT replaces C with the contents of a special register called the Constant Register. This value has the same role as the k or n that it replaces (i.e., when used in place of n it designates a relocation register). The Constant Register is designated by the symbol $C$ and may be set to any value, as indicated below.</td>
</tr>
</tbody>
</table>

In the following examples, assume in each case that relocation register 3 contains 003400 and that the constant register contains 000003.

<table>
<thead>
<tr>
<th>r</th>
<th>Value of r</th>
</tr>
</thead>
<tbody>
<tr>
<td>5:C</td>
<td>000005</td>
</tr>
<tr>
<td>-17:C</td>
<td>177761</td>
</tr>
<tr>
<td>3,0:C</td>
<td>003400</td>
</tr>
<tr>
<td>3,150:C</td>
<td>003550</td>
</tr>
<tr>
<td>3,-1:C</td>
<td>003377</td>
</tr>
<tr>
<td>C:C</td>
<td>000003</td>
</tr>
<tr>
<td>3, C; C</td>
<td>003403</td>
</tr>
<tr>
<td>C,0:C</td>
<td>003400</td>
</tr>
<tr>
<td>C,10; C</td>
<td>003410</td>
</tr>
<tr>
<td>C, C; C</td>
<td>003403</td>
</tr>
</tbody>
</table>

**NOTE**
For simplicity most examples in this section use Form A. All three forms of r are equally acceptable, however.

### 8.3 Commands and Functions

When ODT is started (as explained in Section 8.1) it indicates readiness to accept commands by printing an asterisk on the left margin of the terminal page. Most of the ODT commands can be issued in response to the asterisk. For example, a word can be examined and changed if desired, the object program can be run in its entirety or in segments, or memory can be searched for certain words or references to certain words. The discussion below explains these features. In the following examples, characters output by ODT are underlined to differentiate from user input.

#### 8.3.1 Printout Formats

Normally, when ODT prints addresses (as with the commands ↓, ↑, ←, →, @, <, and >) it attempts to print them in relative form (Form B in Table
8-1). ODT looks for the relocation register whose value is closest but less than or equal to the address to be printed, and then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address is printed in absolute form. Since the relocation registers are initialized to -1 (the highest number) the addresses are initially printed in absolute form. If any relocation register subsequently has its contents changed, it may then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain numbers much higher. Then the following sequence might occur (the slash command causes the contents of the location to be printed; the line feed command (\texttt{<LF>}) accesses the next sequential location):

\begin{verbatim}
\texttt{#774/000000 <LF>}
\texttt{000775/000000 <LF>}
\texttt{1.000000/000000 <LF> (absolute location 1000)}
\texttt{1.000002/000000 <LF> (absolute location 1002)}
\texttt{2.000000/000000 (absolute location 1004)}
\end{verbatim}

The printout format is controlled by the format register, \texttt{SF}. Normally this register contains 0, in which case ODT prints addresses relatively whenever possible. \texttt{SF} may be opened and changed to a non-zero value, however, in which case all addresses will be printed in absolute form (see paragraph 8.3.4, Accessing Internal Registers).

8.3.2 Opening, Changing, and Closing Locations

An open location is one whose contents ODT prints for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command used to open a location when another location is already open causes the currently open location to be closed. The contents of an open location may be changed by typing the new contents followed by a single-character command which requires no argument (i.e., \texttt{<LF>}, \texttt{t}, \texttt{RETURN}, \texttt{+, @, >, <}).

The Slash, /

One way to open a location is to type its address followed by a slash:

\begin{verbatim}
\texttt{*1000/012746}
\end{verbatim}

Location 1000 is open for examination and is available for change.

If the contents of an open location are not to be changed, type the \texttt{RETURN} key and the location is closed; ODT prints an asterisk and waits for another command. However, to change the word, simply type the new contents before giving a command to close the location:

\begin{verbatim}
\texttt{*1000/012746 012345 <CR>}
\texttt{t}
\end{verbatim}
On-Line Debugging Technique

In the example above, location 1000 now contains 012345 and is closed since the RETURN key was typed after entering the new contents, as indicated by ODT's second asterisk.

Used alone, the slash reopens the last location opened:

    *1000/012345 2340 <CR>
    */002340

In the example above, the open location was closed by typing the RETURN key. ODT changed the contents of location 1000 to 002340 and then closed the location before printing the *. The single slash command directed ODT to reopen the last location opened. This allowed verification that the word 002340 was correctly stored in location 1000.

Note again, that opening a location while another is open automatically closes the currently open location before opening the new location.

Also note that if an odd numbered address is specified with a slash, ODT opens the location as a byte, and subsequently behaves as if a backslash had been typed (see the following paragraph).

The Backslash, \

In addition to operating on words, ODT operates on bytes. One way to open a byte is to type the address of the byte followed by a backslash. (On the LT33 or LT35 terminal \ is typed by pressing the SHIFT key while typing the L key.) This causes not only the printing of the byte value at the specified address but also the interpreting of the value as ASCII code, and the printing of the corresponding character (if possible) on the terminal:

    *1001/101 =A

A backslash typed alone reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

    *1002/000004 \004 =

The LINE FEED Key, <LF>

If the LINE FEED key is typed when a location is open, ODT closes the open location and opens the next sequential location:

    *1000/002340 <LF> ( <LF> denotes typing the LINE FEED key)
    001002 /012740

In this example, the LINE FEED key caused ODT to print the address of the next location along with its contents, and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is open. The open location may be modified by typing the new contents.

If a byte location was open, typing the LINE FEED key opens the next byte location.
On-Line Debugging Technique

The Up-Arrow, ↑ or ^

If the up-arrow (or circumflex) is typed when a location is open (up-arrow is produced on an LT33 or LT35 by typing SHIFT N), ODT closes the open location and opens the previous location. To continue from the example above:

```
+001002/012740 ↑
001000 /002340
```

Now location 1002 is closed and 1000 is open. The open location may be modified by typing the new contents.

If the opened location was a byte, then up-arrow opens the previous byte.

The Back-Arrow, ← or _

If the back-arrow (or underline) is typed (via SHIFT O on an LT33 or LT35 terminal) to an open word, ODT interprets the contents of the currently open word as an address indexed by the Program Counter (PC) and opens the addressed location:

```
*1006/000006 ←
001016 /000405
```

Notice in this example that the open location, 1006, was indexed by the PC as if it were the operand of an instruction with address mode 67 as explained in Chapter 5.

A modification to the opened location can be made before a line feed, up-arrow, or back-arrow is typed. Also, the new contents of the location will be used for address calculations using the back-arrow command. Example:

```
*100/000222 .4 <LF> (modify to 4 and open next location)
00102 /00111 6↑ (modify to 6 and open previous location)
00100 /000004 1000 (change to 100 and open location indexed by PC)
000202 /123456
```

Open the Addressed Location, @

The at symbol @ (SHIFT P on the LT33 or LT35 terminal) may be used to optionally modify a location, close it, and then use its contents as the address of the location to open next.

```
*1006/001044 @
001044 /000500 (open location 1044 next)
```

```
*1006/001044 2100@
002100 /000167 (modify to 2100 and open location 2100)
```

8-8
On-Line Debugging Technique

Relative Branch Offset, >

The right-angle bracket, >, will optionally modify a location, close it, and then use its low-order byte as a relative branch offset to the next word to be opened. For example:

\[
\text{\#1032/000407_301> (modify to 301 and interpret as a relative branch)}
\]

\[
\text{00636 /000010}
\]

Note that 301 is a negative offset (-77). The offset is doubled before it is added to the PC; therefore, 1034+(-176)=636.

Return to Previous Sequence, <

The left-angle bracket, <, allows the user to optionally modify a location, close it, and then open the next location of the previous sequence which was interrupted by a back-arrow, @, or right-angle bracket command. Note that back-arrow, @, or right-angle bracket causes a sequence change to the word opened. If a sequence change has not occurred, the left-angle bracket simply opens the next location as a LINE FEED does. This command operates on both words and bytes.

\[
\text{\#1032/000407_301> (} \text{causes a sequence change)}
\]

\[
\text{00636 /000010 < (return to original sequence)}
\]

\[
\text{001034 /001040 @ (} \text{@} \text{causes a sequence change)}
\]

\[
\text{001048 /000405 } \text{\&005 < (} \text{now operates on byte)}
\]

\[
\text{001035 } \text{\&002 < (} \text{acts like } \text{<LF>)}
\]

8.3.3 Accessing General Registers 0-7

The program's general registers 0-7 are opened with a command in the following format:

\[
\text{\#$n/}
\]

where n is the integer representing the desired register (in the range 0 through 7). When opened, these registers can be examined or changed by typing in new data as with any addressable location. For example:

\[
\text{\#$0/000032 <CR> (R0 was examined and closed)}
\]

\[
\text{\#$4/000474_464<CR> (R4 was opened, changed, and closed)}
\]

The example above can be verified by typing a slash in response to ODT's asterisk:

\[
*/000464
\]

The LINE FEED, up-arrow, back-arrow or @ command may be used when a register is open.
On-Line Debugging Technique

8.3.4 Accessing Internal Registers

The program's Status Register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. It is opened by typing $$S$. For example:

```
*$$5/000311
```

$$S$$ represents the address of the Status Register. In response to $$S$$ in the example above, ODT printed the 16-bit word, of which only the low-order eight bits are meaningful. Bits 0-3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted, and bits 5-7 indicate the interrupt priority level (in the range 0-7) of the object program. (Refer to the PDP-11 Processor Handbook for the Status Register format.)

The $ is used to open certain other internal locations listed in Table 8-2:

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B</td>
<td>location of the first word of the breakpoint table (see Section 8.3.6).</td>
</tr>
<tr>
<td>$M</td>
<td>mask location for specifying which bits are to be examined during a bit pattern search (see Section 8.3.9).</td>
</tr>
<tr>
<td>$P</td>
<td>location defining the operating priority of ODT (see Section 8.3.15).</td>
</tr>
<tr>
<td>$S</td>
<td>location containing the condition codes (bits 0-3) and interrupt priority level (bits 5-7) (explained above).</td>
</tr>
<tr>
<td>$C</td>
<td>location of the Constant Register (see Section 8.3.10).</td>
</tr>
<tr>
<td>$R</td>
<td>location of Relocation Register 0, the base of the Relocation Register table (see Section 8.3.13).</td>
</tr>
<tr>
<td>$F</td>
<td>location of Format Register (see Section 8.3.1).</td>
</tr>
</tbody>
</table>

8.3.5 Radix 50 Mode, X

The Radix 50 mode of packing certain ASCII characters three to a word is employed by many DEC-supplied PDP-11 system programs, and may be employed by any programmer via the MACRO Assembler's "RAD50" directive. ODT provides a method for examining and changing memory words packed in this way with the X command.

When a word is opened and the X command is typed, ODT converts the contents of the opened word to its 3-character Radix 50 equivalent and prints these characters on the terminal. One of the responses in Table 8-3 can then be typed:
On-Line Debugging Technique

Table 8-3
Radix 50 Terminators

<table>
<thead>
<tr>
<th>Response</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN key &lt;CR&gt;</td>
<td>Closes the currently open location.</td>
</tr>
<tr>
<td>LINE FEED key &lt;LF&gt;</td>
<td>Closes the currently open location and opens the next one in sequence.</td>
</tr>
<tr>
<td>↑ key</td>
<td>Closes the currently open location and opens the previous one in sequence.</td>
</tr>
<tr>
<td>Any three characters whose</td>
<td>Converts the three specified characters into packed Radix 50 format.</td>
</tr>
<tr>
<td>octal code is 040 (space) or</td>
<td></td>
</tr>
<tr>
<td>greater.</td>
<td></td>
</tr>
<tr>
<td>Legal Radix 50 characters for</td>
<td></td>
</tr>
<tr>
<td>this last response are:</td>
<td></td>
</tr>
<tr>
<td>. $ Space 0 through 9 A through Z</td>
<td></td>
</tr>
</tbody>
</table>

If any other characters are typed, the resulting binary number is unspecified (that is, no error message is printed and the result is unpredictable). Exactly three characters must be typed before ODT resumes its normal mode of operation. After the third character is typed, the resulting binary number is available to be stored in the open location by closing the location in any one of the ways listed in Table 8-3. Example:

*1000/042431 M=KB1_CBA <CR>
*1000/011421 M=CBA

NOTE

After ODT has converted the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command which follows. For example:

*1234/063337 X=PRO XIT/013704

Since the Radix 50 equivalent of XIT is 113574, the final slash in the example will cause ODT to open location 113574 if it is a legal address. (Refer to paragraph 8.5 for a discussion of command legality and detection of errors.)

8.3.6 Breakpoints

The breakpoint feature facilitates monitoring the progress of program execution. A breakpoint may be set at any instruction which is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a trap instruction so that program execution is suspended when a breakpoint
On-Line Debugging Technique

is encountered. The original contents of the breakpoint location are restored, and ODT regains control.

With ODT, up to eight breakpoints, numbered 0 through 7, can be set at any one time. A breakpoint is set by typing the address of the desired location of the breakpoint followed by ;B. Thus r:B sets the next available breakpoint at location r. (If all 8 breakpoints have been set, ODT ignores the r:B command.) Specific breakpoints may be set or changed by the r:nB command where n is the number of the breakpoint. For example:

```
*1020:B (sets breakpoint 0)
*1030:B (sets breakpoint 1)
*1040:B (sets breakpoint 2)
*1032:1B (resets breakpoint 1)
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where n is the number of the breakpoint. For example:

```
*;2B (removes the second breakpoint)
```

A table of breakpoints is kept by ODT and may be accessed by the user. The $B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order, and can be sequentially opened using the LINE FEED key. For example:

```
$$8/001020 <LF>
nnnnn /001032 <LF>
nnnnn /nnnnnn (nnnnnn=address internal to ODT)
```

In this example, breakpoint 2 is not set. The contents printed is an address internal to ODT and can be determined by checking the Linker Load Map (see Chapter 6).

It should be noted that a repeat count in a Proceed command refers only to the breakpoint that has most recently occurred. Execution of other breakpoints encountered is determined by their own repeat counts.

8.3.7 Running the Program, r;G and r;P

Program execution is under control of ODT. There are two commands for running the program: r;G and r;P. The r;G command is used to start execution (Go) and r;P to continue (Proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

Execution is started at location 1000. The program runs until a breakpoint is encountered or until program completion, unless it gets caught in an infinite loop, in which case it must be either restarted or reentered as explained in Section 8.1.

Upon execution of either the r;G or r;P command, the general registers 0-6 are set to the values in the locations specified as $0-$6 and the
On-Line Debugging Technique

processor Status Register is set to the value in the location specified as $S$.

When a breakpoint is encountered, execution stops and ODT prints $Bn$ (where $n$ is the breakpoint number), followed by the address of the breakpoint. Locations can then be examined for expected data. For example:

```
*1010:3B
*1000:G
B3:001010
```

(breakpoint 3 is set at location 1010)
(execution started at location 1000)
(execution stopped at location 1010)

To continue program execution from the breakpoint, type ;P in response to ODT's last *.

When a breakpoint is set in a loop, it may be desirable to allow the program to execute a certain number of times through the loop before recognizing the breakpoint. This can be done by setting a proceed count using the k;P command; this command specifies the number of times the breakpoint is to be encountered before program execution is suspended (on the kth encounter). The count, $k$, refers only to the numbered breakpoint which most recently occurred. A different proceed count may be specified for the breakpoint when it is encountered. Thus:

```
B3:001010
*1026:3B
$4: P
B3:001026
```

(execution halted at breakpoint 3)
(reset breakpoint 3 at location 1026)
(set proceed count to 4 and continue execution; loop through breakpoint three times and halt on fourth occurrence of the breakpoint)

Following the table of breakpoints (as explained in Section 8.3.6) is a table of proceed command repeat counts for each breakpoint. These repeat counts can be inspected by typing $SB/$ and nine LINE FEEDS. The repeat count for breakpoint 0 is printed (the first seven LINE FEEDs cause the table of breakpoints to be printed; the eighth types the single instruction mode, explained in the next section, and the ninth LINE FEED begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7, and the repeat count for the single-instruction trap follow in sequence. Before a proceed count is assigned a value by the user, it is set to 0; after the count has been executed, it is set to -1. Opening any one of these provides an alternative way of changing the count as the location, once open, can have its contents modified in the usual manner by typing the new contents and then the RETURN key. For example:

```
nnnnn /001036 <LF> (address of breakpoint 7)
nnnnn /005530 <LF> (single instruction address)
nnnnn /000000 15 <LF> (count for breakpoint 0; change to 15)
nnnnn /000000 <LF> (count for breakpoint 1)
nnnnn /000000 <LF> (count for breakpoint 7)
nnnnn /nnnnnn <LF> (repeat count for single instruction mode; the single instruction address)
```

8-13
On-Line Debugging Technique

is an address internal to the user program if single instrucion mode is used

The address indicated as the single instruction address and the repeat count for single instruction mode are explained next.

8.3.8 Single Instruction Mode

With this mode the number of instructions to be executed before suspension of the program run can be specified. The Proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled when single instruction mode is operative.

Commands for single instruction mode are:

;NS Enables single instruction mode (n can have any non-zero value and serves only to distinguish this form from the form ;S). Breakpoints are disabled.

nP Proceeds with program run for next n instructions before reentering ODT (if n is missing, it is assumed to be 1). Trap instructions and associated handlers can affect the Proceed repeat count. See Section 8.4.2.

;S Disables single instruction mode.

When the repeat count for single instruction mode is exhausted and the program suspends execution, ODT prints:

B8;nnnnnn

where nnnnnnn is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0-7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single instruction mode. This table entry may be directly modified and thus is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than an assumed 1.

8.3.9 Searches

With ODT all or any specified portion of memory can be searched for any specific bit pattern or for references to a particular location.
On-Line Debugging Technique

Word Search, r;W

Before initiating a word search, the mask and search limits must be specified. The location represented by $M$ is used to specify the mask of the search. $M$/ opens the mask register. The next two sequential locations (opened by LINE FEEDS) contain the lower and upper limits of the search. Bits set to 1 in the mask are examined during the search; other bits are ignored. Then the search object and the initiating command are given using the r;W command where r is the search object. When a match is found, (i.e., each bit set to 1 in the search object is set to 1 in the word being searched over the mask range) the matching word is printed. For example:

```plaintext
$M/000000 177400 <LF>
nnnnn /000000 1000 <LF>
nnnnn /000000 1040 <CR>
*400:W
001010 /000770
001034 /000404
```

In the above example, nnnnnn is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line above, the slash was used to open $M$ which now contains 177400; the LINE FEEDS opened the next two sequential locations which now contain the upper and lower limits of the search.

In the search process an exclusive OR (XOR) is performed with the word currently being examined and the search object, and the result is ANDed to the mask. If this result is zero, a match has been found and is reported on the terminal. Note that if the mask is zero, all locations within the limits are printed.

Typing CTRL U during a search printout terminates the search.

Effective Address Search, r;E

ODT provides a search for words which address a specified location. Open the mask register only to gain access to the low and high limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where r is the effective address) to initiate the search.

Words which are either an absolute address (argument r itself), a relative address offset, or a relative branch to the effective address, are printed after their addresses. For example:

```plaintext
$M/177400 <LF>
nnnnn /001000 1810 <LF>
nnnnn /001040 1060 <CR>
*1034:E
001016 /001006
001054 /002767
*1020:E
001022 /177774
001030 /001020
```

(open mask register only to gain access to search limits)

(initiating search)

(relative branch)

(initiating a new search)

(relative address offset)

(absolute address)
On-Line Debugging Technique

Particular attention should be given to the reported effective address references because a word may have the specified bit pattern of an effective address without actually being so used. ODT reports all possible references whether they are actually used as such or not.

Typing CTRL U during a search printout terminates the search.

8.3.10 The Constant Register, \$r;C

It is often desirable to convert a relocatable address into its value after relocation or to convert a number into its two's complement, and then to store the converted value into one or more places in a program. The Constant Register provides a means of accomplishing this and other useful functions.

When \$r;C is typed, the relocatable expression \$r is evaluated to its 6-digit octal value and is both printed on the terminal and stored in the Constant Register. The contents of the Constant Register may be invoked in subsequent relocatable expressions by typing the letter C. Examples follow:

\$-4432;C=173346  \hspace{1cm} (the two's complement of 4432 is placed in the Constant Register)

\$6632/062701\$C  \hspace{1cm} (the contents of the Constant Register are stored in location 6632)

\$1000;1R  \hspace{1cm} (relocation Register 1 is set to 1000)

\$1.4272;C=005272  \hspace{1cm} (relative location 4272 is reprinted as an absolute location and stored in the Constant Register)

8.3.11 Memory Block Initialization, \$F and \$I

The Constant Register can be used in conjunction with the commands \$F and \$I to set a block of memory to a given value. While the most common value required is zero, other possibilities are plus one, minus one, ASCII space, etc.

When the command \$F is typed, ODT stores the contents of the Constant Register in successive memory words starting at the memory word address specified in the lower search limit, and ending with the address specified in the upper search limit.

When the command \$I is typed, the low-order 8 bits in the Constant Register are stored in successive bytes of memory starting at the byte address specified in the lower search limit and ending with the byte address specified in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000-7776 to zero, and byte locations 10000-14777 to ASCII spaces.
On-Line Debugging Technique

(open mask register to gain access to search limits)
(set lower limit to 7000)
(set upper limit to 7776)
(Constant Register set to zero)
(Locations 7000-7776 set to zero)

(set lower limit to 10000)
(set upper limit to 14777)
(Constant Register set to 40
(SPACE))
(Byte locations 10000-14777 are set to value in low-order 8 bits of Constant Register)

8.3.12 Calculating Offsets, r;0

Relative addressing and branching involve the use of an offset—the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it may be necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;0 command.

The command r;0 causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/000034 414;0 000044 022 22 <CR>
*000022
```

In the example, location 346 is opened and the offsets from that location to location 414 are calculated and printed. The contents of location 346 are then changed to 22 (the 8-bit offset) and verified on the next line.

The 8-bit offset is printed only if it is in the range -128(decimal) to 127(decimal) and the 16-bit offset is even, as was the case above. For example, the offset of a relative branch is calculated and modified as follows:

```
*1034/103421 1034;0 177776 377 \021 = 377 <CR>
*103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

8.3.13 Relocation Register Commands, r;nR, ;nR, ;R

The use of the relocation registers is defined in Section 8.2. At the beginning of a debugging session it is desirable to preset the registers to the relocation biases of those relocatable modules which will be receiving the most attention.

This can be done by typing the relocation bias, followed by a semicolon and the specification of relocation registers, as follows:
On-Line Debugging Technique

\texttt{r;nR}

\(r\) may be any relocatable expression and \(n\) is an integer from 0 to 7. If \(n\) is omitted it is assumed to be 0. As an example:

\begin{verbatim}
*1000:5R  (puts 1000 into relocation register 5)
*5.100:5R  (effectively adds 100 to the contents
*       of relocation register 5)
\end{verbatim}

Once a relocation register is defined, it can be used to reference relocatable values. For example:

\begin{verbatim}
*2000:1R  (puts 2000 into relocation register 1)
*1,2176582466  (examines contents of location 4176)
*1,3712:06  (sets a breakpoint at location 5712)
\end{verbatim}

In certain uses, programs may be relocated to an address below that at which they were assembled. This could occur with PIC code (Position Independent Code) which is moved without the use of the Linker. In this case the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Assume a program was assembled at location 5000 and was moved to location 1000. Then the sequence:

\begin{verbatim}
*1000:1R
*1,-5000:1R
\end{verbatim}

enters the two's complement of 4000 in relocation register 1, as desired.

Relocation registers are initialized to -1, so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to -1, type ;nR. To set all relocation registers to -1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register 0. Successively typing a line feed opens the other relocation registers in sequence. When a relocation register is opened in this way, it may be modified like any other memory location.

8.3.14 The Relocation Calculators, nR and n!

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, type n!, where \(n\) specifies the relocation register. This calculator works with opened bytes and words. If \(n\) is omitted, the relocation register whose contents are closest but less than or equal to the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 2,
On-Line Debugging Technique

which contains 2000. To find the most likely module that a given
opened byte is in:

\*2500\811 = \!2,000500

Typing nR after opening a word causes ODT to print the octal number
which equals the value of the contents of the opened location minus
the contents of relocation register n. If n is omitted, ODT selects
the relocation register whose contents are closest but less than or
equal to the contents of the opened location. For example, assume the
relocation bias stored in relocation register 1 is 7000; then:

\*1,500/000000 1R=1,171000

The value 171000 is the content of 1,500, relative to the base 7000.
An example of the use of both relocation calculators follows.

If relocation register 1 contains 1000, and relocation register 2
contains 2000, then to calculate the relocatable addresses of location
3000 and its contents, relative to 1000 and 2000, the following can be
performed.

\*3000/000410 1\!1=1,002000 2\!2=2,001000 1R=1,177410 2R=2,176410

8.3.15 ODT Priority Level, SP

SP represents a location in ODT that contains the interrupt (or
processor) priority level at which ODT operates. If SP contains the
value 377, ODT operates at the priority level of the processor at the
time ODT is entered. Otherwise SP may contain a value between 0 and 7
corresponding to the fixed priority at which ODT operates.

To set ODT to the desired priority level, open SP. ODT prints the
present contents, which may then be changed:

\*SP/000006 377 <CR>

If SP is not specified, its value is seven.

ODT priority must be set to 0 if ODT is being used in an F/B
environment with another job running.

Breakpoints may be set in routines which run at different priority
levels. For example, a program running at a low priority may use a
device service routine which operates at a higher priority level. If
a breakpoint occurs from a low-priority routine, ODT operates at a low
priority; if an interrupt occurs from a high priority routine, the
breakpoints in the high priority routine will not be recognized since
they were removed when the low priority breakpoint occurred. That is,
interrupts set at a priority higher than the one at which ODT is
running will occur and any breakpoints will not be recognized. ODT
disable all breakpoints from the program whenever it gains control.
On-Line Debugging Technique

Breakpoints are enabled when ;P and ;G commands are executed. For example:

```
#P/00000?_5
#1000;B
#2000;B
#1000;G
$0:001000
^T (an interrupt occurs and is serviced)
```

If a higher level interrupt occurs while ODT is waiting for input the interrupt will be serviced, and no breakpoints will be recognized.

8.3.16 ASCII Input and Output, r;nA

ASCII text may be inspected and changed by the command:

```
r;nA
```

where r is a relocatable expression, and n is a character count. If n is omitted it is assumed to be 1. ODT prints n characters starting at location r, followed by a carriage return/line feed. Type one of the following:

```
<CR> ODT outputs a carriage return/line feed and an asterisk and waits for another command.
<LF> ODT opens the byte following the last byte output.
```

Up to n characters of text
ODT inserts the text into memory, starting at location r. If fewer than n characters are typed, terminate the command by typing CTRL U, causing a carriage return/line feed/asterisk to be output. However, if exactly n characters are typed, ODT responds with a carriage return/line feed, the address of the next available byte and a carriage return/line feed/asterisk.

ODT does not check the magnitude of n.

8.4 PROGRAMMING CONSIDERATIONS

Information in this section is not necessary for the efficient use of ODT. However, it does provide a better understanding of how ODT performs some of its functions and in certain difficult debugging situations, this understanding is necessary.

8.4.1 Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and various utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.
On-Line Debugging Technique

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines exit either to the object program or back to the command decoder.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers.

8.4.2 Breakpoints

The function of a breakpoint is to give control to ODT whenever the user program tries to execute the instruction at the selected address. Upon encountering a breakpoint, all of the ODT commands can be used to examine and modify the program.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the user's code so that the locations may be examined and/or altered. ODT then types a message on the terminal of the form Bnjk where k is the breakpoint address (and n is the breakpoint number). The breakpoints are automatically restored when execution is resumed.

A major restriction in the use of breakpoints is that the word where a breakpoint was set must not be referenced by the program in any way since ODT altered the word. Also, no breakpoint should be set at the location of any instruction that clears the T-bit. For example:

```
MOV #240,177776 ;SET PRIORITY TO LEVEL 5
```

NOTE

Instructions that cause or return from traps (e.g., EMT, RTI) are likely to clear the T-bit, since a new word from the trap vector or the stack is loaded into the Status Register.

A breakpoint occurs when a trace trap instruction (placed in the user program by ODT) is executed. When a breakpoint occurs, the following steps are taken:

1. Set processor priority to seven (automatically set by trap instruction).
2. Save registers and set up stack.
3. If internal T-bit trap flag is set, go to step 13.
4. Remove breakpoints.
5. Reset processor priority to ODT's priority or user's priority.
6. Make sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, go to step 15.
8. Decrement repeat count.
9. Go to step 18 if non-zero; otherwise reset count to one.
10. Save terminal status.
11. Type message about the breakpoint or single-instruction mode interrupt.
On-Line Debugging Technique

12. Go to command decoder.
15. Save terminal status.
16. Type BE (Bad Entry) followed by the address.
17. Clear the T-bit, if set, in the user status and proceed to the command decoder.
18. Go to the Proceed processor, bypassing the TT restore routine.

Note that steps 1-5 inclusive take approximately 100 microseconds during which time interrupts are not permitted (ODT is running at level 7).

When a proceed (;P) command is given, the following occurs:

1. The proceed is checked for legality.
2. The processor priority is set to seven.
3. The T-bit flags (internal and user status) are set.
4. The user registers, status, and Program Counter are restored.
5. Control is returned to the user.
6. When the T-bit trap occurs, steps 1, 2, 3, 13, and 14 of the breakpoint sequence are executed, breakpoints are restored, and program execution resumes normally.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, the following occurs:

1. When the breakpoint occurs as described above, ODT is entered.
2. When ;P is typed, the T-bit is set and the IOT, EMT, TRAP, or other trapping instruction is executed.
3. This causes the current PC and status (with the T-bit included) to be pushed on the stack.
4. The new PC and status (no T-bit set) are obtained from the respective trap vector.
5. The whole trap service routine is executed without any breakpoints.
6. When an RTI is executed, the saved PC and PS (including the T-bit) are restored. The instruction following the trap-causing instruction is executed. If this instruction is not another trap-causing instruction, the T-bit trap occurs, causing the breakpoints to be reinserted in the user program,
On-Line Debugging Technique

or the single-instruction mode repeat count to be
decremented. If the following instruction is a trap-causing
instruction, this sequence is repeated starting at step 3.

NOTE

Exit from the trap handler must be via
the RTI instruction. Otherwise, the
T-bit is lost. ODT can not regain
control since the breakpoints have not
been reinserted yet.

Note that the ;P command is illegal if a breakpoint has not occurred
(ODT responds with ?); ;P is legal, however, after any trace trap
entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for
breakpoints 0-7. (The ninth word contains the address of the
next instruction to be executed in single-instruction mode.)

2. The next eight words contain the respective repeat counts.
The following word contains the repeat count for
single-instruction mode.)

These words may be changed at will, either by using the breakpoint
commands or by direct manipulation with $B.

When program runaway occurs (that is, when the program is no longer
under ODT control, perhaps executing an unexpected part of the program
where a breakpoint has not been placed), ODT may be given control by
pressing the HALT key to stop the computer, and restarting ODT (see
Section 8.1). ODT prints *, indicating that it is ready to accept a
command.

If the program being debugged uses the teleprinter for input or
output, the program may interact with ODT to cause an error since ODT
uses the teleprinter as well. This interactive error will not occur
when the program being debugged is run without ODT.

Note the following rules concerning the ODT break routine:

1. If the teleprinter interrupt is enabled upon entry to the ODT
   break routine, and no output interrupt is pending when ODT is
   entered, ODT generates an unexpected interrupt when returning
   control to the program.

2. If the interrupt of the teleprinter reader (the keyboard) is
   enabled upon entry to the ODT break routine, and the program
   is expecting to receive an interrupt to input a character,
   both the expected interrupt and the character are lost.

3. If the teleprinter reader (keyboard) has just read a
   character into the reader data buffer when the ODT break
   routine is entered, the expected character in the reader data
   buffer is lost.
On-Line Debugging Technique

8.4.3 Searches

The word search allows the user to search for bit patterns in specified sections of memory. Using the $M/$ command, the user specifies a mask, a lower search limit ($M+2$), and an upper search limit ($M+4$). The search object is specified in the search command itself.

The word search compares selected bits (where ones appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word is printed.

The search algorithm is:

1. Fetch a word at the current address.
2. XOR (exclusive OR) the word and search object.
3. AND the result of step 2 with the mask.
4. If the result of step 3 is zero, type the address of the unmasked word and its contents. Otherwise, proceed to step 5.
5. Add two to the current address. If the current address is greater than the upper limit, type * and return to the command decoder, otherwise go to step 1.

Note that if the mask is zero, ODT prints every word between the limits, since a match occurs every time (i.e., the result of step 3 is always zero).

In the effective address search, ODT interprets every word in the search range as an instruction which is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is (where (X) denotes contents of X, and K denotes the search object):

1. Fetch a word at the current address X.
2. If (X)=K [direct reference], print contents and go to step 5.
3. If (X)+X+2=K [indexed by PC], print contents and go to step 5.
4. If (X) is a relative branch to K, print contents.
5. Add two to the current address. If the current address is greater than the upper limit, perform a carriage return/line feed and return to the command decoder; otherwise, go to step 1.

8.4.4 Terminal Interrupt

Upon entering the TT SAVE routine, the following occurs:
On-Line Debugging Technique

1. Save the LSR status register (TKS).
2. Clear interrupt enable and maintenance bits in the TKS.
3. Save the TT status register (TPS).
4. Clear interrupt enable and maintenance bits in the TPS.

To restore the TT:

1. Wait for completion of any I/O from ODT.
2. Restore the TKS.
3. Restore the TPS.

NOTES

If the TT printer interrupt is enabled upon entry to the ODT break routine, the following may occur:

1. If no output interrupt is pending when ODT is entered, an additional interrupt always occurs when ODT returns control to the user.

2. If an output interrupt is pending upon entry, the expected interrupt occurs when the user regains control.

If the TT reader (keyboard) is busy or done, the expected character in the reader data buffer is lost.

If the TT reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) is lost.

8.5 ODT ERROR DETECTION

ODT detects two types of error: illegal or unrecognizable command and bad breakpoint entry. ODT does not check for the legality of an address when commanded to open a location for examination or modification. Thus the command:

*177774/
?N-TRAP TO 4 003362

references nonexistent memory, thereby causing a trap through the vector at location 4. If this vector has not been properly initialized, unpredictable results occur.
On-Line Debugging Technique

Typing something other than a legal command causes ODT to ignore the command, print:

    (echoes illegal command)?

and wait for another command. Therefore, to cause ODT to ignore a command just typed, type any illegal character (such as 9 or RUBOUT) and the command will be treated as an error, i.e., ignored.

ODT suspends program execution whenever it encounters a breakpoint, i.e., traps to its breakpoint routine. If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

    BEnnnnnn

and waits for another command. BEnnnnnn denotes Bad Entry from location nnnnnn. A bad entry may be caused by an illegal trace trap instruction, setting the T-bit in the status register, or by a jump to the middle of ODT.
CHAPTER 9
PROGRAMMED REQUESTS

A number of services at the machine language level which the monitor regularly provides to system programs are also available to user-written programs. These include services for file manipulation, command interpretation, and facilities for input and output operations. User programs call these monitor services by means of "programmed requests", which are assembler macro calls written into the user program and interpreted by the monitor at program execution time.

NOTE

Programmed requests used in Version 2 differ from those used in Version 1; for example, the channel number in Version 1 was limited to the range 0-17, where it is not in Version 2; blank fields in macro calls were not allowed in Version 1, and are in Version 2; a .area argument points to an argument list in Version 2, where arguments were pushed on the stack in Version 1.

Programs written for use under Version 1 will assemble and execute properly when the ..V1.. macro call (explained in Section 9.3.1.5) is included, but it is to the user's advantage to convert these programs so they use the new Version 2 macro calls wherever possible. Only macro calls which are used with the current version of RT-ll (Version 2) are discussed in this chapter. See Section 9.5 for instructions on converting Version 1 macro calls to the Version 2 format.

The macro definitions for both Version 1 and Version 2 requests are included in the file SYSMAC.SML (in 8K systems, the system macro library is called SYSMAC.8K); Appendix D provides a listing of SYSMAC.SML. Refer to Chapter 5 for general information related to the use of macro calls.

The FORTRAN programmer should note that the system subroutine library (SYSLIB) gives him the same capability (under FORTRAN) to use the programmed requests which are available to the machine language programmer and described in this chapter. SYSLIB users should first read this chapter and then read Appendix O.
Programmed Requests

9.1 FORMAT OF A PROGRAMMED REQUEST

The basis of a programmed request is the EMT instruction, used to communicate information to the monitor. When an EMT is executed, control is passed to the monitor, which extracts appropriate information from the EMT and executes the function required. The low-order byte of the EMT instruction contains a code which is interpreted as:

<table>
<thead>
<tr>
<th>Low-Order Byte of EMT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>377</td>
<td>Reserved; RT-ll ignores this EMT and returns control to the user program immediately.</td>
</tr>
<tr>
<td>376</td>
<td>Used internally by the RT-ll monitor; this EMT code should never be used by user programs.</td>
</tr>
<tr>
<td>375</td>
<td>Programmed request with several arguments: R0 must point to a list of arguments which designates the specific function.</td>
</tr>
<tr>
<td>374</td>
<td>Programmed request with one argument: R0 contains a function code in the high-order byte and a channel number (see Section 9.2.1) or 0 in the low-order byte.</td>
</tr>
<tr>
<td>360-373</td>
<td>Used internally by the RT-ll monitor; these EMT codes should never be used by user programs.</td>
</tr>
<tr>
<td>340-357</td>
<td>Programmed request with arguments on the stack and/or in R0.</td>
</tr>
<tr>
<td>0-337</td>
<td>Version 1 programmed request. These EMTs use arguments both on the stack and in R0. They are supported for binary compatibility with Version 1 programs.</td>
</tr>
</tbody>
</table>

A programmed request consists of a macro call followed, where necessary, by one or more arguments. Arguments supplied to a macro call must be legal assembler expressions since arguments will be used as source fields in MOV instructions when the macros are expanded at assembly time. The following two formats are used:

1. PRGREQ ARG1,ARG2,...ARGN

2. PRGREQ AREA,ARG1,ARG2,...ARGN

Form 1 above contains the arguments ARG1 through ARGN; no argument list pointer is required. Macros of this form generate either an EMT 374 or one of the EMTs 340-357. Certain arguments for this form may be omitted; refer to the listing of SYSMAC.SML in Appendix D.

In form 2 above, AREA is a pointer to the argument list which contains the arguments ARG1 through ARGN. This form always causes an EMT 375 to be generated. Blank fields are permitted; however, if the AREA
Programmed Requests

argument is blank, the macro assumes that R0 points to a valid argument block (see Section 9.2.3). If any of the fields ARG1 to ARGN are blank, the corresponding entries in the argument list are left untouched. Thus,

```
.PRGREQ AREA, A1, A2
```

points R0 to the argument block at AREA and fills in the first and second arguments, while:

```
.PRGREQ AREA
```

points R0 to the block, and fills in the first word but does not fill in any other arguments.

The call:

```
.PRGREQ , AI
```

assumes R0 points to the argument block and fills in the A1 argument, but leaves the A2 argument alone. The call:

```
.PRGREQ
```

generates only an EMT 375 and assumes that both R0 and the block to which it points are properly set up.

The arguments to RT-11 programmed request macros all serve as the source field of a MOV instruction which moves a value into the argument block or R0. For example:

```
.PRGREQ CHAR
```

expands into:

```
MOV CHAR, R0
EMT 357
```

Care should be taken to make certain that the arguments specified are legal source fields and that the address accurately represents the value desired. If the value is a constant, immediate mode [\#] should be used; if the value is in a register, the register mnemonic [Rn] should be used; if the value is indirectly addressed, the appropriate register convention is necessary [@Rn], and if the value is in memory, the label of the location whose value is the argument is used.

Following are some examples of both correct and incorrect macro calls. Consider the general request:

```
.PRGREQ .AREA,, ARG1,..., ARGN
```

9-3 January 1976
Programmed Requests

A more common way of writing a request of this form is:

.PROGREQ #AREA,#ARG1,...,#ARGN

In this format, the address of AREA is put directly into the argument list. AREA is the tag which indicates the beginning of the argument block. For example:

.PROGREQ #AREA,#4

... AREA: .BLKW 3

When a direct numerical argument is required, the # causes the correct value to be put into the argument block. For example:

.PROGREQ #AREA,#4

is correct, while:

.PROGREQ #AREA,4

is not. This form interprets the 4 as meaning "move the contents of location 4 into the argument block", where the number 4 itself should be moved into the block.

If the request is written as:

.PROGREQ AREA,#4

it is interpreted as "use the contents of location AREA as the list pointer", when the address of AREA is actually desired. This expansion could be used with the following form:

.PROGREQ LIST1,#4

... LIST1: AREA

AREA: .BLKW3

In this case, the content of location LIST1 is the address of the argument list. Similarly, this form is correct:

.PROGREQ LIST1,NUMBER

LIST1: AREA

NUMBER: 4

In this case, the contents of the locations LIST1 and NUMBER are the argument list pointer and data value, respectively.

NOTE

All registers except R0 are preserved across a programmed request. (In certain cases, R0 may contain information passed back by the monitor; however, unless the description of a request indicates that a specific value is returned in R0, it may be assumed that the contents of R0 are unpredictable upon return from the request). With the exception of calls to the CSI, the position of the stack pointer is also preserved across a programmed request.

9-4
Programmed Requests

9.2 SYSTEM CONCEPTS

Some basic operational characteristics and concepts of RT-11 are described below.

9.2.1 Channel Number (chan)

A channel number is a logical identifier in the range 0 to 377 (octal) for a file or "set of data" used by the RT-11 monitor. Thus, when a file is opened on a particular device, a channel number is assigned to that file. To refer to an open file, it is only necessary to refer to the appropriate channel number for that file.

9.2.2 Device block (dblk)

A device block is a four-word block of radix-50 information which specifies a physical device and file name for an RT-11 programmed request. (Refer to Chapter 5 for an explanation of .RAD50 strings.) For example, a device block representing a file FILE.EXT on device DK could be written as:

```
.RAD50 /DK /
.RAD50 /FIL/
.RAD50 /E /
.RAD50 /EXT/
```

The first word contains the device name, the second and third words contain the file name, and the fourth contains the extension. Device, name, and extension must each be left-justified in the appropriate field. This string could also be written as:

```
.RAD50 /DK FILE EXT/
```

Note that spaces must be used to fill out each field. Note also that the colon and period separators do not appear in the actual RAD50 string. They are used only by the monitor keyboard interface to delimit the various fields.

9.2.3 EMT Argument Blocks

Programmed requests which call the monitor via EMT 375 use R0 as a pointer to an argument list. In general, this argument list appears as follows:

<table>
<thead>
<tr>
<th>address</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Function Channel</td>
</tr>
<tr>
<td></td>
<td>Code Number</td>
</tr>
<tr>
<td>x+2</td>
<td>argument1</td>
</tr>
<tr>
<td>x+4</td>
<td>argument2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9-5
Programmed Requests

R0 points to location x. The even (low-order) byte of location x contains the channel number named in the macro call. If no channel number is required, the byte is set to 0. The odd (high-order) byte of x is a code specifying the function to be performed. Locations x+2, x+4, etc. contain arguments to be interpreted. These are described in detail under each request.

Requests which use EMT 374 set up R0 with the channel number in the even byte and the function code in the odd byte. They require no other arguments.

9.2.4 Important Memory Areas

9.2.4.1 Vector Addresses (0-37, 60-477) - Certain areas of memory between 0 and 477 are reserved for use by RT-11. KMON does not load these locations from the save image file when it initiates a program, i.e., R, RUN, and GET will not load these words. However, no hardware memory protection is supplied. Thus, programs should never alter the contents of the indicated areas at run-time.

<table>
<thead>
<tr>
<th>Locations</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,2</td>
<td>Monitor restart. Executes .EXIT request and returns control to KMON.</td>
</tr>
<tr>
<td>4,6</td>
<td>Time out or bus error trap; RT-11 sets this to point to its internal trap handler.</td>
</tr>
<tr>
<td>10,12</td>
<td>Reserved instruction trap; RT-11 sets this to point to its internal trap handler.</td>
</tr>
<tr>
<td>30,32</td>
<td>EMT trap vector and status.</td>
</tr>
<tr>
<td>40-57</td>
<td>RT-11 system communication area (see below).</td>
</tr>
<tr>
<td>60,62</td>
<td>TTY input interrupt vector and status.</td>
</tr>
<tr>
<td>64,66</td>
<td>TTY output interrupt vector and status.</td>
</tr>
<tr>
<td>100,102</td>
<td>KW11L vector and status.</td>
</tr>
<tr>
<td>204,206</td>
<td>RF11 vector and status.</td>
</tr>
<tr>
<td>214,216</td>
<td>TCL1 vector and status.</td>
</tr>
<tr>
<td>220,222</td>
<td>RK05 vector and status.</td>
</tr>
<tr>
<td>330,332</td>
<td>GT40 shift out interrupt vector and status.</td>
</tr>
</tbody>
</table>

These areas are not replaced by RT-11. If they are destroyed by a program, the system must be re-bootstrapped, or the program must restore them.
Programmed Requests

9.2.4.2 Resident Monitor — Section 2.4 of Chapter 2 describes the placement of monitor components when either the Single-Job Monitor or F/B Monitor is brought into memory; included is the approximate size of each monitor component and the size of the area available for handlers and user programs.

9.2.4.3 System Communication Area — RT-11 uses bytes 40-57 to hold information about the program currently executing, as well as certain information used only by the monitor. A description of these bytes follows:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Meaning and Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>40,41</td>
<td>Start address of job. When a file is linked into an RT-11 memory image, this word is set to the starting address of the job either with the Linker /T switch or as an argument in the .END statement of the program. When a foreground program is executed, the FRUN processor relocates this word to contain the actual starting address of the program.</td>
</tr>
<tr>
<td>42,43</td>
<td>Initial value of the stack pointer. If it is not set by the user program in an .ASECT, it defaults to 1000 or the top of the .ASECT in the background, whichever is larger. If a foreground program does not specify a stack pointer in this word, a default stack (128 decimal words) is allocated by FRUN immediately below the program. The initial stack pointer can also be set with the Linker /M switch option.</td>
</tr>
<tr>
<td>44,45</td>
<td>Job Status Word. Used as a flag word for the monitor. Certain bits are maintained by the monitor exclusively while others must be set or cleared by the user job. Those bits in the following list which are marked by an asterisk are bits which must be set by the user job. Since the currently unassigned bits may be used in future releases of RT-11, user programs should not use these bits for internal flags.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>USR swap bit. (Unused in F/B.) The monitor sets this bit when programs do not require the USR to be swapped. See Section 9.2.5 for details on USR swapping.</td>
</tr>
<tr>
<td>14</td>
<td>Lower-case bit. When set (automatically by EDIT when the EL command is typed), disables conversion of lower-case to upper-case.</td>
</tr>
<tr>
<td>13</td>
<td>Reenter bit. When set, this bit indicates that the program may be restarted from the terminal with the REENTER command.</td>
</tr>
</tbody>
</table>

9-7 January 1976
Programmed Requests

*12 Special mode TT bit. When set, this bit indicates that the job is in a "special" keyboard mode of input. Refer to the explanation of the .TTYIN/.TTINR requests for details.

11-10 For F/B Monitor use only.

9 Overlay Bit. Set (by the Linker) if the job uses the Linker overlay structure.

8 CHAIN bit. If this bit is set in a job's save image, words 500-776 are loaded from the save file when the job is started even if the job is entered via CHAIN. (These words are normally used to pass parameters across CHAINS.) The bit is set when a job is running if and only if the job was actually entered with CHAIN.

*7 Error halt bit. When set, this bit indicates a halt on an I/O error. If the user desires to halt when any I/O device error occurs, this bit should be set. (Unused in F/B.)

*6 Inhibit TT wait bit. For use with the Foreground/Background system. When set, this bit inhibits the monitor from entering a console terminal wait state. Refer to the sections concerning .TTYIN/.TTINR, and .TTYOUT/.TOUTR for more information.

5-0 Unused.

46,47 USR load address. Normally 0, this word may be set to any valid word address in the user's program. See Section 9.2.5, Swapping Algorithm, for details of use.

50,51 High memory address. The monitor maintains the highest address the user program can use in this word. The Linker sets it initially. It is modified only via the .SETTOP (Set Top of Memory) monitor request.

52 EMT error code. If a monitor request results in an error, the code number of the error is always returned in byte 52 and the carry bit is set. Each monitor call has its own set of possible errors. It is recommended that the user program reference byte 52 with absolute addressing, rather than relative addressing. For example:

```
ERRWRD = 52
TSTB ERRWRD ;RELATIVE ADDRESSING
TSTB @@ERRWRD ;ABSOLUTE ADDRESSING
```
Programmed Requests

NOTE

Location 52 must always be addressed as a byte, never as a word, since byte 53 will be used in future releases of RT-11.

53
Reserved for future system use.

54, 55
Address of the beginning of the Resident Monitor. RT-11 always loads the resident into the highest available memory locations; this word points to its first location. It must never be altered by the user. Doing so will cause RT-11 to malfunction.

56
Fill character (7-bit ASCII). Some high-speed terminals require filler (null) characters after printing certain characters. Byte 56 should contain the ASCII 7-bit representation of the character after which fillers are required.

57
Fill count. This byte specifies the number of fill characters required. If bytes 56 and 57=0, no fillers are required.

The required fill characters are:

<table>
<thead>
<tr>
<th>Terminal</th>
<th>No. of fills</th>
<th>Value of Word 56</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial LA30 @ 300 baud</td>
<td>10 after carriage return</td>
<td>5015</td>
</tr>
<tr>
<td>Serial LA30 @ 150 baud</td>
<td>4 after carriage return</td>
<td>2015</td>
</tr>
<tr>
<td>Serial LA30 @ 110 baud</td>
<td>2 after carriage return</td>
<td>1015</td>
</tr>
<tr>
<td>VT05 @ 2400 baud</td>
<td>4 after line feed</td>
<td>2012</td>
</tr>
<tr>
<td>VT05 @ 1200 baud</td>
<td>2 after line feed</td>
<td>1012</td>
</tr>
<tr>
<td>VT05 @ 600 baud</td>
<td>1 after line feed</td>
<td>412</td>
</tr>
</tbody>
</table>

9.2.5 Swapping Algorithm

Programmed requests are divided into two categories according to whether or not they require the USR to be in memory (see Table 9-2). Any request which requires the USR in memory may also require that a portion of the user program be saved temporarily on the system device scratch blocks (i.e., be "swapped out") to provide room for the USR. The USR will be read into the swapped region.

During most normal operations, this swapping is invisible to the user and he need not be concerned about it. However, it is possible to optimize programs so that they require little or no swapping. This is particularly useful when operating in an F/B environment, since under the F/B system, the USR will be swapped for both background and foreground jobs regardless of which job required it. If the USR is not swapped, neither the foreground nor the background job will be slowed down by the swapping process.

The following items should be considered if a swap operation is necessary:

1. The background job - If a .SETTOP request in a background job specifies an address beyond the point at which the USR
Programmed Requests

normally resides, a swap will be required when the USR is called. More details concerning the .SETTOP request are in Section 9.4.36.

2. The value of location 46 - If the user either assembles an address into word 46 or moves a value there while the program is running, RT-11 uses the contents of that word as an alternate place to swap the USR. If location 46 is 0, this indicates that the USR will be at its normal location in high memory.

NOTES

1. If the USR does not require swapping, the value in location 46 is ignored. Swapping is a relatively time-consuming operation and is avoided, if possible.

2. A foreground job should always have a value in location 46 unless it is certain that the USR will never be swapped. If the foreground job does not allow space for the USR and a swap is required, a fatal error occurs. (The SET USR NOSWAP command, explained in Chapter 2, ensures that the USR will be resident.)

3. Care should be taken when specifying an alternate address to location 46. The single-job system does not verify the legality of the USR swap address. Thus, if the area to be swapped overlaps the Resident Monitor, the system is destroyed.

4. The user should also take care that the USR is never swapped over any of the following areas: the program stack; any parameter block for calls to the USR; any I/O buffers, device handlers, or completion routines being used when the USR is called.

The following is an example of the way a background program can avoid unnecessary USR swapping.

```
.MCALL .V21,REGDEF,.SETTOP,.EXT
...V2...
,KEGDEF
RMPTR=54
USRLOC=266
;POINTER TO RMON IS AT 54.
;POINTER TO USR LOCATION IS
1AT 266 BYTES INTO RMON.
START:

MOV #$RMPTR,R1 @H1 -> RESIDENT MONITOR
MOV USRLOC(R1),R0 @R0 -> USR
TST -(R6) @POINT JUST BELOW
```

July 1975 9-10
Programmed Requests

| CMP     | R0,#50  | IDoes USR SWAP over US?
| RHI     | 15      | NO, OK
| MOV     | #2,R0   | YES, USR MUST SWAP
| LSI     | .SETTOP | TASK FOR MEMORY UP TO USR
| MOV     | R0,HILIM| USR = HIGH LIMIT OF MEMORY
|         |         | ACTUALLY GRANTED BY MONITOR.
| .EXIT   |         | .CONTAINS HI LIMIT OF MEMORY
| HILIM1  | .NUM    | 0
| .END    | START   | 0

9.2.6 Offset Words

There are several words which always have fixed positions relative to the start of the Resident Monitor. It is often advantageous for user programs to be able to access these words. This is done with the code:

```
RMON = 54
MOV @RMON,register
MOV OFFSET(register),register
```

Here, register is any general register and OFFSET is a number from the following list:

<table>
<thead>
<tr>
<th>OFFSET (Bytes)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>262</td>
<td>System date. (See .DATE request.)</td>
</tr>
<tr>
<td>266</td>
<td>Start of normal USR area. This is where the USR will reside when it is non-swapping. It is useful to be able to perform a .SETTOP in a background job such that the USR is always resident. (An example is in Section 9.2.5.)</td>
</tr>
<tr>
<td>270</td>
<td>Address of I/O exit routine for all devices. The exit routine is an internal queue management routine through which all device handlers exit once the I/O transfer is complete. Any new devices added to RT-ll must also use this exit location.</td>
</tr>
<tr>
<td>275</td>
<td>Unit number of system device (device from which system was last bootstrapped).</td>
</tr>
<tr>
<td>276</td>
<td>Monitor version number (2-377). The user can always access the version number to determine if the most recent monitor is in use.</td>
</tr>
<tr>
<td>277</td>
<td>Update number. Patches to the monitor always increment the update number. This provides a means of checking that all patches have been made. (This number should be accessed by MOVB rather than MOV).</td>
</tr>
<tr>
<td>300</td>
<td>Configuration word. This is a string of 16 bits used to indicate information about either the hardware configuration of the system, or a software condition. The bits and their meanings are:</td>
</tr>
</tbody>
</table>
### Programmed Requests

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0     | 0 = Single-Job Monitor  
       | 1 = F/B Monitor         |
| 1     | 1 = GT40 display hardware exists |
| 2     | 1 = RT-11 BATCH is in control  
       | of the background        |
| 3     | 0 = 60-cycle clock  
       | 1 = 50-cycle clock       |
| 4     | 1 = 11/45 floating-point  
       | hardware exists          |
| 5     | 0 = No foreground job is in memory  
       | 1 = Foreground job is in memory |
| 6     | 1 = User is linked to the GT40  
       | scroller                 |
| 7     | 1 = USR is permanently resident  
       | (via a SET USR NOSWAP)    |
| 9     | 1 = Processor is an 11/03   |
| 11    | 1 = KWILL clock is present  
       | (always set if 11/03)      |

The other bits are reserved for future use and should not be accessed by user programs.

#### 304-313
These locations contain the addresses of the console terminal control and status registers. The order is:

- 304 Keyboard status
- 306 Keyboard buffer
- 310 Printer status
- 312 Printer buffer

These locations can be changed, for example, to reflect a second terminal; thus RT-11 can be made to run on any terminal present on the system which is connected to the machine via the DILL multiple terminal interface. (Refer to the RT-11 Software Support Manual (DEC-11-ORPGA-B-D).

#### 314
The maximum file size allowed in a 0 length .ENTER. This can be adjusted by the user program or by using the PATCH program to be any reasonable value. The default value is 177777 (decimal) blocks, allowing an essentially unlimited file size.

#### 324
Address of .SYNCH entry. User interrupt routines may enter the monitor through this address to synchronize with the job they are servicing.

#### 354
Address of VTI1 display processor display stop interrupt vector.

January 1976  
9-12
Programmed Requests

9.2.7 File Structure

RT-11 uses a "contiguous" file structure. This type of structure implies that every file on the device is made up of a contiguous group of physical blocks. Thus, a file that is 9 blocks long occupies 9 contiguous blocks on the device.

A contiguous area on a device can be in one of the following categories:

1. Permanent file. This is a file which has been .CLOSEEd on a device. Any named file which appears in a PIP directory listing are permanent files.

2. Tentative file. Any file which has been created via .ENTER, but not .CLOSEd, is a tentative file entry. When the .CLOSE request is given, the tentative entry becomes a permanent file. If a permanent file already exists under the same name, the old file is deleted. If a .CLOSE is never given, the tentative file is treated like an empty entry.

3. Empty entry. When disk space is unused or a permanent file is deleted, an empty entry is created. Empty entries appear in a PIP /E directory listing as <UNUSED> N, where N is the decimal block length of the empty area.

Since a contiguous structure does not automatically reclaim unused disk space, the device may eventually become "fragmented". A device is fragmented when there are many empty entries which are scattered over the device. RT-11 PIP has an option which allows the user to collect all empty areas so that they occur at the end of a device. Refer to Chapter 4 for details.

9.2.8 Completion Routines

Completion routines are user-written routines which are entered following an operation. On entry to a completion routine, R0 contains the channel status word for the operation; R1 contains the octal channel number of the operation. The carry bit is not significant.

Completion routines are handled differently in the Single-Job and the F/B versions of RT-11. In the Single-Job version, completion routines are totally asynchronous and can interrupt one another. In F/B, completion routines do not interrupt each other. Instead they are queued and made to wait until the correct job is running. For example, if a foreground job is running and an I/O transfer initiated by a background job completes and wants to go to a completion routine, the background routine is queued and will not execute until the foreground gives up control of the system. If the foreground is running and a foreground I/O transfer completes and wants a completion routine, that routine will be entered immediately if the foreground is not already inside a completion routine. If it is in a completion routine, that routine continues to termination, at which point any other completion routines are entered in a first in/first out manner. If the background is running and a foreground I/O transfer completes and needs a completion routine, the background is suspended and the foreground routine is entered immediately.
Programmed Requests

The restrictions which must be observed when writing completion routines are:

1. Completion functions cannot issue a request which would cause the USR to be swapped in. They are primarily used for issuing READ/WRITE commands, not for opening or closing files, etc. A fatal monitor error is generated if the USR is called from a completion routine.

2. Completion routines should never reside in the memory space which will be used for the USR, since the USR can be interrupted when I/O terminates and the completion routine is entered. If the USR has overlaid the routine, control passes to a random place in the USR, with a HALT or error trap the likely result.

3. The routine must be exited via an RTS PC, as it is called from the monitor via a JSR PC, ADDR where ADDR is the user-supplied address.

4. If a completion routine uses registers other than R0 or R1, it must save them upon entry and restore them before exiting.

9.2.9 Using the System Macro Library

User programs for RT-11 should always be written using the system macro library (SYSMAC.SML), supplied with RT-11. This ensures compatibility among all user programs and allows easy modification by redefining a macro. A listing of SYSMAC.SML appears in Appendix D.

The system macro library for 8K systems appears on the system device as SYSMAC.8K.

Suggestions for writing foreground programs are in Appendix H, F/B Programming and Device Handlers. This appendix should be read in conjunction with Chapter 9 before coding F/B programs.

9.3 TYPES OF PROGRAMMED REQUESTS

There are three types of services which the monitor makes available to the user through programmed requests. These are:

1. Requests for File Manipulation

2. Requests for Data Transfer

3. Requests for Miscellaneous Services

Table 9-1 summarizes the programmed requests in each of these categories alphabetically. Those marked with an asterisk function only in a F/B environment; they are ignored under the Single-Job Monitor. The EMT and function code for each request (where applicable) are included.
Programmed Requests

Table 9-1
Summary of Programmed Requests

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>EMT &amp; Code</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.CHCOPY</strong></td>
<td>375 13</td>
<td>9.4.3</td>
<td>Establishes a link and allows one job to access another job’s channel.</td>
</tr>
<tr>
<td><strong>.CLOSE</strong></td>
<td>374 6</td>
<td>9.4.4</td>
<td>Closes the specified channel.</td>
</tr>
<tr>
<td><strong>.DELETE</strong></td>
<td>375 0</td>
<td>9.4.10</td>
<td>Deletes the file from the specified device.</td>
</tr>
<tr>
<td><strong>.ENTER</strong></td>
<td>375 2</td>
<td>9.4.13</td>
<td>Creates a new file for output.</td>
</tr>
<tr>
<td><strong>.LOOKUP</strong></td>
<td>375 1</td>
<td>9.4.21</td>
<td>Opens an existing file for input and/or output via the specified channel.</td>
</tr>
<tr>
<td><strong>.RENAME</strong></td>
<td>375 4</td>
<td>9.4.32</td>
<td>Changes the name of the indicated file to a new name.</td>
</tr>
<tr>
<td><strong>.REOPEN</strong></td>
<td>375 6</td>
<td>9.3.33</td>
<td>Restores the parameters stored via a SAVESTATUS request and reopens the channel for I/O.</td>
</tr>
<tr>
<td><strong>.SAVESTATUS</strong></td>
<td>375 5</td>
<td>9.4.34</td>
<td>Saves the status parameters of an open file in user memory and frees the channel for future use.</td>
</tr>
</tbody>
</table>

Data Transfer Requests

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>EMT &amp; Code</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.RCV</strong></td>
<td>375 26</td>
<td>9.4.29</td>
<td>Receives data. Allows a job to read messages or data sent by another job in an F/B environment. The three modes correspond to the READ, .READC, and READW modes.</td>
</tr>
<tr>
<td><strong>.RCV</strong> &amp; <strong>.RCVD</strong></td>
<td>375 10</td>
<td>9.4.30</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.</td>
</tr>
<tr>
<td><strong>.READ</strong></td>
<td>375 10</td>
<td>9.4.30</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers asynchronously to the routine specified in the .READC request.</td>
</tr>
</tbody>
</table>

9-15 (continued on next page)
## Summary of Programmed Requests

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>EMT &amp; Code</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.READW</td>
<td>375 10</td>
<td>9.4.30</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program only after the transfer is complete.</td>
</tr>
<tr>
<td>.SDAT</td>
<td>375 25</td>
<td>9.4.35</td>
<td>Allows the user to send messages or data to the other job in an F/B environment. The three modes correspond to the .WRITE, .WRITC and .WRITW modes.</td>
</tr>
<tr>
<td>.SDATC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.SDATW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.TTYIN</td>
<td>340 --</td>
<td>9.4.43</td>
<td>Transfers one character from the keyboard buffer to R0.</td>
</tr>
<tr>
<td>.TTINR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.TTYOUT</td>
<td>341 --</td>
<td>9.4.44</td>
<td>Transfers one character from R0 to the terminal input buffer.</td>
</tr>
<tr>
<td>.TTOUTR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.WRITE</td>
<td>375 11</td>
<td>9.4.47</td>
<td>Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of the I/O.</td>
</tr>
<tr>
<td>.WRITC</td>
<td>375 11</td>
<td>9.4.47</td>
<td>Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers asynchronously to the routine specified in the .WRITC request.</td>
</tr>
<tr>
<td>.WRITW</td>
<td>375 11</td>
<td>9.4.47</td>
<td>Transfers data via the specified channel to a device and returns control to the user program only after the transfer is complete.</td>
</tr>
</tbody>
</table>

### Miscellaneous Services

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>EMT &amp; Code</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CDFN</td>
<td>375 15</td>
<td>9.4.1</td>
<td>Defines additional channels for doing I/O.</td>
</tr>
<tr>
<td>.CHAIN</td>
<td>374 10</td>
<td>9.4.2</td>
<td>Chains to another program (in the background job only).</td>
</tr>
<tr>
<td>.CMKT</td>
<td>375 23</td>
<td>9.4.5</td>
<td>Cancels an unexpired mark time request.</td>
</tr>
<tr>
<td>.CNTXSW</td>
<td>375 33</td>
<td>9.4.6</td>
<td>Requests that the indicated memory locations be part of the F/B context switch process.</td>
</tr>
<tr>
<td>.CSIGEN</td>
<td>344 --</td>
<td>9.4.7</td>
<td>Calls the Command String Interpreter (CSI) in general mode.</td>
</tr>
<tr>
<td>.CSISPC</td>
<td>345 --</td>
<td>9.4.8</td>
<td>Calls the CSI in special mode.</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 9-1 (Cont.)

**Summary of Programmed Requests**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>EMT &amp; Code</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.CSTAT</td>
<td>375 27</td>
<td>9.4.9</td>
<td>Returns the status of the channel indicated.</td>
</tr>
<tr>
<td>.DATE</td>
<td>--- ---</td>
<td>9.3.1.1</td>
<td>Moves the current date information into R0.</td>
</tr>
<tr>
<td>*.DEVICE</td>
<td>375 14</td>
<td>9.4.11</td>
<td>Allows user to turn off device interrupt enable in F/B upon program termination.</td>
</tr>
<tr>
<td>.DSTATUS</td>
<td>342 --</td>
<td>9.4.12</td>
<td>Returns the status of a particular device.</td>
</tr>
<tr>
<td>.EXIT</td>
<td>350 --</td>
<td>9.4.14</td>
<td>Exits the user program and returns control to the Keyboard Monitor.</td>
</tr>
<tr>
<td>.FETCH</td>
<td>343 --</td>
<td>9.4.15</td>
<td>Loads device handlers into memory.</td>
</tr>
<tr>
<td>.GTIM</td>
<td>375 21</td>
<td>9.4.16</td>
<td>Gets time of day.</td>
</tr>
<tr>
<td>.GTJB</td>
<td>375 20</td>
<td>9.4.17</td>
<td>Gets parameters of this job.</td>
</tr>
<tr>
<td>.HERR</td>
<td>374 5</td>
<td>9.4.18</td>
<td>Specifies termination of the job on fatal errors.</td>
</tr>
<tr>
<td>.HRESET</td>
<td>357 --</td>
<td>9.4.19</td>
<td>Terminates I/O transfers and does a .SRESET operation.</td>
</tr>
<tr>
<td>.INTEN</td>
<td>--- ---</td>
<td>9.3.1.2</td>
<td>Notifies monitor that an interrupt has occurred and to switch to &quot;system state&quot;, and sets the processor priority to the correct value.</td>
</tr>
<tr>
<td>.LOCK</td>
<td>346 --</td>
<td>9.4.20</td>
<td>Makes the monitor User Service Routines (USR) permanently resident until .EXIT or .UNLOCK is executed. The user program is swapped out if necessary.</td>
</tr>
<tr>
<td>.MFPS</td>
<td>--- ---</td>
<td>9.3.1.3</td>
<td>Reads the priority bits in the processor status word (does not read the condition codes).</td>
</tr>
<tr>
<td>*.MRKT</td>
<td>375 22</td>
<td>9.4.22</td>
<td>Marks time; i.e., sets asynchronous routine to occur after a specified interval.</td>
</tr>
<tr>
<td>.MTPS</td>
<td>--- ---</td>
<td>9.3.1.3</td>
<td>Sets the priority bits, condition codes, and T bit in the processor status word.</td>
</tr>
<tr>
<td>*.MWAIT</td>
<td>374 11</td>
<td>9.3.23</td>
<td>Waits for messages to be processed.</td>
</tr>
<tr>
<td>.PRINT</td>
<td>351 --</td>
<td>9.4.24</td>
<td>Outputs an ASCII string to the terminal.</td>
</tr>
<tr>
<td>*.PROTECT</td>
<td>375 31</td>
<td>9.4.25</td>
<td>Requests that vectors in the area from 0-476 be given exclusively to this job.</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>EMT Code</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.PURGE</td>
<td>374 3</td>
<td>9.4.26</td>
<td>Clears out a channel.</td>
</tr>
<tr>
<td>.QSET</td>
<td>353 --</td>
<td>9.4.27</td>
<td>Expands the size of the monitor I/O queue.</td>
</tr>
<tr>
<td>.RCTRLO</td>
<td>355 --</td>
<td>9.4.28</td>
<td>Enables output to the terminal.</td>
</tr>
<tr>
<td>.REGDEF</td>
<td>--- --</td>
<td>9.3.1.4</td>
<td>Defines the PDP-11 general registers.</td>
</tr>
<tr>
<td>.RELEAS</td>
<td>343 --</td>
<td>9.4.31</td>
<td>Removes device handlers from memory.</td>
</tr>
<tr>
<td>*.RSUM</td>
<td>374 2</td>
<td>9.4.39</td>
<td>Causes the main line of the job to be resumed where it was suspended with .SPND.</td>
</tr>
<tr>
<td>.SERR</td>
<td>374 4</td>
<td>9.4.18</td>
<td>Inhibits most fatal errors from causing the job to the aborted.</td>
</tr>
<tr>
<td>.SETTOP</td>
<td>354 --</td>
<td>9.4.36</td>
<td>Specifies the highest memory location to be used by the user program.</td>
</tr>
<tr>
<td>.SFPA</td>
<td>375 30</td>
<td>9.4.37</td>
<td>Sets user interrupt for floating point processor exceptions.</td>
</tr>
<tr>
<td>.SPFUN</td>
<td>375 32</td>
<td>9.4.38</td>
<td>Performs special functions on magtape and cassette units.</td>
</tr>
<tr>
<td>*.SPND</td>
<td>374 1</td>
<td>9.4.39</td>
<td>Causes the running job to be suspended.</td>
</tr>
<tr>
<td>.SRESET</td>
<td>352 --</td>
<td>9.4.40</td>
<td>Resets all channels and releases the device handlers from memory.</td>
</tr>
<tr>
<td>.SYNCH</td>
<td>--- --</td>
<td>9.3.1.5</td>
<td>Enables user program to perform monitor programmed requests from within an interrupt service routine.</td>
</tr>
<tr>
<td>*.TLOCK</td>
<td>374 7</td>
<td>9.4.41</td>
<td>Indicates if the USR is currently being used by another job and performs a .LOCK if available.</td>
</tr>
<tr>
<td>.TRPSET</td>
<td>375 3</td>
<td>9.4.42</td>
<td>Sets a user intercept for traps to locations 4 and 10.</td>
</tr>
<tr>
<td>*.TWAIT</td>
<td>375 24</td>
<td>9.4.45</td>
<td>Suspends the running job for a specified amount of time.</td>
</tr>
<tr>
<td>.UNLOCK</td>
<td>347 --</td>
<td>9.4.20</td>
<td>Releases USR if a LOCK was done. The user program is swapped in if required.</td>
</tr>
<tr>
<td>..V1...</td>
<td>--- --</td>
<td>9.3.1.6</td>
<td>Enables expansions to occur in Version 1 format.</td>
</tr>
<tr>
<td>..V2...</td>
<td>--- --</td>
<td>9.3.1.6</td>
<td>Enables expansions to occur in Version 2 format.</td>
</tr>
<tr>
<td>.WAIT</td>
<td>374 0</td>
<td>9.4.46</td>
<td>Waits for completion of all I/O on a specified channel.</td>
</tr>
</tbody>
</table>

January 1976  
9-18
Programmed Requests

Requests requiring the USR (as explained in Section 9.2.5) differ between the Single-Job and F/B Monitors. Table 9-2 indicates which requests require the USR to be in memory. Those requests marked by an asterisk are Version 2 macros only. The CLOSE request on non-file structured devices (LP, PP, TT, etc.) does not require the USR under either monitor.
This page intentionally blank.
## Programmed Requests

### Table 9-2
Requests Requiring the USR

<table>
<thead>
<tr>
<th>Request</th>
<th>F/B</th>
<th>Single-Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.CDFN</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>*.CHAIN</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.CHCOPY</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>.CLOSE (see Note 1)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*.CMKT</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>*.CNTSW</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>.CSIGEN</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>.CSISPC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*.CSTAT</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>.DELETE</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*.DEVICE</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>*.DSTATUS</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>.ENTER</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>.EXIT</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.FETCH</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*.GTIM</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.GTJB</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.HERR</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.HRESET</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>.LOCK (see Note 2)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>.LOOKUP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>*.MRKT</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>*.MWAIT</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>.PRINT</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.PROTECT</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>*.PURGE</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.QSET</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>.RCTRL</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.RCVD/RCVDC/RCVDW</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>.READ/READC/READW</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.RELSAS</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>.RENAME</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>.REOPEN</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.RSUM</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>*.SAVESTATUS</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.SDAT/SDATC/SDATW</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>*.SERR</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.SETTOP</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.SFPa</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.SPFUN</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.SPND</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>*.SRESET</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>.TLOCK (see Note 3)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.TRPSET</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.TTINR/TTYIN</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.TTOUTR/.TTYOUT</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>*.TWAIT</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>.UNLOCK</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.WAIT</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>.WRITE/WRTC/WRTW</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Programmed Requests

Note 1: Only if channel was opened via .ENTER.
Note 2: Only if USR is in a swapping state.
Note 3: Only if USR is not in use by the other job.

9.3.1 System Macros

The following five macros are included in the system macro library, but are not programmed requests in that they cause no EMT instruction to be generated:

```
.DATE .SYNCH
.INTEN .V2..
.REGDEF
```

They can be used in the same manner as the other macro calls; their explanations follow.

9.3.1.1 DATE

This request moves the current date information from the system date word into R0. The date word returned is in the following format:

Bit:  14  10  9  5  4  0

<table>
<thead>
<tr>
<th>MONTH (1-12.)</th>
<th>DAY (1-31.)</th>
<th>YEAR-72(DECIMAL)</th>
</tr>
</thead>
</table>

Macro Call: .DATE

Errors:

No errors are returned. A zero result in R0 indicates that no DATE command was entered.
Programmed Requests

9.3.1.2 .INTEN

This request is used by user program interrupt service routines to:

1. Notify the monitor that an interrupt has occurred and to switch to "system state",

2. Set the processor priority to the correct value.

In Version 2 of RT-11, all external interrupts cause the processor to go to level 7 (see Appendix H). .INTEN is used to lower the priority to the value at which the device should be run. On return from .INTEN, the device interrupt can be serviced, at which point the interrupt routine returns via an RTS PC. It is very important to note that an RTI will not return correctly from an interrupt routine which specifies an .INTEN.

Macro Call: .INTEN .priority, pic

where: .priority is the processor priority at which the user wishes to run his interrupt routine.

pic is an optional argument which should be non-blank if the interrupt routine is written as a PIC (position independent code) routine. If the routine does not have to be PIC, it is recommended that the PIC field be left blank; the non-PIC version is slightly faster than the PIC version.

The user is advised to read Appendix H for more details concerning the use of .INTEN and .SYNCH.

Errors:
None.

Example:
Refer to Section 9.3.1.5, .SYNCH, for an example.
Programmed Requests

9.3.1.3 .MFPS/.MTPS

The .MFPS and .MTPS macro calls allow processor-independent user access to the processor status word.

The .MFPS call is used to read the priority bits only; condition codes are destroyed during the call and must be directly accessed (using conditional branch instructions) if they are to be read in a processor-independent manner.

Macro Call: .MFPS .addr

where: .addr is the address into which the processor status is to be stored; if .addr is not defined, the value is returned on the stack. Note that only the priority bits are significant.

The .MTPS call is used to set the priority, condition codes, and T bit with the value designated in the call.

Macro Call: .MTPS .addr

where: .addr is the address of the word to be placed in the processor status word; if .addr is not defined, the processor status word is taken from the stack. Note that the high byte on the stack is set to zero when .addr is present. If .addr is not present, the user should set the stack to the appropriate value. In either case, the whole word on the stack is put in the processor status word.

The contents of R0 are preserved across either call.

Errors:

None.
Programmed Requests

Example:

```
MCALL ..V2..,REGDEF,..MFPS,..MTPS,..EXIT
..V2..
..REGDEF

START: JSR PC,PICKQ

PICKQ: MFPS
MOV #QHEAD,R4
MTPS #340
MOV @R4,R5
REG 105
MOV @R5,@R4
.MTPS
EXIT

10$: RTS PC
QHEAD: .WORD Q1

; THREE QUEUE ELEMENTS
Q1: .WORD Q2,0,0
Q2: .WORD Q3,0,0
Q3: .WORD 0,0,0

END START
```
Programmed Requests

**.REGDEF**

9.3.1.4 .REGDEF

This macro call defines the PDP-11 general registers as R0 through R5, SP, and PC.

**Macro Call:**

```
.MCALL .REGDEF,...
.REGDEF
```

**Errors:**
None.

**Example:**

Refer to the example for the .SYNCH request. Appendix D shows the expansion of .REGDEF.

**.SYNCH**

9.3.1.5 .SYNCH

This macro call enables the user program to perform monitor programmed requests from within an interrupt service routine. Unless a .SYNCH is used, issuing programmed requests from interrupt routines is not supported by the system and should not be performed. .SYNCH, like .INTEN and .DATE, is not a programmed request and generates no EMT instructions.

**Macro Call:**

```
.SYNCH .area
```

**where:**

- `.area` is the address of a seven-word area which the user must set aside for use by .SYNCH. The 7-word block appears as:

- **Word 1** RT-11 maintains this word; its contents should not be altered by the user.
- **Word 2** The current job's number. This can be obtained by a .G1JB call.
- **Word 3** Unused.

---

January 1976  9-22
Programmed Requests

Word 4 Unused.
Word 5 R0 argument. When a successful return is made from .SYNCH, R0 contains this argument.
Word 6 Must be -1.
Word 7 Must be 0.

Note:

.SYNCH assumes that the user has not pushed anything on the stack between the .INTEN and .SYNCH calls. This rule must be observed for proper operation.

Errors:

The monitor returns to the location immediately following the .SYNCH if the .SYNCH was rejected. The routine is still unable to issue programmed requests, and R4 and R5 are available for use. Errors returned are due to one of the following:

1. Another .SYNCH which specified the same 7-word block is still pending.

2. An illegal job number was specified in the second word of the block. The only currently legal job numbers are 0 and 2.

3. If the job has been aborted or for some reason is no longer running, the .SYNCH will fail.

Normal return is to the word after the error return with the routine in user state and thus allowed to issue programmed requests. R0 contains the argument which was in word 5 of the block. R0 and R1 are free to be used without having to be saved. (R4 and R5 are not free.) Exit from the routine should be done via an RTS PC. (Refer to Appendix H, Section H.1.4, and to the RT-11 Software Support Manual, Section 6.4.)

Example:

.MCALL .V2....REGEF
 .V2
 .REGEF
 .MCALL
 .STRTA
 MOV #JOB,R5
 .STRTA
 GTJR #AREA,R5
 MOV (R5),SYNBLK+2
 .STRTA
 INTRPTI .INTEN 5
 .SYNCH #SYNBLK
 RR SYNFFAIL

9-23
Programmed Requests

.IRETURN HERE AT PRIORITY 0; NOTE: .SYNCH DOES RTI

.RETC #ARE, #CHAN, #BUFF, #WCNT, #CRTN1, #RLK

.RCS #WRITE A BUFFER

.WPFAIL #FAILED SOMEHOW

.PC #REF-INITIALIZE FOR MORE

.SYNBLK #FINTERRUPTS AND EXIT

.WORD #JOB NUMBER

.WORD #IFM CONTAINS 5 ON SUCCESSFUL

.WORD #ISYNCH

.WORD #SFET UP FOR MONITOR

.V1 .. / .. V2 ..

9.3.1.6 .. V1.. / .. V2..

Any program that uses system MACROS must specify the version format
(Version 1 or Version 2) in which the macro calls are to be expanded.
Assembly errors at macro calls will result if the proper version
designation is not made.

The .. V1 .. macro call enables all macro expansions to occur in Version
1 format. (Note that any requests marked with an asterisk in Table
9-1 are not valid as Version 1 requests, and thus will be flagged as
errors if they are assembled in Version 1 form.)

Macro Call: .. MCALL .. V1 ..

.. V1..

This causes all macros in the program to be assembled in Version 1
form and the symbol .. V1 to be defined. User programs should not use
this symbol. The .. V1 .. macro expands into:

.. V1=1

To cause all macro expansions to occur in Version 2 format, the .. V2..
macro call is used. Using .. V2 .. causes the symbol .. V2 to be
defined. Likewise, user programs should not use this symbol.

Macro Call: .. MCALL .. V2..

.. V2..

The .. V2 .. macro expands into:

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

.. V2=1
Programmed Requests

Note:

It is possible that user programs will exist in which both Version 1 and Version 2 macros are present. To allow proper assembly, the user should include the statements:

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

to define the utility macros (CM1, CM2, etc.) used by other Version 2 macros. This causes all macros which existed in Version 1 to assemble in Version 1 format, while those macros new to Version 2 are correctly generated as Version 2 macros. Note that in this case a macro which existed in Version 1 (such as .READ) will expand in the Version 1 format.

Run-time or assembly errors will occur if both the ..V1.. and ..V2.. macro calls are used in a program.

Example:

All examples in Chapter 9 illustrate the Version 2 format. Users are urged to use the ..V2.. macro call in their programs.

9.4 PROGRAMMED REQUEST USAGE

This section provides a description of each of the programmed requests alphabetically. The following parameters are commonly used as arguments in the various calls:

- `.addr` an address, the meaning of which depends on the request being used
- `.area` a pointer to the EMT argument list (for those requests which require a list); see Section 9.2.3
- `.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 or from which an I/O transfer is to be performed
- `.chan` a channel number in the range 0-377(octal)
- `.crtn` the entry point of a completion routine; see Section 9.2.8
- `.count` file number for magtape/cassette operations (see Appendix H); if this argument is blank, a value of 0 is assumed
- `.dblk` the address of a four-word RAD50 descriptor of the file to be operated upon; see Section 9.2.2
- `.num` a number, the value of which depends on the request
- `.wcnt` a word count specifying the number of words to be transferred to or from the buffer during an I/O operation

Additional information concerning these parameters (and others not defined here) is provided as necessary under each request.

9-25
Programmed Requests

9.4.1 .CDFN

The .CDFN request is used to redefine the number of I/O channels. Each job, whether foreground or background, is initially provided with 16(decimal) I/O channels, numbered 0-15. .CDFN allows the number to be expanded to as many as 255(decimal) channels.

Note that .CDFN defines new channels; the previously-defined channels are not used. Thus, a .CDFN for 20(decimal) channels (while the 16 original channels are defined) causes only 20 I/O channels; the space for the original 16 is unused.

Note that if a program is overlaid, channel 15 is used by the overlay handler and should not be modified. (Other channels can be defined and used as usual.)

Macro Call: .CDFN .area, .addr, .num

where: .addr is the address where the I/O channels begin

.num is the number of I/O channels to be created

Request Format:

R0 ⇒ .area: 15 0

.addr
.num

The space used to contain the new channels is taken from within the user program. Each I/O channel requires 5 words of memory. Thus, the user must allocate 5*N words of memory, where N is the number of channels to be defined.

It is recommended that the .CDFN request be used at the beginning of a program, before any I/O operations have been initiated. If more than one .CDFN request is used, the channel areas must either start at the same location or not overlap at all. The two requests .SRESET and .HRESET cause the user's channels to revert to the original 16 channels defined at program initiation. Hence, any .CDFNs must be reissued after using those directives.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An attempt was made to define fewer channels than already exist.</td>
</tr>
</tbody>
</table>

Example:

```
.MCALL .V2,...,REGDEF
.V2...
.REGDEF
.MCALL .CDFN,.PRINT,.EXIT
.START: .CDFN #RLIST,#CHANL,#48.
.B43 BADCDF
.PRINT #MSG1
.EXIT
BADCDF: .PRINT #MSG2
.EXIT
MSG1: .ASCIZ /,CDFN O.K./
.EVEN
MSG2: .ASCIZ /BAD ,CDFN/ 
```

January 1976 9-26
Programmed Requests

.EVEN
RELIST \BLKW 3 \ITEM ARGUMENT LIST
CHANL \BLKW 40,.#5 \ROOM FOR CHANNELS
\END \START

The example defines 40 (decimal) channels to start at location CHANL. An error occurs if 40 or more channels are already defined.
This page intentionally blank.
Programmed Requests

9.4.2 .CHAIN

This request allows a background program to pass control directly to another background program without operator intervention. Since this process may be repeated, a large "chain" of programs can be strung together.

The area from locations 500-507 contains the device name and file name (in RAD50) to be chained to, and the area from locations 510-777 is used to pass information between the chained programs.

Macro Call: .CHAIN

Notes:

1. No assumptions should be made concerning which areas of memory will remain intact across a .CHAIN. In general, 500-777 is the only area guaranteed to be preserved across a .CHAIN.

2. I/O channels are left open across a .CHAIN for use by the new program. However, I/O channels opened via a .CDFN request are not available in this way. Since the monitor reverts to the original 16 channels during a .CHAIN, programs which leave files open across a .CHAIN should not use .CDFN. Furthermore, non-resident device handlers are released during a .CHAIN, and must be FETCHed again by the new program.

3. A program can determine whether it was CHAINEd to or RUN from the keyboard by examining bit 8 of the JSW. This bit is on during program execution only if the program was entered via CHAIN. If a program normally loads into area 500-777, bit 8 of the JSW should be set during program assembly. This causes the monitor to load the area properly. If the bit is not set, locations 500-777 are preserved from the chaining program, causing the new program to malfunction.

Errors:

.CHAIN is implemented by simulating the monitor RUN command (described in Chapter 2), and can produce any errors which RUN can produce. If an error occurs, the .CHAIN is abandoned and the Keyboard Monitor is entered.

When using .CHAIN, care should be taken for initial stack placement, since the program being "chained to" is started. The Linker normally defaults the initial stack to 1000 (octal); if caution is not observed, the stack may destroy chain date before it can be used (see Chapter 2, the RUN command).
Programmed Requests

Example:

```
MCALL .V2:.REGDEF
.V2:
REGDEF
MCALL .CHAIN,.TTYIN

START:
MOV #500,R1
MOV #CHPTR,R2
/DEVICE, FILE NAME TO 500-511
.REPT 4
MOV (R2)*,(R1)*

ENDR

LOOP:
TTYIN
MOV B R0,(R1)*
CMB8 R0,#12
IN LOCATIONS 512 AND UP
BNE LOOP
CLR8 (R1)*
LPUT IN A NULL BYTE

CHAIN

CHPTR:
.RAD50 /OK /
.RAD50 /TECO /
.RAD50 /SAV/
.END START
```

.CHCOPY

9.4.3 .CHCOPY (F/B only)

The .CHCOPY request opens a channel for input, logically connecting it to a file which is currently open by the other job for either input or output. This request may be used by either the foreground or the background. .CHCOPY must be done before the first .READ or .WRITE.

Macro Call: .CHCOPY .area, .chan, .ochan

where: .chan is the channel which the job will use to read the data.

.offer is the channel number of the other job which is to be copied

Request Format:

```
R0 => .area: 13 .chan .ochan
```

.CHCOPY is legal only on files which are on disk or DECTape; however, no errors are detected by the system if another device is used. (To close a channel following use of .CHCOPY, use either the .CLOSE or .PURGE request.)

Notes:

1. If the other job's channel was opened via an .ENTER in order to create a file, the copier's channel indicates a file which extends to the highest block that the creator of the file had written at the time the .CHCOPY was executed.

January 1976 9-28
Programmed Requests

2. A channel which is open on a nonfile-structured device should not be copied, because intermixture of buffer requests may result.

3. A program can write to a file (which is being created by the other job) on a copied channel just as it could if it were the creator. When the copier's channel is closed, however, no directory update takes place.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Other job does not exist, does not have enough channels defined, or does not have the specified channel (.ochan) open.</td>
</tr>
<tr>
<td>1</td>
<td>Channel (.chan) already open.</td>
</tr>
</tbody>
</table>

Example:

In this example, .CHCOPY is used to read data currently being written by the other job. The correct block number and channel to read is obtained by a .RCVDW command. The channel number will be in MSG+4.

```
.MCALL ..V2..,REGDEF
..V2..
..REGDEF
..MCALL .CHCOPY,.RCVDW,.PURGE,.READW,.EXIT,.PRINT

ST (STI)
..PURGE #0    ;MAKE SURE WE HAVE CLEAR CHANNEL
..RCVDW #AREA,#MSG,#2   ;READ TWO WORDS, BLOCK #1 AND CHANNEL

BCS NOJOB     ;NO JOB THERE
.CHCOPY #AREA,#0,MSG+4 ;CHANNEL # IS IN THERE
BCS BUSY      ;BUT BUSY
.READW #AREA,#0,BUFF,#256,MSG+2 ;GET THE CORRECT BLOCK

BCS RERR      ;READ ERROR
..PRINT #OKMSG
..EXIT

NOJOB: PRINT #MSG1
..EXIT

BUSY: PRINT #MSG2
..EXIT

RERR: PRINT #MSG3
..EXIT

..AREA1 .BLKW 5
..MSG1 .BLKW 5
..BUFF1 .BLKW 256,
..MSG21 .ASCIZ /NO JOB/:
..MSG21 .ASCIZ /BUSY/:
..MSG31 .ASCIZ /READ ERROR/:
..OKMSG1 .ASCIZ /READ OK/:
..EVEN

..EXIT
..END ..ST
```

9-29
Programmed Requests

.CLOSE

9.4.4 .CLOSE

The .CLOSE request terminates activity on the specified channel and frees it for use in another operation. The handler for the associated device must be in memory.

Macro Call: .CLOSE .chan

Request Format:

\[ R0 \rightarrow \text{[6.chan]} \]

A .CLOSE is required on any channel opened for either input or output. A .CLOSE request specifying a channel that is not opened is ignored.

A .CLOSE performed on a file which was opened via .ENTER causes the device directory to be updated to make that file permanent. A file opened via .LOOKUP does not require any directory operations. If the device associated with the specified channel already contains a file with the same name and extension, the old copy is deleted when the new file is made permanent. When an entered file is .CLOSEd, its permanent length reflects the highest block written since it was entered; for example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area which was allocated at .ENTER time, the unused blocks are reclaimed as an empty area on the device.

Errors:

.CLOSE does not return any errors. If the device handler for the operation is not in memory, a fatal monitor error is generated.

Example:

An example which illustrates the .CLOSE request follows the discussion of the .WRITW request in Section 9.4.47.
Programmed Requests

9.4.5 .CMKT (F/B only)

The .CMKT request causes one or more outstanding mark time requests to be cancelled (mark time requests are discussed in Section 9.4.22).

Macro Call: .CMKT .area, .id, .time

where: .id is a number used to identify each mark time request to be cancelled. If more than one mark time request has the same .id, that with the earliest expiration time is cancelled. If .id = 0, all nonsystem mark time requests (i.e., in the range 1-177377) for the issuing job are cancelled.

.time is the pointer to a two-word area in which the Monitor will return the amount of time remaining in the cancelled request. The first word contains the high-order time, the second contains the low-order. If an address of 0 is specified, no value is returned. If .id = 0, the .time parameter is ignored and need not be indicated.

Request Format:

R0 ⇒ .area: 23 0
   .id
   .time

Notes:

1. Cancellation of a mark time request frees the associated queue element for other uses.

2. A mark time request can be converted into a timed wait by issuing a .CMKT followed by a .TWAIT, and specifying the same .time area.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The .id was not zero; a mark time with that identification number could not be found (implying that the request was never issued or that it has already expired).</td>
</tr>
</tbody>
</table>

January 1976
Programmed Requests

Example:

See the example following the description of the .MRKT request.

9.4.6 .CNTXSW (F/B only)

A context switch is an operation performed when a transition is made from running one job to running the other. The .CNTXSW request is used to specify locations to be included in the context switch.

Macro Call: .CNTXSW .area, .addr

where: .addr is a list of addresses terminated by a zero word. The addresses in the list must be even and:

a. in the range 2-476, or
b. in the user job area, or
c. in the I/O page (addresses 160000-177776).

Request Format:

R0 \= .area: \begin{array}{c}
\text{330} \\
.\text{addr}
\end{array}

The system always saves the parameters it needs to uniquely identify and execute a job, including all registers, and the locations:

34/36 Vector for TRAP instruction
40-52 System Communication Area

If an .SPPA request (Section 9.4.37) has been executed with a non-zero address, all floating point registers and the floating point status are also saved.

It is possible that both jobs may want to share the use of a particular location and that location is not included in normal context switch operations. For example, if a program uses the IOT instruction to perform some internal user function (such as print error messages), it must set up the vector at 20 and 22 to point to an internal IOT trap handling routine. If both foreground and background wish to use IOT, the IOT vector must always point to the proper location for the job which is executing. Including locations 20 and 22 in the .CNTXSW list for both jobs will accomplish this.

If .CNTXSW is issued more than once, only the latest list is used; the previous address list is discarded. Thus, all addresses to be switched must be included in one list. If the address (.addr) is zero, no extra locations are switched. The list may not be in an area
Programmed Requests

into which the USR swaps, nor may it be modified while a job is running.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>One or more of the above conditions was violated.</td>
</tr>
</tbody>
</table>

Example:

In this example, .CNTXSW request is used to specify that locations 20 and 22 (IOT vector) and certain necessary EAE registers be context switched. This allows both jobs to use IOT and the EAE simultaneously yet independently.

```
.MCALL    .V2,...,REGDEF,.CNTXSW,.PRINT,.EXIT
 .V2,
 .REGDEF
 START:   MOV   #LIST,R0    /SET R0 TO OUR OWN LIST
 .CNTXSW ,#SWAPLS     /THE LIST OF ADDRS IS
 .BCC  15
 .PRINT  #ADDERR     /ADDRESS ERROR(SHOULD NOT
 .EXIT   OCCUR)        

15:
 .PRINT  #CNTQK
 .EXIT

SWAPLS:  .WORD  20        /ADDRESSES TO INCLUDE IN LIST
 .WORD  22
 .WORD  177302
 .WORD  177304
 .WORD  177310

LISI:     .BYTE  0,33     /FUNCTION CODE WORD
 .WORD  0     /THE MACRO FILLS THIS ONE.

ADDERR:   .ASCIZ    /ADDRESSING ERROR/

CNOTQK:   .ASCIZ    /CONTEXT SWITCH OK/

.EVEN
.END   START
```

9.4.7 .CSIGEN

The .CSIGEN request calls the Command String Interpreter (CSI) in general mode to process a standard RT-11 command string (see Chapter 2 for the description of a standard command string). In general mode, all file .LOOKUPs and .ENTERS as well as handler .FETCHs are
Programmed Requests

performed. When called in general mode, the CSI closes channels 0-10 (octal).

Macro Call: .CSIGEN .devspc, .defext, .cstring

where: .devspc is the address of the memory area where the
device handlers (if any) are to be loaded.

.defext is the address of a four-word block which
contains the RAD50 default extensions. These
extensions are used when a file is specified
without an extension.

.cstring is the address of the ASCIZ input string or a
#0 if input is to come from the console
terminal. (In a F/B environment only, if the
input is from the console terminal, an
.UNLOCK of the USR is automatically
performed, even if the USR is locked at the
time.) If the string is in memory, it must
not contain a <CR><LF>, but must terminate
with a zero byte. If the .cstring field is
left blank, input is automatically taken from
the console terminal.

.CSIGEN loads all necessary handlers and opens the files as specified.
The area specified for the device handlers must be large enough to
hold all the necessary handlers simultaneously. If the device
handlers exceed the area available, the user program may be destroyed.
The system, however, is protected from this.

When the EMT is complete, register 0 points to the first available
location above the handlers.

The four-word block pointed to by .defext is arranged as:

Word 1: default extension for all input channels

Words 2,3, and 4: default extensions for output channels 0,1,2
respectively

If there is no default for a particular position, the associated word
must contain a zero. All extensions are expressed in Radix 50. For
example, the following block can be used to set up default extensions
for a macro assembler:

DEFEXT: .RAD50 "MAC"
.RAD50 "OBJ"
.RAD50 "LST"
.WORD 0

In the command string:

*DT0:ALPHA,DT1:BETA=DT2:INPUT

the default extension for input is MAC; for output, OBJ and LST. The
following cases are legal:

*DT0:OUTPUT=
*DT2:INPUT

In other words, the equal sign is not necessary in the event that only
input files are specified.

January 1976 9-34
Programmed Requests

When control returns to the user program after a call to .CSIGEN, all the specified files have been opened for input and/or output. The association is as follows: the three possible output files are assigned to channels 0, 1, and 2; the six input slots are assigned to channels 3 through 10. A null specification causes the associated channel to remain inactive. For example, in the following string:

```
*,LP:=F1,F2
```

channel 0 is inactive since the first slot is null. Channel 1 is associated with the line printer, and channel 2 is inactive. Channels 3 and 4 are associated with two files on DK1, while channels 5 through 10 are inactive. The user program can determine whether a channel is inactive by issuing a .WAIT request on the associated channel, which returns an error if the channel is not open.

Switches and their associated values are returned on the stack; see Section 9.4.8.1 for a description of the way switch information is passed.

Errors:

If CSI errors occur and input was from the console terminal, an error message describing the fault is printed on the terminal and the CSI retries the command (these messages appear in Section 9.4.8.1). If the input was from a string, the carry bit is set and byte 52 contains the error code. The errors are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Illegal command (bad separators, illegal filename, command too long, etc.).</td>
</tr>
<tr>
<td>1</td>
<td>A device specified is not found in the system tables.</td>
</tr>
<tr>
<td>2</td>
<td>Unused.</td>
</tr>
<tr>
<td>3</td>
<td>An attempt to .ENTER a file failed because of a full directory.</td>
</tr>
<tr>
<td>4</td>
<td>An input file was not found in a .LOOKUP.</td>
</tr>
</tbody>
</table>

Example:

This example uses the general mode of the CSI in a program to copy an input file to an output file. Command input to the CSI is from the console terminal.

```
MCALL .V2,...,REGDEF
.V2,...
REGDEF
MCALL .CSIGEN,READW,PRINT,EXIT,WRITE,CLOSE,RESET
ERRWD=52

START: .CSIGEN #SPACE,#DEXT  GET STRING FROM TERMINAL
MOV R0,BUFF   F0 WAS FIRST FREE LOCATION
CLR INBLK    INPUT BLOCK #
MOV #LIST,R5  JEMT ARGUMENT LIST
READW R5,#3,BUFF,#256,INBLK JREAD CHANNEL 3
BCC 23 NO ERRORS
BCL #ERRWD JEOF ERROR?
BEQ EOF YES
MOV #INERR,R0
```

9-35 January 1976
Programmed Requests

151 .PRINT R0
CLR R0
EXIT

251 .WRITN R5, #0, BUFF, #256, INBLK
WRITE THE BLOCK
BCC NOERR
INC ERROR WRITING
MOV #*TERI, R0
BR 10
HARD OUTPUT ERROR

NOERR: INC INBLK
GET NEXT BLOCK
BR READ
LOOP UNTIL DONE

EOF: .CLOSE #0
CLOSE OUTPUT CHANNEL
.CLOSE #3
AND INPUT CHANNEL
.SRESET
RELEASE HANDLEK FROM MEMORY
BR START
GO FOR NEXT COMMAND LINE

DEXTI .WORD 0, 0, 0, 0
INO DEFAULT EXTENSIONS
BUFFI .WORD 0
I/O BUFFER START
INBLKI .WORD 0
RELATIVE BLOCK TO READ/WRITE
LISTI .BLKW 5
IEMT ARGUMENT LIST
INERRI .ASCIZ /INPUT ERROR/
.EVEN
WTERRI .ASCIZ /OUTPUT ERROR/
.EVEN

DSPACE*, .END START
HANDLER SPACE

.CSISPC

9.4.8 .CSISPC

The .CSISPC request calls the Command String Interpreter in special mode to parse the command string and return file descriptors and switches to the program. In this mode, the CSI does not perform any handler fetches, .CLOSEs, .ENTERs, or .LOOKUPs.

Macro Call: .CSISPC .outspc, .deffext, .cstring

where: .outspc is the address of the 39-word block to contain the file descriptors produced by .CSISPC. This area may overlay the space allocated to .cstring if desired.

.deffext is the address of a four-word block which contains the RAD50 default extensions. These extensions are used when a file is specified without an extension.

July 1975 9-36
Programmed Requests

.cstring is the address of the ASCII input string or a #0 if input is to come from the console terminal. If the string is in memory, it must not contain a <CR><LF> but must terminate with a zero byte. If .cstring is blank, input is automatically taken from the console terminal.

The 39-word file description consists of nine file descriptor blocks (five words for each of three possible output files; four words for each of six possible input files) which correspond to the nine possible filenames (three output, six input). If any of the nine possible filenames are not specified, the corresponding descriptor block is filled with zeroes.

The five-word blocks hold four words of RAD50 representing devifile.ext, and 1 word representing the size specification given in the string. (A size specification is a decimal number enclosed in square brackets [], following the output file descriptor.) For example,

*DT3:LIST.MAC[15]=PR:

Using special mode, the CSI returns in the first five word slot:

16101 .RAD50 for DT3
46173 .RAD50 for LIS
76400 .RAD50 for T
50553 .RAD50 for MAC
00017 Octal value of size request

In the fourth slot (starting at an offset of 36 (octal) bytes into .outspce), the CSI returns:

63320 .RAD50 for PR
0 No file name
0 Specified
0

Since this is an input file, only four words are returned.

Switches and their associated values are returned on the stack. See Section 9.4.8.1.

Errors:

Errors are the same as in general mode. However, since .LOOKUPs and .ENTERs are not done, the error codes which are valid are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Illegal command line</td>
</tr>
<tr>
<td>1</td>
<td>Illegal device</td>
</tr>
</tbody>
</table>

Example:

This example illustrates the use of the special mode of CSI. This example could be a program to read a file which is not in RT-ll format to a file under RT-ll.
Programmed Requests

```
MCALL .V2,.REGDEF
.V2,
REGDEF
MCALL.CSISP,PRINT,EXIT,ENTER,CLOSE

START:CSISP #OUTSPC,#DEXT,#CSTRNG 1GET INPUT FROM A STRING IN MEMORY

BCC 28
MOV #SYNERR,R0 1SYNTAX ERROR
EXIT

131:PRINT 1ERROR MESSAGE
EXIT

231:ENTER #LIST, #0, #OUTSPC, #64, ENTER FILE UNDER RT-11
BCC 38
MOV #ENMSG, R0 1ENTER FAILED
BR 18

381:JBR R5,INPUT 1ROUTINE INPUT WILL USE THE INFORMATION AT #OUTSPC+36 TO READ INPUT FROM THE NON-RT11 DEVICE. INPUT IS PROCESSED AND WRITTEN VIA WRIT REQUESTS TO MAKE OUTPUT FILE PERMANENT

CLOSE #0 1AND EXIT PROGRAM
EXIT

CSTRNG1:ASCIZ "DT41RTFIL,MAC=DT21DOS,MAC"

DEXTI:WORD 0,0,0,0 1NO DEFAULT EXTENSIONS
LISTI:BLKW 5 1LIST FOR EMT CALLS
SYNERRI:ASCIZ "CSI ERROR"
ENMSGI:ABCIZ "ENTER FAILED"

INPUT:RTS R5
OUTSPC=.

.END START 1CSI LIST GOES HERE
```

9.4.8.1 Passing Switch Information

In both general and special modes of the CSI, switches and their associated values are returned on the stack. A CSI switch is a slash (/) followed by any character. The CSI does not restrict the switch to printing characters, although it is suggested that printing characters be used wherever possible. The switch can be followed by an optional value, which is indicated by a ; or | separator. The ; separator is followed by either an octal number or by one to three alphanumeric characters, the first of which must be alphabetic, which are converted to Radix-50. The | separator is followed by a decimal value. Switches can be associated with files with the CSI. For example:

```
*DK:FOO/A,DT4:FILE,OBJ/A:100
```

In this case, there are two A switches. The first is associated with the input file DK:FOO. The second is associated with the input file DT4:FILE,OBJ, and has a value of 100(8). The stack output of the CSI is as follows:

```
9-38
```
Programmed Requests

<table>
<thead>
<tr>
<th>Word #</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
<td>Number of switches found in command string. If N=0, no switches were found.</td>
</tr>
<tr>
<td></td>
<td>(top of stack)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Switch value and file number</td>
<td>Even byte = 7-bit ASCII switch value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits 8-14 = Number (0-10) of the file with which the switch is associated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bit 15 = 1 if the switch had a value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 0 if the switch had no value.</td>
</tr>
<tr>
<td>3</td>
<td>Switch value or next switch</td>
<td>If word 2 was less than 0, word 3 = switch value. If word 2 was greater than 0, this word is the next switch value (if it exists).</td>
</tr>
</tbody>
</table>

For example, if the input to the CSI is:

*FILE/B:20,FIL2/E=DT3:INPUT/X;SY:20

on return, the stack is:

Stack Pointer→  
<table>
<thead>
<tr>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>101530</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>101530</td>
</tr>
<tr>
<td>075250</td>
</tr>
<tr>
<td>505</td>
</tr>
<tr>
<td>100102</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

Three switches appeared.

Last switch=X; with file 3, has a value.

Value of switch X=20

Next switch =X; with file 3, has a value.

Next value of switch X=RAD50 code for SY.

Next switch=E; associated with file 1, no value.

Switch=B; associated with file 0 and has a value.

Value is 20.

As an extended example, assume the following string was input for the CSI in general mode:


Assume also that the default extension block is:

<table>
<thead>
<tr>
<th>DEFEXT:</th>
<th>.RAD50 'MAC' ;INPUT EXTENSION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.RAD50 'OP1' ;FIRST OUTPUT EXTENSION</td>
</tr>
<tr>
<td></td>
<td>.RAD50 'OP2' ;SECOND OUTPUT EXTENSION</td>
</tr>
<tr>
<td></td>
<td>.RAD50 'OP3' ;THIRD OUTPUT EXTENSION</td>
</tr>
</tbody>
</table>

The result of this CSI call would be:

1. A file named FILE.OP1 is entered on channel 0 on device DK; channel 1 is open for output to the device LP; a 20-block file named FILE2.OP3 is entered on the system device on channel 2.
Programmed Requests

2. Channel 3 is open for input from paper tape; channel 4 is open for input from a file IN1.MAC on device DT1; channel 5 is open for input from IN2.MAC on device DT2.

3. The stack contains switches and values as follows:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>102515</td>
</tr>
<tr>
<td>7</td>
<td>2102</td>
</tr>
</tbody>
</table>

Explanation
2 switches found in string.
Second switch is 'M', associated with Channel 5; has a numeric value.
Numeric value is 7.
Switch is 'B', associated with Channel 4; has no numeric value.

If the CSI were called in special mode (Section 9.4.8), the stack would be the same as for the general mode call, and the descriptor table would contain:

```
.OUTSPC: 15270 ;RAD50 "DK'
23364 ;RAD50 "FIL'
17500 ;RAD50 "E'
60137 ;RAD50 "OP1'
10 ;LENGTH OF 8 BLOCKS
46600 ;RAD50 "LP'
0 ;NO NAME OR LENGTH SPECIFIED
0
0
75250 ;RAD50 "SY'
23364 ;RAD50 "FIL'
22100 ;RAD50 "E2'
60141 ;RAD50 "OP3'
24 ;LENGTH OF 20 (DECIMAL)
63320 ;RAD50 "PR'
0
0
16077 ;RAD50 "DT1'
35217 ;RAD50 "IN1'
0 ;RAD50 ''
50553 ;RAD50 "MAC'
16100 ;RAD50 "DT2'
35220 ;RAD50 "IN2'
0 ;RAD50 ''
50553 ;RAD50 "MAC'
0

(twelve more zero words are returned)
```

Keyboard error messages which may occur from incorrect use of the CSI when input is from the console keyboard include:

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ILL CMD?</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>?FIL NOT FND?</td>
<td>Input file was not found.</td>
</tr>
</tbody>
</table>

9-40
Programmed Requests

?DEV FUL?
?ILL DEV?

Output file will not fit.
Device specified does not exist.

Notes:

1. In many cases, the user program does not need to process switches in CSI calls. However, the user at the console may inadvertently enter switches. In this case, it is wise for the program to save the value of the stack pointer before the call to the CSI, and restore it after the call. In this way, no extraneous values will be left on the stack.

2. In the F/B System, calls to the CSI which require console terminal input will always do an implicit .UNLOCK of the USR. This should be kept in mind when using .LOCK calls.

9.4.9 .CSTAT (F/B only)

This request furnishes the user with information about a channel. It is supported only in the F/B environment; no information is returned in the Single-Job Monitor.

Macro Call: .CSTAT .area, .chan, .addr

where: .addr is the address of a 6-word block which is to contain the status

Request Format:

R0 => .area: 27 , chan , addr

The 6 words passed back to the user are:

1. Channel status word (see Section 9.4.34)
2. Starting block number of file (0 if sequential-access device or if channel was opened with a nonfile-structured .LOOKUP or .ENTER)
3. Length of file (no information if nonfile-structured device or if channel was opened with a nonfile-structured .LOOKUP or .ENTER)
4. Highest block written since file was opened (no information if nonfile-structured device)
5. Unit number of device with which this channel is associated
6. RAD50 of the device name with which the channel is associated (this is a physical device name, unaffected by any user name ASSIGNment in effect)
Programmed Requests

The fourth word (highest block) is maintained by the .WRITE requests. If data is being written on this channel, the highest relative block number is kept in this word.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The channel is not open.</td>
</tr>
</tbody>
</table>

Example:
In this example, .CSTAT is used to determine the .RAD50 representation of the device with which the channel is associated.

```
.MCALL ..V2..,REGDEF,CSIGEN,CSTAT
..V2..
.REGISTER

.MCALL ,PRINT,,EXIT

STI
.CSIGEN #DEVSDC,#DEFEXT /OPEN FILES
.CSTAT #AREA,#B,#ADDR /GET THE STATUS
BCS NOCHAN /CHANNEL 0 NOT OPEN
MOV #ADDR+12,R5 /POINT TO UNIT #
MOV (R5),R0 /UNIT # TO R0
ADD (PC)+,R0 /MAKE IT RAD50
.RAD50 / 0/
ADD (R5),R0 /GET DEVICE NAME
MOV R0,DEVNAM /DEVNAM HAS RAD50 DEVICE NAME
.EXIT
AREA1 BLKW 5 /EMIT ARG LIST
ADDR1 BLKW 6 /AREA FOR CHANNEL STATUS
DEVNAM WORD 0 /STORAGE FOR DEVICE NAME
DEFEXTI WORD 0,0,0,0
NOCHAN PRINT #MSG
.EXIT
MSG1 ,ABICIZ /NO OUTPUT FILE/
.EVEN
DEVSDC*.
.END 8T
```

9.4.10 .DELETE
The .DELETE request deletes a named file from an indicated device.
Macro Call: .DELETE ,area, ,chan, ,dblk, ,count

January 1976  9-42
Programmed Requests

where: .count is used by magtape/cassette only, (Refer to Appendix H for more information concerning the magtape and cassette handlers.)

Request Format:

R0 ⇒ .area: 0 .chan
 .dblk
 .count

Note:

The channel specified in the .DELETE request must not be in use when the request is made, or an error will occur. The file is deleted from the device, and an empty (UNUSED) entry of the same size is put in its place. A .DELETE issued to a nonfile-structured device is ignored. .DELETE requires that the handler to be used be in memory at the time the request is made. When the .DELETE is complete, the specified channel is left inactive.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Channel is active</td>
</tr>
<tr>
<td>1</td>
<td>File was not found in the device directory</td>
</tr>
</tbody>
</table>

Example:

This example uses the special mode of CSI to delete files.

```
.MCALL .V2,...,REGDEF
...V2,...
.REGDEF
.MCALL .BRESET,.CSISPC,.DELETE,.PRINT,.EXIT

START: .BRESET

.MAKE SURE CHANNELS ARE FREE

.CSISPC #OUTSPC,#DEFEXT

.GET COMMAND LINE TERMINAL DIALOG WAS IDENTITY

.DELETE #LIST,#0,#INSPC

.USE CHANNEL 0 TO DELETE THE FILE WHICH IS AT THE FIRST INPUT SLOT.

.BCC .PRINT .NOFILE .START

.BR .NOFILE .START

.NOFILE: .ABCIZ /FILE NOT FOUND/

.EVEN

.DEFEXTI .RAD50 /MAC/

.WORD 0,0,0

.LISTI .BLKW 2

.OUTSPC=. INSPC=.36

.BLKW 39,

.END .START
```

INSPC is the address of the first input slot in the CSI input table.
Programmed Requests

.DEVICE

9.4.11 .DEVICE (F/B Only)

This request allows the user to set up a list of addresses to be loaded with specified values when a program is terminated. Upon an .EXIT or CTRL C, this list is picked up by the system and the appropriate addresses are set up with the corresponding values. This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates.

Macro Call: .DEVICE .area, .addr

where: .addr is the address of the list of masks and words.

Request Format:

R0 ➔ .area: [14 0]

.addr

The list is composed of address/value pairs and should be terminated by a 0 address. Only one list can be active at a given time. If multiple .DEVICE requests are given, the last list specified is the one used.

Note:

When the job is terminated for any reason, the list is scanned once. At that point, the monitor disables the feature until another .DEVICE call is executed. Thus, background programs which are re-enterable should include .DEVICE as a part of the reenter code.

Errors:

None.

Example:

The following example shows how .DEVICE is used to disable interrupts from the AFC11 (A-D converter sub-system).

```
.MCALL .V2,...,REGDEF
.V2,
.REGION
.MCALL .DEVICE,.EXIT

START: .DEVICE #LIST
.EXIT

LIST1: .BYTE 0,14

SEM T ARG LIST
```

9-44
Programmed Requests

ATOD: 172570 ATOD
0
0
.END START

ADDRESS IS 172570
I JAM A 0 INTO IT
THIS 0 TERMINATES THE LIST.

.DSTATUS

9.4.12 .DSTATUS

This request is used to obtain information about a particular device.

Macro Call: .DSTATUS .cblk, .devnam

where: .cblk is the 4-word space used to store the status information.

.DSTATUS looks for the device specified by .devnam and, if found, returns four words of status starting at the address specified by .cblk. The four words returned are:

1. Status Word

Bits 7-0: contain a number which identifies the device in question. The values (octal) currently defined are:

0 = RK05 Disk
1 = TC11 DECTape
2 = Reserved
3 = Line Printer
4 = Console Terminal
5,6 = Reserved
7 = PC11 High-speed Reader
10 = PC11 High-speed Punch
11 = Magtape (TM11, TMA11)
12 = RF11 Disk
13 = TAJ1 Cassette
14 = Card Reader (CR11, CM11)
15 = Reserved
16 = RJS03/4 Fixed-head Disks
17 = Reserved
20 = TAJ16 Magtape
21 = RP02, RP03 Disk
22 = RX01 Disk

Bit 15: 1= Random-access device (disk, DECTape)
0= Sequential-access device (line printer, paper tape, card reader, magtape, cassette, terminal)

Bit 14: 1= Read-only device (card reader, paper tape reader)

Bit 13: 1= Write-only device (line printer, paper tape punch)

Bit 12: 1= Non RT-ll directory-structured device (magtape, cassette)
Programmed Requests

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

2. Handler size.

   The size of the device handler, in bytes.

3. Entry point.

   Non-zero implies the handler is now in memory; zero
   implies it must be .FETCHed before it can be used.

4. Device size.

   The size of the device (in 256-word blocks) for block-
   replaceable devices; zero for sequential-access devices.

   The device name may be a user-assigned name.

Refer to the RT-ll Software Support Manual for greater detail.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Device not found in tables.</td>
</tr>
</tbody>
</table>

Example:

This example shows how to determine if a particular device handler
is in memory and, if it is not, how to .FETCH it there.

```
 MCALL ..V2,...REGDEF
 ..V2...
 REGDEF
 MCALL ,DSTATUS, ,PRINT, ,EXIT, ,FETCH

START: ,DSTATUS #CORE,#FPRTR ,GET STATUS OF DEVICE
       BCC 18
       PRINT #ILLDEV ,DEVICE NOT IN TABLES
       EXIT

131 TST CORE+4 ,IS DEVICE RESIDENT?
       BNE 28
       FETCH #HNDLR,#FPRTR ,NO, GET IT
       BCC 28
       PRINT #FEFAIL ,FETCH FAILED
       EXIT

281 PRINT #FECHOK
       EXIT

CORE1 : BLKW 4 ,DSTATUS GOES HERE
FPTR1  : RADS0 ,/DT0/ ,DEVICE NAME
         RADS0 ,/FILE MAC/ ,FILE NAME
FEFAIL1 : ASCIZ ,/FETCH FAILED/
ILLDEV1 : ASCIZ ,/ILLEGAL DEVICE/
         EVEN
FECHOK1 : ASCIZ ,/FETCH OK/,
         EVEN
HNDLR#: ,END START ,HANLER WILL GO HERE
```
Programmed Requests

9.4.13 .ENTER

The .ENTER request allocates space on the specified device and creates a tentative entry for the named file. The channel number specified is associated with the file. (Note that if the program is overlaid, channel 15 is used by the overlay handler and should not be modified.)

Macro Call: .ENTER ,area, ,chan, ,dblk, ,length, ,count

where: length is the file size specification. The file length allocation is as follows:

0 - either 1/2 the largest empty entry or the entire second largest empty entry, whichever is largest. (A maximum size for non-specific .ENTERs may be patched in the monitor.)

M - a file of M blocks. M may exceed the maximum mentioned above.

-1 - the largest empty entry on the device.

.count file number for magtape/cassette (see Appendix H); if this argument is blank, a value of zero is assumed.

Request Format:

R0 \rightarrow ,area: [2, ,chan
 ,dblk
 ,length
 ,count

The file created with an .ENTER is not a permanent file until the .CLOSE on that channel is given. Thus, the newly created file is not available to .LOOKUP and the channel may not be used by .SAVSTATUS requests. However, it is possible to go back and read data which has just been written into the file by referencing the appropriate block number. When the .CLOSE to the channel is given, any already existing permanent file of the same name on the same device is deleted and the new file becomes permanent. Although space is allocated to a file during the .ENTER operation, the actual length of the file is determined when .CLOSE is requested.

Each job may have up to 256 files open on the system at any time. If required, all 256 may be opened for output with the .ENTER function. .ENTER requires that the device handler be in memory when the request is made. Thus, a .FETCH should normally be executed before a .ENTER
Programmed Requests

can be done. On return, R0 contains the size of the area actually allocated for use.

Notes:

When using the 0 length feature of .ENTER, it must be kept in mind that less than the largest empty space is allocated. This can have an important effect in transferring files between devices (particularly DECTape) which have a relatively small capacity. For example, to transfer a 200-block file to a DECTape on which the largest available empty space is 300 blocks, a 0 length transfer will not work. Since the .ENTER allocates half the largest space, only 150 blocks are really allocated and an output error will occur during the transfer. If a specific length of 200 is requested, however, the transfer will proceed without error.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Channel is in use.</td>
</tr>
<tr>
<td>1</td>
<td>In a fixed length request, no space greater than or equal to M was found, or in a non-specific request the device or the directory was found to be full.</td>
</tr>
</tbody>
</table>

Example:

.ENTER may be used to open a file on a specified device, and then write data from memory into that file as follows:

```
.MCALL ..V2...,REGDEF,ENTER,WRIW,CLOSE,PRINT
.MCALL ,SRESET,EXIT,FETCH
..V2...
.REGDEF

.START.

.SRESET

.MAKE SURE ALL CHANNELS ARE CLOSED.

.FETCH #CORSPC,#FPRT

.FETCH DEVICE HANDLER.

.BCS BADPET

.FETCH ERROR, PROBABLY ILLEGAL DEVICE.

.ENTER #AREA,#0,#FPRT

.OPEN A FILE ON THE DEVICE SPECIFIED, LENGTH 0 WILL GIVE 1/2 OF LARGEST EMPTY SPACE NOW AVAILABLE.

.BCS BADENT

.I FAILED, CHANNEL PROBABLY BUSY.

.WRIW #AREA,#0,#BUFF,#END=BUF/2,#0

.WRITE DATA FROM MEMORY, THE SIZE IS # OF WORDS BETWEEN BUF AND END, START AT BLOCK 0.

.BCS BADWRT

.WRITE FAILURE.

.CLOSE #0

.CLOSE THE FILE.

.EXIT

.FPRT1 .RADS0 /DK /

.FILE WILL BE ON DK.

.FPRT1 .RADS0 /FILE EXT/

.NAMED FILE, EXT.

_AREA1 .BLK 10

AREA ARGUMENT LIST.

.BADFET1 .PRINT #FMSG

.FMT ARGUMENT LIST.
```

January 1976 9-48
9.4.14 .EXIT

The .EXIT request causes the user program to terminate. When used from a background job under the P/B Monitor and when used under the Single-Job Monitor, .EXIT causes KMON to run in the background area. All outstanding mark time requests are cancelled. Any I/O requests and completion routines pending for that job are allowed to complete. If part of the background job resides where KMON and USR are to be read, the user job is written onto system device scratch blocks. KMON and USR are then loaded and control goes to KMON in the background area. If R0=0 when the .EXIT is done, an implicit INIT command is executed when KMON is entered, disabling the subsequent use of REENTER, START, or CLOSE.

.EXIT also resets any .CDFN and .QSET calls that were done and executes an .UNLOCK if a .LOCK has been done. Thus, the .CLOSE command from the Keyboard Monitor does not operate for programs which perform .CDFN requests.

In a P/B system, an .EXIT from a completion routine acts as if a double CTRL C has been typed, aborting all I/O in progress before exiting. In general, .EXIT from a completion routine should be avoided.

Macro Call: .EXIT

Errors:

None.
Programmed Requests

.FETCH

9.4.15 .FETCH

The .FETCH request loads device handlers into memory from the system device.

Macro Call: .FETCH .coradd, .devnam

where: .coradd is the address where the device handler is to be loaded.

.devices is the pointer to the RAD50 device name.

The storage address for the device handler is passed on the stack. When the .FETCH is complete, R0 points to the first available location above the handler. If the handler is already in memory, R0 keeps the same value as was initially pushed onto the stack. If the argument on the stack is less than 400(8), it is assumed that a handler .RELEAS is being done. (.RELEAS does not dismiss a handler which was LOADED from the KMON; an UNLOAD must be done.) After a .RELEAS, a .FETCH must be issued in order to use the device again.

Several requests require a device handler to be in memory for successful operation. These include:

.CLOSE .READC .READ
.LOOKUP .WRITC .WRITE
.ENTER .READW .SPFUN
.RENAME .WRITW .DELETE

Since foreground jobs must have handlers resident, a .FETCH from the foreground will give a fatal error if the handler has not been previously LOADED.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The device name specified does not exist, or there is no handler for that device in the system.</td>
</tr>
</tbody>
</table>

Example:

In the following example, the PR and PP handlers are fetched into memory in preparation for their use by a program. The program sets aside handler space from its free memory area.
**Programmed Requests**

```
.MCALL .V2..,REGDEF,.FETCH,.PRINT,.EXIT

.MCALL .V2.,REGDEF

.START

.FETCH FREE,#PRNAME
.BCS FERR
.MOV R0,R2
.FETCH R2,#PPNAME

.IFETCH PR HANDLER
.IMMEDIATELY FOLLOWING
.IP HANDLER, R0 POINTS
.TO THE TOP OF PR
.IHANDLER ON RETURN
.IFROM THAT CALL.

.IFETCH PP HANDLER
.NOP HANDLER
.UPDATE FREE MEMORY
.IPOINTER TO POINT TO
.INEW BOTTOM OF FREE
.IAREA(TOP OF HANDLERS),

.MOV R0,FREE

.PRINT #OK
.EXIT

.OK: .ASCIZ /FETCH O.K./

.FERR: .PRINT #MSG
.EXIT

.MALT

.PRNAME: RAD50 "PR"
.PPNAME: RAD50 "PP"
.MSG: .ASCIZ "DEVICE NOT FOUND" #ERROR MESSAGE

.FREE: .+2
.END START

.IPOINTER TO FREE MEMORY

.GTIM
```

9.4.16 .GTIM

.GTIM allows user programs to access the current time of day. The
time is returned in two words, and is given in terms of clock ticks
past midnight.

Macro Call: .GTIM .area, .addr

where: .addr is a pointer to the two words of time
to be returned.

Request Format:

```
R0 ⇒ .area: 21 0
   .addr
```

9-51
Programmed Requests

The high-order time is returned in the first word, the low-order time in the second word. User programs must make the conversion from clock ticks to hours-minutes-seconds. The basic clock frequency (50 or 60 Hz) may be determined from the configuration word in the monitor (see Section 9.2.6). Under a F/B Monitor, the time of day is automatically reset after 24:00 when a .GTIM is done; under the Single-Job Monitor, it is not. The month is not automatically updated under either monitor.

The clock rate is initially set to 60-cycle. Consult the RT-11 System Generation Manual if conversion to a 50-cycle rate is necessary.

Errors:

None.

Example:

```
.MCALL .V2,.REGDEF,.GTIM,.EXIT
.V2
.REGDEF
START
.GTIM #LIST,#TIME
EXIT .WORD 0,0 INDEX AND HI ORDER TIME IS RETURNED HERE.
LIST .BLKW 2 ARGUMENTS FOR THE EMT
.END START
```

9.4.17 .GTJB

The .GTJB request passes certain job parameters back to the user program.

Macro Call: .GTJB .area, .addr

where: .addr is the address of an eight-word block into which the parameters are passed. The values returned are:

- Word 1 - Job Number, 0=Background, 2=Foreground
- Word 2 - High memory limit
- Word 3 - Low memory limit
- Word 4 - Beginning of I/O channel space
- Word 5-8 - Reserved for future use

Request Format:

```
R0 ⇒ .area: 20 0
   .addr
```

January 1976 9-52
Programmed Requests

In the Single-Job Monitor, the job number is always 0 and the low limit 0.

In the F/B Monitor, the job number can either be 0 or 2. If the job number equals 0 (background job), word 2 equals 0 and word 4 describes where the I/O channel words begin. This is normally an address within the Resident Monitor. When a .CDFN is executed, however, the start of the I/O channel area changes to the user specified area.

Errors:

None.

Example:

Use .GTJB to determine if this program is executing as a foreground or background job.

```
.MCALL .V2,...,REGDEF,.GTJB,.PRINT,.EXIT
.V2...
.REGDEF

START:
.GTJB #LIST,#JOBARG JR0 POINTS TO 1ST WORD ON RETURN FROM CALL.
MOV #FM#G,#R1
TST JOBARG /BACKGROUND?
BNE 1$
MOV #BM#G,#R1
1$: .PRINT #R1
.EXIT

FM#G1 .ASCIZ /PROGRAM IN FOREGROUND/
BM#G1 .ASCIZ /PROGRAM IN BACKGROUND/
.EVEN

LIST1 :BLKN 2 /ARGUMENTS FOR THE EMT
JOBARG1 :BLKN 8 /
.JOB PARAMETERS PASSED BACK HERE.

.END START
```

9.4.18 .HERR/.SERR

.HERR and .SERR are complementary requests used to govern monitor behavior for serious error conditions. During program execution, certain error conditions may arise which cause the executing program to be aborted (for example, trying to pass I/O to a device which has no handler in memory, or trying to load a device handler over the USR). Normally, these errors cause program termination with one of the
Programmed Requests

?M- error messages. However, in certain cases it is not feasible to
abort the program because of these errors; for example, a multi-user
program must be able to retain control and merely abort the user who
has generated the error. .SERR accomplishes this by inhibiting the
monitor from aborting the job. Instead, it causes an error return to
the offending EMT to be taken. On return from that request, the C bit
is set and byte 52 contains a negative value indicating the error
condition which occurred.

.HERR turns off user error interception and allows the system to abort
the job on fatal errors and generate an error message. (.HERR is the
default case.)

Macro Calls: .HERR

.SERR

Errors:

Following is a list of the errors which are returned if soft error
recovery is in effect:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Called USR from completion routine.</td>
</tr>
<tr>
<td>-2</td>
<td>No device handler; this operation needs one.</td>
</tr>
<tr>
<td>-3</td>
<td>Error doing directory I/O.</td>
</tr>
</tbody>
</table>
| -4   | FETCH error. Either an I/O error occurred while reading the handler, or tried to
       load it over USR or RMON. |
| -5   | Error reading an overlay. |
| -6   | No more room for files in the directory. |
| -7   | Illegal address (F/B only); tried to perform a
       monitor operation outside the job partition. |
| -10  | Illegal channel number; number is greater
       than actual number of channels which exist. |
| -11  | Illegal EMT; an illegal function code has
       been decoded. |

Traps to 4 and 10, and floating point exception traps are not
inhibited. These errors have their own recovery mechanism. (See
Section 9.4.42.)

Example:

This example causes a normally fatal error to generate errors back to
the user program. The error returned is used to print an appropriate
message.

```assembly
.MCALL .V2,.REGDEF,.FETCH,.ENTER,.HERR,.SERR
.MCALL .EXIT,.PRINT
.V2..
.REGDEF

STI .SERR

TURN ON SOFTWARE ERROR
RETURNS

.FETCH #MDLR,#PTR
GET A DEVICE HANDLER

BCS FCHERR

.ENTER #AREA,#1,#PTR
OPEN A FILE ON CHANNEL 1

BCS ENERR
```

9-54
Programmed Requests

.MERR
.EXIT

NOW PERMIT ?m-ERRORS.

FCHERRR: MOVB #52, R0
BMI FTLERR
.PRINT #FMSG
.EXIT

HAS IT FATAL
YES
NO... NO DEVICE BY THAT NAME

ENERRR: MOVB #52, R0
BMI FTLERR
.PRINT #EMSG
.EXIT

THIS WILL TURN POSITIVE
ADJUST BY ONE
MAKE IT AN INDEX

FTLERRR: NEG R0
DEC R0
ASL R0
MOV TBL(R0), R0
.PRINT
.EXIT

THIS WILL TURN POSITIVE
ADJUST BY ONE
MAKE IT AN INDEX

TBLR: M1
M2
M3
M4
M5
M6
M7
M8
M10
M11

CAN'T OCCUR IN THIS PROGRAM
NO DEVICE HANDLER IN MEMORY
DIRECTORY I/O ERROR
FETCH ERROR
IMPOSSIBLE FOR THIS PROGRAM
NO ROOM IN DIRECTARY
ILLEGAL ADDRESS (F/B)
ILLEGAL CHANNEL
ILLEGAL EMT

M11: .ASCIZ /NO DEVICE HANDLER/
M3: .ASCIZ "DIRECTORY I/O ERROR"
M4: .ASCIZ /ERROR DOING FETCH/
M5: .ASCIZ /NOT APPLICABLE TO THIS PROGRAM
M8: .ASCIZ /NO ROOM IN DIRECTORY/
M10: .ASCIZ /ADDRESS CHECK ERROR/
M11: .ASCIZ /ILLEGAL CHANNEL/
FMSG: .ASCIZ /ILLEGAL EMT/
EMSG: .ASCIZ /FETCH FAILED/
.EVEN

HDLRR: .BLKW 380
.PTR .RADSO /DT4/
.RADSO /EXAMPL/
.RADSO /MAC/
.AREA .BLKW 4
.END 8T

.HRESET

9.4.19 .HRESET

This request performs the same function as .SRESET, after stopping all
I/O transfers in progress for that job. (.HRESET is not used to clear

9-55
Programmed Requests

a hard-error condition.) Note that in the single-job environment, a hardware RESET instruction is used to terminate I/O, while in a F/B environment, only the I/O associated with the job which issued the .HRESET is affected. All other transfers continue.

Macro call: .HRESET

Errors:

None.

Example:

See the example for .SRESET (Section 9.4.40) for format.

.

9.4.20 .LOCK/.UNLOCK

.LOCK

The .LOCK request is used to "lock" the USR in memory for a series of operations. If all the conditions which cause swapping are satisfied, the user program is written into scratch blocks and the USR is loaded. Otherwise, the USR which is in memory is used, and no swapping occurs. The USR is not released until an .UNLOCK request is given. (Note that in a F/B System, calling the CSI may also perform an implicit .UNLOCK.) A program which has many USR requests to make can .LOCK the USR in memory, make all the requests, and then .UNLOCK the USR; no time is spent doing unnecessary swapping.

In a F/B environment, a .LOCK inhibits the other job from using the USR. Thus, the USR should be locked only as long as necessary.

Macro Call: .LOCK

Note that the .LOCK request reduces time spent in file handling by eliminating the swapping of the USR in and out of memory. If the USR is currently resident, .LOCK is ignored. After a .LOCK has been executed, an .UNLOCK request must be executed to release the USR from memory. The .LOCK/.UNLOCK requests are complimentary and must be matched. That is, if three .LOCK requests are issued, at least three .UNLOCKS must be done, otherwise the USR will not be released. More .UNLOCKS than .LOCKS may occur without error.

Notes:

1. It is vital that the .LOCK call not come from within the area into which the USR will be swapped. If this should occur, the return from the USR request would not be to the user program, but to the USR itself, since the LOCK function inhibits the user program from being re-read.
Programmed Requests

2. Once a .LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an .UNLOCK is done.

3. If a foreground job performs a .LOCK request while the background job owns the USR, foreground execution is suspended until the USR is available. Thus, in this case, it is possible for the background to lock out the foreground (see the .TWAIT request).

Errors:

None.

Example:

See the example following .UNLOCK.

.UNLOCK

The .UNLOCK request releases the User Service Routine from memory if it was placed there with a .LOCK request. If the .LOCK required a swap, the .UNLOCK loads the user program back into memory. If the USR does not require swapping, the .UNLOCK acts as a no-op.

Macro Call: .UNLOCK

Notes:

1. It is important that at least as many .UNLOCKS are given as .LOCKS. If more .LOCK requests were done, the USR remains locked in memory. It is not harmful to give more UNLOCKS than are required; those that are extra are ignored.

2. The .LOCK/.UNLOCK pairs should be used only when absolutely necessary when running two jobs in the F/B system. When a job .LOCKS the USR, the other job cannot get at it until it is .UNLOCKed. Thus, the USR should not be .LOCKed unnecessarily, as this may degrade performance in some cases.

3. In a F/B System, calling the CSI with input coming from the console terminal performs an implicit .UNLOCK.

Errors:

None.

Example:

This example shows the usage of .LOCK, .UNLOCK, and their interaction with the system.

```
.MCALL .V2,.REGDEF,.LOCK,.UNLOCK,.LOOKUP
.MCALL .SETTOP,.PRINT,.EXIT
.V2.
.REGDEF

START:

SYSPTR=54

.SETTOP #SYSPTR

JTRY FOR ALL OF MEMORY

MOV  R0, TOP

1R0 HAS THE TOP
```

January 1976
Programmed Requests

\[\text{LOCK}\] 
\[\text{LOOKUP \#LIST,\#0,\#FILE1}\] 
\[\text{BRING USR INTO MEMORY}\] 
\[\text{LOOKUP A FILE ON CHANNEL 0}\] 
\[\text{BCC 15}\] 
\[\text{ION ERROR, PRINT A}\] 
\[\text{PRINT \#MSG}\] 
\[\text{MESSAGE AND EXIT}\] 
\[\text{EXIT}\] 
\[\text{MOV \#LIST,R0}\] 
\[\text{INC (R0)}\] 
\[\text{MOV \#FILE2,R0}\] 
\[\text{NEW POINTER}\] 
\[\text{LOOKUP}\] 
\[\text{FALL ARGS ARE FILLED IN}\] 
\[\text{BCS 25}\] 
\[\text{NOW RELEASE USR}\] 
\[\text{UNLOCK}\] 
\[\text{EXIT}\]

\[\text{LIST1}: \text{BLKW 3}\] 
\[\text{SPACE FOR ARGUMENTS}\] 
\[\text{FILE1}: \text{RAD50 /DK /}\] 
\[\text{FILE2}: \text{RAD50 /FILE1 MAC/}\] 
\[\text{TOP1}: \text{WORD 0}\] 
\[\text{LMSG1}: \text{ABCIZ /LOOKUP ERROR/}\] 
\[\text{EVEN}\]

\[\text{END START}\]

In the above example, .SETTOP tries to obtain as much memory as it can. Most likely this will, in a background job, make the USR non-resident (i.e., unless a SET USR NOSWAP command is done at the keyboard). Thus, if the USR were non-resident, swapping must take place for each .LOOKUP given. Using the .LOCK, the USR is brought into memory and remains there until the .UNLOCK is given.

The second .LOOKUP makes use of the fact that the arguments have already been set up at LIST. Thus, it is possible to increment the channel number, put in a new file pointer and then give a simple .LOOKUP, which does not cause any arguments to be moved into LIST.

\[\text{.LOOKUP}\]

9.4.21 .LOOKUP

The .LOOKUP request associates a specified channel with a device and/or file, for the purpose of performing I/O operations. The channel used is then "busy" until one of the following requests is executed:

\[\text{.CLOSE}\]
\[\text{.SAVESTATUS}\]
\[\text{.RESET}\]
\[\text{.RESET}\]
\[\text{.PURGE}\]
\[\text{.CSIGEN} \quad \text{(if channel is in range 0-10 octal)}\]

Note that if the program is overlaid, channel 15 is used by the overlay handler and should not be modified.

January 1976

9-58
Programmed Requests

Macro Call: .LOOKUP .area, .chan, .blk, .count

where: .count is an argument which can optionally be used for the cassette and magtape handlers. Refer to Appendix H for details of this parameter. If .count is blank, a value of zero is assumed.

Request Format:

\[ R0 \rightarrow .area: 1 .chan .blk .count \]

If the first word of the file name in .blk is zero and the device is a file-structured device, absolute block 0 of the device is designated as the beginning of the "file". This technique allows I/O to any physical block on the device. If a file name is specified for a device which is not file-structured (i.e. PR:FILE.EXT), the name is ignored.

The handler for the selected device must be in memory for a .LOOKUP. On return from the .LOOKUP, R0 contains the length (number of blocks) of the file just looked up. If the length returned is 0, a nonfile-structured .LOOKUP was done to the device.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Channel already open.</td>
</tr>
<tr>
<td>1</td>
<td>File indicated was not found on the device.</td>
</tr>
</tbody>
</table>

Example:

In the following example, the file "DATA.001" on device DT3 is opened for input on channel 7.

```
.MCALL .V2,...,REGDEF,.FETCH,.LOOKUP,.PRINT,.EXIT
..V2,...
..REGDEF

START:
ERRWD=52

.FETCH #SPACE,#DT3N .GET DEVICE HANDLER
.BCS FERR .DT3 IS NOT AVAILABLE
.LOOKUP #LIST,#7,#DT3N .LOOKUP THE FILE
.BCC LDONE .FILE WAS FOUND
.TSTB #ERRWD .ERROR, WHAT'S WRONG?
.BNE NFD .FILE NOT FOUND
.PRINT #CAMSG .PRINT 'CHANNEL ACTIVE'
.EXIT

NFD1 .PRINT #NFMSG .FILE NOT FOUND
.EXIT

CAMSG1 .ASCIZ 'CHANNEL ACTIVE'
```

9-59 January 1976
Programmed Requests

NFMSGL :ASCIZ /FILE NOT FOUND/ ERROR MESSAGES
DTHMSG :ASCIZ /DT3 NOT AVAILABLE/
,EXIT
FERRI :PRINT #DTHSG
,EXIT
LOONE1 :EXIT

PROGRAM CAN NOW ISSUE READS AND WRITES TO FILE \DATA\001 VIA \CHANNEL 7

LISTI ,8LKW 5
DT3NI ,RAD50 "DT3"
,DEVICE
,RAD50 "DAT"
,FILENAME
,RAD50 "A"
,FILENAME
,RAD50 "001"
,EXTENSION

MSPACEI ,*,+400 RESERVED SPACE FOR DT \HANLER

,END START

9.4.22 .MRKT

The .MRKT request schedules a completion routine to be entered after a specified time interval (clock ticks past midnight) has elapsed.

Macro Call: .MRKT .area, .time, .crtn, .id

where: .time is the pointer to the two words containing the time interval (high-order first; low-order second).

.id is a number assigned by the user to identify the particular request to the completion routine and to any cancel mark time requests. The number must not be within the range of codes from 177400-177777; these are reserved for system use. The number need not be unique (i.e., several .MRKT requests may specify the same .id.) On entry to the completion routine, the .id number is in R0.

Request Format:

R0 \rightarrow .area: 22 0
otime
,crtn
,.id

January 1976 9-60
Programmed Requests

.MRKI requests require a queue element taken from the same list as the I/O queue elements. The element is in use until either the completion routine is entered or a cancel mark time request is issued. The user should allocate enough queue elements to handle at least as many mark time requests as he expects to have pending simultaneously.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No queue element was available.</td>
</tr>
</tbody>
</table>

Example:

In this example, a mark time is set up to time out an I/O transfer. If the mark time expires before the transfer is done, a message is printed. If the I/O transfer completes before the mark time, the mark time is cancelled. (Note that the example assumes the I/O channel is already open.)

```
.MCALL .V2,...,REGDEF,READ,WAIT,MRKT,CMKT
.MCALL .V2,...,GET,PRINT,EXIT,LOOKUP
.REGDEF

.STI
.LOOKUP #AREA,#0,#FILE
.FILE NOT FOUND
.BCS LKERR
.MOV #AREA,(SP)
.SEMT LIST TO STACK
.GET #QUEUE,#5
.ALLOCATE 5 MORE ELEMENTS
.MRK (SP),#INTRVL,#MRRT,#1
.SET TIMER GOING
.BCS NOMRKI
.FAILED
.BCS RDLST
.START I/O TRANSFER
.BCS RDER
.WAIT #0
.I AND WAIT A WHILE
.BCS CMKT (BP),#1
.I SEE IF MARK TIME IS DONE
.BCS NOTDUN
.FAILED, THAT MEANS THAT
.I THE MARK TIME ALREADY
.I EXPIRED.

.EXIT

.MRTN1
.CMK (BP),#1
.OK, KILL THE TIMER,
.PRINT #FAIL
.DON'T WORRY ABOUT AN
.RTS PC
.LKERR1
.PRINT #LM
.EXIT
.RDERR1
.PRINT #RDMG
.EXIT
.NOTDUN1
.PRINT #FAIL
.EXIT
.NOMRK1
.PRINT #NOG
.EXIT
.NOQI
.ASCII /NO QUEUE ELEMENTS AVAILABLE/
.FAII
.ASCII /MARK TIME COMPLETED BEFORE TRANSFER/
.LM1
.ASCII /LOOKUP ERROR/
.RDMG1
.ASCII /READ ERROR/
.EVEN
.INTRVLI
.WORD 0,13
.FALLOW 13 CLOCK
.STICKS FOR TRANSFER.
```

9-61
Programmed Requests

| QUEUE I | .BLKW | 5 x 7 | AREA FOR QUEUE ELEMENTS |
| AREA I | .BLKW | 5    | I A FEW WORDS FOR EMT LIST |
| FILE I | .RADS0 | /DK FILE TST/ |
| ROLST I | .BYTE | 0    | I CHANNEL 0 |
|         | .BYTE | 10   | I A READ |
| BLOCK I | .WORD | 0    | I BLOCK # |
|         | .WORD | BUFF | I BUFFER |
|         | .WORD | 256  | I 1 BLOCK |
|         | .WORD | 1    |
| BUFF I | .BLKW | 256  |
| END     |      | 8T   |

```
.MWAIT
```

9.4.23 .MWAIT

This request is similar to the .WAIT request. .MWAIT, however, suspends execution until all messages sent by the other job have been transmitted or received. It provides a means for ensuring that a required message has been processed. It should be used primarily in conjunction with the .RCVD or .SDAT modes of message handling, where no action is taken when a message is completed.

Macro Call: .MWAIT

Errors:

None.

Example:

This program requests a message, does some intermediate processing, and then waits until the message is actually sent.

```
;MCALL ,V2,,REGDEF,,MWAIT,,RCVD,,EXIT,,PRINT
;V2:
;REGDEF

WORDS=255,
START:
     ;RCVD  #AREA,#RBUFF,#WORDS ;GET MESSAGE,
     ;INTERMEDIATE PROCESS

     MOV  #RBUFF+2,R5
     ;MAKE SURE WE HAVE IT,
     CMPB  (R5)*,#'A'
     ;FIRST CHARACTER AN A?
     BNE  BADMSG
     INO, INVALID MESSAGE

     ,EXIT
     BADMSG1 ,PRINT #MSG
     ,EXIT
```

9-62
Programmed Requests

M861  :ASCIZ /BAD MESSAGE/
AREA1  :BLKW 10
RBUFF1  :BLKW 256.
          :EVEN
          :END   START

9.4.24  .PRINT

The .PRINT request causes output to be printed at the console terminal. When a foreground job is running and a change occurs in the job producing output, a B> or F> appears. Any text following the message has been printed by the job indicated (foreground or background) until another B> or F> is printed. The string to be printed may be terminated with either a null (0) byte or a 200 byte. If the null (ASCIZ) format is used, the output is automatically followed by a <CR><LF>. If a 200 byte terminates the string, no <CR><LF> is generated.

Macro Call: .PRINT .addr

where:  .addr is the address of the string to be printed.

Control returns to the user program after all characters have been placed in the output buffer.

The foreground job issues a message immediately using .PRINT no matter what the state of the background job. Thus, for urgent messages, .PRINT should be used (rather than .TTYIN or .TTYOUT).

Errors:

None.

Example:

```
:MCALL ,.V2,,REGDEF,.PRINT,.EXIT
   ,.V2
   ,REGDEF

START:
   .PRINT #32
   .PRINT #51
   .EXIT

S11  :ASCIZ /THIS WILL HAVE CR=LF FOLLOWING/
S21  :ASCIZ /THIS WILL NOT HAVE CR=LF/
   .BYTE 200
   .EVEN
   .END   START
```
Programmed Requests

.PROTECT

9.4.25 .PROTECT

The .PROTECT request is used by a job to obtain exclusive control of a vector (two words) in the region 0-476. If it is successful, it indicates that the locations are not currently in use by another job or by the monitor, in which case the job may place an interrupt address and priority into the protected locations and begin using the associated device.

Macro Call: .PROTECT .area, .addr

where: .addr is the address of the word pair to be protected. .addr must be a multiple of four, and must be less than 476 (octal). The two words at .addr and .addr+2 will be protected.

Request Format:

R0 → .area: 31 0
    .addr

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Protect failure; locations already in use.</td>
</tr>
<tr>
<td>1</td>
<td>Address greater than 476 or not a multiple of 4.</td>
</tr>
</tbody>
</table>

Example:

This example shows the use of .PROTECT to gain control of the UDC11 vectors.

```
.MCALL ,V2,...,REGDEF,.PROTECT,.PRINT,.EXIT
...V2...

REGDEF

STI MOVC #AREA, -(SP)
MOVC #234,R5
.PROTECT (SP),R5
/PROTECT 234,236
BCS ERR  /YOU CAN'T
MOV #UDCINT,(R5)+  /INITIALIZE THE VECTORS:
MOV #48,(R5)      /AT LEVEL 7

,EXIT ERR1 ,PRINT #NOVEC
,EXIT
AREAI ,BLKW 5
NOVECI ,ASCIZ /VECTORS ALREADY IN USE/
```

9-64
9.4.26 .PURGE

The .PURGE request is used to de-activate a channel without performing a .HRESET, .SRESET, .SAVESTATUS, or .CLOSE request. It merely frees a channel without taking any other action. If a tentative file has been ENTERed on the channel, it will be discarded. Purging an inactive channel acts as a no-op.

Macro Call: .PURGE .chan

Errors:
None.

Example:

The following code is used to make certain that channels 0-7 are free:

```
,MCALL ,,V2,,REGDEF,,PURGE,,EXIT
,,V2,,
,REGDEF
START:
CLR R1
10I .PURGE R1
INC R1
CMP R1,#8
BLO 10
,EXIT
,END START
```

9.4.27 .QSET

All RT-11 I/O transfers are done through a centralized queue management system. If I/O traffic is very heavy and not enough queue elements are available, the program issuing the I/O requests may be
Programmed Requests

suspended until a queue element becomes available. In a F/B system, the other job runs while the first program waits for the element.

The .QSET request is used to make the RT-11 I/O queue larger (i.e., add available entries to the queue). A general rule to follow is that each program should contain one more queue element than the total number of I/O requests which will be active simultaneously. Timing requests such as .TWAIT and .MRKT also cause elements to be used and must be considered when allocating queue elements for a program. Note that if synchronous I/O is done (i.e., .READW/.WRITW, etc.) and no timing requests are done, no additional queue elements need be allocated.

Macro Call: .QSET .addr, .q leng

where: .addr is the address at which the new elements are to start.
       .q leng is the number of entries to be added. Each queue entry is seven words long; hence the space set aside for the queue should be .q leng * 7 words.

Each time .QSET is called, a contiguous area of memory is divided into seven-word segments and is added to the queue for that job. .QSET may be called as many times as required. The queue set up by multiple .QSET requests is a linked list. Thus, .QSET need not be called with strictly contiguous arguments. The space used for the new elements is allocated from the user's program space. Thus, care must be taken so that the program in no way alters the elements once they are set up. The .SRESET and .HRESET requests discard all user-defined queue elements; therefore any .QSETs must be reissued.

Care should also be taken to allocate enough memory for the queue. The elements in the queue are altered by the monitor; if enough space is not allocated, destructive references will occur in an unexpected area of memory.

Errors:

None.

Example:

```
.MCALL .V2,.REGDEF,.QSET,.EXIT
..V2...
..REGDEF

START
.QSET #O1,#5 /ADD 5 ELEMENTS TO THE QUEUE
.QSET #O3,#3 /STARTING AT Q1
..EXIT
.O11 .BLKW 7*5
.O31 .BLKW 7*3

.END  START
```

Note that Q1 and Q3 need not have been contiguous.
Programmed Requests

9.4.28  .RCTRLO

The .RCTRLO request ensures that the console terminal is able to print. Since CTRL O (^O) struck while output is directed to the console terminal inhibits the output from printing until either another ^O is struck or until the program resets the ^O switch, a program that has a message which must appear at the console can override ^O struck at the keyboard.

Macro Call:  .RCTRLO

Errors:

None.

Example:

In this example, the user program first calls the CSI in general mode, then processes the command. When finished, it returns to the CSI for another command line. To make certain that the prompting "*" typed by the CSI is not inhibited by a CTRL O in effect from the last operation, terminal output is re-enabled via a .RCTRLO command prior to the CSI call.

```
.MCALL  .V2, .REGDEF, .RCTRLO, .CSIEN, .EXIT
.V2, .REGDEF
START:  .RCTRLO
        ;MAKE SURE TT OUTPUT IS ENABLED
        .CSIEN  #DSPACE, #DEXT, #0  ;CALL CSI=IT WILL TYPE
                             ;"

        ;PROCESS COMMAND

        JMP  START
        ;GET NEXT COMMAND

        DEXTI  0
        ;NO DEFAULT EXTENSIONS
        DEXTI  0
        DSPACEI  *+400  ;HANDLER SPACE

.END  START
```

9-67
Programmed Requests

**.RCVD/.RCVDC/.RCVDW**

9.4.29  .RCVD/.RCVDC/.RCVDW (P/B Only)

There are three forms of the receive data request; these are used in conjunction with the .SDAT (Send Data) requests to allow a general data/message transfer system. .RCVD requests can be thought of as .READ requests, where data transfer is not from a peripheral device but from the other job in the system. Additional queue elements should be allocated for buffered I/O operations in .RCVD and .RCVDC requests (see .QSET).

**.RCVD**

This request is used to receive data and continue execution. The request is posted and the issuing job continues execution. At some point when the job needs to have the transmitted message, an .MWAIT should be executed. This causes the job to be suspended until the message has been received.

Macro Call:  .RCVD .area, .buff, .wcnt

where:  .buff is the address of the buffer to which the message is to be sent.

.wcnt is the number of words to be transferred.

Request Format:

```
R0  \rightarrow .area: 26 0
    (unused)
    .buff
    .wcnt
    1
```

Word 0 (the first word) of the message buffer will contain the number of words transmitted whenever the .RCVD is complete. Thus, the space allocated for the message should always be at least one word larger than the actual message size expected.

The word count is a variable number, and as such, the .SDAT/.RCVD combination can be used to transmit a few words or entire buffers. The .RCVD operation is only complete when a .SDAT is issued from the other job.

Programs using .RCVD/.SDAT must be carefully designed to either always transmit/receive data in a fixed format or have the capability of handling variable formats. The messages are all processed in FIFO (first in-first out) order. Thus, the receiver must be certain it is receiving the message it actually wants.

January 1976  9-68
Programmed Requests

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No other job exists in the system.</td>
</tr>
</tbody>
</table>

Example:

An example follows the .RCVDW section.

.RCVDC

The .RCVDC request receives data and enters a completion routine when the message is received. The .RCVDC request is posted and program execution stays with the issuing job. When the other job sends a message, the completion routine specified will be entered.

Macro Call: .RCVDC .area, .buff, .wcnt, .crtn

where: .buff is the address of the buffer to which the message is to be sent.

.wcnt is the number of words to be transmitted.

.crtn is the completion routine to be entered (see Section 9.2.8).

As in the others, word 0 of the buffer contains the number of words transmitted when the transfer is complete.

Request Format:

<table>
<thead>
<tr>
<th>R0</th>
<th>area: 26 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(unused)</td>
</tr>
<tr>
<td>.buff</td>
<td></td>
</tr>
<tr>
<td>.wcnt</td>
<td></td>
</tr>
<tr>
<td>.crtn</td>
<td></td>
</tr>
</tbody>
</table>

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No other job exists in the system.</td>
</tr>
</tbody>
</table>

Example:

An example follows the .RCVDW section.

.RCVDW

.RCVDW is used to receive data and wait. A message request is posted and the job issuing the request is suspended until the other job sends a message to the issuing job. When the issuing job runs again, the message has been received, and word 0 of the buffer indicates the number of words which were transmitted.
Programmed Requests

Macro Call: .RCVDW .area, .buff, .wcnt

where: .buff is the address of the buffer to which the message is to be sent.

.wcnt is the number of words to be transmitted.

Request Format:

\[
\begin{array}{c|c}
R0 & .area: \hline
& 26 \ 0 \\
& \text{(unused)} \\
& .buff \\
& .wcnt \\
& 0 \\
\end{array}
\]

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No other job exists in the system.</td>
</tr>
</tbody>
</table>

Example:

In this example, the running job receives a message from the second job and interprets it as the device and filename of a file to be opened and used. In this case, the message was in RADSO format, and the receiving program did not use the transmitted length for any purpose.

\[
\begin{align*}
\text{START:} \\
\text{MOV} & \text{ #AREA, RS} \\
\text{.RCVDW} & \text{ RS, #FILE, #4} \\
\text{BGS} & \text{ MERR} \\
\text{.PURGE} & \text{ #0} \\
\text{.LOOKUP} & \text{ RS, #0, #FILE+2} \\
\text{BGS} & \text{ LKERR} \\
\text{.EXIT} \\
\text{AREA:} & \text{BLKW} 10 \\
\text{FILE:} & \text{BLKW} 1 \\
\text{MERR:} & \text{PRINT #MMSG} \\
\text{LKERR:} & \text{PRINT #LKMSG} \\
\text{MMGS:} & \text{ASCIZ /MESSAGE ERROR/} \\
\text{LKMSG:} & \text{ASCIZ /LOOKUP ERROR/} \\
\text{END} & \text{START}
\end{align*}
\]

The issuing job is suspended until the indicated data is transmitted. Either of the other modes could have also been used to receive the message.
Programmed Requests

9.4.30 .READ/.READC/.READW

RT-ll provides three modes of I/O: .READ/.WRITE, .READC/.WRITC, and .READW/.WRITW. Section 9.4.47 explains the output operations. The input operations are described next.

Note that in the case of .READ and .READC, additional queue elements should be allocated for buffered I/O operations (see .QSET).

.READ

The .READ request transfers a specified number of words from the specified channel to memory. Control returns to the user program immediately after the .READ is initiated. No special action is taken when the transfer is completed.

Macro Call: .READ .area, .chan, .buff, .wcnt, .blk

where: .buff is the address of the buffer to receive the data read.
 .wcnt is the number of words to be read.
 .blk is the block number to be read relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again. If .blk=0, TT: gives ^ prompt and LP: gives form feed. (This is true for all .READ and .WRITE requests.)

Request Format:

R0 $ .area: 10 | .chan
 .blk
 .buff
 .wcnt
 l

When the user program needs to access the data read on the specified channel, a .WAIT request should be issued. This ensures that the data has been read completely. If an error occurred during the transfer, the .WAIT request indicates the error.
Programmed Requests

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Attempt to read past end-of-file</td>
</tr>
<tr>
<td>1</td>
<td>Hard error occurred on channel</td>
</tr>
<tr>
<td>2</td>
<td>Channel is not open</td>
</tr>
</tbody>
</table>

Example:
Refer to the .WRITE/.WRITC/.WRITW examples.

.READC

The .READC request transfers a specified number of words from the indicated channel to memory. Control returns to the user program immediately after the .READC is initiated. Execution of the user program continues until the .READC is complete, then control passes to the routine specified in the request. When an RTS PC is executed in the completion routine, control returns to the user program.

Macro Call: .READC .area, .chan, .buff, .wcnt, .crtn, .blk

where: .buff is the address of the buffer to receive the data read.

.wcnt is the number of words to be read.

.crtn is the address of the user's completion routine (refer to Section 9.2.8).

.blk is the block number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.

Request Format:

R0 $\Rightarrow$ .area: 10 .chan
                .blk
                .buff
                .wcnt

address of completion routine

When entering a .READC completion function the following are true:

1. R0 contains the channel status word for the operation. If bit 0 of R0 is set, a hardware error occurred during the transfer. The data may not be reliable.

2. R1 contains the octal channel number of the operation. This is useful when the same completion function is to be used for several different transfers.

January 1976 9-72
Programmed Requests

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Attempt to read past end-of-file</td>
</tr>
<tr>
<td>1</td>
<td>Hard error occurred on channel</td>
</tr>
<tr>
<td>2</td>
<td>Channel is not open</td>
</tr>
</tbody>
</table>

Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

.READW

The .READW request transfers a specified number of words from the indicated channel to memory. Control returns to the user program when the .READW is complete or if an error is detected.

Macro Call: .READW .area, .chan, .buff, .wcnt, .blk

where: .buff is the address of the buffer to receive the data read.

.wcnt is the number of words to be read. The number must be positive.

.blk is the block number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.

Request Format:

R0 ⇒ .area: 10 .chan
            .blk
            .buff
            .wcnt
            0

On return from this call, the C bit set indicates an error has occurred. If no error occurred, the data is in memory at the specified address. In an F/B system, the other job can be run while the issuing job is waiting for the I/O to complete.

Note:

Upon return from any READ programmed request, R0 will contain no information if the read is from a sequential-access device (for example, magtape). If the read is from a random-access device (disk, DECTape) R0 will contain the actual number of words that will be read (.READ or .READC) or have been read (.READW). This will be less than the requested word count if an attempt is made to read past the end-of-file, but a partial transfer is possible. In the case of a partial transfer, the C bit is set and error code 0 is returned. Therefore, a
Programmed Requests

Program should always use the returned would count as the number of words available. For example, suppose a file is 5 blocks long (i.e., it has block numbers 0 to 4) and a request is issued to read 512 words, starting at block 4. The request is shortened to 256 words; no error is indicated. Also note that since the request will be shortened to an exact number of blocks, a request for 256 words will either succeed or fail, but cannot be shortened.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Attempt to read past end-of-file</td>
</tr>
<tr>
<td>1</td>
<td>Hard error occurred on channel</td>
</tr>
<tr>
<td>2</td>
<td>Channel is not open</td>
</tr>
</tbody>
</table>

Example:

Refer to the .WRITE/.WRITC/.WRITW examples.

.RELEAS

9.4.31 .RELEAS

The .RELEAS request removes the handler for the specified device from memory. The .RELEAS is ignored if the handler is:

1. Part of RAMON (i.e., the system device),
2. Not currently resident, or
3. Resident because of a LOAD command to the Keyboard Monitor,

.RELEAS from the foreground is always ignored, since the foreground can only use handlers which have been LOADED.

Macro Call: .RELEAS .devname

where: .devname is the pointer to the .RAD50 device name.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Handler name was illegal.</td>
</tr>
</tbody>
</table>
Programmed Requests

Example:

In the following example, the DECtape handler (DT) is loaded into memory, used, then released. If the system device is DECtape, the handler is already resident, and .FETCH will return HSPACE in R0.

```
.MCALL .V2... ,REDEFINE, FETCH, RELEASE, EXIT
.V2...
.REDEFINE

START: .FETCH
#HSPACE,#DTNAME /LOAD DT HANDLER
BCB FERR NOT AVAILABLE

; USE HANDLER
.RELEASE #DTNAME MARK DT NO LONGER IN
BR START MEMORY
FERR MALT
DTNAME: RAD50 /DT /
HSPACE:

.END START
```

9.4.32 .RENAME

The .RENAME request causes an immediate change of name of the file specified. An error occurs if the channel specified is already open.

Macro Call: .RENAME .area, .chan, .dblk

Request Format:

```
R0 => .area: 4 .chan .dblk
```

The .dblk argument consists of two consecutive .RAD50 device and file specifications. For example:

```
.RENAME #AREA,#7,#DBLK ;USE CHANNEL 7
BCS RNMERR ;NOT FOUND

; DBLK:
.Rad50 /DT3/
.Rad50 /OLDFIL/
.Rad50 /MAC/
.Rad50 /DT3/
.Rad50 /NEWFIL/
.Rad50 /MAC/
```

9-75 January 1976
Programmed Requests

The first string represents the file to be renamed and the device it is found on. The second represents the new file name. If a file with the same name as the new file name specified already exists on the indicated device, it is deleted. The second occurrence of the device name DT3 is necessary for proper operation, and should not be omitted. The specified channel is left inactive when the .RENAMEs is complete.

.RENAME requires that the handler to be used be resident at the time the .RENAME request is made. If it is not, a monitor error occurs. Note that .RENAME is legal only on files which are on disk or DECTape. (.RENAMEs to other devices are ignored.)

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Channel open</td>
</tr>
<tr>
<td>1</td>
<td>File not found</td>
</tr>
</tbody>
</table>

Example:

In the following example, the file DATA.TMP on DT0 is renamed to DATA.001:

```
.MCALL .V2, ,REGDEF, .FETCH, .PRINT
.MCALL .EXIT, .RENAME
.V2
.REGDEF

START: .FETCH #HSPACE, #NAMBLK ,GET HANDLER
?BCS FERR #SOME ERROR
?.RENAME #AREA, #0, #NAMBLK ,IDO THE RENAME
?BCS RNMERR #ERROR
.EXIT
.PRINT #FMSG
.EXIT
.PRINT #RNMSG
.EXIT
.AREA RNMSG 5 /ROOM FOR ARGS.
.NAMBLK1 .RAD50 /DT0DATA TMP/ /OLD NAME
.RAD50 /DT0DATA 001/ /NEW NAME
.FMSG1 .ASCIZ /FETCH?/ #ERROR MESSAGES
.RNMSG1 .ASCIZ /RENAME?/ #ERROR MESSAGES
HSPACE*

.END START
```
Programmed Requests

9.4.33 .REOPEN

The .REOPEN request reassociates the specified channel with a file on which a .SAVESTATUS was performed. The .SAVESTATUS/.REOPEN combination is useful when a large number of files must be operated on at one time. As many files as are needed can be opened with .LOOKUP, and their status preserved with .SAVESTATUS. When data is required from a file, a .REOPEN enables the program to read from the file. The .REOPEN need not be done on the same channel as the original .LOOKUP and .SAVESTATUS.

Macro Call: .REOPEN .area, .chan, .cblk

where: .cblk is the address of the five-word block where the channel status information was stored.

Request Format:

R0 ⇒ .area: 6 .chan
 .cblk

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified channel is in use. The .REOPEN has not been done.</td>
</tr>
</tbody>
</table>

Example:

Refer to the example following the description of .SAVESTATUS.

9.4.34 .SAVESTATUS

The .SAVESTATUS request stores five words of channel status information into a user-specified area of memory. These words contain
Programmed Requests

all the information RT-ll requires to completely define a file. When
a .SAVSTATUS is done, the data words are placed in memory, and the
specified channel is again available for use. When the saved channel
data is required, the .REOPEN request is used.

.SAVSTATUS can only be used if a file has been opened with .LOOKUP.
If .ENTER was used, .SAVSTATUS is illegal and returns an error. Note
that .SAVSTATUS is legal only on files which are not on magtape or
cassette.

Macro Call: .SAVSTATUS .area, .chan, .cblk

where: .cblk is the address of the user memory block (5
words) where the channel status information
is to be stored.

Request Format:

R0 ➔ .area: 5 .chan
            .cblk

The five words stored are the five words normally contained in the
channel area, as follows:

<table>
<thead>
<tr>
<th>Word #</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Channel status word. The contents of the bits of this word are:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 - a hardware error occurred on this channel.</td>
</tr>
<tr>
<td>1-5</td>
<td>Index into monitor tables. This describes the physical device with which the channel is associated.</td>
</tr>
<tr>
<td>6</td>
<td>1 - a .RENAME operation is in progress on the channel.</td>
</tr>
<tr>
<td>7</td>
<td>1 - a .CLOSE operation must rewrite the directory (i.e., set when a .ENTER is done).</td>
</tr>
<tr>
<td>8-12</td>
<td>Contains the directory segment number (1-37(8)) in which the current open file can be found.</td>
</tr>
<tr>
<td>13</td>
<td>1 - An end-of-file was found on the channel.</td>
</tr>
<tr>
<td>14</td>
<td>Unused.</td>
</tr>
<tr>
<td>15</td>
<td>1 - This channel is currently in use (i.e., a file is open on this channel).</td>
</tr>
</tbody>
</table>

| 2      | Starting block number of the file. Zero for sequential-access devices. |
| 3      | Length of file (in 256-word blocks). |
| 4      | Data length of file; currently unused. |
| 5      | Even Byte: I/O count. Count of how many I/O requests have been made on this channel. Odd Byte: |
Programmed Requests

unit number of the device associated with the channel (between 0 - 7).

While the .SAVESTATUS/.REOPEN combination is very useful, care must be observed when using it. In particular, the following cases should be avoided:

1. If a .SAVESTATUS is performed and the same file is then deleted before it is reopened, it becomes available as an empty space which could be used by the .ENTER command. If this sequence occurs, the contents of the file supposedly saved will change.

2. Although the device handler for the required peripheral need not be in memory for execution of a .REOPEN, if the handler is not in memory when a .READ or .WRITE is executed, a fatal error is generated.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The file was opened via .ENTER, or is a magtape or cassette file, and a .SAVESTATUS is illegal.</td>
</tr>
</tbody>
</table>

Example:

One of the more common uses of .SAVESTATUS and .REOPEN is to consolidate all directory access motion and code at one place in the program. All files necessary are opened and their status saved, then they are re-opened one at a time as needed. USR swapping can be minimized by locking in the USR, doing .LOOKUPS as needed, using .SAVESTATUS to save the file data, and then .UNLOCKING the USR.

In the program segment below, three input files are specified in the command string; these are then processed one at a time.

```
MCALL .V2,.REGEF,.CSIGEN,.SAVESTATUS,.REOPEN
MCALL .READ,.EXIT
..V2.
.REGEF
START: MOV #AREA,R5
        .CSIGEN #SPACE,#EXT    IGET INPUT FILES
MOV R8,BUFF    ISAVE POINTER TO FREE CORE

.SAVESTATUS R5,#3,#BLOCK1 ISAVE FIRST INPUT FILE
.SAVESTATUS R5,#4,#BLOCK2 ISAVE SECOND FILE
.SAVESTATUS R5,#5,#BLOCK3 ISAVE THIRD FILE

MOV #BLOCK1,R4
PROCESS: .REOPEN R5,#8,R4    IREOPEN FILE ON
          ICHANNEL 0

.READ R5,#0,BUFF,COUNT,BLOCK IPROCESS FILE ON CHANNEL 0

DONE: ADD #12,R4
       CMP R4,#BLOCK3    IPOINT TO NEXT SAVESTATUS BLOCK
       LAST FILE PROCESSED?
```

9-79
Programmed Requests

```
.BLOG  .PROCESS  .EXIT

.BLOCK1: .WORD 0,0,0,0,0  ;MEMORY BLOCKS FOR
.BLOCK2: .WORD 0,0,0,0,0  ;SAVE STATUS INFORMATION
.BLOCK3: .WORD 0,0,0,0,0
.AREA:  .BLKW 10

.BUFF1: .WORD 0
.BLOCK1: .WORD 0
.COUNT:  .WORD 256,

.DEXT:  .WORD 0,0,0,0
.DSPACE:  .END  .START
```

**.SDAT/.SDATC/.SDATW**

9.4.35  .SDAT/.SDATC/.SDATW

These requests are used in conjunction with the .RCVD/.RCWD/.RCVDC calls to allow message transfers with RT-ll. .SDAT transfers can be considered similar to .WRITE requests in which data transfer is not from a peripheral, but from one job to another. Additional I/O queue elements should be allocated for buffered I/O operations in .SDAT and .SDATC requests (see .QSET).

**.SDAT**

Macro Call: .SDAT .area, .buff, .wcnt

where: .buff is the buffer address of the beginning of the message to be transferred.

.wcint is the number of words to transfer.

**Request Format:**

```
R0 → .area: 25 0
unused .buff .wcint
1
```

**Errors:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No other job exists.</td>
</tr>
</tbody>
</table>

January 1976  9-80
Programmed Requests

Example:

See the example following .SDATW.

.SDATC

Macro Call: .SDATC .area, .buff, .wcnt, .crt

where: .buff is the buffer address of the beginning of the message to be transferred.

.wcnt is the number of words to transfer.

.crt is the address of the completion routine to be entered when the message has been transmitted (refer to Section 9.2.8).

Request Format:

R0 ➔ .area:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No other job exists.</td>
</tr>
</tbody>
</table>

Errors:

Example:

See the example following .SDATW.

.SDATW

Macro Call .SDATW .area, .buff, .wcnt

where: .buff is the buffer address of the beginning of the message to be transferred

.wcnt is the number of words to transfer

Request Format:

R0 ➔ .area:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No other job exists.</td>
</tr>
</tbody>
</table>
Programmed Requests

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No other job exists.</td>
</tr>
</tbody>
</table>

Example:

In this example, the job first sends a message interrogating the other job about the status of an operation, and then looks for an acknowledgement from the job.

```
.MCALL .V2,,.REGDEF,.SDAT,.RCVD,.MWAIT,.PRINT,.EXIT
   .V2,,.REGDEF

START1
   MOV #AREA,R5          ; SETUP EMT BLOCK
   ,SDAT R5,#$BUFF,#MLGTH ; ASK HIM A QUESTION
   BCS NOJOB              ; NO OTHER JOB AROUND!

   ; MISCELLANEOUS PROCESSING
   ,RCVD R5,#$BUFF2,#20,  ; RECEIVE 20 DECIMAL WORDS
   ,MWAIT
   MOV #BUFF2+2,R1        ; POINT TO ACTUAL ANSWER.
   CMPB (R1)*,#'Y        ; IS FIRST WORD Y FOR YES?
   BNE PRNEG              ; INEGATIVE ACKNOWLEDGE
   ,PRINT #POSACK
   ,EXIT

PRNEG1 ,PRINT #NEGACK    ; NEGATIVE ON OUR INQUIRY
   ,EXIT
   ,EXIT

$BUFF1 ,ASCII /IS THE REQUIRED PROCESS GOING?/
   MLGTH=#BUFF
   BUFFER WORD 0          ; ACTUAL LENGTH IS HERE
   ,BLKW 20               ; SPACE FOR 20 WORDS

NOJOB1 ,PRINT #NJMSG
   ,EXIT
NEGACK1 ,ASCII /NEGATIVE ACKNOWLEDGE/
POSACK1 ,ASCII /POSITIVE ACKNOWLEDGE/
NJMSG1 ,ASCII /NO JOB/
   ,EVEN
AREA 1 ,BLKW 10,

,END START
```

.SETTOP

9.4.36 .SETTOP

The .SETTOP request allows the user program to request that a new address be specified as a program's upper limit. The monitor
Programmed Requests
determines whether this address is legal and whether or not a memory
swap is necessary when the USR is required. For instance, if the
program specified an upper limit below the start address of USR, no
swapping is necessary, as the USR is not overlaid. If .SETTOP from
the background specifies a high limit greater than the address of the
USR and a SET USR NOSWAP command has not been given, a memory swap is
required. Section 9.2.5 gives details on determining where the USR is
in memory and how to optimize the .SETTOP.

On return from .SETTOP, both R0 and the word at location 50 (octal)
contain the highest memory address allocated for use. If the job
requested an address higher than the highest address which is legal
for the requesting job, it is adjusted down to that address.

Macro Call: .SETTOP .addr

where: .addr is the address of the word immediately
following the free area desired.

Notes:

1. A program should never do a .SETTOP and assume that its new
   upper limit is the address it requested. It must always examine the returned contents of R0 or location 50 to
determine its actual high address.

2. In Version 1 of RT-11, .SETTOP did not return the high
   address in R0, but only in word 50.

3. It is imperative that the value returned in R0 or location 50
   be used as the absolute upper limit. If this value is ever
   exceeded, vital parts of the monitor may be destroyed, and
   the system integrity will be violated.

Errors:
None.

Example:
Following is an example in two parts. The first indicates how a
small background job (i.e., one with free space between itself and the
USR) can be assured of reserving space up to but not including the USR.
This in effect gives the job all the space it can without causing the
USR to become non-resident.
The second part indicates how to always reserve the maximum amount of
space by making the USR non-resident.

I)

   .MCALL .V2...,REGDEF,.SETTOP,.EXIT
   .V2...
   .REGDEF

START
RMON=54
USR=266

;POINTER TO START OF RESIDENT
;OFFSET FROM RESIDENT TO POINTER
;WHERE USR WILL START.

9-83         July 1975
Programmed Requests

```
MOV #RMON,R1       ;START OF RMON TO R1
MOV USR(R1),R0     ;POINT TO LOWEST USR WORD
TST -(R0)          ;POINT TO HIGHEST WORD NOT IN USR
SETTOP             ;AND ASK FOR IT
MOV R0,MI CORE     ;R0 CONTAINS THE HIGH ADDRESS
EXIT                ;THAT WAS RETURNED.
```

II)

```
SETTOP #=2
```

```
MOV R0,MI CORE     ;IF WE ASK FOR A VALUE GREATER
EXIT                ;THAN START OF RESIDENT, WE
              ;WILL GET BACK THE ABSOLUTELY
              ;HIGHEST USABLE ADDRESS,
              ;THAT IS OUR LIMIT NOW
```

If a SET USR NOSWAP command is executed, the USR cannot be made
non-resident. In this case, in both I & II above, R0 would return a
value just below the USR.

Caution should be used concerning technique I, above. If the
background program is so large that the USR is normally positioned
over part of it, the high limit value returned by the .SETTOP may
actually be lower than the original limit. The USR is then resident,
with a portion of the user program destroyed. The example in Section
9.2.5 shows how to include checks that will avoid this situation.

```
.SFPA
```

9.4.37 .SFPA

.SFPA allows users with floating point hardware (FP on 11/45 and FIS
on 11/40) to set trap addresses to be entered when a floating point
exception occurs. If no user trap address is specified and a floating
point (FP) exception occurs, a ?M-FP TRAP occurs, and the job is
aborted.

Macro Call: .SFPA .area, .addr

where: .addr is the address of the routine to be entered
when an exception occurs.

Request Format:

```
R0 ⇒ .area: 30 0
            .addr
```

Notes:

1. If the address argument is 0, user floating point routines
   are disabled and the fatal ?M-FP TRAP error is produced.

2. In the F/B environment, an address value of 1 indicates that
   the FP registers should be switched when a context switch
   occurs, but no user traps are enabled. This allows both jobs
   to use the FP unit. An address of 1 to the Single-Job
   Monitor is equivalent to an address of 0.

July 1975 9-84
Programmed Requests

3. When the user routine is activated, it is necessary to re-execute an .SFPA request, as the monitor inhibits user traps when any one is serviced. It does this to inhibit any possible infinite loop being set up by repeated FP exceptions.

4. If the 11/45 FPP is being used, the instruction STST -(SP) is executed by the monitor before entering the user's trap routine. Thus, the trap routine must pop the two status words off the stack before doing an RTI. The program can tell if FPP hardware is available by examining the configuration word in the monitor (see Section 9.2.6).

Errors:
None.

Example:
This example sets up a user FP trap address.

```
.MCALL ..V2..,REGDEF,.SFPA,.EXIT
..V2..
,REGDEF
START:
,SFPA  #AREA,#FPTRAP
,EXIT
FPTRAP:
:
MOV  R0,-(SP)  1RM USED BY .SFPA
,SFPA  #AREA,#FPTRAP
MOV  (SP)+,R0  IRSTORE R0
RTI
.
AREA1  ,BLKW  10
,END  START
```

9.4.38 .SFUN
This request is used principally with cassettes and magtape handlers to do device-dependent functions, such as rewind and backspace, to those devices. (It may also be used with diskette to allow reading and writing of absolute sectors; specific information is in Appendix H.)

Macro Call: .SFUN .area, .chan, .code, .buff, .wcnt, .blk, .crtn

where: .code is the numerical code of the function to be performed
Programmed Requests

.buff is the buffer address; this parameter must be set to zero if no buffer is required.
.crtm is the entry point of a completion routine. If left blank, 0 is automatically inserted.

Request Format:

RO \[.area: 32 .chan
   .blk
   .buff
   .wcnt
   .code 377
   .crtm

All other arguments are the same as those defined for .READ/.WRITE requests (Sections 9.4.30 and 9.4.47). They are only required when doing a .WRITE with extended record gap to MT. If the .crtm argument is left blank, the requested operation will complete before control returns to the user program. .crtm=1 is equivalent to executing a .READ or .WRITE in that the function is initiated and returns immediately to the user program. A .WAIT on the channel ensures that the operation is completed. If .crtm=N, it is taken as a completion routine address to be entered when the operation is complete.

The available functions and their codes are:

<table>
<thead>
<tr>
<th>Function</th>
<th>MT</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward to last file</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>Forward to last block</td>
<td>376</td>
<td></td>
</tr>
<tr>
<td>Forward to next file</td>
<td>375</td>
<td></td>
</tr>
<tr>
<td>Forward to next block</td>
<td>374</td>
<td></td>
</tr>
<tr>
<td>Rewind to load point</td>
<td>373</td>
<td>373</td>
</tr>
<tr>
<td>Write file gap</td>
<td>372</td>
<td></td>
</tr>
<tr>
<td>Write EOF</td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>Forward 1 record</td>
<td>376</td>
<td></td>
</tr>
<tr>
<td>Backspace 1 record</td>
<td>375</td>
<td></td>
</tr>
<tr>
<td>Write with extended file gap</td>
<td>374</td>
<td></td>
</tr>
<tr>
<td>Offline</td>
<td>372</td>
<td></td>
</tr>
</tbody>
</table>

To use the .SPFUN request, the handler must be in memory and a channel associated with a file via a .LOOKUP request.

Refer to Appendix H for details of MT and CT handlers.

Errors:

Errors are detected in the same way as for the .READ/.READC/.READW requests. Refer to Section 9.4.30 for details.

Example:

The following example rewinds a cassette and writes out a 256-word buffer and then a file gap.

```cpp
.MCALL .V2 ,.REGDEF,.FETCH,.LOOKUP,.SPFUN,.WRITE
.MCALL .EXIT,.PRINT,.WAIT,.CLOSE
.V2
.REGDEF

START: .FETCH #HSPC,#CT IGET A HANDLER
```

July 1975 9-86
Programmed Requests

BCS FERR ;FETCH ERROR
.LOOKUP #AREA,#4,#CT ;LOOK IT UP ON CHANNEL 4
BCS LKERR ;LOOKUP ERROR
.SPFUN #AREA,#4,#373,#O ;REWIND SYNCHRONOUSLY
BCS SPERR ;AN ERROR OCCURRED.
MDV #3,R5 ;COUNT
.IBLOCK 0,
.WRIT #AREA,#4,#BUFF,#256,#BLK
BCS WTER #AREA,#4,#372,#0,#1 ;SYNCHRONOUS FILE GAP
.PRINT #DONE
.WAIT #4 ;WAIT FOR DONE
.CLOSE #4 ;CLOSE THE FILE
.EXIT

.AREA1 .BLKW 10
.FERR1 .PRINT #FMSG
.EXIT
.LKERR1 .PRINT #LKMSG
.EXIT
.SPERR1 .PRINT #SPMSG
.EXIT
.WTER1 .PRINT #WTMSG
.EXIT
.DONE1 .ASCII /ALL DONE/
.FMSG1 .ASCII /FETCH?/
.LKMSG1 .ASCII /FILE?/
.SPMSG1 .ASCII /SPECIAL FUNCTION ERROR/
.WTMSG1 .ASCII /WRITE ERROR/
.EVEN
.CT1 .RADD /CT /
.WORD 0,0,0
.BUFF1 .BLKW 256
.BLK1 .WORD 0
.MSPE1 .END .START

9.4.39 .SPND/.RSUM (P/B only)

The .SPND/.RSUM requests allow a job to control execution of its mainstream code (that code which is not executing as a result of a completion routine). .SPND suspends the mainstream and allows only completion routines (for I/O and mark time requests) to run. .RSUM from one of the completion routines resumes the mainstream code. These functions enable a program to wait for a particular I/O or mark time request by suspending the mainstream and having the selected event's completion routine issue a .RSUM. This differs from the .WAIT request, which suspends the mainstream until all I/O operations on a specific channel have completed.

.SPND

Macro Call: .SPND

where: R0 ⇒ 10

9-87
Programmed Requests

.RSUM

Macro Call: .RSUM

where: R0 \rightarrow 20

Notes:

1. The monitor maintains a suspension counter for each job. This counter is decremented by .SPND and incremented by .RSUM. A job will actually be suspended only if this counter is negative. Thus, if a .RSUM is issued before a .SPND, the latter request will return immediately.

2. A program must issue an equal number of .SPNDs and .RSUMs.

3. A .RSUM request from the mainstream code increments the suspension counter.

4. A .SPND request from a completion routine decrements the suspension counter, but does not suspend the mainstream. If a completion routine does a .SPND, the mainstream continues until it also issues a .SPND, at which time it is suspended and will require two .RSUMs to proceed.

5. Since a .TWAIT is simulated in the monitor using suspend and resume, a .RSUM issued from a completion routine without a matching .SPND may cause the mainstream to continue past a timed wait before the entire time interval has elapsed.

6. A .SPND or .RSUM, like most other programmed requests, may be issued from a user-written interrupt routine if the .INTEN/.SYNCH sequence is followed. All notes referring to .SPND/.RSUM from a completion routine also apply to this case.

Errors:

None.

Example:

In this example, the program starts a number of read operations and suspends itself until at least two of them are complete.

```
.MCALL ...,V2,,REGDEF,..SPND,..RSUM,..READC,..EXIT,..LOOKUP
.MCALL ...,PRINT,,WAIT
...,V2,,
.REGDEF

START:

...LOOKUP #AREA,#2,#FILE2
BCS 15
...LOOKUP #AREA,#3,#FILE3
BCS 15
...LOOKUP #AREA,#4,#FILE4
BCC 35
```

January 1976 9-88
Programmed Requests

181 .PRINT #2S
182 .EXIT
281 .ASCIZ /LOOKUP ERROR/
381 .EVEN

MOV #2, R8YCTR  ; WAIT FOR 2 COMPLETIONS
MOV AREA, R5
READC R5, #2, #BUF1, COUT1, #CROUTN, BLOK1
BCS ERROR
READC R5, #3, #BUF2, COUT2, #CROUTN, BLOK2
BCS ERROR
READC R5, #4, #BUF3, COUT3, #CROUTN, BLOK3
BCS ERROR
.SPND

.WAIT #2
.WAIT #3
.WAIT #4
.EXIT

CROUTN: ABL R1  ; DOUBLE CHANNEL # FOR INDEXING

INC DONEFL(R1)  ; R1 CHANNEL THAT IS DONE
ROR R0  ; SET A FLAG SAYING 00,
ADC ERRFLG(R1)  ; ANY ERRORS?
DEC R8YCTR  ; IF CARRY SET, SET ERROR FLAG FOR CHANNEL
BNE 180  ; ARE WE THE SECOND TO FINISH?
BNE 190  ; NO
RSUM 1  ; YES, START UP
191 RTS PC

ERROR1 .PRINT #RDMSG
.ERROR1 .EXIT
RDMSG: .ASCIIZ /READ ERROR/

AREA1 .BLKW 10
AREA2 .BLKW 10
R8YCTR1 .WORD 0
COUNT1 .WORD 256
COUNT2 .WORD 256
COUNT3 .WORD 256
BLOK1 .WORD 0
BLOK2 .WORD 0
BLOK3 .WORD 0
FILE2 .RADS /DK TEST2 TMP/
FILE3 .RADS /DK TEST3 TMP/
FILE4 .RADS /DK TEST4 TMP/
DONEFL .WORD 0, 0, 0
ERRFLG .WORD 0, 0, 0
BUF1 .BLKW 256
BUF2 .BLKW 256
BUF3 .BLKW 256
.END START
Programmed Requests

**.SRESET**

9.4.40 .SRESET

The .SRESET (software reset) request performs the following functions:

1. Dismisses any device handlers which were brought into memory via a .FETCH call. Handlers which were loaded via the Keyboard Monitor LOAD command remain resident, as does the system device handler.

2. Purges any currently open files. Files opened for output with .ENTER will never be made permanent.

3. Reverts to using only 16 (decimal) I/O channels. Any channels defined with .CDFN are discarded. A .CDFN must be reissued to open more than 16 (decimal) channels after a .SRESET is performed.

4. Resets the I/O queue to one element. A .QSET must be reissued to allocate extra queue elements.

5. Clears completion queue of any completion routines.

**Macro Call .SRESET**

Errors:

None.

Example:

In the example below, .SRESET is used prior to calling the CSI to ensure that all handlers are removed from memory and the CSI is started with a free handler area.

```
.MCALL .V2,...,REGDEF,,CSIGEN,,SRESET,,EXIT
...V2,,
,REGDEF

STARTI .CSIGEN #DSPACE,#DEXT,#0 GET COMMAND STRING
MOV $R0,BUFFER $R0 POINTS TO FREE MEMORY

DONEI .SRESET IRELEASE HANDLERS, DELETE
OR START IAND REPEAT PROGRAM.
```

9-90
Programmed Requests

\texttt{DEXTI ,WORD 0,0,0,0 \ NO DEFAULT EXTENSIONS}
\texttt{BUFFER: 0 \ START OF HANDLER AREA.}
\texttt{END \ START}

If the .SRESET had not been performed prior to the second call of .CSIGEN, it is possible that the second command string would load a handler over one that the monitor thought was resident from the first command line.

\textbf{.TLOCK}

9.4.41 .TLOCK

.TLOCK is used in an P/B system to attempt to gain ownership of the USR. It is similar to .LOCK in that if successful, the user job returns with the USR in memory. However, if a job attempts to .LOCK the USR while the other job is using it, the requesting job is suspended until the USR is free. With .TLOCK, if the USR is not available, control returns immediately with the C bit set to indicate the .LOCK request failed.

Macro Call: .TLOCK

The .TLOCK request allows the job to continue running, with only one sub-job affected. With a .LOCK request, all sub-jobs would be automatically suspended, and the other job in the system would run.

Request Format:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{R0: area:} & \textbf{70} \\
\hline
\end{tabular}
\end{center}

Errors:

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Code} & \textbf{Explanation} \\
\hline
0 & USR is already in use by another job. \\
\hline
\end{tabular}
\end{center}

Example:

In the following example, the user program needs the USR for a sub-job it is executing. If it fails to get the USR it suspends that sub-job and runs another sub-job. This type of procedure is useful to schedule several sub-jobs within a background or foreground program.

\texttt{.MCALL \,..V2\,..REGDEF\,TLOCK\,LOOKUP\,UNLOCK\,EXIT\,PRINT}
\texttt{\,..V2\,..}
\texttt{\,..REGDEF}
\texttt{START!}

9-91
Programmed Requests

.TRPSET

9.4.42 .TRPSET

.TRPSET allows the user job to intercept traps to 4 and 10 instead of having the job aborted with a ?M-TRAP TO 4 or ?M-TRAP TO 10 message. If .TRPSET is in effect when a trap occurs, the user-specified routine is entered. The sense of the C bit on entry to the routine determines which trap occurred: C bit clear indicates a trap to 4; set indicates a trap to 10. The user routine should exit via an RTI instruction.

Macro Call: .TRPSET .area .addr

where .addr is the address of the user's trap routine. If an address of 0 is specified, the user's trap interception is disabled.

Request Format:

R0 → .area: 3 0 .addr

Notes:

It is necessary to reissue a .TRPSET request whenever a trap occurs and the user routine is entered. The monitor inhibits servicing user traps prior to entering the first user trap routine. Thus, if a trap should occur from within the user's trap routine, a ?M-TRAP message is generated. The last operation the user routine should perform before an RTI is to reissue the .TRPSET request.

9-92
Programmed Requests

Errors:
None.

Example:
The following example sets up a user trap routine and, when the trap occurs, prints an appropriate error message.

```
;MCALL $V2,...,REGDEF,..TRPSET,.EXIT,.PRINT
;V2...
;REGDEF

START:

;TRPSET #AREA,#TRPLOC
MOV #101,R0
TST (R0)*
WORD 67
;EXIT

SET TO PRODUCE A TRAP
THIS WILL TRAP TO 4.
THIS WILL TRAP TO 10.

TRPLOC: MOV R0,=(SP)
BCS 13
;PRINT #TRP4
;TRAP TO 4
BR 25

131 ;PRINT #TRP10
251 ;TRPSET #AREA,#TRPLOC
MOV (SP)+,R0
;RESET TRAP ADDRESS
RTI

AREA1 ;BLKW 10
TRP4 ;ASCII /TRAP TO 4/
TRP10 ;ASCII /TRAP TO 10/
;EVEN

.END START
```

9.4.43 .TTYIN/.TTINR

These requests are used to transfer characters from the console terminal to the user program. The character thus obtained appears right-justified (even byte) in R0.

9-93
Programmed Requests

The expansion of .TTYIN is:

EMT 340  
BCS .-2

while that for .TTINR is:

EMT 340

If no characters or lines are available when an EMT 340 is executed, return is made with the C bit set. The implication of these calls is that .TTYIN causes a tight loop waiting for a character/line to appear, while the user can either wait or continue processing using .TTINR.

Macro Calls: .TTYIN .char

.TTINR

where: .char is the location where the character in R0 is stored. If not specified, the character is left in R0.

If the carry bit is set when execution of the .TTINR request is completed, it indicates that no character was available; the user has not yet typed a valid line. Under the F/B Monitor, .TTINR does not return the carry bit set unless bit 6 of the Job Status Word was on when the request was issued (see below).

There are two modes of doing console terminal input. This is governed by bit 12 of the Job Status Word. If bit 12 = 0, normal I/O is performed. In this mode, the following conditions apply:

1. The monitor echoes all characters typed; lower case characters are converted to upper case.
2. CTRL U (+U) and RUBOUT perform line deletion and character deletion, respectively.
3. A carriage return, line feed, CTRL Z, or CTRL C must be struck before characters on the current line are available to the program. When carriage return is typed, characters on the line typed are passed one-by-one to the user program; both carriage return and line feed are passed to the program.
4. ALTMODEs (octal codes 175 and 176) are converted to ESCAPES (octal 33).

If bit 12 = 1, the console is in special mode. The effects are:

1. The monitor does not echo characters typed except for CTRL C and CTRL O.
2. CTRL U and RUBOUT do not perform special functions.
3. Characters are immediately available to the program.
4. No ALTMODE conversion is done.

In special mode, the user program must echo the characters received. However, CTRL C and CTRL O are acted on by the monitor in the usual
Programmed Requests

way. Bit 12 in the JSW must be set by the user program. This bit is
cleared when control returns to RT-11.

CTRL F and CTRL B are not affected by the setting of bit 12. The
monitor always acts on these characters.

CTRL S and CTRL Q are intercepted by the monitor (unless, under the
F/B monitor, the SET TTY NOPAGE command is issued).

Under the F/B Monitor, if a terminal input request is made and no
character is available, job execution is blocked until a character is
ready. This is true for both .TTYIN and .TTINR, and for both normal
and special modes. If a program really requires execution to continue
and the carry bit to be returned, it must turn on bit 6 of the JSW
(location 44) before the .TTINR request. Bit 6 is cleared when a
program terminates.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No characters available in ring buffer.</td>
</tr>
</tbody>
</table>

Example:

Refer to the example following the description of .TTYOUT/.TTOUTR.

9.4.44 .TTYOUT/.TTOUTR

These requests cause a character to be transmitted from R0 to the
console terminal. The difference, as in the .TTYIN/.TTINR requests,
is that if there is no room for the character in the monitor's buffer,
the .TTOUT request waits for room before proceeding, while the
.TTOUTR does not wait for room and the character in R0 is not output.

Macro Calls: .TTYOUT .char

.TTOUTR

where: .char is the location containing the character to
be loaded in R0 and printed. If not
specified, the character in R0 is printed.
Upon return from the request, R0 still
contains the character.

If the carry bit is set when execution of the .TTOUTR request is
completed, it indicates that there is no room in the buffer and that
no character was output. Under the F/B Monitor, .TTOUTR normally does
not return the carry bit set. Instead, the job is blocked until room
Programmed Requests

is available in the output buffer. If a job really requires execution
to continue and the carry bit to be returned, it must turn on bit 6 of
the Job Status Word (location 44) before issuing the request.

The .TTINR and .TTOUTR requests have been supplied as a help to those
users who do not wish to suspend program execution until a console
operation is complete. With these modes of I/O, if a no-character or
no-room condition occurs, the user program can continue processing and
try the operation again at a later time.

Note:

If a foreground job leaves bit 6 on in the JSW, any further foreground
.TTYIN or .TTYOUT requests will cause the system to lock out the
background. Note also that each partition has its own JSW, and
therefore can be in different terminal modes independently.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Output ring buffer full.</td>
</tr>
</tbody>
</table>

Example:

As an example of the various terminal requests, the following program
is coded in two ways. The program itself accepts a line from the
keyboard, then repeats it on the terminal.

The first example uses .TTYIN and .TTYOUT, which are synchronous. The
monitor retains control until both requests are satisfied, hence there
is no time available for any other processing while waiting.

```
.MCALL ..V2...,REGDEF,TTYIN,TTYOUT
..V2:,REGDEF
START: MOV #BUFFER,R1  IPOINT R1 TO BUFFER
       CLR R2  ICLEAR CHARACTER COUNT
       INC R2  IBUMP COUNT
       CMPB #12,R0  IWAS LAST CHAR=LF?
       BNE INLOOP  INO=GET NEXT
       MOV #BUFFER,R1  IYES=POINT R1 TO BUFFER
OUTLOOP  .TTYOUT (R1)+  IPRINT CHAR
       DEC R2  IDECREASE COUNT
       BEQ START  IDONE IF COUNT = 0
       BR OUTLOOP

BUFFER*

.END   START
```

Rather than wait for the user to type something at INLOOP or wait for
the output buffer to have available space at OUTLOOP, the routine can
be recoded using .TTINR and .TTOUTR as follows:
Programmed Requests

```
.MCALL ....V2....REGDEF...TTYIN...TTYOUT
....V2....REGDEF
.MCALL ...TTINR...TTOUTR..EXIT

START:  MOV  #BUFFER,R1       ;POINT R1 TO BUFFER
         CLR  R2               ;CLEAR CHARACTER COUNT
         BIS  #100,#44         ;WE REALLY WANT CARRY SET
INLOOP:  .TTINR
         BCS  NOCHAR          ;NONE AVAILABLE
         CHRN:  MOVB  R0,(R1)+ ;PUT CHAR IN BUFFER
                  INC  R2         ;INCREASE COUNT
                  CMPB R0,#12      ;WAS LAST CHAR = LF?
                  BNE  INLOOP       ;NO-GET NEXT
                  MOV  #BUFFER,R1  ;YES-POINT R1 TO BUFFER
OUTLOOP: MOVB (R1),R0       ;PUT CHAR IN R0
         .TTOUTR
         BCS  NOROOM          ;NO ROOM IN OUTPUT BUFFER
CHRIN:   DEC  R2              ;DECREASE COUNT
         BEQ  START          ;DONE IF COUNT=0
                  INC  R1          ;BUMP BUFFER POINTER
                  BR  OUTLOOP       ;AND TYPE NEXT

NOCNHR:

.TTINR
     ;PERIODIC CHECK FOR
     ;CHARACTER AVAILABILITY
     ;GOT ONE
     ;
     ;(code to be executed
     ;while waiting)
     ;
     ;BR  NOCHAR

NOROOM:

MOV8  (R1),R0                 ;PERIODIC ATTEMPT TO TYPE
     ;CHARACTER
.TTOUTR
     ;SUCCESSFUL
     ;
     ;(code to be executed
     ;while waiting)
     ;
     ;TYPEIT:  BIC  #100,#44      ;MUST CLEAR THIS BIT
                  ;SO HANG WHILE
                  ;WAITING FOR ROOM.
     .TTYOUT (R1)
     ;PUT CHAR
     BIS  #100,#44            ;RESTORE NO-WAIT
     CHROUT

BUFFER:  .BLKW  100.
.END  START
```

9-97

January 1976
Programmed Requests

.TWAIT

9.4.45 .TWAIT

The .TWAIT request suspends the user's job for an indicated length of
time. .TWAIT requires a queue element, and thus, should be considered
when the .QSET request is executed.

Macro call: .TWAIT .area, .time

where: .time is a pointer to two words of time
(high-order first, low-order second).

Request Format:

R0 ➔ .area: 24 0
.time

Note:

Since a .TWAIT is simulated in the monitor using suspend and resume, a
.RSUM issued from a completion routine without a matching .SPND may
cause the mainstream to continue past a timed wait before the entire
time interval has elapsed.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No queue element was available.</td>
</tr>
</tbody>
</table>

Example:

.TWAIT can be used in applications where a program must be only
activated periodically. This example will 'wake up' every 10 seconds
to perform a task, and then 'sleep' again.

```
.MCALL .V2,...,REGDEF,.TWAIT,.QSET,.EXIT,.PRINT
...V2...
.REGDEF

GOI: .QSET #GAREA,#7
.START: MOV #EMTST,R0
        .TWAIT
        BCS NOQ
        JSR PC,TASK
        BR START
        GAREA: .BLKW 7*7
        EMTST: .BYTE 0,24
        TIME: .WORD TIME
        TASK: .WORD 0,10,*60,10 second intervals
```

January 1976 9-98
Programmed Requests

```
INC MPTR
BIT #1,MPTR
BEQ 15
"PRINT MSG
MBS PC
1$1 "PRINT MSG1
RTS PC
MPTR: "WORD 0
MSG1: "ASCIZ /TICK/
MSG1: "ASCIZ /TOCK/
EVEN
NOQ1: "EXIT
"END 00
```

9.4.46 .WAIT

The .WAIT request suspends program execution until all input/output requests on the specified channel are completed. The .WAIT request combined with the .READ/.WRITE requests makes double-buffering a simple process.

.WAIT also conveys information back through its error returns. An error is returned if either the channel is not currently open or if the last I/O operation resulted in a hardware error.

Macro Call: .WAIT .chan

Note:

In a F/B system, executing a .WAIT when I/O is pending causes that job to be suspended and the other job to run, if possible.

Request Format:

```
R0 \[0 .chan
```

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Channel specified is not open.</td>
</tr>
<tr>
<td>1</td>
<td>Hardware error occurred on the previous I/O operation on this channel.</td>
</tr>
</tbody>
</table>

These error codes make the .WAIT request useful in checking channel status.
Programmed Requests

Example:

For an example of .WAIT used for I/O synchronization, see the examples in the next section.

An example of the use of .WAIT for error detection is its use in conjunction with .CSIGEN to determine which file fields in the command string have been specified. For example, a program such as MACRO might use the following code to determine if a listing file is desired.

```
.MCALL "V2",REGDEF,.WAIT,.CSIGEN,.EXIT
"V2",
REGDEF

START:

.CSIGEN DSPACE,DEXT,D0 IPROCESS COMMAND STRING
.WAIT #0 ICHECK FOR FILE IN FIRST FIELD
BCS NOBINARY I NO BINARY DESIRED

NOBINARY:

.WAIT #1 ICHECK FOR LISTING SPECIFICATION
BCS NOLISTING I NO LISTING DESIRED

NOLISTING:

.WAIT #3 ICHECK FOR INPUT FILE OPEN
BCS ERROR I NO INPUT FILE

ERROR: .EXIT

DEXT: .RAD50 /MAC/ 
.RAD50 /OBJ/ 
.RAD50 /LIST/ 
.WORD 0
DSPACE *,
.END START
```

.WRITE/.WRITC/.WRITW

9.4.47 .WRITE/.WRITC/.WRITW

Note that in the case of .WRITE and .WRITC, additional queue elements should be allocated for buffered I/O operations (see .QSET).

.WRITE

The .WRITE request transfers a specified number of words from memory to the specified channel. Control returns to the user program immediately after the request is queued.
Programmed Requests

Macro Call: .WRITE ,area, ,chan, ,buff, ,wcnt, ,blk

where: .buff is the address of the memory buffer to be used for output.
       .wcnt is the number of words to be written.
       .blk is the number of the block to be written.

Request Format:

```
R0 = .area:
11  .chan
 .blk
 .buff
 .wcnt
  l
```

Notes:

See the note following .WRITW.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Attempted to write past end-of-file.</td>
</tr>
<tr>
<td>1</td>
<td>Hardware error.</td>
</tr>
<tr>
<td>2</td>
<td>Channel was not opened.</td>
</tr>
</tbody>
</table>

Example:

Refer to the examples following .WRITW.

.WRITC

The .WRITC request transfers a specified number of words from memory to a specified channel. Control returns to the user program immediately after the request is queued. Execution of the user program continues until the .WRITC is complete, then control passes to the routine specified in the request. When an RTS PC is encountered in the routine, control returns to the user program.

Macro Call: .WRITC ,area, ,chan, ,buff, ,wcnt, ,crtn, ,blk

where: .buff is the address of the memory buffer to be used for output.
       .wcnt is the number of words to be written.
       .crtn is the address of the completion routine to be entered (see Section 9.2.8).
       .blk is the number relative to the start of the file, not block 0 of the device. The monitor translates the block supplied into an absolute device block number. The user program normally updates .blk before it is used again.
Programmed Requests

Request Format:

R0 \rightarrow .area:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When entering a .WRITC completion function the following are true:

1. R0 contains the channel status word for the operation. If bit 0 of R0 is set, a hardware error occurred during the transfer. The data may not be reliable.

2. R1 contains the octal channel number of the operation. This is useful when the same completion function is to be used for several different transfers.

Notes:

See the note following .WRITW.

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End-of-file on output. Tried to write outside limits of file.</td>
</tr>
<tr>
<td>1</td>
<td>Hardware error occurred.</td>
</tr>
<tr>
<td>2</td>
<td>Specified channel is not open.</td>
</tr>
</tbody>
</table>

Example:

Refer to the examples following .WRITW.

.WRITW

The .WRITW request transfers a specified number of words from memory to the specified channel. Control returns to the user program when the .WRITW is complete.

Macro Call: .WRITW .area, chan, .buff, .wcnt, .blk

where: .buff is the address of the buffer to be used for output.

.wcnt is the number of words to be written. The number must be positive.

.blk is the number of the block to be written.
Programmed Requests

Request Format:

\[
\begin{array}{|c|c|}
\hline
R0 & \text{.area} & 11 \text{.chan} \\
\text{.blk} & \text{.buff} & \text{.wcont} \\
\hline
\end{array}
\]

Note:
Upon return from any WRITE programmed request, R0 will contain no information if the write is to a sequential-access device (for example, magtape). If the write is to a random-access device (disk, DECtape), R0 contains the number of words that will be written (.WRITE or .WRITC) or have been written (.WRITW). If a request is made to write past the end-of-file on a random-access device, the word count is shortened and an error is returned. Note that the write will be done and a completion routine, if specified, will be entered, unless the request cannot be partially filled (shortened word count = 0).

Errors:

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Attempted to write past EOF.</td>
</tr>
<tr>
<td>1</td>
<td>Hardware error.</td>
</tr>
<tr>
<td>2</td>
<td>Channel was not opened.</td>
</tr>
</tbody>
</table>

Examples:

The following routine illustrates the differences between the three types of .READ/.WRITE requests and is coded in three ways, each using a different mode of monitor I/O. The routine itself is a simple program to duplicate a paper tape.

In the first example, .READW and .WRITW are used. The I/O is completely synchronous, with each request retaining control until the buffer is filled (or emptied).

\[
\text{.MCALL} \quad \text{.V2..} \quad \text{REGDEF,.FETCH,.READW,.WRITW}
\text{.MCALL} \quad \text{ENTER,.LOOKUP,.PRINT,.EXIT,.CLOSE,.WAIT}
\text{.V2..}
\text{.REGDEF}
\]

ERRNO=52

START:
\text{.FETCH} \quad \text{#SPACE,#PRNAME} \quad \text{GET PR HANDLER}
\text{BCS} \quad \text{FERR} \quad \text{PR NOT AVAILABLE}
\text{MOV} \quad \text{R8,R2} \quad \text{PR HAS NEXT FILE LOCATION}
\text{.FETCH} \quad \text{R2,#PPNAME} \quad \text{GET PP HANDLER}
\text{BCS} \quad \text{FERR} \quad \text{NOT AVAILABLE}
\text{MOV} \quad \text{#AREA,R5} \quad \text{GETM ARGUMENT AREA}
\text{CLR} \quad \text{R4} \quad \text{R4 IS OUTPUT CHANNEL 1}
\text{MOV} \quad \text{#1,R3} \quad \text{R3 IS INPUT CHANNEL 1}
\text{.ENTER} \quad \text{R5,R4,#PPNAME} \quad \text{ENTER THE FILE}
\text{BCS} \quad \text{ENERR} \quad \text{SOME ERROR IN ENTER}
\text{.LOOKUP} \quad \text{R5,R3,#PRNAME} \quad \text{LOOKUP FILE ON CHANNEL 1}
\text{BCS} \quad \text{LKERR} \quad \text{ERROR IN LOOKUP}
\text{CLR} \quad \text{R1} \quad \text{USE R1 AS BLOCK NUMBER}

LOOP:
\text{.READW} \quad \text{R5,R3,#BUFF,#256,R1} \quad \text{READ ONE BLOCK}
\text{BCS} \quad \text{RDERR}
\text{.WRITW} \quad \text{R5,R4,#BUFF,#256,R1} \quad \text{WRITE THAT BLOCK}

9-103 January 1976
Programmed Requests

BCS   WTERR
INC   R1

;BUMP BLOCK, NOTE: THIS IS
;NOT NECESSARY FOR NON-FILE
;DEVICES IN GENERAL, IT IS
;USED HERE AS AN EXAMPLE OF
;A GENERAL TECHNIQUE,

JMP   KEEP GOING
ROERR: TSTB ERRWD
BEQ   10

;ERROR IS IT EOF?
;YES
;INC, MAKE READ ERROR

IS1:  CLOSE R3
      CLOSE R4

;CLOSE INPUT AND OUTPUT

EXIT

WTERR: PRINT #WTMSG
EXIT

PRNAME = RADS0 /PR /
WORD 0

;NOTE THAT PR NEEDS NO FILE NAME
;FILE NAME NEED ONLY BE 0.

PPNAME = RADS0 /PP /
WORD 0

FERRI = PRINT #FMSG
EXIT

;ERROR ACTIONS GO HERE, IT IS
;GENERALLY UNDESIRABLE TO
;EXECUTE A MALT OR RESET
;INSTRUCTION ON ERROR.

ENERRI = PRINT #EMSG
EXIT

LERRRI = PRINT #LMSG
EXIT

FMSGI = ASCII /NO DEVICE?/
EMSGI = ASCII /ENTRY ERROR?/
LMMSGI = ASCII /LOOKUP ERROR?/
RDMSGI = ASCII /READ ERROR?/
WTMSGI = ASCII /WRITE ERROR?/

EVEN
AREA1 = BLKW 10
BUFFI = BLKW 256
HSPACE =

.END START

The same routine can be coded using .READ and .WRITE as follows. The
.WAIT request is used to determine if the buffer is full or empty
prior to its use.

.MCALL .V2,...,REGDEF,FETCH,READ,WRITE
.MCALL ,ENTER,LOOKUP,PRINT,EXIT,CLOSE,WAIT
.V2,...
.REGDEF

ERRWD=52

START: INC R2

;GET PR HANDLER

BCS FERR  ;PR NOT AVAILABLE
MOV R0,R2 ;R0 HAS NEXT FREE LOCATION

;GET PP HANDLER

BCS FERR  ;NOT AVAILABLE
MOV #AREA,R5 ;SET ARGUMENT AREA

CLR R4  ;R4 IS OUTPUT CHANNEL 1
MOV #1,R3  ;R3 IS INPUT CHANNEL 11

,ENTER R5,R4,#PPNAME ;ENTER THE FILE

9-104
Programmed Requests

BCS   ENERR  ; SOME ERROR IN ENTER
       .LOOKUP R5, R3, #PRNAME ; LOOKUP FILE ON CHANNEL 1
       BCS   LKERR  ; ERROR IN LOOKUP
       CLR    R1  ; USE R1 AS BLOCK NUMBER
LOOP:  .READ R5, R3, #BUFF, #256, , R1 ; READ A BUFFER
       BCS   RDERR  ; ERROR IN READ
       .WAIT R3  ; WAIT FOR BUFFER
       BCS   IGERR  ; ERROR HERE IS HARMFUL
       .WRITE R5, R4, #BUFF, #256, , R1 ; WRITE THE BUFFER
       BCS   IGERR  ; I/O ERROR
       INC    R1  ; KEEP GOING
       BR     LOOP  ; JOB IS NOT OVER YET
       .TSTB ERRWD ; ERROR, IS IT EOF?
       .ONE IGERR  ; NO, HARMFUL ERROR
       .CLOSE R3  ; CLOSE INPUT AND OUTPUT
       .CLOSE R4
       .EXIT
       IOERR1 .PRINT #IOMSG ; IAND EXIT,
       .EXIT
       .PRINT #PRMSG ; AND, HARD READ ERROR
       .EXIT
       PRNAME1 .RAD50 /PR/ ; NOTE THAT PR NEEDS NO FILE NAME
       .WORD 0  ; IF FILE NAME NEED ONLY BE 0.
       PPNAME1 .RAD50 /PP/ ;
       .WORD 0
FERR1  .PRINT #FMSG ; ERROR ACTIONS GO HERE, IT IS
       .EXIT
       ENERR1 .PRINT #EMSG ; USUALLY UNDESIRABLE TO
       .EXIT
       LKERR1 .PRINT #LMSG ; EXECUTE A HALT OR RESET
       .EXIT
       FMSG1  .ASCIZ /NO DEVICE?/
       EMSG1  .ASCIZ /ENTRY ERROR?/
       LMSG1  .ASCIZ /LOOKUP ERROR?/
       IOMSG1 .ASCIZ "I/O ERROR?"
       WTMGS1 .ASCIZ /WRITE ERROR?/
       .EVEN
       AREA1 .BLKW 10
       .BUFF1 .BLKW 256,
       .MSPACE=.END START

.RAND and .WRITE are also often used for double-buffered I/O. The basic double-buffering algorithm for input is:

Explanation

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ BUFFER 1</td>
<td>Fill BUFFER 1</td>
</tr>
<tr>
<td>WAIT</td>
<td>Wait for BUFFER 1 to fill</td>
</tr>
<tr>
<td>READ BUFFER 2</td>
<td>Start filling BUFFER 2</td>
</tr>
<tr>
<td>USE BUFFER 1</td>
<td>Process BUFFER 1 while BUFFER 2 fills</td>
</tr>
<tr>
<td>WAIT</td>
<td>Wait for BUFFER 2 to fill</td>
</tr>
<tr>
<td>READ BUFFER 1</td>
<td>Start filling BUFFER 1</td>
</tr>
<tr>
<td>USE BUFFER 2</td>
<td>Process BUFFER 2 while BUFFER 1 fills</td>
</tr>
</tbody>
</table>
Programmed Requests

Correspondingly, the basic double-buffering algorithm for output is:

<table>
<thead>
<tr>
<th>LOOP</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILL BUFFER 1</td>
<td>Prepare BUFFER 1 for output</td>
</tr>
<tr>
<td>WRITE BUFFER 1</td>
<td>Start emptying BUFFER 1</td>
</tr>
<tr>
<td>FILL BUFFER 2</td>
<td>Fill BUFFER 2 while BUFFER 1 empties</td>
</tr>
<tr>
<td>WAIT BUFFER 1</td>
<td>Wait for BUFFER 1 to empty</td>
</tr>
<tr>
<td>WRITE BUFFER 2</td>
<td>Start emptying BUFFER 2</td>
</tr>
<tr>
<td>FILL BUFFER 1</td>
<td>Fill BUFFER 1 while BUFFER 2 empties</td>
</tr>
<tr>
<td>WAIT BUFFER 2</td>
<td>Wait for BUFFER 2 to empty</td>
</tr>
<tr>
<td>BR LOOP</td>
<td></td>
</tr>
</tbody>
</table>

The previous example program can be coded using completion routines via .READC and .WRITC as follows. Once the initial read is performed, the remainder of the I/O is performed by the completion routines.

```
;MCALL V2...REDF, FENTC,FETCH, READC, WRITC
;MCALL ENTER, LOOKUP, PRINT, EXIT, CLOSE, WAIT;
;V2...
;REDF

ERRBYTE=52

START: FETCH #HSPACE,#PPNAME: IGFT PR HANDLER
RCS FLNK: IPR NOT AVAILABLE
MOV RO,RR: IR0 HAS NEXT FREE LOCATION
FETCH R2,#PPNAME: IGFT PP HANDLER
FLNK RCS FERR: INOT AVAILABLE
MOV #AREA,R5: IEMT ARGUMENT AREA
CLR R4: IR4 IS OUTPUT CHANNEL 0
MOV #1,R3: IR7 IS INPUT CHANNEL 1
ENTER R5,RA,#PPNAME: ENTER THE FILE
RCS ENERR: ISOME ERROR IN ENTER
LOOKUP RS,R3,#PPNAME: LOOKUP FILE ON CHANNEL 1
RCS LKPR: IERROR IN LOOKUP
CLR R1: IUSE R1 AS BLOCK NUMBER
DCL RG, FFLG: ICLEAR DONG/ERROR FLAG
READC RS,R3,#BUFF,#256,#RDCOMP,R1; IREAD ONE BLOCK
RCS EOF: INO ERROR WILL HAPPEN HERE
181 TST DFLG: IDONE FLAG SET?
REG RS, R4: INO, WAIT FOR IT TO RE SET;
RMI TDFR: YES, BUT HARD ERROR OCCURRED
FOF: CLOSE R3: ICLOSE INPUT AND OUTPUT CHANNELS
CLOSE R4: IDONE
EXIT: IALL DONE

;ENABL ILSB
RDTCOM: ROR R0: IIF BIT 0 SET
RCS RWRR: IAN ERROR OCCURRED
;WRITE #AREA,#0,#BUFF,#254,#WRCOMP,BLKN IWRITE THAT BLOCK
RWRITC MCS: IERROR HERE IS HARDWARE
RWRITC MOV #=1, FFLG: IFLAG THE ERROR
RWRITC 2S1 PC
RDWRITC ROR R0: IHARDWARE ERROR
TNC RKL: IBUMP BLOCK NUMBER
;READC #AREA,#1,#BUFF,#254,#RDCOMP,BLKN
RCC 53: INO ERROR
TSTB ERRBYTE: IEOF?
```

January 1976

9-106
Programmed Requests

RNE RWERP
INN DF1G
RTS PC

.DSRL
I SM

FER1 MOV #FMSG,R0
MR TYPIT

ERROR ACTIONS GO HERE. IT IS
GENERALLY UNDESIRABLE TO
EXECUTE A HALT OR RESET
INSTRUCTION ON ERROR.

ENERR1 MOV #FMSG,R0
MR TYPIT

TOERR1 MOV #LMSG,R0
MR TYPIT

LKERR1 MOV #LMSG,R0

TYPIT .PRINT

.EXIT

.NLIST BFX

FMSG1 ,ASCIIZ /NO DEVICE/

EMSG1 ,ASCIIZ /ENTRY ERROR/

LMSG1 ,ASCIIZ /LOOKUP ERROR/

TOMSG1 ,ASCIIZ "$0 ERROR"

.LIST REV

.EVEN

DFLGI .WORD 0

PRNAME1 ,RAD50 /P / NOTE THAT PR NEEDS NO FILE NAME

PPNAME1 ,RAD50 /P /

RLK1 .WORD 0

AREA1 .BLKW 10

BUFF1 .BLKW 256

WSPACE .END START

The following example incorporates the .LOOKUP, .READ, and .CLOSE
requests. The program opens the file RT11.MAC which is on the system
device, ST1, for input on channel 0. The first block is read and the
file is then closed.

.MCALL .V2,...,REGEF,.CLOSE,.LOOKUP

.MCALL .PRINT,.EXIT,.READ,.FETCH

..V2...

.REGEF

.START1 MOV #LIST,R5 ; EMT ARGUMENT LIST POINTER

CLR R4

CLR R3

.CORADD,#FPTR

IFETCH DEVICE HANDLER

BCC 25

MOV #FETMSG,R0

IFETCH ERROR

IPRINT ERROR MESSAGE

25: .PRINT

.EXIT

25: .LOOKUP R5,R3,#FPTR

BCC 35

MOV #LMSG,R0

IPRINT FAILURE MESSAGE

BR 15

35: .READ R5,R3,#BUFF,#25b,,R4 1READ ONE BLOCK

BCC 45

MOV #RDMGS,R0

IREAD ERROR

BR 15

45: .CLOSE R3

ICLOSE THE CHANNEL

LIST1 .BLKW 5

.FPthr1 .RAD50 /SY RT11 MAC/

.HAD50 OF FIEL NAME,DEVICE

.FETMSG1 ,ASCIIZ /FETCH FAILED/ ASCII MESSAGES

.LKMSG1 ,ASCIIZ /LOOKUP FAILED/

9-107
Programmed Requests

\[ \text{RDMSG} \quad \text{ASCIZ} \quad / \text{READ FAILED}/ \]
\[ \text{EVEN} \]
\[ \text{COPADDI} \quad \text{BLKW} \quad 2000 \quad / \text{SPACE FOR LARGEST HANDLERS} \]
\[ \text{BUFF} \quad *, \quad \text{END} \quad \text{START} \]

9.5 CONVETING VERSION 1 MACRO CALLS TO VERSION 2

As mentioned in the introduction of this chapter, RT-11 Version 2 supports a slightly modified format for system macro calls than Version 1. This section details the conversion process from the Version 1 format to Version 2.

9.5.1 Macro Calls Requiring No Conversion

Version 1 macro calls which need no conversion are:

- \text{.CSIGN}
- \text{.RCTLO}
- \text{.CSISPC}
- \text{.RELEAS}
- \text{.DATE}
- \text{.SETTOP*}
- \text{.DSTATUS}
- \text{.SRESET}
- \text{.EXIT}
- \text{.TTINR**}
- \text{.FETCH}
- \text{.TTOUTR}
- \text{.HRESET}
- \text{.TTYIN}
- \text{.LOCK}
- \text{.TTYOUT}
- \text{.PRINT}
- \text{.UNLOCK}
- \text{.QSET}

*Provided location 50 is examined for the maximum value.

**Except in F/B System.

9.5.2 Macro Calls Which May Be Converted

The following Version 1 macro calls may be converted:

- \text{.CLOSE}
- \text{.RENAME}
- \text{.DELETE}
- \text{.REOPEN}
- \text{.ENTER}
- \text{.SAVESTATUS}
- \text{.LOOKUP}
- \text{.WAIT}
- \text{.READ}
- \text{.WRITE}

The general format of the V1 macro is:

\[ \text{.PRGREQ} \quad \text{.chan, .arg1 .arg2,...argn} \]

In this form, \text{.chan} is an integer between 0 and 17 (inclusive), and is not a general assembler argument. The channel number is assembled into the EMT instruction itself. The arguments \text{arg1-argn} are always pushed either into R0 or on the stack.

The V2 equivalent of the above call is:

\[ \text{.PRGREQ} \quad \text{.area, .chan, .arg1,...argn} \]
Programmed Requests

In the V2 call, the .chan argument can be any legal assembler argument; it need not be in the range 0 to 17 (octal), but should be in the range 0-377 (octal). .area points to a memory list where the arguments arg1...argn will go.

As an example, consider a .READ request in both forms:

V1:     .READ 5,#BUFF,#256.,BLOCK
V2:     .READ #AREA,#5,#BUFF,#256.,BLOCK

AREA:   .WORD 0 ;CHANNEL/FUNCTION CODE HERE
        .WORD 0 ;BLOCK NUMBER HERE
        .WORD 0 ;BUFFER ADDRESS HERE
        .WORD 0 ;WORD COUNT HERE
        .WORD 0 ;A 1 GOES HERE.

Thus, the difference in the calls is that in Version 2 the channel number becomes a legal assembler argument and the .area argument has been added.

Following is a complete list of the conversions necessary for each of the EMT calls. Both the Version 1 and Version 2 formats are given. Note that parameters inside [] are optional parameters. Refer to the appropriate section in this chapter for more details of each request.

<table>
<thead>
<tr>
<th>Version</th>
<th>Programmed Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1:</td>
<td>.DELETE .chan,.dblk</td>
</tr>
<tr>
<td>V2:</td>
<td>.DELETE .area,.chan,.dblk,[.count]</td>
</tr>
<tr>
<td>V1:</td>
<td>.LOOKUP .chan,.dblk</td>
</tr>
<tr>
<td>V2:</td>
<td>.LOOKUP .area,.chan,.dblk,[.count]</td>
</tr>
<tr>
<td>V1:</td>
<td>.ENTER .chan,.dblk,[.length]</td>
</tr>
<tr>
<td>V2:</td>
<td>.ENTER .area,.chan,.dblk,[.length],[.count]</td>
</tr>
<tr>
<td>V1:</td>
<td>.RENAME .chan,.dblk</td>
</tr>
<tr>
<td>V2:</td>
<td>.RENAME .area,.chan,.dblk</td>
</tr>
<tr>
<td>V1:</td>
<td>.SAVESTAT .chan,.cblk</td>
</tr>
<tr>
<td>V2:</td>
<td>.SAVESTAT .area,.chan,.cblk</td>
</tr>
<tr>
<td>V1:</td>
<td>.RFOPEN .chan,.cblk</td>
</tr>
<tr>
<td>V2:</td>
<td>.REOPEN .area,.chan,.cblk</td>
</tr>
<tr>
<td>V1:</td>
<td>.CLOSE .chan</td>
</tr>
<tr>
<td>V2:</td>
<td>.CLOSE .chan</td>
</tr>
<tr>
<td>V1:</td>
<td>.READ/.READW .chan,.buff,.wcnt,.blk</td>
</tr>
<tr>
<td>V2:</td>
<td>.READ/.READW .area,.chan,.buff,.wcnt,.blk</td>
</tr>
<tr>
<td>V1:</td>
<td>.READC .chan,.buff,.wcnt,.crtn,.blk</td>
</tr>
<tr>
<td>V2:</td>
<td>.READC .area,.chan,.buff,.wcnt,.crtn,.blk</td>
</tr>
<tr>
<td>V1:</td>
<td>.WRITE/.WRITW .chan,.buff,.wcnt,.blk</td>
</tr>
<tr>
<td>V2:</td>
<td>.WRITE/.WRITW .area,.chan,.buff,.wcnt,.blk</td>
</tr>
</tbody>
</table>

9-109
Programmed Requests

V1: .WRITC .chan,.buff,.wcnt,.crtn,.blk
V2: .WRITC .area,.chan,.buff,.wcnt,.crtn,.blk

V1: .WAIT .chan
V2: .WAIT .chan

Important features to keep in mind for Version 2 calls are:

1. Version 2 calls require the .area argument, which points to the area where the other arguments will be.

2. Enough memory space must be allocated to hold all the required arguments.

3. .chan must be any legal assembler argument, not just an integer between 0-17 (octal).

4. Blank fields are permitted in the Version 2 calls. Any field not specified (left blank) is left alone in the argument block.
CHAPTER 10
EXPAND UTILITY PROGRAM

EXPAND is an RT-ll system program which processes the macro references in a macro assembly language source file. EXPAND accepts a subset of the complete macro language and, using the system library file SYSMAC.SK, produces an output file in which all legal macro references are expanded into macro-free source code. EXPAND is normally used with ASEMBL, the RT-ll assembler designed for minimum memory configurations (refer to Chapter 11).

10.1 LANGUAGE

EXPAND simply copies its input files to its output file unless it encounters any of the following directives (see Chapter 5 for more information about these directives):

1. .MCALL
   Directs EXPAND to search the file SY:SYSMAC.SML to find the macro names listed in the .MCALL directive. If the macro names are found, EXPAND stores their definitions in its internal tables.

2. .MACRO
   Directs EXPAND to copy a macro definition from the user's input file into the internal tables.

3. .name
   If .name is the name of a macro defined in either a .MCALL or .MACRO directive, then .name is expanded according to the definition stored for it in the EXPAND internal tables.

4. .ENDM
   If encountered while storing a macro definition, the .ENDM directive terminates the definition. It is not recognized outside macro definitions.

10.2 RESTRICTIONS

Unlike the full macro assembler (MACRO), EXPAND only expands macros that observe the following restrictions:
EXPAND Utility Program

1. The following directives may not be used:

```
.ERROR    .NARG
.IF DIF    .NCHR
.IF IDN    .NTYPE
.IRP       .PRINT
.IRPC      .REPT
.MEXIT
```

2. Macros cannot be nested. Recursive macros that call themselves directly or indirectly are illegal and cause an error message.

3. Macros cannot be redefined. Once a name has been used for a macro name, it cannot be used again in the program for a macro or symbol name.

4. Macro names must begin with a dot (.). If the dot is missing, an error message is printed.

5. Dummy argument names must begin with a dot (.). Such names cannot be used as dummy argument names in the macro but can be used for other purposes outside of the macro.

6. The backslash operator is not available.

7. Automatically created symbols are not available.

8. No more than 30 arguments may be used in any MACRO directive.

10.3 CALLING AND USING EXPAND

To run EXPAND, type:

```
R EXPAND
```

in response to the dot printed by the Keyboard Monitor. EXPAND responds with an asterisk indicating that it is ready to accept a command string. A command string must be of the following form:

```
*ofile=ifile1,ifile2,...,ifile6
```

ifile2 through ifile6 are optional. Each file specification follows the general RT-II command string syntax (dev:filnam,ext). The default value for each file specification is noted below:

<table>
<thead>
<tr>
<th>I/O File</th>
<th>Dev</th>
<th>Ext</th>
</tr>
</thead>
<tbody>
<tr>
<td>ofile</td>
<td>DK</td>
<td>PAL</td>
</tr>
<tr>
<td>ifile1,..., ifile6</td>
<td>device used for last source file specified or DK</td>
<td>MAC</td>
</tr>
</tbody>
</table>

Type CTRL C to halt EXPAND and return control to the monitor. To restart EXPAND, type R EXPAND or the REENTER command in response to the monitor's dot.

10-2
EXPAND Utility Program

EXPAND copies sequentially the specified input files to the specified output file until a macro directive is encountered. EXPAND then changes the macro directive to a comment by inserting a semicolon so that it will not be seen later by the assembler (usually ASEMBl).

If the directive is .MCALL, EXPAND searches the system library file (SYSMAC8K) for the requested macro definitions. The requested definitions are then included in the user's program in the order in which they are found in the library.

For the .MACRO directive, EXPAND reads each line following the directive up to the next .ENDM directive. Each line is stored in the internal definition table and then changed to a comment in the output file so that it is not processed later by the assembler. Also, any occurrence of a macro argument name within the definition is flagged internally so that it can be replaced by the real argument value whenever the macro is later referenced.

For macro references, EXPAND locates the stored macro definition in its internal tables, binds the actual argument values to the argument names, and changes the macro reference to a comment line. EXPAND then begins copying the stored definition to the output file. Whenever a macro argument name is encountered in the definition, it is replaced by the corresponding actual argument value.

Examples:

The following are examples of input and corresponding EXPAND output.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT=11  MACRO EXPAND VR2=02</td>
<td>RT=11  MACRO EXPAND VR2=02</td>
</tr>
<tr>
<td>R1=1 SP=6 PC=7</td>
<td>R1=1 SP=6 PC=7</td>
</tr>
<tr>
<td>.MACRO CALL SUBR</td>
<td>.MACRO CALL SUBR</td>
</tr>
<tr>
<td>JSR PC, SUBR</td>
<td>JSR PC, SUBR</td>
</tr>
<tr>
<td>.ENDM</td>
<td>.ENDM</td>
</tr>
<tr>
<td>.MCALL LOOKUP, READ, VP</td>
<td>.MCALL LOOKUP, READ, VP</td>
</tr>
<tr>
<td>...V2=1</td>
<td>...V2=1</td>
</tr>
<tr>
<td>.ENDM</td>
<td>.ENDM</td>
</tr>
<tr>
<td>.MACRO LOOKUP AREA, CHAN, CHANN, SPF</td>
<td>MOV CHAN, %0</td>
</tr>
<tr>
<td>IF OF ...V1</td>
<td>FMT 0&lt;20+, AREA&gt;</td>
</tr>
<tr>
<td>IF NR CHAN</td>
<td></td>
</tr>
<tr>
<td>.ENDC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>.MACRO AREA</td>
<td>MOV AREA, %0</td>
</tr>
<tr>
<td>IF NR AREA</td>
<td>MOVB #1, (0)</td>
</tr>
<tr>
<td>.ENDC</td>
<td></td>
</tr>
<tr>
<td>.MACRO CHAN</td>
<td>MOVB CHAN, %0</td>
</tr>
<tr>
<td>IF NR CHAN</td>
<td></td>
</tr>
<tr>
<td>.ENDC</td>
<td></td>
</tr>
</tbody>
</table>
EXPAND Utility Program

. ENDC
. IF NB 0
. ENDC
. IF NB
. ENDC
. IF NB
. IFF
. ENDC
. ENDC

CLR R1
READ #AREA,#0,#BUFF,#256,,R1

MOV "INBLK,%0
MOV #1,1(0)

MOV $0,0(0)$

MOV $2,0(0)$

MOV $4,0(0)$

CLR $4,0(0)$

EMT $0375$

CLR R1
READ #AREA,#0,#BUFF,#256,,R1

. IF NF ... V1
. IF NF #256.
. ENDC

MOV $256,#0$

MOV $1,-(6)$

MOV $BUFF,-(6)$

MOV $0,-(6)$

EMT $0<200+#AREA$

. IFF
. IF NF #AREA
. ENDC

MOV #AREA,#0

MOV #8,1(0)

MOV $0,(0)$

. IF NF #R1
. ENDC

. IF NF R1
. ENDC

. IF NF #BUFF
. ENDC

. IF NF #BUFF
. ENDC

MOV $BUFF,4,(0)$

MOV #256,6,(0)

MOV $1,8,(0)$

EMT $0375$

HALT
. END START

. END START

HALT
EXPAND Utility Program

10.4 EXPAND ERROR MESSAGES

The following messages are caused by fatal errors detected by EXPAND. They print on the console terminal and cause EXPAND to restart:

<table>
<thead>
<tr>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>?BAD SWITCH?</td>
<td>An unrecognized command string switch was specified.</td>
</tr>
<tr>
<td>?INPUT ERROR?</td>
<td>Hardware error in reading an input file.</td>
</tr>
<tr>
<td>?INSUFFICIENT CORE?</td>
<td>Not enough memory to store macro definitions.</td>
</tr>
<tr>
<td>?MISSING END IN MACRO?</td>
<td>End of input was encountered while storing a macro definition; probably missing an .ENMD.</td>
</tr>
<tr>
<td>?NO INPUT FILE?</td>
<td>There must be at least one input file.</td>
</tr>
<tr>
<td>?OUTPUT DEVICE FULL?</td>
<td>No room to continue writing output; try to compress the device with PIP.</td>
</tr>
<tr>
<td>?WRONG NUMBER OF OUTPUT FILES?</td>
<td>There must be exactly one output file.</td>
</tr>
</tbody>
</table>

The following errors are non-fatal but indicate that something is wrong in the input file(s). These errors appear in the output file as a line in the following form:

?*** ERROR *** message

After each run of EXPAND, the total number of non-fatal errors is printed on the console terminal.

<table>
<thead>
<tr>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD MACRO ARG</td>
<td>The macro argument is not formatted correctly.</td>
</tr>
<tr>
<td>LINE TOO LONG</td>
<td>A line has become longer than 132 characters.</td>
</tr>
<tr>
<td>MACRO ALREADY DEFINED</td>
<td>A macro was defined more than once.</td>
</tr>
<tr>
<td>MACRO(S) NOT FOUND</td>
<td>Macros listed in an .MCALL statement were not found in SYSMAC.8K (make sure SYSMAC.8K is present on system).</td>
</tr>
<tr>
<td>MISSING COMMA IN MACRO ARG</td>
<td>Found spaces or tabs within a macro argument when a comma was expected; try using brackets around the arguments, e.g., &lt;arg with spaces&gt;.</td>
</tr>
</tbody>
</table>

January 1976

10-6
<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISSING DOT</td>
<td>A macro name or argument name does not begin with a dot.</td>
</tr>
<tr>
<td>NAME DOESN'T MATCH</td>
<td>Optional name given in .ENDM directive does not match name given in corresponding .MACRO directive.</td>
</tr>
<tr>
<td>NESTED MACROS</td>
<td>A macro is being defined or invoked within another macro.</td>
</tr>
<tr>
<td>NO NAME</td>
<td>A macro definition has no name.</td>
</tr>
<tr>
<td>SYNTAX</td>
<td>A macro directive is not constructed correctly.</td>
</tr>
<tr>
<td>TOO MANY ARGS</td>
<td>A macro directive has more than 30 arguments.</td>
</tr>
</tbody>
</table>
CHAPTER 11
ASUREL, THE 8K ASSEMBLER

ASUREL is designed for use on an RT-11 system with minimum memory space (or larger systems where system table space is critical) and is a subset of the RT-11 MACRO assembler described in Chapter 5. ASSEMBL has the same features as MACRO with the following exceptions:

1. MACRO directives (MACRO, .MCALL, .ENDM, .IRP, etc.) are not recognized
2. DATE is not printed in listings
3. Wide line-printer output is not available
4. There is no lower-case mode
5. There is no enable/disable punch directive
6. There are no floating point directives
7. There are no local symbols or local symbol blocks
8. CREF is not available

Many of the macro features are supported by the EXPAND program (as described in Chapter 10).

11.1 CALLING AND USING ASSEMBL

ASUREL is loaded in response to the dot printed by the Keyboard Monitor with the RT-11 monitor R Command as follows:

    R ASSEMBL

followed by the RETURN key. ASSEMBL responds with an asterisk (*) and waits for specification of the output and input files in the standard RT-11 format as follows:

    *object,listing=source1,...,source6
ASEMVL, the 8K Assembler

where:

object is a binary object file output by ASEMVL.

listing is the assembly listing file containing the assembly listing and symbol table.

source1,...,source6 are the ASCII source files containing the ASEMVL source program(s). A maximum of six source files is allowed.

A null specification in any of the file fields signifies that the associated input or output file is not desired. ASEMVL file specifications follow the standard RT-ll convention (dev:filnam.ext). The default value for each file specification is noted below:

<table>
<thead>
<tr>
<th>I/O File</th>
<th>Dev</th>
<th>Ext</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>DK</td>
<td>.OBJ</td>
</tr>
<tr>
<td>listing</td>
<td>device used for object output</td>
<td>.LST</td>
</tr>
<tr>
<td>source1,...,sourceN</td>
<td>device used for last source file specified or DK</td>
<td>.PAL</td>
</tr>
</tbody>
</table>

Type CTRL C to halt ASEMVL and return control to the monitor. To restart ASEMVL, type R ASEMVL or the REENTER command in response to the monitor's dot.

Table 11-1 lists the RT-ll macro directives which are not available in ASEMVL.

Table 11-1
Directives not Available in ASEMVL

<table>
<thead>
<tr>
<th>Directive</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.MACRO</td>
<td>Macros cannot be defined in ASEMVL.</td>
</tr>
<tr>
<td>.ENDM</td>
<td></td>
</tr>
<tr>
<td>.MEXIT</td>
<td></td>
</tr>
<tr>
<td>.MCALL</td>
<td></td>
</tr>
<tr>
<td>.NCHR</td>
<td>The number of characters in an argument cannot be obtained with a macro.</td>
</tr>
<tr>
<td>.NARG</td>
<td>The number of arguments in a macro cannot be obtained with a macro.</td>
</tr>
<tr>
<td>\ (backslash)</td>
<td>Symbols used as macro arguments cannot be passed as a numeric string.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 11-1 (Cont.)
Directives not Available in ASEMBL

<table>
<thead>
<tr>
<th>Directive</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ERROR</td>
<td>Messages cannot be flagged with a P error code output as part of the assembly listing. Comment lines can be used to replace .ERROR.</td>
</tr>
<tr>
<td>.IF IDN</td>
<td>Strings cannot be compared.</td>
</tr>
<tr>
<td>.IF DIF</td>
<td>)</td>
</tr>
<tr>
<td>.IRP</td>
<td>Indefinite repeat blocks cannot be created.</td>
</tr>
<tr>
<td>.IRPC</td>
<td>)</td>
</tr>
<tr>
<td>.NTYPE</td>
<td>A macro cannot be modified based on the addressing mode of an argument.</td>
</tr>
<tr>
<td>.PRINT</td>
<td>Messages cannot be output as part of the assembly listing. Comment lines can be used to replace .PRINT.</td>
</tr>
<tr>
<td>.REPT</td>
<td>A block of code cannot be duplicated a number of times in-line with other source code using a directive.</td>
</tr>
<tr>
<td>.LIST ME</td>
<td>)</td>
</tr>
<tr>
<td>.NLIST ME</td>
<td>)</td>
</tr>
<tr>
<td>.LIST MEB</td>
<td>)</td>
</tr>
<tr>
<td>.NLIST MEB</td>
<td>)</td>
</tr>
<tr>
<td>.LIST MD</td>
<td>)</td>
</tr>
<tr>
<td>.NLIST MD</td>
<td>)</td>
</tr>
<tr>
<td>.LIST MC</td>
<td>)</td>
</tr>
<tr>
<td>.NLIST MC</td>
<td>)</td>
</tr>
<tr>
<td>.LIST TTM</td>
<td>)</td>
</tr>
<tr>
<td>.NLIST TTM</td>
<td>)</td>
</tr>
<tr>
<td>.ENABL LC</td>
<td>All lower case ASCII input is converted to upper case.</td>
</tr>
<tr>
<td>.DSABL LC</td>
<td>)</td>
</tr>
<tr>
<td>.ENABL LSB</td>
<td>Local symbols and local symbol blocks are not available in ASEMBL.</td>
</tr>
<tr>
<td>.DSABL LSB</td>
<td>)</td>
</tr>
<tr>
<td>.ENABLE PNC</td>
<td>Binary output is always enabled.</td>
</tr>
<tr>
<td>.DSABL PNC</td>
<td>)</td>
</tr>
<tr>
<td>.ENABL FPT</td>
<td>Floating point directives are not available.</td>
</tr>
<tr>
<td>.DSABL FPT</td>
<td>)</td>
</tr>
<tr>
<td>.FLT2</td>
<td>)</td>
</tr>
<tr>
<td>.FLT4</td>
<td>)</td>
</tr>
</tbody>
</table>

Example:

This example uses the output produced by the EXPAND program as input to ASEMBL. The assembly listing follows.
ASEMFL, the 8K Assembler

```
MAIN, RT=11 MACRO V82-10 PAGE 1

1
2  RT-11 MACRO EXPAND V82-02
3
4  000001   R1=1
5  000006   SPX=6
6  000007   PCX=7
7
8    .MACRO .CALL .SUBR
9    JSR PCX, SUBR
10    .ENDM
11
12    .MACRO .LOOKUP .AREA, .CHAN, .DEVLK, .SPF
13    .TF DF ..., V1
14    .TF NB , .CHAN
15
16    .FNDC
17    MOV , .CHAN, %0
18    .FNDEC
19    EMT %0<20+, .AREA>
20
21    .FNDC
22    MOV , .AREA, %0
23    .FNDEC
24    MOVB #1, 1(0)
25    .FNDC
26    MOVB , .CHAN, (0)
27    .FNDC
28    MOV , .DEVLK, P, (0)
29    .FNDC
30    MOV , .SPF, 4, (0)
31    .FNDC
32    CLR 4, (0)
33    .FNDC
34    EMT %0<75
35    .FNDC
36    .FNDC
37    .FNDC
38    .MACRO .READ .AREA, .CHAN, .BUFF, .WCNT, .BLK
39    .TF DF ..., V1
40    .TF NB , .WCNT
41
42    .FNDC
43    MOV #1, = (6)
44    MOV , .BUFF, = (4)
45    MOV , .CHAN, = (4)
46    EMT %0<200+, .AREA>
47
48    .FNDC
49    MOV , .AREA, %0
50    .FNDC
51    MOVB #8, 1(0)
52    .FNDC
53    MOVB , .CHAN, (0)
54    .FNDC
55    MOVB , .BLK, 2, (0)
56    .FNDC
57
```

11-4
ASEMBL, the 8K Assembler

`.MAIN`, RT=11 MACRO V502-10 PAGE 1+

58       JTF NB 'BUFF
59
60       JENDC
61       JTF NB 'WENT
62
63       JENDC
64       J
65       MOV #1,8,(0)
66       JENDC
67       JENDM
68       J
69       000001...V2=1
70       000008...CRECT MAIN
71       GLOBL SORT
72       STACK:BLK 10R
73       ARFA:BLK 10
74       BUFRI:BLK 10R
75       INGK:BLK 5
76       00432 012706 START:MOV #STACK,SP
77       00436 010146 A1
78       00440 R11:CALL SORT
79       00440 004767 JSR PC,SORT
80       J
81       IF DF...V1
82       IF NB 0
83       ENDC
84
85       IF
86       IF NB #INGK
87       00444 012706 000429
88       112767 000001 000001
89       MOV #INBK,8R
90       ENDC
91       IF NB 0
92       00456 116710 000000
93       ENDC
94       IF NB
95       ENDC
96       IF NB
97       00E62 005060 000004
98       IFF
99       CLR 4,(0)
100      00462 104375 ENDC
101      EMT 0475
102      00470 005001 ENDC
103      CLR R1 1 BLOCK NUMBER
104      READ #AREA,#0,#BUFRI,#254,R1
105
106      IF DF...V1
107      IF NB #254.
ASEML, the 8K Assembler

```
108 .ENDC
109 MOV  \$256,,\%0
110
111 MOV  \$1,=\(6,)\%
112 MOV  \$BUF,=\(6,)\%
113 MOV  \$0,=\(6,)\%
114 EMT  \$0<200+\#AREA>
115 .IF NB \#AREA
116 0472 012700
117 0476 112760
118 000200*
119 000010
120 000001
121 .ENDC
122 .IF NB \#R1
123 0510 010160
124 000002
125 .ENDC
126 0514 012670
127 000220*
128 000004
129 .ENDC
12A 0522 012760
130 000200
131 000006
132 .ENDC
133 0530 012760
134 000001
135 000010
136 .ENDC
137 0536 104375
138 .ENDC
139 .HALT
140 0540 000000
141 .END
```

```
SYMBOL TABLE

A   000436R 002 AREA 000280R 002 R  004400R 002
BUF 000220R 002 INRLK 000420R 002 PC  0000007
R1  0000001 SP  0000006 SRT  0000000 G
STACK 000000R 002 START 000432R 002 \...V2 = 000001

ABS. 000000 000
000000 000
000000 001

MAIN 000542 002
ERRORS DETECTED: 0
FREE CORE: 1908B, WORDS

LPI=TEST, PAL
```
### 11.2 ASEMBL ERROR MESSAGES

The system error messages output for ASEMBL are abbreviated as follows:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>?BSW?</td>
<td>The switch specified was not recognized by the program.</td>
</tr>
<tr>
<td>?CORE?</td>
<td>There are too many symbols in the program being assembled. Try dividing program into separately assembled subprograms.</td>
</tr>
<tr>
<td>?NIF?</td>
<td>No input file was specified and there must be at least one input file.</td>
</tr>
<tr>
<td>?ODF?</td>
<td>No room to continue writing output; try to compress device with PIP.</td>
</tr>
<tr>
<td>?TMO?</td>
<td>Too many output files were specified.</td>
</tr>
</tbody>
</table>
CHAPTER 12

BATCH

12.1 INTRODUCTION TO RT-11 BATCH

RT-11 BATCH is a complete job control language that allows RT-11 to operate unattended. RT-11 BATCH processing is ideally suited to frequently run production jobs, large and long running programs, and programs that require little or no interaction with the user. Using BATCH, the user can prepare his job on any RT-11 input device and leave it for the operator to start and run.

RT-11 BATCH provides the ability to:

- Execute an RT-11 BATCH stream from any legal RT-11 input device.
- Output a log file to any legal RT-11 output device (except magtape or cassette).
- Execute the BATCH stream either with the Single-Job Monitor or in the background with a Foreground/Background Monitor.
- Generate and support system independent BATCH language jobs.
- Execute RT-11 monitor commands from the BATCH stream.

RT-11 BATCH consists of two main sections: the BATCH compiler and the BATCH run-time handler. The BATCH compiler reads the batch input stream created by the user, translates it into a format suitable for the RT-11 BATCH run-time handler, and stores it in a file. The BATCH run-time handler takes the file generated by the compiler and executes it in conjunction with the RT-11 monitor. As each command in the batch stream is executed, the command, along with any terminal output generated by executing the command, is output to the BATCH log device.

12.1.1 Hardware Requirements to Run BATCH

RT-11 BATCH can be run on any Single-Job system configured with at least 12K words of memory. A minimum system of 16K words of memory is required to run BATCH in a Foreground/Background environment. A line printer, although optional, is highly desirable as the log device.
12.1.2 Software Requirements to Run BATCH

BATCH uses certain RT-11 system programs to perform its operations. For example, the $BASIC command runs BASIC.SAV. The user must ensure that the following RT-11 programs are on the system device, with exactly the following names, before running BATCH.

- BASIC.SAV    (BASIC users only)
- BA.SYS
- BATCH.SAV    (MACRO users only)
- CREF.SAV     (FORTRAN users only)
- FORLIB.OBJ   (FORTRAN users only)
- FORTRA.SAV   (FORTRAN users only)
- LINK.SAV     (MACRO users only)
- MACRO.SAV    (MACRO users only)
- PIP.SAV

12.2 BATCH CONTROL STATEMENT FORMAT

Input to RT-11 BATCH is either a file generated using the RT-11 Editor and input from any legal RT-11 input device, or punched cards input from the card reader. In both cases, the input consists of BATCH control statements. A BATCH control statement consists of three fields, separated from one another with spaces: command fields, specification fields, and comment fields. The control statement has the form:

$command/switch specification/switch !comment

Each control statement requires a specific combination of command and specification fields and switches (see Section 12.4). Control statements may not be longer than 80 characters, excluding multiple spaces, tabs, and comments. A line continuation character (-) may be used to indicate that the control statement is continued on the next line (see Table 12-4). Even if the line continuation character is used, the maximum control statement length is still 80 characters.

The following example of a $FORTRAN command illustrates the various fields in a control statement.

$FORTRAN/LIST/RUN PROGA/LIBRARY PROGB/EXE !RUN FORTRAN,

command/switches spec fields/switches comment field

12.2.1 Command Fields

The command field in a BATCH control statement indicates the operation to be performed. It consists of a command name and certain command field switches. The command field is indicated by a $ in the first character position and is terminated by a space, tab, blank, or carriage return.

12.2.1.1 Command Names -- The command name must appear first in a BATCH control statement. All BATCH command names have a dollar sign ($) in the first position of the command, e.g., $JOB. No intervening
spaces are allowed in the command name. BATCH recognizes only two forms of a command name: the full name and an abbreviation consisting of $ and the first three characters of the command name. For example, the $FORTRAN command may be entered as:

$FORTRAN

or

$FOR

but cannot be entered as:

$FORT

or

$FORTR

12.2.1.2 Command Field Switches -- Switches that appear in a command field are command qualifiers and their functions apply to the entire control statement. All switch names must begin with a slash (/) that immediately follows the command name. Table 12-1 describes the command field switches that are legal in BATCH and indicates the commands on which they can be used. Those switch characters that appear in braces are optional. These switches will be mentioned again in the sections pertaining to the commands with which they can be used.

All /NO switches are the defaults, except the /WAIT switch in the $MOUNT and $DISMOUNT commands and the /OBJECT switch in the $LINK command.

Table 12-1
Command Field Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/BAN{NER}</td>
<td>Print header of job on the log file. This switch is allowed only on the $JOB command.</td>
</tr>
<tr>
<td>/NOBAN{NER}</td>
<td>Do not print a job header.</td>
</tr>
<tr>
<td>/CRE{F}</td>
<td>Produce a cross reference listing during compilation. This switch is allowed only on the $MACRO command.</td>
</tr>
<tr>
<td>/NOCRE{F}</td>
<td>Do not create a cross reference listing.</td>
</tr>
<tr>
<td>/DEL{ETE}</td>
<td>Delete input files after the operation is complete. This switch is allowed on the $COPY and $PRINT commands.</td>
</tr>
<tr>
<td>/NODEL{ETE}</td>
<td>Do not delete input files after operation is complete.</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 12-1 (cont.)
Command Field Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Function</th>
</tr>
</thead>
</table>
| /DOL[LARS] | The data following this command may have a $ in the first character position of a line. This switch is allowed on the $CREATE, $DATA, $FORTRAN, and $MACRO commands. Reading of the data is terminated by one of the following BATCH commands:  

$JOB  
$SEQUENCE  
$EOO  
$EOJ  

or by a physical end-of-file on the BATCH input stream. |
| /NODOL[LARS] | Following data may not have a $ in the first character position; a $ in the first character position signifies a BATCH control command. |
| /LIB[RARY] | Include the default library in the link operation. This switch is allowed on the $LINK and $MACRO commands. |
| /NOLIB[RARY] | Do not include the default library in the link operation. |
| /LIS[T] | Produce a temporary listing file (see Section 12.2.5) on the listing device (LST:) or write data images on the log device (LOG:). This switch is allowed on the $BASIC, $CREATE, $DATA, $FORTRAN, $JOB, and $MACRO commands. When used on the $JOB command, /LIST sends data lines in the job stream to the log device (LOG:). |
| /NOLIS[T] | Do not produce a temporary listing file. |
| /MAP | Produce a temporary linkage map on the listing device (LST:). This switch is allowed on the $FORTRAN, $LINK, and $MACRO commands. |
| /NOMAP | Do not create a MAP file. |
| /OBJ[ECT] | Produce a temporary object file as output of compilation or assembly (see Section 12.2.5). This switch is allowed on the $FORTRAN, $LINK, and $MACRO commands. When used on $LINK, this switch means that temporary files are to be included in the link operation. |
| /NOOBJ[ECT] | Do not produce object file as output of compilation, or, on $LINK, do not include temporary files in the link operation. |
| /RT11 | Set BATCH to operate in RT-11 mode (see Section 12.5). This switch is allowed only on the $JOB command. |
| /NORT11 | Do not set BATCH to operate in RT-11 mode. |

(continued on next page)
### Table 12-1 (cont.)
#### Command Field Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/RUN</td>
<td>Link (if necessary) and execute programs compiled since the last &quot;link-and-go&quot; operation or start of job. This switch is allowed on the $BASIC, $FORTRAN, $LINK, and $MACRO commands.</td>
</tr>
<tr>
<td>/NORUN</td>
<td>Do not execute or link and execute the program after performing the specified command.</td>
</tr>
<tr>
<td>/TIM(E)</td>
<td>Write the time of day to the log file when commands are executed. This switch is allowed only on the $JOB command.</td>
</tr>
<tr>
<td>/NOTIM(E)</td>
<td>Do not write time of day to log file.</td>
</tr>
<tr>
<td>/UNI{QUE}</td>
<td>Check for unique spelling of switches and keynames (see Section 12.4.13). This switch is allowed only on the $JOB command.</td>
</tr>
<tr>
<td>/NOUNI{QUE}</td>
<td>Do not check for unique spelling.</td>
</tr>
<tr>
<td>/WAI{T}</td>
<td>Pause to wait for operator action. This switch is allowed on the $DISMOUNT, $MESSAGE, and $MOUNT commands.</td>
</tr>
<tr>
<td>/NOWAI{T}</td>
<td>Do not pause for operator action.</td>
</tr>
<tr>
<td>/WRI{TE}</td>
<td>Indicate that the operator is to WRITE-ENABLE a specified device or volume. This switch is allowed only on the $MOUNT command.</td>
</tr>
<tr>
<td>/NOWRI{TE}</td>
<td>Indicate that no writes are allowed or that the specified volume is read-only; the operator is informed and must WRITE-LOCK the appropriate device.</td>
</tr>
</tbody>
</table>

### 12.2.2 Specification Fields

Specification fields immediately follow command fields in a BATCH control statement and are used to name the devices and files involved in the command. These fields are separated from the command field, and from each other, by blanks or spaces.

If a specification field contains more than one file to be used in the same operation, the + operator is used between these files. For example, to assemble files F1 and F2 to produce an object file F3 and a temporary listing file, type:

\[ \text{$MACRO/LIST F1+F2/SOURCE F3/OBJECT} \]

If a command is to be repeated for more than one field specification, the ""," operator is used between these files. For example, the following command assembles F1 to produce F2, a temporary listing file, and a map file F3. It then assembles F4 and F5 to produce F6 and a temporary listing file.

\[ \text{$MACRO/LIST F1/SOURCE F2/OBJECT F3/MAP,F4+F5/SOURCE- F6/OBJECT} \]
Note that the command field switches apply to the entire line, but the specification field switches apply only to the field they follow.

Depending on the command being used, specification fields may contain a device specification, file specification, or an arbitrary ASCII string. Any of these three may be followed by an appropriate specification field switch (see Table 12-3).

12.2.2.1 Physical Device Names -- Each device in an RT-11 BATCH specification field is referenced by means of a standard 2- or 3-character device name. Table 2-2 in Chapter 2 lists each name and its related device. If no unit number is specified for devices which have more than one unit, unit 0 is assumed.

In addition to the permanent names shown in Table 2-2, devices can be assigned logical device names. A logical device name takes precedence over a physical name and thus provides device independence. With this feature, a program that is coded to use a specific device does not need to be rewritten if the device is unavailable. For example, DK: is normally assigned to the system device, but that name can be assigned to DECtape unit 0 with an RT-11 monitor ASSIGN command.

Certain logical names must be assigned prior to running any BATCH job since BATCH uses them as default devices. These devices are:

- **LOG**: BATCH log device (cannot be magtape or cassette)
- **LST**: default for listing files generated by BATCH stream

The following are not legal device names in RT-11; if used, the operator must assign them as logical names with the ASSIGN command. These names are included here because they can be used in BATCH streams written for other DIGITAL systems.

- **DF**: Fixed-head disk (RF).
- **LL**: Line printer with upper and lower case characters.
- **M7**: 7-track magtape.
- **M9**: 9-track magtape.
- **PS**: Public storage (DK: as assigned by RT-11).

Refer to Sections 2.7.2.4 and 12.7.1 for instructions on assigning logical names to devices.

12.2.2.2 File Specifications -- Files are referenced symbolically in a BATCH control statement by a name of up to six alphanumeric characters followed, optionally, by a period and an extension of three alphanumeric characters. The extension to a file name generally indicates the format of a file. In most cases, it is a good practice to conform to the standard file name extensions for RT-11 BATCH. If an extension is not specified for an output file, BATCH and most other RT-11 system programs assign appropriate default extensions. If an extension for an input file is not specified, the system searches for that file name with a default extension. Table 12-2 lists the standard extensions used in RT-11 BATCH.
Table 12-2
File Name Extensions

<table>
<thead>
<tr>
<th>Extension</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.BAS</td>
<td>BASIC source file (BASIC input).</td>
</tr>
<tr>
<td>.BAT</td>
<td>BATCH command file.</td>
</tr>
<tr>
<td>.CTL</td>
<td>BATCH control file generated by the BATCH compiler.</td>
</tr>
<tr>
<td>.CTT</td>
<td>BATCH temporary file generated by the BATCH compiler.</td>
</tr>
<tr>
<td>.DAT</td>
<td>BASIC or FORTRAN data file.</td>
</tr>
<tr>
<td>.DIR</td>
<td>Directory listing file.</td>
</tr>
<tr>
<td>.FOR</td>
<td>FORTRAN IV source file (FORTRAN input).</td>
</tr>
<tr>
<td>.LST</td>
<td>Listing file.</td>
</tr>
<tr>
<td>.LOG</td>
<td>BATCH log file.</td>
</tr>
<tr>
<td>.MAC</td>
<td>MACRO or EXPAND source file (MACRO, EXPAND, SRCCOM input).</td>
</tr>
<tr>
<td>.MAP</td>
<td>Linkage map output from $LINK operation.</td>
</tr>
<tr>
<td>.OBJ</td>
<td>Object file, output from compilation or assembly.</td>
</tr>
<tr>
<td>.SOU</td>
<td>Temporary source file.</td>
</tr>
<tr>
<td>.SAV</td>
<td>$RUNable file or program image output from $LINK.</td>
</tr>
</tbody>
</table>

12.2.2.3 Wild Card Construction -- The wild card construction means that the file name or extension in certain BATCH control statements (i.e., $COPY, $CREATE, $DELETE, $DIRECTORY, $PRINT) may be replaced totally with an asterisk (*). The asterisk is used as a wild card to designate the entire file name or extension. See Chapter 4, Section 4.1.1, for a complete description of the wild card construction.

12.2.2.4 Specification Field Switches -- Specification field switches follow file specifications in a BATCH control statement and designate how the file will be used. These switches apply only to the field in which they appear. Switch names begin with a slash. The specification field switches legal in RT-11 BATCH are listed in Table 12-3. Optional characters in the switch names are shown in braces.
Table 12-3  
Specification Field Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>/BAS(IC)</td>
<td>BASIC source file.</td>
</tr>
<tr>
<td>/EXE CUTABLE</td>
<td>Indicates the runnable program image file to be created as the result of a link operation.</td>
</tr>
<tr>
<td>/FOR TRAN</td>
<td>FORTRAN source file.</td>
</tr>
<tr>
<td>/INP UT</td>
<td>Input file; default if no switches are specified.</td>
</tr>
<tr>
<td>/LIB RARY</td>
<td>Library file to be included in link operation (prior to default library).</td>
</tr>
<tr>
<td>/LIS T</td>
<td>Listing file.</td>
</tr>
<tr>
<td>/LOG ICAL</td>
<td>Indicates that the device is a logical device name; used in $DISMOUNT and $MOUNT commands.</td>
</tr>
<tr>
<td>/MAC RO</td>
<td>MACRO or EXPAND source file.</td>
</tr>
<tr>
<td>/MAP</td>
<td>Linker map file.</td>
</tr>
<tr>
<td>/OBJ ECT</td>
<td>Object file (output of assembly or compilation).</td>
</tr>
<tr>
<td>/OUT PUT</td>
<td>Output file.</td>
</tr>
<tr>
<td>/PHY SICAL</td>
<td>Indicates physical device name.</td>
</tr>
<tr>
<td>/SOU RCE</td>
<td>Indicates source file.</td>
</tr>
<tr>
<td>/VID</td>
<td>Volume identification.</td>
</tr>
</tbody>
</table>

12.2.3 Comment Fields

Comment fields, used to document a BATCH stream, are identified by an exclamation point (!) appearing anywhere except the first character position in the control statement. Any character following the ! and preceding the carriage return/line feed combination is treated as a comment and is ignored by the BATCH processor. For example, the following command:

```
$RUN PIP !DELETE FILES ON DK:
```

runs the RT-11 system program PIP; the comment is ignored.

Comments can also be included as separate comment lines by typing a $ in character position 1, followed immediately by the ! operator and the comment, e.g.,

```
$!DELETE FILES ON DK:
```

12.2.4 BATCH Character Set

The RT-11 BATCH character set is limited to the 64 upper-case characters (i.e., ASCII 40 through 137). The current ASCII set is assumed (i.e., character 137 is underscore and not left arrow, and
character 136 is circumflex, not up-arrow). No control characters other than tab, carriage return, and line feed are supported by the BATCH job control language.

Table 12-4 details the way in which BATCH normally interprets certain characters. Character interpretations are different when RT-11 mode is used (see Section 12.5).

<table>
<thead>
<tr>
<th>Character</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank/space</td>
<td>Specification field delimiter. Separates arguments in control statements. Any string of consecutive spaces and tabs (except in quoted strings) is considered a blank and is equivalent to a single space.</td>
</tr>
<tr>
<td>!</td>
<td>Comment delimiter. All characters after the exclamation point are ignored by the input routine, up to the carriage return/line feed.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Used to pass a text string containing delimiting characters where the normal precedence rules would create the wrong action, e.g., to include a space in a volume identification (/VID).</td>
</tr>
<tr>
<td>$</td>
<td>BATCH control statement recognition character. A dollar sign ($) in the first character position of a BATCH input stream line indicates that the line is a control statement.</td>
</tr>
<tr>
<td>.</td>
<td>Delimiter for file extension (type).</td>
</tr>
<tr>
<td>-</td>
<td>Indicates line continuation if the character after the hyphen is one of the following:</td>
</tr>
<tr>
<td>* a carriage return/line feed</td>
<td></td>
</tr>
<tr>
<td>* any number of spaces followed by a carriage return/line feed</td>
<td></td>
</tr>
<tr>
<td>* a comment delimiter (!)</td>
<td></td>
</tr>
<tr>
<td>* spaces followed by a comment delimiter (!)</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>If any other character follows the hyphen, the hyphen is assumed to be a minus sign indicating a negative value in a switch.</td>
</tr>
<tr>
<td>0-9</td>
<td>Start of a switch name. Must be followed immediately by an alphanumeric string.</td>
</tr>
<tr>
<td>:</td>
<td>Immediately follows a device name. Also can be used to separate a switch name from its value or to separate a switch value from its subvalue (: can be used interchangeably with = for this).</td>
</tr>
<tr>
<td>A-Z</td>
<td>Alphabetic string components.</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Character</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Separates switch name from value.</td>
</tr>
<tr>
<td>\</td>
<td>Illegal character except when preceding a directive to the BATCH run-time handler from the operator. (To include \ in an RT-11 mode command, use \ \ .)</td>
</tr>
<tr>
<td>+</td>
<td>Delimiter separating multiple files in a single specification field. Also used to indicate a positive value in switches.</td>
</tr>
<tr>
<td>,</td>
<td>The comma (,) is used to separate sets of arguments for which the command is to be repeated.</td>
</tr>
<tr>
<td>*</td>
<td>The * is used as a wild card in utility command file specifications.</td>
</tr>
<tr>
<td>CR/LF</td>
<td>Carriage return/line feed. Indicates end-of-line (or end of logical record) for records in the BATCH input stream.</td>
</tr>
</tbody>
</table>

**12.2.5 Temporary Files**

When field specifications are not included in a BATCH command line, BATCH sometimes generates temporary files. For example, a $FORTRAN command which is followed in the BATCH stream by the FORTRAN source program could be entered as:

```
$FORTRAN/RUN/OBJECT/LIST
FORTRAN source program
$EOD
```

This command generates a temporary source file from the source statements that follow, a temporary object file, a temporary listing file, and a temporary save image file.

BATCH sends temporary files to the default device (DK:) or the listing device (LST:) according to their nature. If the device is file-structured, BATCH assigns file names and extensions as follows:

- `nnnmnm.LST` for temporary listing files (sent to LST:)
- `nnnmnm.MAP` for temporary map files (sent to LST:)
- `nnnppp.OBJ` for temporary object files (sent to DK:)
- `nnnppp.SAV` for temporary save image files (sent to DK:)
- `nnnppp.SOU` for temporary source files (sent to DK:)

where:

- `nnn` is the last three digits of the sequence number assigned to the job by the $SEQUENCE command (see Section 12.4.22). Thus, a sequence number of 12345 produces a file name beginning 345. If no $SEQUENCE command is used, nnn is set to 000.

January 1976 12-10
is the mth listing (or map) file since the BATCH run-time handler (BA.SYS) was loaded. The first such file, listing or map, is 000. Each time a new temporary file is generated, the file name is incremented by 1. Thus, the second listing file produced under job sequence number 12345 is 345001.LST, and the first map file produced is 345000.MAP.

is the pth object, save image, or source file in the current BATCH run. The first such file (object, save image, or source) is 000. Each time a new temporary file is generated, the file name is incremented by 1. These file names are reset to 000 every time that BATCH is run and after every $LINK, $MACRO, or $FORTRAN command that uses the temporaries.

12.3 General Rules and Conventions

The following general rules and conventions are associated with RT-11 BATCH processing.

1. A dollar sign ($) is always in the first character position of a command line.

2. Each job must have a $JOB and $EOJ command (or card).

3. Command and switch names can be spelled out entirely or the first three characters of the command and required characters of the switch can be specified.

4. Wild card construction (*) can be specified only for the utility commands: $COPY, $CREATE, $DELETE, $DIRECTORY, and $PRINT.

5. Comments can be included at the end of command lines or in a separate comment line. When comments are included in a command line, they must follow the command and be preceded by an exclamation mark.

6. Only 80 characters per control statement (card record) are allowed, excluding multiple spaces, tabs, and comments.

7. When file specifications are omitted from BATCH commands and data is supplied in the BATCH stream, the system creates a temporary file with a default name (see Section 12.2.5).

8. The RT-11 monitor type-ahead feature is restricted to BATCH handler directives (see Section 12.7.3) to be inserted into a BATCH program. No other terminal input (except to the foreground) can be entered while a BATCH stream is executing.
12.4 BATCH COMMANDS

BATCH commands are placed in the input stream to indicate to the system which functions to perform in the job. All BATCH commands have a dollar sign ($) in the first character position of the command, e.g., $JOB. Intervening spaces are not allowed in command names. The command name must always start in the first character position of the line (card column 1).

BATCH commands are presented in alphabetical order in this chapter for ease of reference. However, a user who is unfamiliar with BATCH may prefer to read the commands in a functional order as detailed in Table 12-5. The characters shown in braces are optional.

Table 12-5
BATCH Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Section</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SEQ{ENCE}</td>
<td>12.4.22</td>
<td>Assigns an arbitrary identification number to a job.</td>
</tr>
<tr>
<td>$JOB</td>
<td>12.4.13</td>
<td>Indicates the start of a job.</td>
</tr>
<tr>
<td>$EOJ</td>
<td>12.4.11</td>
<td>Indicates the end of a job.</td>
</tr>
<tr>
<td>$MOU{NT}</td>
<td>12.4.18</td>
<td>Signals the operator to mount a volume on a device and optionally assigns a logical device name.</td>
</tr>
<tr>
<td>$DIS{MOUNT}</td>
<td>12.4.9</td>
<td>Signals the operator to dismount a volume from a device and deassigns a logical device name.</td>
</tr>
<tr>
<td>$FOR{TRAN}</td>
<td>12.4.12</td>
<td>Compiles a FORTRAN source program.</td>
</tr>
<tr>
<td>$BAS{IC}</td>
<td>12.4.1</td>
<td>Compiles a BASIC source program.</td>
</tr>
<tr>
<td>$MAC{RO}</td>
<td>12.4.16</td>
<td>Assembles a MACRO source program.</td>
</tr>
<tr>
<td>$LIB{RARY}</td>
<td>12.4.14</td>
<td>Specifies libraries that are to be used in linkage operations.</td>
</tr>
<tr>
<td>$LIN{K}</td>
<td>12.4.15</td>
<td>Links modules for execution.</td>
</tr>
<tr>
<td>$RUN</td>
<td>12.4.21</td>
<td>Causes a program to execute.</td>
</tr>
<tr>
<td>$CAL{L}</td>
<td>12.4.2</td>
<td>Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.</td>
</tr>
<tr>
<td>$CHA{IN}</td>
<td>12.4.3</td>
<td>Passes control to another BATCH file.</td>
</tr>
<tr>
<td>$DAT{A}</td>
<td>12.4.6</td>
<td>Indicates the start of data.</td>
</tr>
<tr>
<td>$EOD</td>
<td>12.4.10</td>
<td>Indicates the end of data.</td>
</tr>
<tr>
<td>$MES{AGE}</td>
<td>12.4.17</td>
<td>Issues a message to the operator.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 12-5 (cont.)
BATCH Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Section</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$COP{Y}</td>
<td>12.4.4</td>
<td>Copies files.</td>
</tr>
<tr>
<td>$CRE{ATE}</td>
<td>12.4.5</td>
<td>Creates new files from data included in BATCH stream.</td>
</tr>
<tr>
<td>$DEL{ETE}</td>
<td>12.4.7</td>
<td>Deletes files.</td>
</tr>
<tr>
<td>$DIR{ECTORY}</td>
<td>12.4.8</td>
<td>Provides a directory of the specified device.</td>
</tr>
<tr>
<td>$PRI{NT}</td>
<td>12.4.19</td>
<td>Prints files.</td>
</tr>
<tr>
<td>$RT1{1}</td>
<td>12.4.20</td>
<td>Specifies that the following lines are RT-11 mode commands.</td>
</tr>
</tbody>
</table>

12.4.1 $BASIC Command

The $BASIC command calls RT-11 Single-User BASIC to execute a BASIC source program. The $BASIC command has the following format:

```
$BASIC[/switch] {dev:filnam.ext/sw} {!comments}
```

where:

- `/switch` indicates the switches that can be appended to the $BASIC command. The switches are as follows:
  - `/RUN` indicates that the source program is to be executed.
  - `/NORUN` indicates that the program is to be compiled only; error messages are sent to the log file.
  - `/LIST` writes data images that are contained in the job stream to the log file (LOG:).
  - `/NOLIST` writes data images to the log file only if $JOB/LIST was specified.

- `dev:filnam.ext` indicates the name of the source file and the device on which it resides. If `dev:` is omitted, DK: is assumed. If `ext` is omitted, the extension .BAS is assumed. If this specification is omitted, the
source statements must immediately follow the $BASIC command in the input stream.

The source program included after a $BASIC statement can be terminated either by a $EOD command or by any other BATCH command that starts with a $ in the first position.

/sw indicates the switches that may follow the source file name. Any file name with no switch appended is assumed to be the name of a source file. This switch can have one of the following values or can be omitted.

/BASIC indicates that the file name specified is a BASIC source program.
/SOURCE performs the same function as /BASIC.
/INPUT performs the same function as /BASIC.

The $BASIC command can be followed by the source program, by legal BASIC commands such as RUN, and by data. The following two BATCH streams, for example, produce the same results.

```
#BASIC
10 INPUT A
20 PRINT A
30 END
RUN
123
$EOD

#BASIC/RUN
10 INPUT A
20 PRINT A
30 END
#DATA
123
$EOD
```

12.4.2 $CALL Command

The $CALL command transfers control to another BATCH control file, temporarily suspending execution of the current control file. The CALLeD file is executed until $EOJ is reached or until the job aborts; control then returns to the statement following the $CALL in the originating BATCH control file. Calls can be nested up to 31 levels. The log file for the CALLeD BATCH file is included in the log file for the originating BATCH program.

The format of the $CALL command is:

```
$CALL dev:filnam.ext {!comments}
```

No switches are allowed in the $CALL command. $JOB command switches are saved across a $CALL but do not apply to the called BATCH file.
If .CTL is specified as the file extension, a precompiled BATCH control file is assumed. If no .ext is specified, .BAT is assumed, and the called BATCH stream is compiled before execution.

Please note that if the program called generates temporary files, those files may supersede currently existing temporary files if the two jobs have the same sequence number. For example, consider the following two BATCH streams:

\[
\begin{align*}
&\$\FOR/OBJ~A \\
&\$\FOR/OBJ~B \\
&\$\LINK/\RUN
\end{align*}
\]

\[
\begin{align*}
&\$\FOR/OBJ~A \\
&\$\CALL\ C \\
&\$\FOR/OBJ~B \\
&\$\LINK/\RUN
\end{align*}
\]

The called BATCH file (C.BAT) contains the following:

\[
\begin{align*}
&\$\JOB \\
&\$\FOR/OBJ~A1 \\
&\$\FOR/OBJ~B1 \\
&\$\LINK/\RUN \\
&\$\EOJ
\end{align*}
\]

The temporary object files generated by C.BAT would change the behavior of the above two BATCH statement sequences since the first temporary file created by C.BAT (000000.OBJ) would supersede the temporary file produced by the first $FORTRAN command (000000.OBJ). This could be prevented by giving the BATCH job C.BAT a unique sequence number (see Section 12.4.22).

12.4.3 $CHAIN Command

The $CHAIN command transfers control to a named BATCH control file but does not return to the input stream which executed the $CHAIN command. The format of the $CHAIN command is:

\[
\text{\$CHAIN dev:filnam.ext \{licomments\}}
\]

No switches are allowed in the $CHAIN command. If .CTL is specified as the file extension, a precompiled BATCH control file is assumed. If no .ext is specified, .BAT is assumed, and the chained BATCH stream is compiled before execution.

An $EOJ command should always follow the $CHAIN command in the BATCH stream.
NOTE

The values of BATCH run-time variables remain constant across a $CALL, $CHAIN, or return from call. See Section 12.5.2.2 for a description of these variables.

The $CHAIN command is useful for transferring control to programs that need only be run once at the end of a BATCH stream. For example, the following BATCH program (PRINT.BAT) could be used to print and then delete all temporary listing files generated during the current BATCH job.

```
$JOB !PRINT ALL LIST FILES
$PRINT/DELETE *.LST
$EOJ

PRINT.BAT can then be run with the $CHAIN command, e.g.,

$JOB
$MACRO/RUN/LIST FILE1,FILE2,FILE3
$CHAIN PRINT
$EOJ
```

$COPY

12.4.4 $COPY Command

The $COPY command copies files in image mode from one device to another. The wild card construction (see Section 12.2.2.3) can be used in the input and output file specifications. More than one input file can be concatenated to form one output file so long as the output specification does not contain a wild card. The $COPY command has the following format.

```
$COPY[/switch] dev:filnam.ext/OUTPUT dev:filnam1.ext[/INPUT]-
{comments}
```

where:

/switch indicates switches that can be appended to the $COPY command.

/DELETE indicates that input files are to be deleted after the copy operation.

/NODELETE indicates that input files are not to be deleted after the copy operation.
dev: indicates the device containing the files to be copied for the input portion of the command or the device to which the files are to be copied for the output portion of the command. If dev: is not specified, DK: is assumed.

filnam indicates the name to be assigned to the output file; a wild card may be used instead of an explicit file name. Additional output files can be specified if they are separated from each other with commas.

ext indicates the file extension and must be specified. For input files, a wild card can be used instead of an explicit extension. For output files, wild cards can be used so long as no concatenation is specified.

/OUTPUT is appended to a file specification to indicate that it is for the output file.

filnam1 specifies the name of the input file. Wild cards can be used instead of an explicit file name. Additional input files can be specified if they are separated from each other with commas. Files are copied to the output file in the order specified.

/INPUT is appended to the input file specifier(s). The system assumes input if no switch is used.

The following are examples of the $COPY command:

$COPY *.BAS/OUTPUT DT1:*.*.BAS

The above command copies all files with the extension .BAS from the DECTape on unit 1 to the default storage device DK.

$COPY FILE2.FOR/OUTPUT FILE0.FOR+FILE1.FOR

The above command merges the input files FILE0.FOR and FILE1.FOR to form one file called FILE2.FOR and stores FILE2.FOR on device DK.

$COPY FILE2.FOR+FILE3.FOR/OUTPUT FILE0.FOR+FILE1.FOR

The above command copies FILE0.FOR to DK: as FILE2.FOR and FILE1.FOR as FILE3.FOR.

$COPY */.*/OUT DT0:*.FOR,DT1:*.*/OUT DT0:*.*

The above command copies all files with the extension .FOR from DT0: to DK: and all files on DT0: to DT1:.
$CREATE

12.4.5 $CREATE Command

The $CREATE command generates a file consisting of data records from data that follows the $CREATE command in the input stream. An error occurs if the data does not immediately follow the $CREATE command. A $DATA command must not precede the data records.

The data associated with $CREATE can be followed by a $EOD command to signify the end of data or any other BATCH control statement can be used to indicate end of data and initiate a new function. The $CREATE command has the following format:

$CREATE[/switch] dev:filnam.ext {!comments}

where:

/switch indicates switches that can be appended to the $CREATE command.
/DOLLARS indicates that the data following this command may have a $ in the first character position of a line.
/NODOLLARS indicates that a $ may not be in the first character position of a line.
/LIST writes data image lines to log file.
/NOLIST does not write data image lines to log file. If $JOB/LIST was specified, this switch is ignored.

dev: device on which the file is to reside; DK: if not specified.
filnam indicates the name to be assigned to the file. The file name must be specified.
ext indicates the file extension. If extension is omitted, filnam must be followed by a period.

The following is an example of the $CREATE command:

$CREATE/LIST PROG.FOR
    FORTRAN source file

$EOD

The data records following the $CREATE command become a new file (PROG.FOR) on the default device (DK:) and a listing is generated on logical device LOG:
12.4.6 $DATA Command

The $DATA command is used to include data records in the input stream. No file name is associated with the data; the data is transferred to the appropriate program as though input from the console terminal. For example, the $RUN command for a particular program can be followed by a $DATA command and the data records to be processed by the program. The data records must be valid data for the program that is to use them.

The $DATA command has the following format:

$DATA[/switch] {!comments}

Four switches can be used with the $DATA command.

/DOLLARS indicates that the data following this command may have a $ in the first character position of a line.
/NODOLLARS indicates that a $ may not be in the first character position of a line.
/LIST writes data image lines to the log file.
/NOLIST does not write data image lines to the log file. If $JOB/LIST was specified, this switch is ignored.

An $EOD command normally follows the last data record. However, any other BATCH command may also signal the end of the data so long as $DATA/DOLLARS is not specified (see Table 12-1).

The following example shows data being entered into a BASIC program (TEST1.BAS).

```
#BASIC/RUN TEST1.BAS
#DATA
25.75,125.146
180.210,520.874
$EOD
```
12.4.7 $DELETE Command

The $DELETE command is used to delete files from the specified device. This command has the form:

```
$DELETE dev:filnam1.ext{,dev:filnam2.ext,...,dev:filnamn.ext} -
```

where filnam1 through filnamn are the names of the files to be deleted. The file name and extension are required. Wild cards can be used in the file name and extension positions.

The following example deletes all files named TEST1 on the default device (DK:).

```
$DELETE TEST1.*
```

The following example deletes all files with .FOR extensions on DT1; then deletes all files with .MAC extensions on DK:

```
$DELETE DT1:*.*.FOR:*.*.MAC
```

12.4.8 $DIRECTORY Command

The $DIRECTORY command outputs a directory of the specified device to a listing file. If no listing file is specified, the listing goes to the BATCH log file. Wild cards can be used in the specification fields. This command has the form:

```
$DIRECTORY {dev:filnam.ext/LIST} {dev:filnam.ext}[/INPUT] -
```

12-20
BATCH

where:

/LIST indicates name of directory listing file.

/INPUT indicates input files to be included in directory (default).

The following are examples of the $DIRECTORY command:

$DIRECTORY

The above command outputs a directory of the device DK: to the BATCH log file.

$DIRECTORY FOR DIR/List *.FOR

The above command creates a directory file (FOR.DIR) on the device DK:. The directory contains the names, lengths, and dates of creation of all FORTRAN source files on the device DK:.

12.4.9 $DISMOUNT Command

The $DISMOUNT command removes the logical device name assigned by a $MOUNT command. When $DISMOUNT is encountered during the execution of a job, the entire $DISMOUNT command line is printed on the console terminal to inform the operator of the specific device to unload. This command has the form:

$DISMOUNT[/switch] ldn: [/LOGICAL] ![comments]

where:

/switch indicates the switches that can be appended to the $DISMOUNT command.

/WAIT indicates that the job is to pause until the operator enters a response. If neither /WAIT nor /NOWAIT is specified, /WAIT is assumed. BATCH rings a bell at the terminal, prints the physical device name to be dismounted and a ?, and waits for a response. (Input to the BATCH handler can be entered, see Section 12.7.3.)

/NOWAIT does not pause for operator response. BATCH prints the physical device name to be dismounted.
BATCH

ldn: is the logical device name to be deassigned from the physical device.

/LOGICAL identifies the device specification as a logical device name.

The following example instructs the operator to dismount the physical device with the logical device name OUT and removes the logical assignment of device OUT. In this case, OUT is DT0. The operator dismounts DT0 and types a carriage return.

$DISMOUNT/WAIT OUT:/LOGICAL DT0?

$EOD

12.4.10 $EOD Command

The $EOD command indicates the end of data record or the end of a source program in the job stream. The format of this command is:

$EOD {!comments}

The $EOD command can signal the end of data associated with any of the following commands:

$BASIC
$CREATE
$DATA
$FORTRAN
$MACRO

The $EOD command in the following example indicates the end of a source program that is to be compiled, linked, and executed.

$FORTRAN/RUN
  source program

$EOD
12.4.11 $EOJ Command

The $EOJ command indicates the end of a job. This command must be the last statement in every BATCH job. The command has the following format:

$EOJ {!comments}

If a $JOB command, a $SEQUENCE command, or a physical end-of-file is encountered in the input stream before $EOJ, the error message NO $EOJ appears in the log file.

12.4.12 $FORTRAN Command

The $FORTRAN command calls the FORTRAN compiler to compile a source program. Optionally, this command can provide printed listings or list files and may produce a linkage map in the listing. The $FORTRAN command has the following format:

$FORTRAN[/switch] {dev:filnam1.ext/sw} {dev:filnam2.ext/OBJECT}-
{dev:filnam3.ext/LIST} {dev:filnam4.ext/EXECUTE}-
{dev:filnam5.ext/MAP} {dev:filnam6.ext/LIBRARY} {!comments}

where:

/switch indicates the switches that can be appended to the $FORTRAN command. The switches are as follows:

/RUN indicates that the source program is to be compiled, linked with the default library (initially FORLIB.OBJ, may be reset with the $LIBRARY command), and executed.

12-23
/NORUN indicates that the program is to be compiled only.

/OBJECT indicates that a temporary object file is to be produced.

/NOOBJECT indicates that a temporary object file is not to be produced.

/LIST indicates that a list file is to be produced on the listing device (LST:).

/NOLIST indicates that a list file is not to be produced.

/MAP produces a linkage map on the listing device (LST:).

/NOMAP does not create MAP file.

/DOLLARS indicates that the data following this command may have a $ in the first character position of a line.

/NODOLLARS indicates that a $ may not be in the first character position of a line.

devoilnam1.ext indicates the device, file name, and extension of the FORTRAN source file. If filnam1 is not specified, the $FORTRAN source statements must immediately follow the $FORTRAN command in the input stream; BATCH generates a temporary source file that is deleted after it is compiled (see Section 12.2.5).

The source program included after a $FORTRAN statement can be terminated either by a $EOD command or by any other BATCH command so long as $FORTRAN/DOLLARS is not specified (see Table 12-1). A BATCH command is one that starts with a $ in the first position.

/sw can have one of the following values or can be omitted:

/FORTRAN indicates that the file name specified is a FORTRAN source program. Any file name with no switch appended is assumed to be the name of a source file.

/SOURCE performs the same function as /FORTRAN.

/INPUT performs the same function as /FORTRAN.

devoilnam2.ext/OBJECT indicates the device, file name, and extension of the object file produced by compilation. The object file remains on the specified device after the job finishes. The object file specification, if included, must be followed by the /OBJECT switch.

If the object file specification is omitted but $FORTRAN/OBJECT is specified, a temporary object file is created, included in any $LINK operations that follow it in the job, and deleted after the link operation.

devoilnam3.ext/LIST indicates the name to be assigned to the list file created by the compiler. The list file is not printed automatically if LST: is assigned to
file-structured device, but it can be listed using the $PRINT command. The list file specification must be followed by the /LIST switch.

`dev:filnam4.ext/EXECUTE` indicates the name to be assigned to a save image file. The save image file specification must be followed by the /EXECUTE switch. If this field is not included, BATCH generates a temporary save image file (see Section 12.2.5) and then deletes the temporary file.

`dev:filnam5.ext/MAP` indicates the name to be assigned to the linkage map file created by the Linker. The map specification must be followed by the /MAP switch.

`dev:filnam6.ext/LIBRARY` indicates that the specified file is to be included in the link procedure as a library before FORLIB.OBJ. The file must be a library file (produced by RT-11 LIBR). The library specification must be followed by the /LIBRARY switch.

The following are examples of $FORTRAN commands:

```
$FORTRAN/RUN PROGA.FOR
```

The above command calls FORTRAN to compile a source program named PROGA.FOR. The program is compiled and executed.

```
$FORTRAN/N0OBJ/LIST
   source program
```

```
$E0D
```

The above command sequence compiles the FORTRAN program but does not produce an object file. A temporary listing file is created on LST:.

```
$JOB
```

12.4.13 $JOB Command

The $JOB command indicates the beginning of a job. Each job must have its own $JOB command. This command has the following format:

```
$JOB{[/switch]{[/switch2]{[/switchn} {!comments}}
```
The switches allowed in the $JOB command are:

/BANNER    print header (a repetition of the $JOB command) on the log file.
/NOBANNER  do not print job header.
/LIST      write data image lines that are contained in the job stream to the log file.
/NOLIST    write data image lines to the log file only when a /LIST switch exists on a $BASIC, $CREATE, or $DATA command that has data lines following it.
/RT11      if no $ appears in column 1 when one is expected, assume that the line or card is an RT-11 mode command (see Section 12.5).
/NORT11    do not process RT-11 mode commands.
/TIME      write the time of day to the log file when command lines are executed (see NOTE on following page).
/NOTIME    do not write time of day.
/UNIQUE    check for unique spelling of switches and keynames. When this switch is used, commands and switches may be abbreviated to the least number of characters that still make their names unique. For example, the /DOLLARS switch can be abbreviated to /DO since no other switches begin with the characters DO.
/NOUNIQUE  check only for normal switch and keyname spellings.

Each job must be ended with a $EOJ command if it is to be run. If an input stream consists of more than one job, BATCH automatically terminates one job when the $JOB command for the next job is encountered. A job terminated with another $JOB command will never be run; an error message (NO $EOJ) will appear in the log.

The following $JOB command specifies that the time of day be written to the log file before each BATCH command beginning with a $ is executed and that unique abbreviations of BATCH commands and switches be accepted.

$JOB/TIME/UNIQUE
NOTE

If the /TIME switch is used on the $JOB command, the $DATA command cannot be used. For example, this job will not run properly:

```
$JOB/TIME
$RUN PROG
$DATA
 123
$EOD
$EOJ
```

The /TIME switch uses the KMON TIME command to print the current time on the log for each BATCH command, including $DATA, causing an abort of the program that was to use the data. To avoid the problem, use RT-ll mode:

```
$JOB/TIME
$RT11
:R PROG
 123
$E0J
```
This page intentionally blank.
12.4.14 $LIBRARY Command

The $LIBRARY command allows the user to specify a list of library files that will be included in FORTRAN links or with other linkage operations that specify the /LIBRARY switch. By default, the list of libraries contains only FORLIB.OBJ, the RT-ll FORTRAN library. This command has the form:

```
$LIBRARY mylib {!comments}
```

or

```
$LIBRARY mylib+FORLIB {!comments}
```

where:

FORLIB is the RT-ll FORTRAN library and mylib is a user library. Libraries are linked in order of their appearance in the $LIBRARY command.

The following example shows two user libraries (LIB1.OBJ and LIB2.OBJ) to be included in FORTRAN links before FORLIB.OBJ.

```
$LIBRARY LIB1.OBJ+LIB2.OBJ+FORLIB.OBJ
```

12.4.15 $LINK Command

The $LINK command is used to produce save image files from object files. This command links the specified files (if any) with all temporary object files created since the last link or "link-and-go" operation (if any).
Temporary object files are those created as a result of a $FORTRAN or $MACRO command in which object files were neither specifically named by using the /OBJECT switch nor suppressed by using the /NOOBJECT switch. Permanent object files are created by using the /OBJECT switch on a $FORTRAN or $MACRO file descriptor.

Files are linked in the following order:

1. First, temporary files are linked in the order in which they were compiled.

2. Then, permanent files are linked in the order in which they are specified in the $LINK command.

3. If a library is specified in the $LINK command, it is linked next, providing that unresolved references remain.

4. If $LINK/LIBRARY is specified, the default library list is searched and linked.

The format for this command is:

$LINK[/switch] {filnam1.ext/OBJECT} {filnam2.ext/LIBRARY} {filnam3.ext/MAP} {filnam4.ext/EXECUTE} {!comments}

where:

switch indicates the switches that can be appended to the $LINK command. The switches are as follows:

/LIBRARY indicates that the FORTRAN library (FORLIB.OBJ) and any default libraries specified in the $LIBRARY command are to be included in this $LINK operation. This switch is normally used when the files being linked do not include any temporary FORTRAN object files or when $FORTRAN was specified without the /RUN or /MAP switch but the default library list is to be searched for unresolved references.

/NOLIBRARY Indicates that the default libraries are not to be included.

/MAP produces a temporary load map on the listing device (LST:).

/NOMAP indicates that a map file is not to be produced.

/OBJECT indicates that temporary object files are to be included in the link. If neither /OBJECT nor /NOOBJECT is specified, $LINK/OBJECT is assumed.

/NOOBJECT indicates that temporary files are not to be included in the link.

/RUN indicates that the save image files associated with this $LINK command are to be executed when the link is complete.

/NORUN indicates that program linking only is to occur.
BATCH

filnam1.ext/OBJECT
indicates the name of the object file to be linked. If /OBJECT is not specified, it is assumed as the default.

filnam2.ext/LIBRARY
indicates that the specified file is to be included in the link procedure as a library. The file specified must be a library file (produced by RT-11 LIBR).

filnam3.ext/MAP
indicates the load map file to be created as a result of the $LINK command.

filnam4.ext/EXECUTE
indicates the save image file to be created as a result of the $LINK command.

The following are examples of the $LINK command:

$LINK/RUN
The above command links all temporary object files created since the last $LINK command or the last $FORTRAN/OBJ or $MACRO/OBJ command.

$LINK/MAP PROG1.OBJ+PROG2.OBJ PROGA.SAV/EXE
The above command links the temporary files and the object files PROG1.OBJ and PROG2.OBJ to form a save image file named PROGA.SAV. It also creates and outputs a temporary map file.

$MACRO

12.4.16 $MACRO Command

The $MACRO command calls the MACRO assembler to assemble a source program and, optionally, to provide printed listings or list files. MACRO listing directives, if any, must be specified in the source program to enable their use, as they cannot be entered at BATCH command level.

The $MACRO command has the following format:

$MACRO[/switch] {filnam1.ext/sw} {filnam2.ext/OBJECT}-
{filnam3.ext/LIST} {filnam4.ext/MAP} {filnam5.ext/LIBRARY}-
{filnam6.ext/EXECUTE} {!comments}

where:
/switch  indicates the switches that can be appended to the $MACRO command. The switches are as follows:

/RUN      indicates that the source program is to be assembled, linked, and run.
/NORUN    indicates that the source program is to be assembled only.
/OBJECT   indicates that a temporary object file is to be produced.
/NOOBJECT indicates that a temporary object file is not to be produced.
/LIST     indicates that a listing file is to be produced on the listing device (LST:).
/NOLIST   indicates that a list file is not to be produced.
/CREF     specifies that a cross reference listing is to be produced during assembly.
/NOCREF   indicates that a cross reference listing is not to be produced during assembly.
/MAP      produces a linkage map as part of the listing file on LST:.
/NOMAP    does not create MAP file.
/DOLLARS  indicates that the data following this command may have a $ in the first character position of a line.
/NODOLLARS indicates that a $ may not be in the first character position of a line.
/LIBRARY  indicates that the default library is to be included in the link operation.
/NOLIBRARY indicates that the default library is not to be included in the link operation.

filnam1.ext indicates the name of the source file in the format dev:filnam.ext. If filnam is not specified, the $MACRO source statements must immediately follow the $MACRO command in the input stream.

The source program included after a $MACRO statement can be terminated either by a $EOD command or by any other BATCH command so long as $MACRO/DOLLARS is not specified. A BATCH command is one that starts with a $ in the first position.

/sw can have one of the following values or can be omitted:

/MACRO     indicates that the file name specified is a MACRO source program. Any file name with no switch appended is assumed to be the name of a source file.
/SOURCE    performs the same function as /MACRO.
/INPUT     performs the same function as /MACRO.

filnam2.ext/OBJECT indicates the name (in the format dev:filnam.ext) to be assigned to the object file produced by compilation. The object file remains on the specified device after the job finishes. The object file specification, if included, must be followed by the /OBJECT switch.

12-30
BATCH

If the object file specification is omitted but $MACRO/OBJECT is specified, a temporary object file is created, included in any $LINK operations that follow the $MACRO command in the job, and deleted after the link operation (see Section 12.2.5).

filnam3.ext/LIST
indicates the name to be assigned to the list file created by the assembler. The list file is not printed automatically if LST: is assigned to a file-structured device, but can be listed using the $PRINT command. The list file specification must be followed by the /LIST switch.

filnam4.ext/MAP
indicates the file to which the storage map is to be output.

filnam5.ext/LIBRARY
indicates that the specified file is to be included in the link procedure as a library. The library file specification must be followed by the /LIBRARY switch.

filnam6.ext/EXECUTE
indicates the name to be assigned to a save image file. The save image file specification must be followed by the /EXECUTE switch. If this field is not included, BATCH generates a temporary save image file (see Section 12.2.5), runs it, and then deletes the temporary file.

The following $MACRO command assembles a program named PROG0.MAC and creates a temporary object file and a temporary listing file.

$MACRO/OBJECT PROG0.MAC

12.4.17 $MESSAGE Command

The $MESSAGE command is used to issue a message to the operator at the console terminal. It provides a means for the job to communicate with the operator. The $MESSAGE command has the form:

$MESSAGE[/switch] message {!comments}

where:
BATCH

/switch indicates the switches that can be appended to the $MESSAGE command. These switches are:

/WAIT indicates that the job is to pause until the operator types a carriage return to continue or enters commands to the BATCH handler followed by a carriage return (see Section 12.7.3).

/NOWAIT do not pause for operator response.

message is a string of characters that must fit on one console line. The message is printed on the console.

For example, if the following message is included in the input stream:

$MESSAGE/WAIT MOUNT SCRATCH TAPE ON MT0

The message:

MOUNT SCRATCH TAPE ON MT0
?

appears on the console terminal and a bell sounds. The operator mounts the tape and types carriage return to allow further processing of the job. (See Section 12.7.3 for operator interaction with BATCH.)

$MOUNT

12.4.18 $MOUNT Command

The $MOUNT command assigns a logical device name and other characteristics to a physical device. When $MOUNT is encountered during the execution of a job, the entire $MOUNT command line is printed on the console terminal to notify the operator of the volume to be used.

The $MOUNT command has the form:

$MOUNT[/switch] dev:/PHYSICAL[/VID=x] {ldn:/LOGICAL} {!comments}

where:

/switch indicates the switches that can be appended to the $MOUNT command. The switches are:

/WAIT indicates that the job is to pause until the operator enters a response. If neither /WAIT nor /NOWAIT is specified,
/WAIT is assumed. BATCH rings a bell, prints a ?, and waits for a response. (The response can be input to the BATCH handler; see Section 12.7.3.

/NOWAIT does not pause for operator response.

/WRITE tells the operator to WRITE-ENABLE the volume.

/NOWRITE tells the operator to WRITE-PROTECT the volume.

dev is required and specifies the physical device name and an optional unit number followed by a colon, e.g., DT1:. If dev is specified without a unit number, the operator can enter one in response to the ? printed by the $MOUNT command. If the operator is to supply a unit number, do not use the /NOWAIT switch because it will assume unit 0.

/PHYSICAL identifies the device specification as a physical unit specification. If neither /PHYSICAL nor /LOGICAL is specified, /PHYSICAL is assumed.

/VID=x /VID="x" provides volume identification. The volume identification is the name physically attached to the volume. It is included to help the operator locate the volume. This switch may appear only on the physical device file specification. If x contains spaces, it must be input as "x".

1dn:/LOGICAL is required to identify the logical device name, if any, to be assigned to the device. The logical device name specification must be followed by the /LOGICAL switch.

The following are examples of the $MOUNT command:

$MOUNT /WAIT /WRITE DT: /VID=BAT01 2:/LOGICAL

This command instructs the operator to select a DECTape unit and mount DECTape volume BAT01 on that unit, WRITE-ENABLED. It informs the operator by printing:

$MOUNT /WAIT /WRITE DT: /VID=BAT01 2:/LOGICAL

The operator selects a unit, mounts DECTape volume BAT01, WRITE-ENABLED, and responds to the ? by typing the unit number (e.g., 1) followed by a carriage return. BATCH assigns logical device name 2 to the physical device (e.g., DT1:) and proceeds.

If no unit number response is necessary, e.g.,

$MOUNT /WAIT /WRITE DT1: 2:/LOGICAL

the operator responds with a carriage return after mounting the DECTape and WRITE-ENABLING the device.

12-33
$PRINT

12.4.19 $PRINT Command

The $PRINT command is used to print the contents of the specified files on the listing device (LST:). This command has the form:

$PRINT[/switch] dev:filnam.ext[/INPUT]{,dev:filnam2.ext,...,dev:-filnamn.ext} {!comments}

where:

/switch indicates the switches that can be appended to the $PRINT command. The switches are:

/DELETE indicates that input files are to be deleted after printing.
/NODELETE indicates that input files are not to be deleted after printing.

dev: is the device containing the files to be printed; if dev: is not specified, DK: is assumed.

gnlam1.ext-filnamn.ext indicate names of the files to be printed. Wild cards can be used for the file name or extension.

/INPUT indicates that the file is an input file; /INPUT is assumed if it is not entered.

The following command prints a listing of files with extension .MAC that are stored on default device DK:

$PRINT *.MAC

The following example creates listing files for the programs A and B, prints the listing files, and then deletes them.

#MACRO A.MAC A/LIST
#MACRO B.MAC B/LIST
$PRINT/DELETE A.LST,B.LST

12-34
12.4.20 $RTll Command

The $RTll command allows the BATCH job to communicate directly with the RT-ll system. This command puts BATCH in RT-ll mode, i.e., until a line beginning with $ is encountered, all data images are interpreted as commands to the RT-ll monitor, RT-ll system programs, or to the BATCH run-time system. The $RTll command has the form:

$RTll {!comments}

See Section 12.5 for a complete description of the RT-ll mode.

12.4.21 $RUN Command

The $RUN command requests execution of a program for which a save image file (.SAV) was previously created. It can also be used to run RT-ll system programs.

The $RUN command has the form:

$RUN dev:filnam.ext {!comments}

where filnam.ext is the name of the program (system or user) to be executed. If no extension is specified, .SAV is assumed.

For example, the user can run PIP to print a directory listing.

$RUN PIP
$DATA
LP:=DK:/L
$EOD
12.4.22 $SEQUENCE Command

The $SEQUENCE command is an optional command. If used, it must immediately precede a $JOB command. The $SEQUENCE command assigns a job an arbitrary identification number. The last three characters of a sequence number are assigned by BATCH as the first three characters of a temporary listing or object file (see Section 12.2.5). If a sequence number is less than three characters long, it is padded with zeros on the left.

The form of this command is:

$SEQUENCE id {!comments}

where id is an unsigned decimal number indicating the identification number of a job.

The following are examples of the $SEQUENCE command:

```
$SEQUENCE 3     !SEQUENCE NUMBER IS 003
$JOB
$SEQUENCE 100   !SEQUENCE NUMBER IS 100
$JOB
```

12.4.23 Example BATCH Stream

The following example BATCH stream creates a MACRO program, assembles and links that program, and runs the save image file. It then deletes the object, save image, and source files created and prints a directory of DK: showing the files created by the BATCH stream.

```
$JOB
$MESSAGE     THIS IS AN EXAMPLE BATCH STREAM
$MESSAGE     NOW CREATE A MACRO PROGRAM
$CREATE/LIST EXAMPL.MAC
.TITLE EXAMPL FOR BATCH
 .MCALL .REGDEF...V2....PRINT..EXIT
 .REGDEF
 .V2..
START: .PRINT #MESSAG
 .EXIT
MESSAGE: .ASCIZ /EXAMPLE MACRO PROGRAM FOR BATCH/
 .END   START
```

July 1975
12-36
To run this batch stream, the user types the following at the console. Messages are printed by BATCH.

```
.LOAD BR(LP
.ASSIGN LP:LOG
.ASSIGN LP:LST
R BATCH
*EXAMPLE
THIS IS AN EXAMPLE BATCH STREAM
NOW CREATE A MACRO PROGRAM
RUN THE MACRO PROGRAM
PRINT A DIRECTORY
END OF THE EXAMPLE BATCH STREAM

END BATCH
```

The above example BATCH stream produces the following log file on the line printer.

```
$JOB
$MESSAGE THIS IS AN EXAMPLE BATCH STREAM
$MESSAGE NOW CREATE A MACRO PROGRAM
$CREATE/LIST EXAMPLE.MAC
$TITLE EXAMPLE FOR BATCH
*CALL .REGDEF,...V2...,PRINT,EXIT
.*REGDEF
**VP**
START: .PRINT #MESSAGE
.QUIT
MESSAGE: ASCIZ
.EXIT
/EXAMPLE MACRO PROGRAM FOR BATCH/
3

$END
$MACRO EXAMPLE OBJECT LIST ASSEMBLE
*ERRORS DETECTED: 0
FREE CORE: 10764, WORDS
*
$LINK EXAMPLE EXECUTE LINK
$PRINT/DELETE EXAMPLE.LST
$MESSAGE RUN THE MACRO PROGRAM
$RUN EXAMPLE EXECUTE
```

12-37
BATCH

EXAMPLE MACRO PROGRAM FOR BATCH
$DELETE EXAMPL,OBJ+EXAMPL,SAV+EXAMPL,MAC

$MESSAGE PRINT A DIRECTORY
$DIRECTORY OK:EXAMPL,*
9=MAY-75
EXAMPL.BAT 2 28=APR-75
EXAMPL.CTL 3 9=MAY-75
2 FILES, 5 BLOCKS
1436 FREE BLOCKS

$MESSAGE END OF THE EXAMPLE BATCH STREAM
$EOJ

12.5 RT-II MODE

RT-II mode provides the capability to enter commands to the RT-II monitor or to system programs and to create BATCH programs. RT-II mode may be entered with either the $JOB/RTII command or the $RTII command. If entered with the $JOB/RTII command, RTII mode remains in effect until the next $JOB command is encountered in the BATCH stream. If entered with the $RTII command, RT-II mode is in effect until a $ is encountered in the first position of the command line.

The characters ., $, *, and tab or space appearing in the first position of a line (or card column 1) are interpreted as control characters and indicate the following:

. command to the RT-II monitor, e.g.,
.R PIP

* data line; any line not intended to go to the RT-II monitor or to the BATCH run-time handler, e.g., a command to the RT-II PIP program:

*FILE1.DAT/D

NOTE

The * is not passed as data to the program.
Comment lines (!) cannot appear on data lines as they would be considered as data.

$ BATCH command. Causes exit from RT-II mode if RT-II mode was entered with the $RTII command. For example:

$RTII !ENTER RT-II MODE
.R PIP
*DIRB:FILE3.DAT/D
$DIRECTORY DIRB: !LEAVE RT-II MODE

space/tab separator to indicate line directed to BATCH run-time handler. This separator is indicated by a ! in the following descriptions.

12-38
12.5.1 Running RT-11 System Programs

The most common use of RT-11 mode is to send commands to the RT-11 monitor and to system programs. For example, the following commands can be inserted in the BATCH stream to run PIP and save backup copies of files on DECTape.

```plaintext
$RT11
.R PIP
#DT1:*.*=.*,FOR/X
```

The user must anticipate and include in the BATCH input stream responses that are required by the called program, e.g., the Y response to PIP's ARE YOU SURE? query.

BATCH standard commands cannot be mixed with RT-11 mode data lines (i.e., lines beginning with an asterisk). For example, the proper way to do a $MOUNT within a sequence of RT-11 mode data commands is:

```plaintext
$JOB/RT11
.R MACRO
#A1=A1
#A2=A2
#MOUNT DT8:/PHYSICAL
.R MACRO
#B1=DT:B1
#B2=DT:B2
```

12.5.2 Creating RT-11 Mode BATCH Programs

RT-11 mode may be used by advanced system programmers to create BATCH programs. These BATCH programs consist of standard RT-11 mode commands (monitor commands, data lines for input to system programs, etc.) plus special RT-11 mode commands. These special commands are interpreted by the BATCH run-time handler to allow dynamic calculations and conditional execution of the RT-11 mode standard commands. The following facilities allow the user to create BATCH programs and to dynamically control the execution of these programs at run-time.

A. Labels
B. Variable modification
   1) equating a variable to a constant or character (LET statement)
   2) incrementing the value of a variable by 1
   3) reading a value into a variable
   4) conditional transfers on comparison of variable values with numeric or character values (IF and GOTO statements)
C. Commands to control terminal I/O
D. Other Control Characters
E. Comments

12.5.2.1 Labels -- Labels in RT-11 mode are user-defined symbols that provide a symbolic means of referring to a specific location within a BATCH program. If present, a label must begin in the first character position, must be unique within the first six characters, and must terminate with a colon (:) and a carriage return/line feed combination.

12-39    July 1975
12.5.2.2 Variables -- A variable in RT-11 mode is a symbol representing a value that can change during program execution. The 26 variables permitted in a BATCH program have the names A-Z; each variable requires one byte of physical storage. The user may assign values to variables in a LET statement. These values may then be tested by an IF statement to control the direction of program execution.

Variables may be assigned values with a LET statement of the following form:

\[ \text{LET } x = "c} \]

where \( x \) is a variable name A-Z and "c indicates the ASCII value of a character. For example:

\[ \text{LET } A = "0} \]

indicates that the value of variable \( A \) is equal to the 7-bit ASCII value of the character 0 (60).

The LET statement can also specify an octal value in the form:

\[ \text{LET } A = n \]

where \( n \) is an 8-bit signed octal value in the range 0 to 377; positive numbers range from 0 to 177 and negative numbers from 200 to 377 (-200 to -1).

Variables may be used to introduce control characters, such as ALTMODE, into a BATCH stream. For example, wherever 'A' appears in the following BATCH stream, BATCH substitutes the contents of variable \( A \) (the code for an ALTMODE):

\[
\text{#JOB/RT11}
\text{LET } A = 33
\text{'A IS AN ALTMODE}
\text{.R EDIT}
\text{*EBFIL.MAC 'A''A'}
\text{*EDIT FILE TO CHANGE THE VERSION NUMBER TO 2}
\text{*VERSION='A'D12'"A'}
\text{*EX 'A''A'}
\text{#EOJ}
\]

The value of a variable can be incremented by 1 by placing a percent sign (%) before the variable. For example:

\[ \%A \]

indicates that the unsigned contents of variable \( A \) are to be increased by 1.

Conditional transfers of control according to the value of a variable are indicated with an IF statement. The IF statement has the form:

\[ \text{IF}(x = "c) \text{ labell, label2, label3} \]

or

\[ \text{IF}(x = n) \text{ labell, label2, label3} \]

where \( x \) is the variable to be tested, "c is the ASCII value to be
BATCH

compared with the contents of the variable, or n is an octal value in the range 0 to 377, and label1, label2, and label3 are the names of labels included in the BATCH stream.

When BATCH evaluates the expression (x-=c) or (x=n), the BATCH run-time handler transfers control to:

- label1 if the value of the expression is less than zero.
- label2 if the value of the expression is equal to zero.
- label3 if the value of the expression is greater than zero.

If one of the labels is omitted, and the condition is met for the omitted label, control transfers to the line following the IF statement.

NOTE

Since this comparison is a signed byte comparison, 377 is considered to be -1.

The characters + and - allow the user to control where BATCH begins searching for label1, label2, and label3. If the label is preceded by a minus sign (-), the label search starts just after the $JOB command. If a plus sign (+) or no sign precedes the label, the label search starts after the IF statement. For example, the following statement:

→ IF (B=-9) -LOOP, LOOP1,

transfers program control to the label LOOP following the $JOB command if the contents of variable B are less than the ASCII value of 9 or to the label LOOP1 following the IF statement if B is equal to ASCII 9. If the contents of variable B are greater than the ASCII value of 9, program control goes to the next BATCH statement in sequence.

The GOTO statement unconditionally transfers program control to a label specified as the argument of the statement. This statement may be one of the following three forms:

→ GOTO label transfers control to the first occurrence of label that appears after this GOTO statement in the BATCH stream.
→ GOTO +label same as GOTO label.
→ GOTO -label transfers control to the first occurrence of label that appears after the $JOB command.

If the label is not found, transfer goes to $EOJ.

The following GOTO statement transfers control unconditionally to the next label LOOP if such a label appears in the BATCH stream following the GOTO statement.

→ GOTO LOOP

12-41
NOTE

If a label cannot be found, e.g., a minus sign was intended but omitted, the BATCH handler searches until the end of the CTL file is reached and ends the job.

12.5.2.3 Terminal I/O Control -- Commands may be issued directly to the BATCH run-time handler to control console terminal input/output to the log file. If none of the following commands is entered, TTYOUT is assumed.

- NOTTY  do not write terminal input/output to the log file. Comments to the log will still be logged.
- TTYIN   write only terminal input to the log file.
- TTYIO   write terminal input and output to the log file.
- TTYOUT  write only terminal output to the log file (default).

12.5.2.4 Other Control Characters -- Other control characters allowed in an RT-11 mode command that begins with a period (.) or an asterisk (*) indicate the following:

- 'text' command to BATCH run-time handler, where text can be one of the following:

  CTY      accept input from the console terminal; notify the operator that action is required by ringing a bell and printing a ?.
  FF       output current log buffer.
  NL       insert a new line (line feed) in the BATCH stream.
  x        insert contents of variable where x is an alphanumeric variable A-Z, indicates that the contents of the variable are to be inserted as an ASCII character at this place in the command string.
  "message" direct the message to the console terminal.

Example 1:

The following commands allow the operator to enter the name of a MACRO program to be assembled. The BATCH stream contains:

```
$JOB/RT11
.R MACRO
"ENTER MACRO COMMAND STRING"/'CTY'
$EOJ
```

The operator receives the following message at the terminal; he types a response, followed by carriage return, and BATCH processing continues.

July 1975

12-42
Example 2:
The user may want to run the same BATCH file on several systems with different configurations and would want to assign a device dynamically. The following RT-11 mode command allows the user to request that the listing device name be entered by the operator.

```
.ASSIGN "PLEASE TYPE LST DEVICE NAME"/"CTY"/LST
```
The operator receives the message and responds with the device to be used as the listing device (DT2:).

```
PLEASE TYPE LST DEVICE NAME?DT2:
```

12.5.2.5 Comments -- Comments can be included in RT-11 mode as separate comment statements. This is accomplished by typing a separator followed by a ! and the comment, e.g.,

```
!OPERATOR ACTION IS REQUESTED IN THIS JOB. BE PREPARED.
```

12.5.3 RT-11 Mode Examples
The following are examples of BATCH programs using the RT-11 mode.

Example 1:
This BATCH program assembles, lists, and maps 10 programs with only 12 BATCH commands.

```
$JOB/RT11
$ASSEMBLE, LIST, MAP PROG0 TO PROG9
TTYIO
!WRITE TERMINAL I/O TO THE LOG FILE
LET N=0
!START AT PROG0
LOOP:
.R MACRO
"PROG'N",LOG:/C=PROG'N'/N/TTM
.R LINK
",LOG:=PROG'N'
%N
!INCREMENT VARIABLE N
IF(N"5)-LOOP,-LOOP,END
!TEST FOR END
END:
$E0J
```

Example 2:
The following program allows the user to set up a master control stream to run several BATCH jobs with one call to BATCH. First the user sets up a BATCH job (INIT.BAT) that will $CHAIN to the master control stream:
 BATCH

$JOB/R711
LET I="0"
!INITIALIZE INDEX
$CHAIN MASTER !GO TO MASTER
$EOJ

The following is the master control stream (MASTER.BAT) to which .INIT
chains.

$JOB/R711 !MASTER CONTROL STREAM
$I
!BUMP INDEX BY 1
IF(I="?"),.END
.R BATCH
!THIS IS A $CHAIN
*$JOB/1
!RUNS JOB1-JOB7
END:
$MESSAGE END OF BATCH RUN
$EOJ

Each job to be run by MASTER.BAT must contain the following:

$JOB
!BATCH COMMANDS
$CHAIN MASTER
$EOJ

The master control stream is activated by calling BATCH as follows:

.R BATCH
*INIT

12.6 CREATING BATCH PROGRAMS ON PUNCHED CARDS

To create a BATCH program on punched cards, the user punches into the
cards the commands as described in Section 12.4. Each command line
occupies a single punched card. Only one card, the EOF card, is
different from the standard BATCH commands. The EOF (end-of-file)
card terminates the list of jobs from the card reader.

To create the EOF card, hold the MULT PCH key on the keypad while typing the following characters (DEC029 codes, see Appendix H for
DEC026 codes):

- & 0 1 6 7 8 9

This procedure produces an EOF card with holes punched in the first
column (see Figure 12-1).
12.6.1 Terminating BATCH Jobs on Cards

To terminate BATCH, type \F, followed by a carriage return, put two EOF cards and a blank card in the reader, and ensure that the card reader is ready. Note that a small card deck (less than 512 characters) may require more than two EOF cards to terminate the deck.

12.7 OPERATING PROCEDURES

12.7.1 Loading BATCH

After the RT-11 system has been bootstrapped and the date and time entered, the BATCH run-time handler must be made resident by typing the RT-11 LOAD command as follows:

```
.LOAD BA
```

The BATCH run-time handler must be detached with the /U switch to the BATCH compiler command line (see Section 12.7.2) before it is removed with the RT-11 command UNLOAD.

Unless the log device is SY: or any device for which the handler is already resident, the BATCH log device must be made resident. The log device is loaded by typing:

```
.LOAD log
```

where log is the device to which the BATCH log is to be written, e.g.,
BATCH

LOAD LP

The loading of device handlers can, of course, be done with a single LOAD command, e.g.,

LOAD BA, LP

The log device must then be assigned the logical device name LOG. This is accomplished with the RT-11 monitor command ASSIGN in the form:

ASSIGN log:LOG

For example, if LP: is the log device, type:

ASSIGN LP:LOG

The device to be used for listing files must be made resident with the LOAD command and then assigned the logical device name LST:. This is accomplished with the RT-11 ASSIGN command in the form:

ASSIGN lisdev:LST

where lisdev is the physical device to be used for listings. If, for example, listings are to be produced on the line printer, type:

ASSIGN LP:LST

NOTE

The ASSIGN command with no arguments should not be used in a BATCH program since it would then deassign the log and list devices, possibly causing the BATCH job to terminate.

The BATCH Run-Time Handler input device (Compiler output device) must also be made resident. If this device is already resident or is SY:, it need not be loaded. For example, to load the DECTape handler as the input device, type:

LOAD DT

If the input file to the BATCH Compiler is on cards, the card reader handler must be loaded by typing:

LOAD CR

NOTE

If input is on cards, the RT-11 monitor command SET must be used (before the handler is loaded) to specify CRLF and NOIMAGE modes, i.e.,

SET CR CRLF

to append a carriage return/line feed combination to each card image, and

SET CR NOIMAGE

to translate the card by packing card code into ASCII data, one column per byte.

January 1976 12-46
12.7.2 Running BATCH

When all necessary handlers have been loaded, the BATCH Compiler is run as follows:

```
>R BATCH
```

BATCH responds by printing an asterisk (*) to indicate readiness to accept command string input. In response to the *, type the output file specifications for the control file followed by an equal sign or left angle bracket, followed by the input file specifications for the BATCH file as follows:

```
{{spc0},{spc1}{/switch}={spc3,...}{,spc8/switch}
```

where:

- **spc0** is the BATCH compiler output device and file to be used by the BATCH run-time handler. The device specified must be random-access. This file should not be deleted or moved by the job that is within it; nor should the system ever be compressed from within a batch job. If spc0 is omitted, a file is generated on the default device (DK:) with the same name as the first input file but with a .CTL extension. If no extension is specified, .CTL is assumed.

- **spc1** is the log file created by the BATCH run-time handler. If no log device is specified, LOG: is assumed. The device name specified for spc1 must be that assigned to LOG:.

The size of a log file on a file-structured device may be changed from the default size of 64 (decimal) by enclosing the requested size in square brackets. For example:

```
*,FILE.LOG[10]=FILE
```

The default extension for spc1 is .LOG.

- **spc3**-**spc8** are input file specifications. If no input extensions are specified, the default extension is .BAT. If a .CTL file is specified, a precompiled file is assumed and it must be the only file in the input list.

- **/switch** is one or more of the following:

  - **/N** compiles but does not execute; creates BATCH control file (.CTL), generates an ABORT JOB message at the beginning of the log file, and returns to RT-ll monitor.

  - **/T:n** if n=0, sets the /NOTIME switch as default on the $JOB command; if n=1, the default switch on the $JOB command is /TIME.
indicates that BATCH compiler is to detach the BATCH run-time handler from the RT-ll monitor for subsequent removal. The prompting message !UNLOAD BA! is printed.

Note

The RT-ll monitor command UNLOAD BA must be specified to actually remove the handler. The handler must be reloaded before BATCH is run again.

/V

prints version number of BATCH compiler

/X

indicates that input is a precompiled BATCH program (used when .CTL is not the extension or when the extension is omitted).

Example 1:

In the following example, the user calls BATCH to compile and execute the three input files (PROG1.BAT, PROG2.BAT, PROG3.BAT), to generate on DR: the compiler output files and to generate on LOG: a log file.

.R BATCH
  *PROG1.BAT, PROG2.BAT, PROG3.BAT

Example 2:

The following commands compile and run SYBILD.BAT, printing the version number of BATCH.

.R BATCH
  *SYBILD/V
  BATCH V01-02

Example 3:

The following commands compile PROTO.BAT to create PROTO.CTL but do not run the compiled program.

.R BATCH
  *PROTO/N

Example 4:

Type the following commands to unlink BA.SYS from the monitor and to unload it.

.R BATCH
  */U
  'UNLOAD BA!

  .UNLOAD BA
Example 5:
The following commands compile FILE.BAT from magtape to create FILE.CTL on DK1:, execute the compiled file, and create a log file named FILE.LOG of size 20 on LOG1:

```
.R BATCH
```

Example 6:
The following commands execute a precompiled job called FILE.TST.

```
.R BATCH
*FILE.TST/X
```

Example 7:
The following commands execute a precompiled job called FILE.CTL.

```
.R BATCH
*FILE/X
```

Example 8:
The following commands accept input from the card reader to create on DK: a file called TEMP.CTL and execute that file.

```
.R BAT CH
*CR:
```

Example 9:
The following commands accept input from the card reader to create on DK: a file called JOB.CTL and execute that file.

```
.R BAT CH
*JOB=CR:
```

12.7.3 Communicating with BATCH Jobs
During the execution of a BATCH stream, the operator may be requested to supply action or provide information or to insert a command line into the BATCH stream. The operator can accomplish this by typing directives to the BATCH handler via the console terminal.

NOTE
These directives are equivalent to the compiler output generated by BATCH in the .CTL file. The .CTL file is an ASCII file that can be listed using PIP.

These directives have the form:

```
\dir
```

where dir is one of the directives listed in Table 12-6.
To use these directives, the operator must get control of the BATCH run-time handler by typing a carriage return on the console terminal. When BATCH executes a command, it notices the carriage return and prints a carriage return/line feed combination at the terminal. The directives in Table 12-6 may then be entered by the operator; those most useful to the operator are marked with an asterisk (*).

Table 12-6
Operator Directives to BATCH Run-Time Handler

<table>
<thead>
<tr>
<th>Directive</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Send the characters that follow to the console terminal.</td>
</tr>
<tr>
<td>*\A</td>
<td>Change the input source to be the console terminal.</td>
</tr>
<tr>
<td>*\B</td>
<td>Change the input source to be the BATCH stream.</td>
</tr>
<tr>
<td>*\C</td>
<td>Send the following characters to the log device.</td>
</tr>
<tr>
<td>*\D</td>
<td>Consider the following characters as user data.</td>
</tr>
<tr>
<td>*\E</td>
<td>Send the following characters to the RT-11 monitor.</td>
</tr>
<tr>
<td>*\F</td>
<td>Force the output of the current log block. If this directive is followed by any characters other than another BATCH backslash () directive, the BATCH job prints an error message and terminates; control returns to the RT-11 monitor.</td>
</tr>
<tr>
<td>\G</td>
<td>Get characters from the console terminal until a carriage return is encountered.</td>
</tr>
<tr>
<td>\Hn</td>
<td>Help function to change the logging mode where n specifies the following:</td>
</tr>
<tr>
<td></td>
<td>0 Log only .TTYOUT and .PRINT.</td>
</tr>
<tr>
<td></td>
<td>1 Log .TTYOUT, .PRINT, and .TTYIN</td>
</tr>
<tr>
<td></td>
<td>2 Do not log .TTYOUT, .PRINT, and .TTYIN</td>
</tr>
<tr>
<td></td>
<td>3 Log only .TTYIN</td>
</tr>
<tr>
<td>\ivxlabell?label2?label3?</td>
<td>IF statement which causes conditional transfer, where v is a variable name in the range A-Z; x is a value for the signed 8-bit comparison (v-x); and label1, label2, label3 are 6-character labels to which control is transferred under certain conditions. (All labels must be six characters in length; if too short, pad with spaces.) If v-x is less than 0, control transfers to label1; if v-x is equal to 0, control goes to label2; if v-x is greater than 0, control goes to label3. The direction for the label search is indicated by ?; if ? is 0, the search begins at the beginning of this job; if ? is 1, the label search begins after the IF statement.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 12-6 (cont.)
Operator Directives to BATCH Run-Time Handler

<table>
<thead>
<tr>
<th>Directive</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\Jlabel?</td>
<td>Jump, unconditional transfer; where label is a 6-character label and ? is 0 or 1. (All labels must be six characters in length; if too short, pad with spaces.) If ?=0, label is a backward reference; if ?=1, label is a forward reference.</td>
</tr>
<tr>
<td>\Kv0</td>
<td>Increment variable v where v is a variable name in the range A-Z.</td>
</tr>
<tr>
<td>\Kvln</td>
<td>Store the 8-bit number n in variable v.</td>
</tr>
<tr>
<td>\Kv2</td>
<td>Take the value in variable v and return it to the program (via TTYIN).</td>
</tr>
<tr>
<td>\Llabel</td>
<td>Insert label as a 6-character alphanumeric string in the BATCH stream. (All labels must be six characters in length; if too short, pad with spaces.) Labels must not include backslash characters. Characters beyond six are ignored.</td>
</tr>
</tbody>
</table>

Example 1:

The operator may wish to interrupt the BATCH handler to enter information from the console as a result of a /WAIT or 'CTY' in the BATCH stream. For example, the following message appears at the terminal.

```
#MESSAGE/WAIT WRITE NECESSARY FILES TO DISK
```

To divert BATCH stream input from the current file to the console terminal, the operator types a \E and can then enter commands to the RT-11 monitor until he types a \B. Control then returns to the BATCH stream. For example, to respond to the preceding message, the operator types:

```
WRITE NECESSARY FILES TO DISK
\N\E\ER PIP
\E\ER PIP
\DRK:*.*=DT2:FILE.MAC/X
\E\F\B
```

changes input source to TT: and calls the RT-11 monitor to run PIP writes FILE.MAC from DT2: to RK: outputs log block and returns control to the BATCH stream

Example 2:

The following BATCH program allows the user to make frequent edits to a file and list only the edits. First the user creates a BATCH program which will assemble with a listing and link the file. This BATCH program, called COMPIL.BAT, contains:

```
$JOB/RT11
TTYIO
!WRITE TERMINAL I/O TO THE LOG FILE
.R MACRO
```

12-51

July 1975
BATCH

!CALL THE MACRO ASSEMBLER
*FILE, FILE/C=FILE
#MESSAGE/WHIT OK TO TYPE EDIT COMMANDS
.R LINK
!CALL THE RT-11 LINKER
*FILE, LOG:=FILE
#EOJ

At run-time, the user can insert commands into the BATCH stream via the console terminal. These commands will search for the section of the listing file that has been edited and then list this section to the log. The command must be inserted after the R MACRO command but before the R LINK command. The operator types the following to use COMPIL.BAT.

LOAD BH, LP
ASSIGN LP;LOG
.R BATCH
*COMPILE
OK TO TYPE EDIT COMMANDS
?\A\E
ER EDIT
:
*ERFILE, LST$#
*ERFILE, LST$#
*PBELL:$=J$#
*/L$#
*EX$#
\E\C\B
return to BATCH

END BATCH

12.7.4 Terminating BATCH

When BATCH terminates normally, it prints:

END BATCH

and returns control to the RT-11 monitor.

To abort BATCH while it is executing a BATCH stream, the best method is to interrupt the BATCH handler by typing a carriage return. When BATCH executes the next command after the carriage return was typed, it prints a carriage return/line feed combination at the console terminal and the operator has control. The operator then types \F followed by a carriage return. The BATCH handler responds with the FE (forced exit) error message and writes the remainder of the log buffer. Control returns to the RT-11 monitor.

CTRL C can be typed to terminate BATCH but is a more drastic method than that described above. CTRL C should be used when BATCH is in a loop or when a long assembly is running.

12.8 DIFFERENCES BETWEEN RT-11 BATCH AND RSX-11D BATCH

For those users who may wish to run their RT-11 BATCH programs under RSX-11D, or vice versa, Table 12-7 lists the differences between the two programs. A user who plans to run his BATCH programs under both systems should ensure that the programs are compatible with both RT-11 and RSX-11D BATCH.

January 1976 12-52
### Table 12-7
Differences Between RT-11 and RSX-11D BATCH

<table>
<thead>
<tr>
<th>Differing Aspects</th>
<th>RT-11</th>
<th>RSX-11D</th>
</tr>
</thead>
<tbody>
<tr>
<td>File descriptors</td>
<td>dev:filnam.ext/sw</td>
<td>SY:filnam.ext/sw</td>
</tr>
<tr>
<td>Default listing extension</td>
<td>.LST (or .LIS)</td>
<td>.LIS</td>
</tr>
<tr>
<td>Executable file extension</td>
<td>.SAV</td>
<td>.EXE</td>
</tr>
<tr>
<td>Incompatible commands</td>
<td>$BASIC $CALL $CHAIN $LIBRARY $RT11 $SEQUENCE</td>
<td>$MCR</td>
</tr>
<tr>
<td>Incompatible switches</td>
<td>$COPY/DELETE $CREATE/DOLLARS $CREATE/LIST $DATA/DOLLARS $DATA/LIST $DIR file/LIST $DISMOUNT/WAIT $DISMOUNT lun:/LOGICAL $PORT/RAN/DOLLARS $PORT/RAN/MAP $JOB/BANNER $JOB/LIST $JOB/RT11 $JOB/TIME $JOB/UNIQUE $LINK/LIBRARY $LINK/OBJECT $MACRO/CREF $MACRO/DOLLARS $MACRO/LIBRARY $MACRO/MAP $MESSAGE/WAIT $MESSAGE/WRITE $PRINT/DELETE</td>
<td>$DIR file/DIRECTORY $JOB/NAME $JOB/LIMIT $JOB/MCR $LINK/MCR</td>
</tr>
<tr>
<td>$DATA input</td>
<td>appears as if from input</td>
<td>appears as if from a file named FOR001.DAT</td>
</tr>
<tr>
<td>Logical device names</td>
<td>in $MOUNT and $DISMOUNT</td>
<td>logical unit numbers only</td>
</tr>
<tr>
<td>$RUN</td>
<td>file name must be specified</td>
<td>RSX11DBAT.EXE is default</td>
</tr>
</tbody>
</table>

#### 12.9 ERROR MESSAGES

Two types of errors are reported by the BATCH system: user BATCH stream errors found by the compiler and noted in the BATCH log file, and system or operator errors noted at the console at run-time. Note that when BATCH is called from within a BATCH stream, both types of error messages are sent to the log.
BATCH

The following messages detail errors found by the compiler and noted in the log file. An error is indicated by the printing of the erroneous line, an uparrow (or circumflex) under the error, and one of the following messages.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT JOB</td>
<td>An error has occurred in compiling a BATCH program; the compiler forces the job to abort. All error messages are printed in the log file.</td>
</tr>
<tr>
<td>BAD CONSTRUCTION</td>
<td>In RT-ll mode, an IF statement is not in the correct form or there was an illegal 'text' directive.</td>
</tr>
<tr>
<td>BAD SEQUENCE ARGUMENT</td>
<td>The identification number specified in a $SEQUENCE command is not numeric.</td>
</tr>
<tr>
<td>BAD VARIABLE</td>
<td>The variable specified is not one of the characters A-Z.</td>
</tr>
<tr>
<td>BAD VID</td>
<td>The volume identification specified in a $MOUNT command is not in the correct form, e.g., the equal sign (=) or the text is missing.</td>
</tr>
<tr>
<td>BATCH HANDLER NOT RESIDENT</td>
<td>The BATCH run-time handler was not loaded with the RT-ll LOAD command (see Section 12,7,1).</td>
</tr>
<tr>
<td>BATCH STACK OVERFLOW</td>
<td>There are too many nested $CALL commands in the BATCH stream.</td>
</tr>
<tr>
<td>CL 'MAND NOT UNIQUE</td>
<td>The spelling of a command is not unique (appears only when $JOB/UNIQUE was specified).</td>
</tr>
<tr>
<td>DISMOUNT ERROR</td>
<td>The logical device name specified does not exist.</td>
</tr>
<tr>
<td>'$' MISSING</td>
<td>A $ is not present in the first position of the command line (or card column 1).</td>
</tr>
<tr>
<td>ILLEGAL '+'</td>
<td>The + construction was used when not allowed (e.g., in a $RUN or $BASIC input file descriptor), there is a + in an output file descriptor, or a + terminates a control statement.</td>
</tr>
<tr>
<td>ILLEGAL CHARACTER</td>
<td>The character specified is not used in proper context.</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILLEGAL SWITCH</strong></td>
<td>The switch name specified is not a legal RT-ll BATCH switch or is not legal for this field.</td>
</tr>
<tr>
<td><strong>ILLEGAL SWITCH COMBINATION</strong></td>
<td>More than one switch of the same type, e.g., /INPUT and /SOURCE exists on same command line.</td>
</tr>
<tr>
<td><strong>LINE TOO LONG</strong></td>
<td>The input line entered is greater than 80 characters.</td>
</tr>
<tr>
<td><strong>MULTIPLE SWITCH</strong></td>
<td>The same switch is specified more than once in a single command line.</td>
</tr>
<tr>
<td><strong>NO $EOJ</strong></td>
<td>A $JOB or $SEQUENCE command appears without a preceding $EOJ to end the previous job.</td>
</tr>
<tr>
<td><strong>NO FILE</strong></td>
<td>No file descriptor was found in the BATCH stream where one was expected, or no file name was entered in the $CREATE command.</td>
</tr>
<tr>
<td><strong>NO FILE NAME BEFORE &quot;.&quot;</strong></td>
<td>An extension was specified but no file name preceded it.</td>
</tr>
<tr>
<td><strong>NO LOGICAL DEVICE</strong></td>
<td>No logical device is specified in a $MOUNT command.</td>
</tr>
<tr>
<td><strong>NO PHYSICAL DEVICE</strong></td>
<td>No physical device is specified in a $MOUNT command.</td>
</tr>
<tr>
<td><strong>PLEASE ASSIGN LOG,LST</strong></td>
<td>The log device (LOG:) and/or the listing device (LST:) were not assigned (see Section 12.7.1).</td>
</tr>
<tr>
<td><strong>PLEASE LOAD LOG HANDLER</strong></td>
<td>The log device handler is not resident (see Section 12.7.1).</td>
</tr>
<tr>
<td><strong>SEPARATOR MISSING</strong></td>
<td>A file descriptor was not terminated by a space, a +, a , or a carriage return.</td>
</tr>
<tr>
<td><strong>SWITCH NOT UNIQUE</strong></td>
<td>The spelling of a switch is not unique (appears only when $JOB/UNIQUE is specified).</td>
</tr>
<tr>
<td><strong>TOO MANY FILE DESCRIPTORS</strong></td>
<td>More than six file descriptors are specified in a $command line.</td>
</tr>
<tr>
<td><strong>UNKNOWN COMMAND</strong></td>
<td>The command specified with a $ in character position 1 is not a legal BATCH command.</td>
</tr>
</tbody>
</table>
The following messages, generated by system or operator errors, are noted on the console at run-time.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD COPY OF HANDLER</td>
<td>The copy of BA.SYS in memory is bad. BATCH cannot be run until BA.SYS is unloaded and reloaded (see Section 12.7.1).</td>
</tr>
<tr>
<td>BAD SWITCH</td>
<td>The command line to the BATCH Compiler contains an illegal switch.</td>
</tr>
<tr>
<td>BATCH FATAL ERROR</td>
<td>A nonrecoverable error has occurred (e.g., the system device is WRITE-LOCKed); BATCH must be rerun. The system may have to be rebootsed.</td>
</tr>
<tr>
<td>BATCH STACK OVERFLOW</td>
<td>There are too many nested $CALL commands in the BATCH stream.</td>
</tr>
<tr>
<td>BC</td>
<td>Fatal error. A Bad Code was found in the control file by the BATCH handler. This can happen when a .CTL file has been garbled or if an editing mistake was made by the programmer when altering or creating the file with EDIT.</td>
</tr>
<tr>
<td>END BATCH</td>
<td>A BATCH job has been terminated.</td>
</tr>
<tr>
<td>EOF WITH NO EOJ</td>
<td>A file was not terminated with a $EOJ command.</td>
</tr>
<tr>
<td>FE</td>
<td>Fatal error. A Forced End occurred due to the appearance in the .CTL file of an illegal $F followed by a carriage return or BATCH was terminated from the console with a $F followed by a carriage return. The $F must be followed by another BATCH control directive.</td>
</tr>
<tr>
<td>ILLEGAL COMMAND LINE</td>
<td>Command line to the BATCH compiler is incorrect.</td>
</tr>
<tr>
<td>ILLEGAL DEVICE</td>
<td>A $DISMOUNT command was attempted on device that was not assigned.</td>
</tr>
<tr>
<td>ILLEGAL LOG DEVICE</td>
<td>Magtape, cassette, or a read-only device (e.g., PTR:) was specified as the log device.</td>
</tr>
<tr>
<td>INPUT ERROR</td>
<td>An error occurred while BATCH was attempting to read the compiler input file (.BAT).</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Meaning</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT FILE?</td>
<td>An input file descriptor was not specified to the BATCH compiler command line.</td>
</tr>
<tr>
<td>IO</td>
<td>Fatal error. An Input or Output error occurred when the BATCH handler was attempting to read the .CTL file or write to the log file. The probable cause of this error is a log file overflow.</td>
</tr>
<tr>
<td>LOG DEVICE ERROR</td>
<td>An error occurred on the log device.</td>
</tr>
<tr>
<td>LU</td>
<td>Fatal error. A Lock Up occurred in the BATCH handler because it could not find a free channel or a channel it could use. This can happen if all channels are opened for magtape or cassette operations within the BATCH stream.</td>
</tr>
<tr>
<td>NO CONTROL FILE</td>
<td>The .CTL file was probably sent to a nonfile-structured device (e.g., LP:) without using the /N switch to inhibit execution.</td>
</tr>
<tr>
<td>OUTPUT DEVICE FULL</td>
<td>The temporary file (.CTL) created by BATCH is too large for the specified device. Try compressing the device with PIP, or use another device.</td>
</tr>
<tr>
<td>OUTPUT ERROR</td>
<td>An error occurred while BATCH was attempting to write the compiler output file (.CTL) or magtape or cassette was specified as the .CTL output device.</td>
</tr>
<tr>
<td>OUTPUT FILE NOT OPEN</td>
<td>Fatal error. BATCH attempted to write on a .CTL file without opening it.</td>
</tr>
<tr>
<td>RETURN FROM CALL ERROR</td>
<td>Fatal error. BATCH cannot read control file which called a subprogram. BATCH cannot resume execution on return from call.</td>
</tr>
<tr>
<td>TOO MANY OUTPUT FILES</td>
<td>A third output file was specified to the BATCH compiler; only two output files may be specified.</td>
</tr>
<tr>
<td>!UNLOAD BA!</td>
<td>Prompting message printed when /U switch is given to the compiler.</td>
</tr>
</tbody>
</table>
APPENDIX A

ASSEMBLY, LINK, AND BUILD INSTRUCTIONS

The information that formerly appeared in this appendix has now been incorporated into the RT-ll System Generation Manual, DEC-ll-ORGMA-A-D.
APPENDIX B

COMMAND AND SWITCH SUMMARIES

Command and switch summaries of the various RT-11 system and utility programs are grouped here for the user's convenience. Refer to the appropriate chapter for details.

B.1 KEYBOARD MONITOR (Chapter 2)

B.1.1 Command Summary

Only those command characters underlined need be entered; all command lines are terminated by typing a carriage return.

<table>
<thead>
<tr>
<th>Command Format</th>
<th>Used Under</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGN dev:udev</td>
<td>F/B,S/J</td>
<td>Assigns a user-defined name (udev) as an alternate name for a device (dev). Deassigns synonyms when used without any arguments.</td>
</tr>
<tr>
<td>B location</td>
<td>F/B,S/J</td>
<td>Sets a relocation base (location), which is an octal address to be used as a base address for subsequent Examine and Deposit commands.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>F/B,S/J</td>
<td>Causes all currently open files to become permanent. (B only)</td>
</tr>
<tr>
<td>DATE dd-mmm-yy</td>
<td>F/B,S/J</td>
<td>Enters the indicated day-month-year (dd-mmm-yy); this date is then assigned to newly created files, new device directory entries, and listing output. When used without an argument, the current date (as entered) is printed.</td>
</tr>
<tr>
<td>D location = value1,value2,...,valuen</td>
<td>F/B,S/J</td>
<td>Deposits the specified values starting at the given location (location represents an octal address which is added to the base address to obtain the actual address at which values will be deposited).</td>
</tr>
<tr>
<td>Command Format</td>
<td>Used Under</td>
<td>Explanation</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>E location m-location n</td>
<td>F/B,S/J</td>
<td>Prints the contents of the specified locations in octal on the console terminal (location represents an octal address which is added to the base address to obtain the actual address examined).</td>
</tr>
<tr>
<td>FRUN dev:filnam.ext/N:n/S:n/P</td>
<td>F/B (F only)</td>
<td>Initiates a foreground job which exists on the device indicated (dev) under the specified filename and extension. /N:n is optionally used to allocate n (decimal) words over and above actual program size; /S:n is optionally used to allocate n words for stack space; /P is used for debugging purposes (the load address is printed but the program must be started using RSUME).</td>
</tr>
<tr>
<td>GET dev:filnam.ext</td>
<td>F/B,S/J</td>
<td>Loads the specified memory image file (filnam.ext) into memory from the indicated device (dev:).</td>
</tr>
<tr>
<td>GT OFF</td>
<td>F/B,S/J</td>
<td>Used (after GT ON) to clear the display processor and resume printout on the console terminal.</td>
</tr>
<tr>
<td>GT ON/L:n/T:n</td>
<td>F/B,S/J</td>
<td>Enables the display processor so that the display screen replaces the console as the terminal output device. /L:n may be optionally used to designate the number of lines to display (12&quot; screen - 1&lt;=n&lt;=37 octal; 17&quot; screen - 1&lt;=n&lt;=50 octal). /T:n may be optionally used to indicate the top position of the scroll display (12&quot; screen - 1&lt;=n&lt;=1350 octal; 17&quot; screen - 1&lt;=n&lt;=1750 octal).</td>
</tr>
<tr>
<td>INITIALIZE</td>
<td>F/B,S/J (B only)</td>
<td>Resets background system tables; makes nonresident all handlers not loaded and purges background's I/O channels.</td>
</tr>
<tr>
<td>LOAD dev,...</td>
<td>F/B,S/J</td>
<td>Makes a device handler resident for use.</td>
</tr>
<tr>
<td>LOAD dev=B,dev=F,...</td>
<td>F/B</td>
<td>Makes a device handler resident for use with background and foreground jobs.</td>
</tr>
<tr>
<td>R filnam.ext</td>
<td>F/B,S/J (B only)</td>
<td>Loads the specific memory image file (filnam.ext) into memory from the system device and starts execution.</td>
</tr>
<tr>
<td>REENTER</td>
<td>F/B,S/J</td>
<td>Starts a program at its reentry address (i.e., its start address -2).</td>
</tr>
<tr>
<td>RSUME</td>
<td>F/B (F only)</td>
<td>Resumes execution of a foreground program where it was suspended.</td>
</tr>
<tr>
<td>RUN dev:filnam.ext</td>
<td>F/B,S/J (B only)</td>
<td>Loads the specified memory image file (filnam.ext) into memory from the indicated device (dev:) and starts execution.</td>
</tr>
</tbody>
</table>
## Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command Format</th>
<th>Used Under</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE dev:filnam.ext areal,area2-arean</td>
<td>F/B,S/J</td>
<td>Writes the area(s) of user memory specified into the named file (filnam.ext) in save image format. Memory is transferred in 256-word blocks.</td>
</tr>
<tr>
<td>SET dev:{NO}option=value</td>
<td>F/B,S/J</td>
<td>Used to change device (dev) handler characteristics and certain system configuration parameters. See Table 2-5 (Section 2.7.2.8) for a list of options.</td>
</tr>
<tr>
<td>START address</td>
<td>F/B,S/J</td>
<td>Begins execution of the program currently in memory at the specified address. If an address is not indicated, the starting address in location 40 is used.</td>
</tr>
<tr>
<td>SUSPEND</td>
<td>F/B</td>
<td>Suspends execution of the foreground job currently running.</td>
</tr>
<tr>
<td>TIME hh:mm:ss</td>
<td>F/B,S/J</td>
<td>Enters time of day in hours, minutes, seconds past midnight (hh:mm:ss). If all three arguments are omitted, the current time of day is output.</td>
</tr>
<tr>
<td>UNLOAD dev,dev,...</td>
<td>F/B,S/J</td>
<td>Makes previously loaded handlers (dev) nonresident and frees the memory space they were using.</td>
</tr>
</tbody>
</table>

### B.1.2 Special Function Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Used Under</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL A</td>
<td>F/B,S/J</td>
<td>Valid when the monitor GT ON command has been typed and the display is in use. Does not echo on the terminal. Used after CTRL S has been typed to effectively page output.</td>
</tr>
<tr>
<td>CTRL B</td>
<td>F/B</td>
<td>Echoes B› on the terminal and causes all keyboard input to be directed to the background job. At least one line of output will be taken from the background job (the foreground job has priority and control will revert to it if it has output). B› does not echo if output is already coming from the background job. (Control can be redirected to the foreground job via CTRL F.)</td>
</tr>
</tbody>
</table>
## Command and Switch Summaries

<table>
<thead>
<tr>
<th>Key</th>
<th>Used Under</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL C</td>
<td>F/B,S/J</td>
<td>Echoes tC on the terminal, interrupts current program execution, and returns control to the Keyboard Monitor. If a program is waiting for terminal input or is using the device handler TT: for input, typing a single CTRL C interrupts execution and returns control to the monitor command level. Otherwise, two CTRL C's must be typed in order to interrupt execution.</td>
</tr>
<tr>
<td>CTRL E</td>
<td>F/B,S/J</td>
<td>Valid when the monitor GT ON command has been typed and the display is in use. Does not echo on the terminal, but causes all I/O to appear on both the display screen and the console terminal simultaneously. A second CTRL E disables console terminal output.</td>
</tr>
<tr>
<td>CTRL F</td>
<td>F/B</td>
<td>Echoes F&gt; on the terminal and directs all keyboard input to the foreground job and all output to be taken from the foreground job. Control remains with the foreground job until redirected to the background job (via CTRL B) or until the foreground job terminates.</td>
</tr>
<tr>
<td>CTRL O</td>
<td>F/B,S/J</td>
<td>Echoes tO on the terminal and causes suppression of teleprinter output while continuing program execution. Teleprinter output is reenabled when one of the following occurs: 1. A second CTRL O is typed 2. A return to the monitor is indicated via CTRL C 3. The running program issues a .RCTRO directive (see Chapter 9)</td>
</tr>
<tr>
<td>CTRL Q</td>
<td>F/B,S/J</td>
<td>Does not echo. Resumes printing characters on the terminal from the point at which printing was previously stopped (via CTRL S).</td>
</tr>
<tr>
<td>CTRL S</td>
<td>F/B,S/J</td>
<td>Does not echo. Temporarily suspends output to the terminal until a CTRL Q is typed. If GT ON is in effect, each subsequent CTRL A causes output to proceed until the screen has been refilled once.</td>
</tr>
<tr>
<td>CTRL U</td>
<td>F/B,S/J</td>
<td>Echoes tU followed by a carriage return on the terminal and deletes the current input line.</td>
</tr>
<tr>
<td>CTRL Z</td>
<td>F/B,S/J</td>
<td>Echoes tZ on the terminal and terminates input when used with the terminal device handler (TT:).</td>
</tr>
</tbody>
</table>
Command and Switch Summaries

<table>
<thead>
<tr>
<th>Key</th>
<th>Used Under</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUBOUT</td>
<td>F/B,S/J</td>
<td>Deletes the last character from the current line. Echoes a backslash plus the character deleted; each succeeding RUBOUT deletes and echoes another character; an enclosing backslash is printed when a key other than RUBOUT is typed.</td>
</tr>
</tbody>
</table>

B.2 EDITOR (Chapter 3)

B.2.1 Command Arguments

<table>
<thead>
<tr>
<th>Format</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>A decimal integer (in the range -16383 to +16383) which may, except where noted, be preceded by a + or -. Whenever an argument is acceptable in a command, its absence implies an argument of 1.</td>
</tr>
<tr>
<td>0</td>
<td>Refers to the beginning of the current line.</td>
</tr>
<tr>
<td>/</td>
<td>Refers to the end of the text in the current Text Buffer.</td>
</tr>
<tr>
<td>=</td>
<td>Is used with the J, D and C commands only and represents (-n), where n is equal to the length of the last text argument used.</td>
</tr>
</tbody>
</table>

B.2.2 Input and Output Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT BACKUP</td>
<td>EB dev:filnam.ext[n]$</td>
<td>Opens a file for editing, creating a backup copy (.BAK).</td>
</tr>
<tr>
<td>EDIT READ</td>
<td>ER dev:filnam.ext$</td>
<td>Opens a file for input.</td>
</tr>
<tr>
<td>EDIT WRITE</td>
<td>EW dev:filnam.ext[n]$</td>
<td>Creates a new file for output.</td>
</tr>
<tr>
<td>END FILE</td>
<td>EF</td>
<td>Closes the current output file without performing any further input/output operations.</td>
</tr>
<tr>
<td>EXIT</td>
<td>EX</td>
<td>Outputs the remainder of the input file to the output file and returns control to the monitor.</td>
</tr>
<tr>
<td>LIST</td>
<td>(-)nL</td>
<td>Prints a specified number of lines on the console terminal.</td>
</tr>
<tr>
<td></td>
<td>0L</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/L</td>
<td></td>
</tr>
</tbody>
</table>

B-5
## Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command</th>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT</td>
<td>nN</td>
<td>Outputs the contents of the Text Buffer to the output file, clears the buffer, and reads in the next page of the input file.</td>
</tr>
<tr>
<td>READ</td>
<td>R</td>
<td>Reads a page of text from the input file and appends it to the contents of the buffer.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>V</td>
<td>Prints the current text line (the line containing the pointer) on the console terminal.</td>
</tr>
<tr>
<td>WRITE</td>
<td>(-)nW</td>
<td>Outputs a specified number of lines of text from the Text Buffer to the output file.</td>
</tr>
</tbody>
</table>

### B.2.3 Pointer Relocation Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVANCE</td>
<td>(-)nA</td>
<td>Moves the pointer over a specified number of lines in the Text Buffer. The pointer is positioned at the beginning of the line.</td>
</tr>
<tr>
<td>BEGINNING</td>
<td>B</td>
<td>Moves the current location pointer to the beginning of the Text Buffer.</td>
</tr>
<tr>
<td>JUMP</td>
<td>(-)nJ</td>
<td>Moves the pointer over a specified number of characters in the Text Buffer.</td>
</tr>
</tbody>
</table>

### B.2.4 Search Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND</td>
<td>nPtext$</td>
<td>Beginning at the current location pointer, searches the entire text file for the nth occurrence of the specified character string. Pages of text are read into the Text Buffer, searched, and then written to the output file until the text string is found.</td>
</tr>
<tr>
<td>GET</td>
<td>nGtext$</td>
<td>Searches the contents of the Text Buffer, beginning at the current location pointer, for the next occurrence of the text string.</td>
</tr>
<tr>
<td>POSITION</td>
<td>nPtext$</td>
<td>Searches the input file for the nth occurrence of the text string; if the text string is not found, the buffer is cleared and a new page is read from the input file.</td>
</tr>
</tbody>
</table>
Command and Switch Summaries

B.2.5 Text Modification Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANGE</td>
<td>(-)nCtext$</td>
<td>Replaces n characters, beginning at the pointer, with the indicated text string.</td>
</tr>
<tr>
<td></td>
<td>0Ctext$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/Ctext$</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>(-)nD</td>
<td>Removes a specified number of characters from the Text Buffer, beginning at the current location pointer.</td>
</tr>
<tr>
<td></td>
<td>0D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>=D</td>
<td></td>
</tr>
<tr>
<td>EXCHANGE</td>
<td>(-)nXtext$</td>
<td>Replaces n lines, beginning at the pointer, with the indicated text string.</td>
</tr>
<tr>
<td></td>
<td>0Xtext$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/Xtext$</td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td>Itext$</td>
<td>Inserts text immediately following the current location pointer; an ALTMODE terminates the text.</td>
</tr>
<tr>
<td>KILL</td>
<td>(-)nK</td>
<td>Removes n lines from the Text Buffer beginning at the current location pointer.</td>
</tr>
<tr>
<td></td>
<td>0K</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/K</td>
<td></td>
</tr>
</tbody>
</table>

B.2.6 Utility Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT CONSOLE</td>
<td>EC</td>
<td>If scroller is in use, EC returns scroller to its normal mode (using full screen for display of text and commands); if scroller is not in use, EC clears screen and returns control to console terminal (following ED).</td>
</tr>
<tr>
<td>EDIT DISPLAY</td>
<td>ED</td>
<td>If scroller is in use, ED recalls the text window (following EC) and arranges text and commands on screen; if scroller is not in use, ED displays text window only.</td>
</tr>
<tr>
<td>EXECUTE MACRO</td>
<td>nE</td>
<td>Executes the command string specified in the last macro command.</td>
</tr>
<tr>
<td>MACRO</td>
<td>M/command string/</td>
<td>Inserts a command string into the Macro Buffer.</td>
</tr>
<tr>
<td></td>
<td>0M</td>
<td>Clears the Macro Buffer and reclaims the area for text.</td>
</tr>
<tr>
<td></td>
<td>M//</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td>nS</td>
<td>Copies the specified number of lines, beginning at the pointer, into the Save Buffer.</td>
</tr>
<tr>
<td>UNSAVE</td>
<td>U</td>
<td>Inserts the entire contents of the Save Buffer into the Text Buffer at the position of the current location pointer.</td>
</tr>
</tbody>
</table>

B-7
Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command</th>
<th>Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT VERSION</td>
<td>EV</td>
<td>Displays the version number of the Editor on the console terminal.</td>
</tr>
<tr>
<td>EDIT LOWER</td>
<td>EL</td>
<td>Enables editing in upper- and lower-case.</td>
</tr>
<tr>
<td>EDIT UPPER</td>
<td>EU</td>
<td>Returns editing to upper-case only (after EL).</td>
</tr>
</tbody>
</table>

B.2.7 Immediate Mode Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL N</td>
<td>Advances the pointer (cursor) to the beginning of the next line.</td>
</tr>
<tr>
<td>CTRL G</td>
<td>Moves the pointer (cursor) to the beginning of the previous line.</td>
</tr>
<tr>
<td>CTRL D</td>
<td>Moves the pointer (cursor) forward by one character.</td>
</tr>
<tr>
<td>CTRL V</td>
<td>Moves the pointer (cursor) back by one character.</td>
</tr>
<tr>
<td>RUBOUT</td>
<td>Deletes the character immediately preceding the pointer (cursor).</td>
</tr>
<tr>
<td>ALTMODE (two)</td>
<td>Enters Immediate Mode. Returns control to Editor Command Mode.</td>
</tr>
<tr>
<td>(one only)</td>
<td></td>
</tr>
<tr>
<td>Any character other</td>
<td>Inserts the character as text positioned immediately before the pointer (cursor).</td>
</tr>
<tr>
<td>than the above (with the exception of CTRL C)</td>
<td></td>
</tr>
</tbody>
</table>

B.2.8 Key Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTMODE</td>
<td>Echoes $. A single ALTMODE terminates a text string. A double ALTMODE executes the command string. (When used alone on a line, two ALTMODES cause control to enter Immediate Mode, while a single ALTMODE returns control to Editor Command Mode.)</td>
</tr>
<tr>
<td>CTRL C</td>
<td>Echoes at the terminal as ^C and a carriage return. Terminates execution of EDIT commands, closes any open files, and returns to monitor command mode.</td>
</tr>
<tr>
<td>CTRL O</td>
<td>Echoes ^O and a carriage return. Inhibits printing on the terminal until completion of the current command string. Typing a second CTRL O resumes output.</td>
</tr>
<tr>
<td>CTRL U</td>
<td>Echoes ^U and a carriage return. Deletes all the characters on the current terminal input line.</td>
</tr>
</tbody>
</table>
Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUBOUT</td>
<td>Deletes character from the current line.</td>
</tr>
<tr>
<td>TAB</td>
<td>Spaces to the next tab stop. Tab stops are positioned every eight spaces on the terminal.</td>
</tr>
<tr>
<td>CTRL X</td>
<td>Echoes ^X and a carriage return. CTRL X causes the Editor to ignore the entire command string currently being entered. The Editor prints a &lt;CR&gt;&lt;LF&gt; and an asterisk to indicate that the user may enter another command.</td>
</tr>
</tbody>
</table>

B.3 PIP (Chapter 4)

B.3.1 Switch Summary

<table>
<thead>
<tr>
<th>Switch</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/A</td>
<td>Copies file(s) in ASCII mode; ignores nulls and rubouts; converts to 7-bit ASCII.</td>
</tr>
<tr>
<td>/B</td>
<td>Copies files in formatted binary mode.</td>
</tr>
<tr>
<td>/C</td>
<td>Used in conjunction with another switch; causes only files with current date to be included in the specified operation.</td>
</tr>
<tr>
<td>/D</td>
<td>Deletes file(s) from specified device.</td>
</tr>
<tr>
<td>/E</td>
<td>Lists the device directory including unused spaces and their sizes. Sequence numbers are listed for cassettes.</td>
</tr>
<tr>
<td>/F</td>
<td>Prints a short directory (filenames only) of the specified device.</td>
</tr>
<tr>
<td>/G</td>
<td>Ignores any input errors which occur during a file transfer and continues copying.</td>
</tr>
<tr>
<td>/I or no switch</td>
<td>Copies file(s) in image mode (byte by byte).</td>
</tr>
<tr>
<td>/K</td>
<td>Scans the specified device and types the absolute block numbers (in octal) of any bad blocks on the device.</td>
</tr>
<tr>
<td>/L</td>
<td>Lists the directory of the specified device. Sequence numbers are listed for cassettes.</td>
</tr>
<tr>
<td>/M:n</td>
<td>Used when I/O transfers involve either cassette or magtape, n represents the numeric position of the file to be accessed in relation to the physical position of the cassette or magtape on the drive.</td>
</tr>
</tbody>
</table>
### Command and Switch Summaries

<table>
<thead>
<tr>
<th>Switch</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/N:n</td>
<td>Used with /Z to specify the number of directory blocks (n) to allocate to the directory.</td>
</tr>
<tr>
<td>/O</td>
<td>Bootstraps the specified device (DT0, RKn, RF, DPn, DSn, DXn).</td>
</tr>
<tr>
<td>/Q</td>
<td>Causes PIP to type each filename which is eligible for a wild card operation and to ask for a confirmation of its inclusion in the operation.</td>
</tr>
<tr>
<td>/R</td>
<td>Renames the specified file.</td>
</tr>
<tr>
<td>/S</td>
<td>Compresses the files on the specified directory device so that free blocks are combined into one area.</td>
</tr>
<tr>
<td>/T</td>
<td>Extends number of blocks allocated for a file.</td>
</tr>
<tr>
<td>/U</td>
<td>Copies the bootstrap from the specified file into absolute blocks 0 and 2 of the specified device.</td>
</tr>
<tr>
<td>/V</td>
<td>Types the version number of the PIP program being used.</td>
</tr>
<tr>
<td>/W</td>
<td>Includes the absolute starting block and any extra directory words in the directory listing for each file on the device (numbers in octal). Used with /F, /L, or /E.</td>
</tr>
<tr>
<td>/X</td>
<td>Copies files individually (without concatenation).</td>
</tr>
<tr>
<td>/Y</td>
<td>Causes system flies and .BAD files to be operated on by the command specified.</td>
</tr>
<tr>
<td>/Z:n</td>
<td>Zeroes (initializes) the directory of the specified device; n is used to allocate extra words per directory entry. When used with /N, the number of directory segments for entries may be specified.</td>
</tr>
</tbody>
</table>

### B.4 MACRO/CREF (Chapter 5)

Refer to Appendix C for a complete summary of MACRO features. CREF switches are also included in that appendix.
Command and Switch Summaries

B.5 LINKER (Chapter 6)

B.5.1 Switch Summary

The Linker switches (and the command line on which each must appear) are:

<table>
<thead>
<tr>
<th>Switch Name</th>
<th>Command Line</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/A</td>
<td>1st</td>
<td>Alphabetizes the entries in the load map.</td>
</tr>
<tr>
<td>/B:n</td>
<td>1st</td>
<td>Bottom address of program is indicated as n (illegal for foreground links).</td>
</tr>
<tr>
<td>/C</td>
<td>any</td>
<td>Continues input files on another command line (must be used with /O).</td>
</tr>
<tr>
<td>/F</td>
<td>1st</td>
<td>Indicates that the Linker will use the default FORTRAN library, FORLIB.OBJ.</td>
</tr>
<tr>
<td>/I</td>
<td>1st</td>
<td>Includes the global symbols to be searched from the library.</td>
</tr>
<tr>
<td>/L</td>
<td>1st</td>
<td>Produces an output file in LDA format (illegal for foreground links).</td>
</tr>
<tr>
<td>/M:n</td>
<td>1st</td>
<td>Allows terminal keyboard specification of the user's stack address. n represents an optional 6-digit unsigned octal number.</td>
</tr>
<tr>
<td>/O:n</td>
<td>any but the 1st</td>
<td>Indicates that the program will be an overlay structure; n specifies the overlay region to which the module is assigned.</td>
</tr>
<tr>
<td>/R</td>
<td>1st</td>
<td>Produces output in REL format; only files in REL format will run in the foreground (REL format files may not be run under a Single-Job system).</td>
</tr>
<tr>
<td>/S</td>
<td>1st</td>
<td>Allows the maximum amount of space in memory to be available for the Linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.)</td>
</tr>
<tr>
<td>/T or /T:n</td>
<td>1st</td>
<td>Transfer address is to be specified at terminal keyboard via n.</td>
</tr>
</tbody>
</table>
Command and Switch Summaries

B.6 LIBRARIAN (Chapter 7)

B.6.1 Switch Summary

The Librarian (LIBR) switches (and the command line on which each must appear) are:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Command Line</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/C</td>
<td>Any</td>
<td>The command is too long for the current line and is continued on the next line.</td>
</tr>
<tr>
<td>/D</td>
<td>1st</td>
<td>Deletes modules from a library file.</td>
</tr>
<tr>
<td>/G</td>
<td>1st</td>
<td>Global deletion; deletes entry points from the library directory.</td>
</tr>
<tr>
<td>/R</td>
<td>1st</td>
<td>Replaces modules in a library file.</td>
</tr>
<tr>
<td>/U</td>
<td>1st</td>
<td>Update; inserts and replaces modules in a library file.</td>
</tr>
</tbody>
</table>

B.7 ODT (Chapter 8)

B.7.1 Command Summary

In the command format shown below, r represents a relocatable expression and n represents an octal number.

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN</td>
<td></td>
<td>Closes open location and accepts the next command.</td>
</tr>
<tr>
<td>LINE FEED</td>
<td></td>
<td>Closes current location and opens next sequential location.</td>
</tr>
<tr>
<td>↑ or ^</td>
<td>↑ or ^</td>
<td>Opens previous location.</td>
</tr>
<tr>
<td>← or _</td>
<td>← or _</td>
<td>Indexes the contents of the opened location by the contents of the PC and opens the resulting location.</td>
</tr>
<tr>
<td>＞</td>
<td>＞</td>
<td>Uses the contents of the opened location as a relative branch instruction and opens the referenced location.</td>
</tr>
<tr>
<td>＜</td>
<td>＜</td>
<td>Returns to sequence prior to last ＞, ＞, or ← command and opens the succeeding location.</td>
</tr>
<tr>
<td>＠</td>
<td>＠</td>
<td>Uses the contents of the opened location as an absolute address and opens that location.</td>
</tr>
</tbody>
</table>
### Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>/</td>
<td>Reopens the last opened location.</td>
</tr>
<tr>
<td>r/</td>
<td></td>
<td>Opens the word at location r.</td>
</tr>
<tr>
<td>\ \</td>
<td>r\</td>
<td>Reopens the last opened byte (SHIFT L).</td>
</tr>
<tr>
<td>l n!</td>
<td></td>
<td>Opens the byte at location r.</td>
</tr>
<tr>
<td>$ sn/</td>
<td></td>
<td>After a word or byte has been opened, prints the address of the opened location relative to relocation register n. If n is omitted, ODT selects the relocation register whose contents are closest to but less than or equal to the address of the opened location.</td>
</tr>
<tr>
<td>$B/</td>
<td></td>
<td>Opens the first word of the breakpoint table.</td>
</tr>
<tr>
<td>$C/</td>
<td></td>
<td>Opens Constant Register.</td>
</tr>
<tr>
<td>$F/</td>
<td></td>
<td>Opens Format Register.</td>
</tr>
<tr>
<td>$P/</td>
<td></td>
<td>Opens Priority Register.</td>
</tr>
<tr>
<td>$R/</td>
<td></td>
<td>Opens first Relocation Register (register 0).</td>
</tr>
<tr>
<td>$S/</td>
<td></td>
<td>Opens Status Register.</td>
</tr>
<tr>
<td>A r;A</td>
<td></td>
<td>Starting at location r, prints n bytes in their ASCII format; then inputs n bytes from the terminal starting at location r.</td>
</tr>
<tr>
<td>B</td>
<td>;B</td>
<td>Removes all Breakpoints.</td>
</tr>
<tr>
<td>r;B</td>
<td></td>
<td>Sets Breakpoint at location r.</td>
</tr>
<tr>
<td>r;nB</td>
<td></td>
<td>Sets Breakpoint n at location r.</td>
</tr>
<tr>
<td>;nB</td>
<td></td>
<td>Removes the nth Breakpoint.</td>
</tr>
<tr>
<td>C r;C</td>
<td></td>
<td>Prints the value of r and stores it in the Constant Register.</td>
</tr>
<tr>
<td>E r;E</td>
<td></td>
<td>Searches for instructions that reference effective address r.</td>
</tr>
<tr>
<td>F ;F</td>
<td></td>
<td>Fills memory words with contents of the Constant Register.</td>
</tr>
<tr>
<td>G r;G</td>
<td></td>
<td>Goes to location r and starts program.</td>
</tr>
<tr>
<td>I ;I</td>
<td></td>
<td>Fills memory bytes with the low-order 8 bits of the Constant Register.</td>
</tr>
<tr>
<td>O r;O</td>
<td></td>
<td>Calculates offset from currently open location to r.</td>
</tr>
<tr>
<td>P ;P</td>
<td></td>
<td>Proceeds with program execution from breakpoint. In single instruction mode only, executes next instruction.</td>
</tr>
</tbody>
</table>

B-13
### Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>k;P</td>
<td></td>
<td>Proceeds with program execution from breakpoint; stops after encountering the breakpoint k times. In single instruction mode only, executes next k instructions.</td>
</tr>
<tr>
<td>R</td>
<td>;R</td>
<td>Sets all Relocation Registers to -1 (highest address value).</td>
</tr>
<tr>
<td></td>
<td>;nR</td>
<td>Sets Relocation Register n to -1.</td>
</tr>
<tr>
<td>r;nR</td>
<td></td>
<td>Sets Relocation Register n to the value of r. If n is omitted, it is assumed to be 0.</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>Selects the Relocation Register whose contents are closest to but less than or equal to contents of the opened location. Subtracts the contents of the register from the contents of the opened word and prints the result.</td>
</tr>
<tr>
<td>nR</td>
<td></td>
<td>Subtracts the contents of the Relocation Register n from the contents of the opened word and prints the result.</td>
</tr>
<tr>
<td>S</td>
<td>;S</td>
<td>Disables single instruction mode; reenables breakpoints.</td>
</tr>
<tr>
<td></td>
<td>;nS</td>
<td>Enables single instruction mode (n can have any value and is not significant); disables breakpoints.</td>
</tr>
<tr>
<td>W</td>
<td>r;W</td>
<td>Searches for words with bit patterns which match r.</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>Performs a Radix 50 unpack of the binary contents of the current opened word; then permits the storage of a new Radix 50 binary number in the same location.</td>
</tr>
</tbody>
</table>

### B.8 PROGRAMMED REQUESTS (Chapter 9)

Appendix E summaries the programmed requests available under RT-11, Version 2C.

### B.9 BATCH (Chapter 12)

Only those characters underlined need be entered.

#### B.9.1 Switch Summary

(Command Field)

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/BANNER</td>
<td>Prints header of job on the log file.</td>
</tr>
<tr>
<td>/CREF</td>
<td>Specifies that a cross reference listing is to be produced during compilation.</td>
</tr>
<tr>
<td>/DELETE</td>
<td>Indicates that input is to be deleted after the operation is complete.</td>
</tr>
</tbody>
</table>

January 1976

B-14
Command and Switch Summaries

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/DOLLARS</td>
<td>Indicates that the data following this command may have an $ in the first character position of a line. Reading of the data is terminated by one of the following BATCH commands:</td>
</tr>
<tr>
<td></td>
<td>$JOB</td>
</tr>
<tr>
<td></td>
<td>$SEQUENCE</td>
</tr>
<tr>
<td></td>
<td>$EOD</td>
</tr>
<tr>
<td></td>
<td>$EOJ</td>
</tr>
<tr>
<td></td>
<td>or by a physical end-of-file on the BATCH input stream.</td>
</tr>
<tr>
<td>/LIBRARY</td>
<td>Includes the default library in the link operation.</td>
</tr>
<tr>
<td>/LIST</td>
<td>Produces a temporary listing on the listing device (LST:) or writes data images on the log device (LOG:).</td>
</tr>
<tr>
<td>/MAP</td>
<td>Produces a temporary linkage map on the listing device (LST:).</td>
</tr>
<tr>
<td>/OBJECT</td>
<td>Produces a temporary object file as output of compilation or assembly.</td>
</tr>
<tr>
<td>/RT11</td>
<td>Sets BATCH to operate in RT-11 mode.</td>
</tr>
<tr>
<td>/RUN</td>
<td>Causes linking (if necessary) and execution of programs compiled since the last link and go operation or start of job.</td>
</tr>
<tr>
<td>/TIME</td>
<td>Writes the time of day to the log file when commands are executed.</td>
</tr>
<tr>
<td>/UNIQUE</td>
<td>Checks for unique spelling of switches and keynames.</td>
</tr>
<tr>
<td>/WAIT</td>
<td>Pauses to wait for operator action.</td>
</tr>
<tr>
<td>/WRITE</td>
<td>Indicates that the operator is to WRITE-ENABLE a specified device or volume.</td>
</tr>
<tr>
<td>/NOBANNER</td>
<td>Does not print a job header.</td>
</tr>
<tr>
<td>/NOCREF</td>
<td>Does not create a cross reference listing.</td>
</tr>
<tr>
<td>/NODELETE</td>
<td>Does not delete input after operation is complete.</td>
</tr>
<tr>
<td>/NODOLLARS</td>
<td>Following data may not have a $ in the first character position; a $ in the first character position signifies a BATCH control command.</td>
</tr>
<tr>
<td>/NOLIBRARY</td>
<td>Does not include the default library in the link operation.</td>
</tr>
</tbody>
</table>
## Command and Switch Summaries

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/NOLIST</td>
<td>Does not produce a listing.</td>
</tr>
<tr>
<td>/NOMAP</td>
<td>Does not create MAP file.</td>
</tr>
<tr>
<td>/NOOBJECT</td>
<td>Does not produce object file as output of compilation.</td>
</tr>
<tr>
<td>/NORT11</td>
<td>Does not set BATCH to operate in RT-ll mode.</td>
</tr>
<tr>
<td>/NORUN</td>
<td>Does not execute or link and execute the program after performing the specified command.</td>
</tr>
<tr>
<td>/NOTIME</td>
<td>Does not write time of day to log file.</td>
</tr>
<tr>
<td>/NOUNIQUE</td>
<td>Does not check for unique spelling.</td>
</tr>
<tr>
<td>/NOWAIT</td>
<td>Does not pause for operator action.</td>
</tr>
<tr>
<td>/NOWRITE</td>
<td>Indicates that no writes are allowed or that the volume specification is read-only; the operator is informed and must WRITE-LOCK the appropriate device.</td>
</tr>
</tbody>
</table>

(Specification Field)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>/BASIC</td>
<td>BASIC source file.</td>
</tr>
<tr>
<td>/EXECUTABLE</td>
<td>Indicates that runnable program image file is to be created as the result of a link operation.</td>
</tr>
<tr>
<td>/FORTRAN</td>
<td>FORTRAN source file.</td>
</tr>
<tr>
<td>/INPUT</td>
<td>Input file; default if no switches are specified.</td>
</tr>
<tr>
<td>/LIBRARY</td>
<td>Library file to be included in link operation (prior to default library).</td>
</tr>
<tr>
<td>/LIST</td>
<td>Listing file.</td>
</tr>
<tr>
<td>/LOGICAL</td>
<td>Indicates that the device is a logical device name.</td>
</tr>
<tr>
<td>/MACRO</td>
<td>MACRO or EXPAND source file.</td>
</tr>
<tr>
<td>/MAP</td>
<td>Linker map file.</td>
</tr>
<tr>
<td>/OBJECT</td>
<td>Object file (output of assembly or compilation).</td>
</tr>
<tr>
<td>/OUTPUT</td>
<td>Output file.</td>
</tr>
<tr>
<td>/PHYSICAL</td>
<td>Indicates physical device name.</td>
</tr>
</tbody>
</table>
Command and Switch Summaries

Switch Meanings
/SOURCE Indicates source file.
/VID Volume identification.

B.9.2 Command Summary

Command Meanings
$BASIC Compiles a BASIC source program.
$CALL Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.
$CHAIN Passes execution to another BATCH file.
$COPY Copies files.
$CREATE Creates new files from data included in BATCH stream.
$DATA Indicates the start of data.
$DELETE Deletes files.
$DIRECTORY Provides a directory of the specified device.
$DISMOUNT Signals operator to dismount a volume from a device and deassigns a logical device name.
$EOD Indicates the end of data.
$EOJ Indicates the end of a job.
$FORTRAN Compiles a FORTRAN source program.
$JOB Indicates the start of a job.
$LIBRARY Specifies library that is to be used in linkage operations.
$LINK Links modules for execution.
$MACRO Assembles a MACRO source program.
$MESSAGE Issues message to the operator.
$MOUNT Signals operator to mount a volume on a device and optionally assigns a logical device name.
$PRINT Prints files.
$RT11 Specifies that the following lines are RT-11 mode commands.
Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$RUN</td>
<td>Causes a program to execute.</td>
</tr>
<tr>
<td>$SEQUENCE</td>
<td>Assigns an arbitrary identification number to a job.</td>
</tr>
</tbody>
</table>

B.10  DUMP (Appendix I)

B.10.1 Switch Summary

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B</td>
<td>Outputs octal bytes.</td>
</tr>
<tr>
<td>/E:n</td>
<td>Ends output at block n.</td>
</tr>
<tr>
<td>/G</td>
<td>Ignores input errors.</td>
</tr>
<tr>
<td>/N</td>
<td>Suppresses ASCII output.</td>
</tr>
<tr>
<td>/O:n</td>
<td>Outputs only block number n.</td>
</tr>
<tr>
<td>/S:n</td>
<td>Starts output with block n.</td>
</tr>
<tr>
<td>/W</td>
<td>Outputs octal words.</td>
</tr>
<tr>
<td>/X</td>
<td>Outputs RAD50 characters.</td>
</tr>
</tbody>
</table>

B.11  FILEX (Appendix J)

B.11.1 Switch Summary

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/A</td>
<td>Indicates a character-by-character ASCII transfer in which rubouts and nulls are deleted; when /T is also used, each PDP-10 word is assumed to contain five 7-bit ASCII bytes.</td>
</tr>
<tr>
<td>/D</td>
<td>Deletes the named file from the device; valid only for DOS/BATCH and RSTS DECTape.</td>
</tr>
<tr>
<td>/F</td>
<td>Causes a &quot;fast&quot; listing of the device directory by listing filenames only.</td>
</tr>
</tbody>
</table>
Command and Switch Summaries

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/I</td>
<td>Performs an image mode transfer; if the input is either DOS/BATCH, RSTS or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /I indicates that the file resembles a file created on DECsystem-10 by MACYII, MACXII, or LNXXII with the /I switch; in this case, each DECsystem-10 36-bit word contains one PDP-11 8-bit byte in its low-order bits.</td>
</tr>
<tr>
<td>/L</td>
<td>Causes a complete listing of the device directory, including filenames, block lengths, and creation dates.</td>
</tr>
<tr>
<td>/P</td>
<td>Performs a packed image transfer; if the input is from either DOS/BATCH, RSTS-11 or RT-11, this is a word-for-word transfer; if the input is from DECsystem-10, /P indicates that the file resembles a file created on DECsystem-10 by MACYII, MACXII, or LNXXII with the /P switch, in which case each DECsystem-10 36-bit word contains four PDP-11 8-bit bytes aligned on bits 0, 8, 16, and 26. This mode is assumed if no mode switch (/A, /I) is indicated in a command line.</td>
</tr>
<tr>
<td>/S</td>
<td>Indicates the device is a DOS/BATCH (or RSTS) file-structured device.</td>
</tr>
<tr>
<td>/T</td>
<td>Indicates the device is a DECsystem-10 file-structured device.</td>
</tr>
<tr>
<td>/V</td>
<td>Types out version number of FILEX.</td>
</tr>
<tr>
<td>/Z</td>
<td>zeroes the directory of the specified device in the proper format (valid only for DOS/BATCH and RSTS DECTape).</td>
</tr>
</tbody>
</table>

B.12 SRCCOM (Appendix K)

B.12.1 Switch Summary

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B</td>
<td>Compares blank lines. Without this switch, blank lines are ignored.</td>
</tr>
<tr>
<td>/C</td>
<td>Ignores comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs).</td>
</tr>
<tr>
<td>/F</td>
<td>Includes form feeds in the output file (form feeds are still compared if /F is not used, but they are not included in the output of differences).</td>
</tr>
</tbody>
</table>
### Command and Switch Summaries

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/H</td>
<td>Types list of switches available (help text).</td>
</tr>
<tr>
<td>/L:n</td>
<td>Specifies the number of lines that determine a match (where n is an octal number &lt;=310). The default value for n is 3.</td>
</tr>
<tr>
<td>/S</td>
<td>Ignores spaces and tabs.</td>
</tr>
</tbody>
</table>

---

### B.13 PATCH (Appendix L)

#### B.13.1 Command Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/O</td>
<td>Indicates overlay-structured file.</td>
</tr>
<tr>
<td>/M</td>
<td>Indicates monitor file.</td>
</tr>
<tr>
<td>Vr;nR</td>
<td>Sets relocation register n to value Vr.</td>
</tr>
<tr>
<td>b;B</td>
<td>Sets bottom address of overlay file to b.</td>
</tr>
<tr>
<td>[s:]r,o/</td>
<td>Opens word location Vr + o in overlay segment s.</td>
</tr>
<tr>
<td>[s:]r,o</td>
<td>Opens byte location Vr + o in overlay segment s.</td>
</tr>
<tr>
<td>&lt;CR&gt;</td>
<td>Closes currently open word/byte.</td>
</tr>
<tr>
<td>&lt;LF&gt;</td>
<td>Closes currently open word/byte and opens the next one.</td>
</tr>
<tr>
<td>↑ or ^</td>
<td>Closes the currently open word/byte and opens the previous one.</td>
</tr>
<tr>
<td>$</td>
<td>Closes the currently open word and opens the word addressed by it.</td>
</tr>
<tr>
<td>F</td>
<td>Begins patching a new file.</td>
</tr>
<tr>
<td>E</td>
<td>Exits to RT-11 monitor.</td>
</tr>
</tbody>
</table>
## Command and Switch Summaries

**B.14 PATCHO (Appendix M)**

### B.14.1 Command Summary

<table>
<thead>
<tr>
<th>Command</th>
<th>Format</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BYTE</strong></td>
<td>BYTE CSECT + OFFSET = [#NAME OF [#CSECT OR #GLOBAL] OFFSET]</td>
<td>Modifies a given byte in an object module.</td>
</tr>
<tr>
<td><strong>DEC</strong></td>
<td>*DEC</td>
<td>Used when the proper checksum for the patch being made is unknown. PATCHO computes a checksum and prints out its value.</td>
</tr>
<tr>
<td><strong>DUMP</strong></td>
<td>*DUMP</td>
<td>Prints the contents of an object module in octal and causes an automatic POINT to the next module, if any, in the input file.</td>
</tr>
<tr>
<td><strong>EXIT</strong></td>
<td>*EXIT</td>
<td>Terminates the patch session by closing the file and returning control to the monitor.</td>
</tr>
<tr>
<td><strong>HELP</strong></td>
<td>*HELP</td>
<td>Prints an explanation of the PATCHO commands.</td>
</tr>
<tr>
<td><strong>LIST</strong></td>
<td>*LIST</td>
<td>Lists the names of all object modules in the input file in the order in which they appear in the file. (A POINT command should be used after a LIST.)</td>
</tr>
<tr>
<td><strong>OPEN</strong></td>
<td>*OPEN</td>
<td>Opens files for input and output.</td>
</tr>
<tr>
<td><strong>POINT</strong></td>
<td>*POINT modnam</td>
<td>Locates the specified object module (modnam) and prepares it for subsequent WORD, BYTE, or DUMP operations.</td>
</tr>
<tr>
<td><strong>WORD</strong></td>
<td>WORD CSECT + OFFSET = [#NAME OF [#CSECT OR #GLOBAL] OFFSET]</td>
<td>Modifies a given word in an object module.</td>
</tr>
</tbody>
</table>
**APPENDIX C**

**MACRO ASSEMBLER, INSTRUCTION, AND CHARACTER CODE SUMMARIES**

### C.1 ASCII CHARACTER SET

<table>
<thead>
<tr>
<th>Even Parity</th>
<th>Octal Bit</th>
<th>7-Bit Code</th>
<th>Character</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>NUL</td>
<td>Null, Tape Feed, CTRL SHIFT P.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>Start of Heading; also SOM (Start Of Message), CTRL A.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>002</td>
<td>STX</td>
<td>Start of Text; also EOA (End Of Address), CTRL B.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>003</td>
<td>ETX</td>
<td>End of Text; also EOM (End Of Message), CTRL C.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>004</td>
<td>EOT</td>
<td>End of Transmission (END); Shuts off TWX machines, CTRL D.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>005</td>
<td>ENQ</td>
<td>Enquiry (ENQRY); also WRU, CTRL E.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>006</td>
<td>ACK</td>
<td>Acknowledge; also RU, CTRL F.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>007</td>
<td>BEL</td>
<td>Rings the Bell. CTRL G.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>BS</td>
<td>Backspace; also FEO, Format Effector. Backspaces some machines, CTRL H.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>HT</td>
<td>Horizontal TAB. CTRL I.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>012</td>
<td>LF</td>
<td>Line Feed or Line Space (New Line); Advances paper to next line; duplicated by CTRL J.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>013</td>
<td>VT</td>
<td>Vertical TAB (VTAB). CTRL K.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>014</td>
<td>FF</td>
<td>FORM FEED to top of next page (PAGE). CTRL L.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>015</td>
<td>CR</td>
<td>Carriage Return to beginning of line. Duplicated by CTRL M.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>016</td>
<td>SO</td>
<td>Shift Out; Changes ribbon color to red. CTRL N.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>017</td>
<td>SI</td>
<td>Shift In; Changes ribbon color to black. CTRL O.</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>020</td>
<td>DLE</td>
<td>Data Link Escape. CTRL B (DC0).</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>021</td>
<td>DC1</td>
<td>Device Control 1, turns transmitter (reader) on, CTRL Q (X ON).</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>022</td>
<td>DC2</td>
<td>Device Control 2, turns punch or auxiliary on, CTRL R (TAPE, AUX ON).</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>023</td>
<td>DC3</td>
<td>Device Control 3, turns transmitter (reader) off, CTRL S (X OFF).</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>024</td>
<td>DC4</td>
<td>Device Control 4, turns punch or auxiliary off, CTRL T (AUX OFF).</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>025</td>
<td>NAK</td>
<td>Negative Acknowledge; also ERR, Error, CTRL U.</td>
<td></td>
</tr>
</tbody>
</table>
### MACRO Assembler, Instruction, and Character Code Summaries

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>026</td>
<td>SYN</td>
<td>Synchronous File (SYNC), CTRL V.</td>
</tr>
<tr>
<td>0</td>
<td>027</td>
<td>ETB</td>
<td>End of Transmission Block; also LEM, Logical End of Medium, CTRL W.</td>
</tr>
<tr>
<td>0</td>
<td>030</td>
<td>CAN</td>
<td>Cancel (CANCL), CTRL X.</td>
</tr>
<tr>
<td>1</td>
<td>031</td>
<td>EM</td>
<td>End of Medium, CTRL Y.</td>
</tr>
<tr>
<td>1</td>
<td>032</td>
<td>SUB</td>
<td>Substitute, CTRL Z.</td>
</tr>
<tr>
<td>0</td>
<td>033</td>
<td>ESC</td>
<td>Escape, CTRL SHIFT K.</td>
</tr>
<tr>
<td>1</td>
<td>034</td>
<td>FS</td>
<td>File Separator, CTRL SHIFT L.</td>
</tr>
<tr>
<td>0</td>
<td>035</td>
<td>GS</td>
<td>Group Separator, CTRL SHIFT M.</td>
</tr>
<tr>
<td>0</td>
<td>036</td>
<td>RS</td>
<td>Record Separator, CTRL SHIFT N.</td>
</tr>
<tr>
<td>1</td>
<td>037</td>
<td>US</td>
<td>Unit Separator, CTRL SHIFT O.</td>
</tr>
<tr>
<td>0</td>
<td>040</td>
<td>SP</td>
<td>Space.</td>
</tr>
<tr>
<td>0</td>
<td>041</td>
<td>!</td>
<td>Apostrophe or Acute Accent.</td>
</tr>
<tr>
<td>0</td>
<td>042</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>043</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>044</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>045</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>046</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>047</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>050</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>051</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>052</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>053</td>
<td>,</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>054</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>055</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>056</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>057</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>060</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>061</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>062</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>063</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>064</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>065</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>066</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>067</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>070</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>071</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>1</td>
<td>072</td>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>0</td>
<td>073</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>1</td>
<td>074</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>1</td>
<td>075</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>0</td>
<td>077</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>@</td>
<td>@</td>
</tr>
<tr>
<td>0</td>
<td>101</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>102</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>103</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>104</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>105</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>106</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>0</td>
<td>107</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>1</td>
<td>112</td>
<td>J</td>
<td>J</td>
</tr>
<tr>
<td>0</td>
<td>113</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>1</td>
<td>114</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>0</td>
<td>115</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>0</td>
<td>116</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>117</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0</td>
<td>120</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>1</td>
<td>121</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

1 122 R
0 123 S
1 124 T
0 125 U
0 126 V
1 127 W
1 130 X
1 131 Y
0 132 Z
1 133 {  \SHIFT K.
0 134 \SHIFT L.
1 135 } \SHIFT M.
1 136 t (Appears as ^ on some machines).
0 137 ~ (Appears as _ (Underscore) on some machines).
0 140 \ Accent Grave.
1 141 a
1 142 b
0 143 c
1 144 d
0 145 e
0 146 f
1 147 g
1 150 h
0 151 i
0 152 j
1 153 k
0 154 l
1 155 m
1 156 n
0 157 o
1 160 p
0 161 q
0 162 r
1 163 s
0 164 t
1 165 u
1 166 v
0 167 w
0 170 x
1 171 y
1 172 z
0 173 { This Code Generated by ALTMODE.
0 176 ~ This Code Generated by PREFIX key (if Present)
1 177 DEL DELETE, RUBOUT.

C.2 RADIX-50 CHARACTER SET

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Octal Equivalent</th>
<th>Radix-50 Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>A-Z</td>
<td>101-132</td>
<td>1-32</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

<table>
<thead>
<tr>
<th>Character</th>
<th>44</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>56</td>
<td>34</td>
</tr>
<tr>
<td>unused</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>0-9</td>
<td>60-71</td>
<td>36-47</td>
</tr>
</tbody>
</table>

The maximum Radix-50 value is, thus:

\[
\text{2} \\
47 \times 50 + 47 \times 50 + 47 = 174777
\]

The following table provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents. For example, given the ASCII string X2B, the Radix-50 equivalent is (arithmetic is performed in octal):

\[
X = 113000 \\
2 = 002400 \\
B = 000002 \\
X2B = 115402
\]

<table>
<thead>
<tr>
<th>Single Char. or First Char.</th>
<th>Second Character</th>
<th>Third Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 003100</td>
<td>A 000050</td>
<td>A 000001</td>
</tr>
<tr>
<td>B 006200</td>
<td>B 000120</td>
<td>B 000002</td>
</tr>
<tr>
<td>C 011300</td>
<td>C 000170</td>
<td>C 000003</td>
</tr>
<tr>
<td>D 014400</td>
<td>D 000240</td>
<td>D 000004</td>
</tr>
<tr>
<td>E 017500</td>
<td>E 000310</td>
<td>E 000005</td>
</tr>
<tr>
<td>F 022600</td>
<td>F 000360</td>
<td>F 000006</td>
</tr>
<tr>
<td>G 025700</td>
<td>G 000430</td>
<td>G 000007</td>
</tr>
<tr>
<td>H 031000</td>
<td>H 000500</td>
<td>H 000010</td>
</tr>
<tr>
<td>I 034100</td>
<td>I 000550</td>
<td>I 000011</td>
</tr>
<tr>
<td>J 037200</td>
<td>J 000620</td>
<td>J 000012</td>
</tr>
<tr>
<td>K 042300</td>
<td>K 000670</td>
<td>K 000013</td>
</tr>
<tr>
<td>L 045400</td>
<td>L 000740</td>
<td>L 000014</td>
</tr>
<tr>
<td>M 050500</td>
<td>M 001010</td>
<td>M 000015</td>
</tr>
<tr>
<td>N 053600</td>
<td>N 001060</td>
<td>N 000016</td>
</tr>
<tr>
<td>O 056700</td>
<td>O 001130</td>
<td>O 000017</td>
</tr>
<tr>
<td>P 062000</td>
<td>P 001200</td>
<td>P 000020</td>
</tr>
<tr>
<td>Q 065100</td>
<td>Q 001250</td>
<td>Q 000021</td>
</tr>
<tr>
<td>R 070200</td>
<td>R 001320</td>
<td>R 000022</td>
</tr>
<tr>
<td>S 073300</td>
<td>S 001370</td>
<td>S 000023</td>
</tr>
<tr>
<td>T 076400</td>
<td>T 001440</td>
<td>T 000024</td>
</tr>
<tr>
<td>U 101500</td>
<td>U 001510</td>
<td>U 000025</td>
</tr>
<tr>
<td>V 104600</td>
<td>V 001560</td>
<td>V 000026</td>
</tr>
<tr>
<td>W 107700</td>
<td>W 001630</td>
<td>W 000027</td>
</tr>
<tr>
<td>X 113000</td>
<td>X 001700</td>
<td>X 000030</td>
</tr>
<tr>
<td>Y 116100</td>
<td>Y 001750</td>
<td>Y 000031</td>
</tr>
<tr>
<td>Z 121200</td>
<td>Z 002020</td>
<td>Z 000032</td>
</tr>
<tr>
<td>$ 124300</td>
<td>$ 002070</td>
<td>$ 000033</td>
</tr>
<tr>
<td>. 127400</td>
<td>. 002140</td>
<td>. 000034</td>
</tr>
<tr>
<td>0 132500</td>
<td>0 002210</td>
<td>0 000035</td>
</tr>
<tr>
<td>1 135600</td>
<td>1 002260</td>
<td>1 000036</td>
</tr>
<tr>
<td>2 140700</td>
<td>2 002330</td>
<td>2 000037</td>
</tr>
<tr>
<td>3 144000</td>
<td>3 002400</td>
<td>3 000040</td>
</tr>
</tbody>
</table>
### C.3 MACRO SPECIAL CHARACTERS

<table>
<thead>
<tr>
<th>Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>form feed</td>
<td>Source line terminator, forces a new listing page</td>
</tr>
<tr>
<td>line feed</td>
<td>Source line terminator</td>
</tr>
<tr>
<td>carriage return</td>
<td>Formatting character</td>
</tr>
<tr>
<td>vertical tab</td>
<td>Source line terminator</td>
</tr>
<tr>
<td>:</td>
<td>Label terminator</td>
</tr>
<tr>
<td>=</td>
<td>Direct assignment indicator</td>
</tr>
<tr>
<td>%</td>
<td>Register term indicator</td>
</tr>
<tr>
<td>tab</td>
<td>Item terminator, field terminator</td>
</tr>
<tr>
<td>space</td>
<td>Item terminator, field terminator</td>
</tr>
<tr>
<td>#</td>
<td>Immediate expression indicator</td>
</tr>
<tr>
<td>@</td>
<td>Deferred addressing indicator</td>
</tr>
<tr>
<td>(</td>
<td>Initial register indicator</td>
</tr>
<tr>
<td>)</td>
<td>Terminal register indicator</td>
</tr>
<tr>
<td>, (comma)</td>
<td>Operand field separator</td>
</tr>
<tr>
<td>;</td>
<td>Comment field indicator</td>
</tr>
<tr>
<td>+</td>
<td>Arithmetic addition operator or autoincrement indicator</td>
</tr>
<tr>
<td>-</td>
<td>Arithmetic subtraction operator or autodecrement indicator</td>
</tr>
<tr>
<td>*</td>
<td>Arithmetic multiplication operator</td>
</tr>
<tr>
<td>/</td>
<td>Arithmetic division operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND operator</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>' (apostrophe)</td>
<td>Double ASCII character indicator</td>
</tr>
<tr>
<td>.</td>
<td>Single ASCII character indicator</td>
</tr>
<tr>
<td>&lt;</td>
<td>Assembly location counter</td>
</tr>
<tr>
<td>&gt;</td>
<td>Initial argument indicator</td>
</tr>
<tr>
<td>†</td>
<td>Terminal argument indicator</td>
</tr>
<tr>
<td>†</td>
<td>Universal unary operator</td>
</tr>
<tr>
<td>@</td>
<td>Argument indicator</td>
</tr>
<tr>
<td>\</td>
<td>MACRO numeric argument indicator</td>
</tr>
</tbody>
</table>

### C.4 ADDRESS MODE SYNTAX

In the following syntax table, n represents an integer between 0 and 7; R is a register expression; E represents any expression; ER represents either a register expression or an absolute expression in the range 0 to 7.

<table>
<thead>
<tr>
<th>0n</th>
<th>Register</th>
<th>R</th>
<th>Register R contains the operand. R is a register expression.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1n</td>
<td>Deferred Register</td>
<td>@R or (R)</td>
<td>Register R contains the operand address.</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2n</td>
<td>Autoincrement (ER)+</td>
<td>The contents of the register specified by ER are incremented after being used as the address of the operand.</td>
</tr>
<tr>
<td>3n</td>
<td>Deferred Autoincrement @(ER)+</td>
<td>ER contains a pointer to the address of the operand. ER is incremented after use.</td>
</tr>
<tr>
<td>4n</td>
<td>Autodecrement -(ER)</td>
<td>The contents of register ER are decremented before being used as the address of the operand.</td>
</tr>
<tr>
<td>5n</td>
<td>Deferred Autodecrement @-(ER)</td>
<td>The contents of register ER are decremented before being used as a pointer to the address of the operand.</td>
</tr>
<tr>
<td>6n</td>
<td>Index by the Register Specified E(ER)</td>
<td>The value obtained when E is combined with the contents of the register specified (ER) is the address of the operand.</td>
</tr>
<tr>
<td>7n</td>
<td>Deferred index by the Register Specified @E(ER)</td>
<td>E added to ER produces a pointer to the address of the operand.</td>
</tr>
<tr>
<td>27</td>
<td>Immediate Operand #$E</td>
<td>E is the operand.</td>
</tr>
<tr>
<td>37</td>
<td>Absolute address @$E</td>
<td>E is the operand address.</td>
</tr>
<tr>
<td>67</td>
<td>Relative address E</td>
<td>E is the address of the operand.</td>
</tr>
<tr>
<td>77</td>
<td>Deferred relative address @E</td>
<td>E is a pointer to the address of the operand.</td>
</tr>
</tbody>
</table>

C.5 INSTRUCTIONS

The tables of instructions which follow are grouped according to the operands they take and according to the bit patterns of their op-codes.

The following symbols are used to indicate the instruction type format:

OP Instruction mnemonic
R Register Expression
E Expression
ER Register expression or expression $0<=$ER$<=$7
AC Floating point register expression
A General address specification

C-6
MACRO Assembler, Instruction, and Character Code Summaries

In the representation of op-codes, the following symbols are used:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td>Source operand Specified by a 6-bit address mode</td>
</tr>
<tr>
<td>DD</td>
<td>Destination operand Specified by a 6-bit address mode</td>
</tr>
<tr>
<td>XX</td>
<td>8-bit offset to a Branch instructions location</td>
</tr>
<tr>
<td>R</td>
<td>Integer between 0 Represents a general and 7 register</td>
</tr>
</tbody>
</table>

Symbols used in the description of instruction operations are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE</td>
<td>Source Effective Address</td>
</tr>
<tr>
<td>FSE</td>
<td>Floating Source Effective Address</td>
</tr>
<tr>
<td>DE</td>
<td>Destination Effective Address</td>
</tr>
<tr>
<td>FDE</td>
<td>Floating Destination Effective Address</td>
</tr>
<tr>
<td></td>
<td>Absolute Value of</td>
</tr>
<tr>
<td>()</td>
<td>Contents of</td>
</tr>
<tr>
<td>+</td>
<td>Becomes</td>
</tr>
</tbody>
</table>

The condition codes in the processor status word (PS) are affected by the instructions; these condition codes are represented as follows:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Negative bit: Set if the result is negative</td>
</tr>
<tr>
<td>Z</td>
<td>Zero bit: Set if the result is zero</td>
</tr>
<tr>
<td>V</td>
<td>Overflow bit: Set if the operation caused an overflow</td>
</tr>
<tr>
<td>C</td>
<td>Carry bit: Set if the operation caused a carry</td>
</tr>
</tbody>
</table>

In the representation of the instruction's effect on the condition codes, the following symbols are used:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Conditionally set</td>
</tr>
<tr>
<td>-</td>
<td>Not affected</td>
</tr>
<tr>
<td>0</td>
<td>Cleared</td>
</tr>
<tr>
<td>1</td>
<td>Set</td>
</tr>
</tbody>
</table>

To set conditionally means to use the instruction's result to determine the state of the code.

Logical operators are represented by the following symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Inclusive OR</td>
</tr>
<tr>
<td>⋅</td>
<td>Exclusive OR</td>
</tr>
<tr>
<td>&amp;</td>
<td>AND</td>
</tr>
<tr>
<td>-</td>
<td>Used over a symbol to represent the 1's complement of the symbol</td>
</tr>
</tbody>
</table>
### C.5.1 Double Operand Instructions (OP A,A)

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>01SSDD</td>
<td>MOV</td>
<td>MOVE</td>
<td>(SE) → (DE)</td>
<td>*  *  0  -</td>
</tr>
<tr>
<td>11SSDD</td>
<td>MOVB</td>
<td>MOVE Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02SSDD</td>
<td>CMP</td>
<td>CoMPare</td>
<td>(SE)−(DE)</td>
<td>*  *  *  *</td>
</tr>
<tr>
<td>12SSDD</td>
<td>CMPB</td>
<td>CoMPare Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03SSDD</td>
<td>BIT</td>
<td>BIT Test</td>
<td>(SE) &amp; (DE)</td>
<td>*  *  0  -</td>
</tr>
<tr>
<td>13SSDD</td>
<td>BITB</td>
<td>BIT Test Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04SSDD</td>
<td>BIC</td>
<td>BIT Clear</td>
<td>(SE) &amp; (DE) + (DE)</td>
<td>*  *  0  -</td>
</tr>
<tr>
<td>14SSDD</td>
<td>BICB</td>
<td>BIT Clear Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05SSDD</td>
<td>BIS</td>
<td>BIT Set</td>
<td>(SE) l (DE) + (DE)</td>
<td>*  *  0  -</td>
</tr>
<tr>
<td>15SSDD</td>
<td>BISB</td>
<td>BIT Set Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06SSDD</td>
<td>ADD</td>
<td>ADD</td>
<td>(SE) + (DE) → (DE)</td>
<td>*  *  *  *</td>
</tr>
<tr>
<td>16SSDD</td>
<td>SUB</td>
<td>SUBtract</td>
<td>(DE) − (SE) → (DE)</td>
<td>*  *  *  *</td>
</tr>
</tbody>
</table>

### C.5.2 Single Operand Instructions (OP A)

<table>
<thead>
<tr>
<th>Op-code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0050DD</td>
<td>CLR</td>
<td>CLeaR</td>
<td>0 + (DE)</td>
<td>0  1  0  0</td>
</tr>
<tr>
<td>1050DD</td>
<td>CLRB</td>
<td>CLeaR Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0051DD</td>
<td>COM</td>
<td>COMplement</td>
<td>(DE) + (DE)</td>
<td>*  *  0  1</td>
</tr>
<tr>
<td>1051DD</td>
<td>COMB</td>
<td>COMplement Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0052DD</td>
<td>INC</td>
<td>INCrement</td>
<td>(DE) + 1 + (DE)</td>
<td>*  *  *  1</td>
</tr>
<tr>
<td>1052DD</td>
<td>INCB</td>
<td>INCrement Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0053DD</td>
<td>DEC</td>
<td>DECrement</td>
<td>(DE) − 1 + (DE)</td>
<td>*  *  *  -</td>
</tr>
<tr>
<td>1053DD</td>
<td>DECB</td>
<td>DECrement Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0054DD</td>
<td>NEG</td>
<td>NEGate</td>
<td>(DE) + 1 + (DE)</td>
<td>*  *  *  *</td>
</tr>
<tr>
<td>1054DD</td>
<td>NEGB</td>
<td>NEGate Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0055DD</td>
<td>ADC</td>
<td>Add Carry</td>
<td>(DE) + (C) + (DE)</td>
<td>*  *  *  *</td>
</tr>
<tr>
<td>1055DD</td>
<td>ADCB</td>
<td>Add Carry Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0056DD</td>
<td>SBC</td>
<td>SubTract Carry</td>
<td>(DE) − (C) + (DE)</td>
<td>*  *  *  *</td>
</tr>
<tr>
<td>1056DD</td>
<td>SBCB</td>
<td>SubTract Carry Byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0057DD</td>
<td>TST</td>
<td>TeST</td>
<td>(DE)</td>
<td>*  *  0  0</td>
</tr>
<tr>
<td>1057DD</td>
<td>TSTB</td>
<td>TeST Byte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## MACRO Assembler, Instruction, and Character Code Summaries

### C.5.3 Rotate/Shift

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0060DD</td>
<td>ROR</td>
<td>ROtate Right</td>
<td><img src="image" alt="ROR Diagram" /></td>
<td>* * * *</td>
</tr>
<tr>
<td>1060DD</td>
<td>RORB</td>
<td>ROtate Right Byte</td>
<td><img src="image" alt="RORB Diagram" /></td>
<td>* * * *</td>
</tr>
<tr>
<td>0061DD</td>
<td>ROL</td>
<td>ROtate Left</td>
<td><img src="image" alt="ROL Diagram" /></td>
<td>* * * *</td>
</tr>
<tr>
<td>1061DD</td>
<td>ROLB</td>
<td>ROtate Left Byte</td>
<td><img src="image" alt="ROLB Diagram" /></td>
<td>* * * *</td>
</tr>
<tr>
<td>0062DD</td>
<td>ASR</td>
<td>Arithmetic Shift Right</td>
<td><img src="image" alt="ASR Diagram" /></td>
<td>* * * *</td>
</tr>
<tr>
<td>1062DD</td>
<td>ASRB</td>
<td>Arithmetic Shift Right Byte</td>
<td><img src="image" alt="ASRB Diagram" /></td>
<td>* * * *</td>
</tr>
<tr>
<td>0063DD</td>
<td>ASL</td>
<td>Arithmetic Shift Left</td>
<td><img src="image" alt="ASL Diagram" /></td>
<td>* * * *</td>
</tr>
<tr>
<td>1063DD</td>
<td>ASLB</td>
<td>Arithmetic Shift Left Byte</td>
<td><img src="image" alt="ASLB Diagram" /></td>
<td>* * * *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001DD</td>
<td>JMP</td>
<td>JuMP</td>
<td>DE → (PC)</td>
<td>- - - -</td>
</tr>
<tr>
<td>0003DD</td>
<td>SWAB</td>
<td>SWAp Bytes</td>
<td><img src="image" alt="SWAB Diagram" /></td>
<td>* * 0 0</td>
</tr>
</tbody>
</table>

The following instructions are available on the PDP-11/03 (LSI/11) only:

### C.5.4 Move/Shift/Store

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1067DD</td>
<td>MFPS</td>
<td>Move byte From Processor Status word</td>
<td>(DE)+PSW</td>
<td>* * 0 -</td>
</tr>
<tr>
<td>1064SS</td>
<td>MTPS</td>
<td>Move byte To Processor Status word</td>
<td>(SE)→PSW</td>
<td>* * * *</td>
</tr>
</tbody>
</table>

The following instructions are available on the PDP-11/35, 40, 45 as noted:

### C.5.5 Other Instructions

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0065SS</td>
<td>MFPI</td>
<td>Move From Previous Instruction space</td>
<td>(SE) → (TEMP)</td>
<td>* * 0 -</td>
</tr>
</tbody>
</table>

January 1976
## MACRO Assembler, Instruction, and Character Code Summaries

### 11/45 with KT11 only

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1065SS</td>
<td>MFPD</td>
<td>Move From (SE) + (TEMP) * 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Previous (SP) - (SP)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data space (TEMP) + (SP)</td>
<td></td>
</tr>
</tbody>
</table>

### 11/35, 11/40, 11/45 with KT11

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0066DD</td>
<td>MTPI</td>
<td>Move To ((SP)) + (TEMP) * 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Previous (SP+2) + (SP)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Instruction space (TEMP) + (DE)</td>
<td></td>
</tr>
</tbody>
</table>

### 11/45 with KT11 only

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1066DD</td>
<td>MTPD</td>
<td>Move To ((SP)) + (TEMP) * 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Previous (SP+2) + (SP)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data space (TEMP) + (DE)</td>
<td></td>
</tr>
</tbody>
</table>

### 11/35, 11/40, 11/45

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1701DD</td>
<td>LDFPS</td>
<td>Load FPP DE + FPS</td>
<td></td>
</tr>
</tbody>
</table>

### 11/45 with FP11-B

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0707DD</td>
<td>NEGD</td>
<td>NEGate Double - (FDE) + FDE</td>
<td></td>
</tr>
<tr>
<td>0707DD</td>
<td>NEGF</td>
<td>NEGate Floating - (FDE) + FDE * 0 0</td>
<td></td>
</tr>
<tr>
<td>1702DD</td>
<td>STFPS</td>
<td>Store Floating Point processor status See Chapter 7 in PDP-11/45 Processor Handbook</td>
<td></td>
</tr>
<tr>
<td>1703DD</td>
<td>STST</td>
<td>Store floating point processor status</td>
<td></td>
</tr>
<tr>
<td>1704DD</td>
<td>CLRD</td>
<td>CLeaR Double 0 + (FDE) 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>1704DD</td>
<td>CLRF</td>
<td>CLeaR Floating 0 + (FDE) 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>1705DD</td>
<td>TSTD</td>
<td>TeST Double (FDE) * 0 0</td>
<td></td>
</tr>
<tr>
<td>1705DD</td>
<td>TSTF</td>
<td>TeST Floating (FDE) * 0 0</td>
<td></td>
</tr>
<tr>
<td>1706DD</td>
<td>ABSD</td>
<td>make ABSolute Double</td>
<td>(FDE) + (FDE) 0 * 0 0</td>
</tr>
<tr>
<td>1706DD</td>
<td>ABSF</td>
<td>make ABSolute Floating</td>
<td>(FDE) + (FDE) 0 * 0 0</td>
</tr>
</tbody>
</table>

---

January 1976 C-10
## MACRO Assembler, Instruction, and Character Code Summaries

### C.5.4 Operate Instructions (OP)

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>HALT</td>
<td>HALT</td>
<td>The computer stops all functions.</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000001</td>
<td>WAIT</td>
<td>WAIT</td>
<td>The computer stops and waits for an interrupt.</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000002</td>
<td>RTI</td>
<td>RetTurn from Interrupt (Return from Trap)</td>
<td>The PC and ST are popped off the SP stack: ((SP)) + (PC) (SP)+2 + (SP) ((SP)) + (ST) (SP)+2 + (SP)</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000005</td>
<td>RESET</td>
<td>RESET</td>
<td>Returns all I/O devices to power-on state.</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000241</td>
<td>CLC</td>
<td>CLean Carry bit</td>
<td>0 → C</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000261</td>
<td>SEC</td>
<td>SEt Carry bit</td>
<td>1 → C</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000242</td>
<td>CLV</td>
<td>CLean overflow</td>
<td>0 → V</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000262</td>
<td>SEV</td>
<td>SEt overflow bit</td>
<td>1 → V</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000244</td>
<td>CLZ</td>
<td>CLean Zero bit</td>
<td>0 → Z</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000264</td>
<td>SEZ</td>
<td>SEt Zero bit</td>
<td>1 → Z</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000250</td>
<td>CLN</td>
<td>CLean Negative bit</td>
<td>0 → N</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000270</td>
<td>SEN</td>
<td>SEt Negative bit</td>
<td>1 → N</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000257</td>
<td>CCC</td>
<td>Clear all Condition Codes</td>
<td>0 → N 0 0 0 0</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000277</td>
<td>SCC</td>
<td>Set all Condition Codes</td>
<td>1 → N 1 1 1 1</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td>000240</td>
<td>NOP</td>
<td>No OPeration</td>
<td></td>
<td>N  Z  V  C</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

The following instructions are available on the PDP-11/45 with FP11-B only:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>170000</td>
<td>CFCC</td>
<td>Copy Floating Condition Code</td>
<td>Copy FPP condition codes into CPU condition codes.</td>
<td>* * * *</td>
</tr>
<tr>
<td>170011</td>
<td>SETD</td>
<td>SET Double floating mode</td>
<td>FPP set to double precision</td>
<td>- - - -</td>
</tr>
<tr>
<td>170001</td>
<td>SETF</td>
<td>SET Floating mode</td>
<td>FPP set to single precision mode</td>
<td>- - - -</td>
</tr>
<tr>
<td>170002</td>
<td>SETI</td>
<td>SET Integer mode</td>
<td>FPP set for integer data (16 bits)</td>
<td>- - - -</td>
</tr>
<tr>
<td>170012</td>
<td>SETL</td>
<td>SET Long integer mode</td>
<td>FPP set for long integer data (32 bits)</td>
<td>- - - -</td>
</tr>
</tbody>
</table>

All 11/45's, with and without FP11-B

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>000006</td>
<td>RTT</td>
<td>ReTurn from inTerrupt</td>
<td>Same as RTI instruction but inhibits trace trap</td>
<td>* * * *</td>
</tr>
</tbody>
</table>

C.5.5 Trap Instructions (OP or OP e where 0<=$E<=$377(8)) *(OP (only))

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>*000003</td>
<td>BPT</td>
<td>BreakPoint Trap</td>
<td>Trap to location 14. This is used to call ODT.</td>
<td>* * * *</td>
</tr>
<tr>
<td>*000004</td>
<td>IOT</td>
<td>Input Output Trap</td>
<td>Trap to location 20. This is used to call IOK.</td>
<td>* * * *</td>
</tr>
</tbody>
</table>

C-12
MACRO Assembler, Instruction, and Character Code Summaries

<table>
<thead>
<tr>
<th>Start Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>104000-104377</td>
<td>EMT</td>
<td>EMulator Trap to location 30. This is used to call system programs.</td>
</tr>
<tr>
<td>104400-104771</td>
<td>TRAP</td>
<td>TRAP Trap to location 34. This is used to call any routine desired by the programmer.</td>
</tr>
</tbody>
</table>

C.5.6 Branch Instructions OP E
(where –128(decimal)≤(E–2)/2<127(decimal))

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Condition to be met if branch is to occur</th>
</tr>
</thead>
<tbody>
<tr>
<td>0004XX</td>
<td>BR</td>
<td>Branch always</td>
<td></td>
</tr>
<tr>
<td>0010XX</td>
<td>BNE</td>
<td>Branch if Not Equal (to zero)</td>
<td>Z=0</td>
</tr>
<tr>
<td>0014XX</td>
<td>BEQ</td>
<td>Branch if Equal (to zero)</td>
<td>Z=1</td>
</tr>
<tr>
<td>0020XX</td>
<td>BGE</td>
<td>Branch if Greater than or Equal (to zero)</td>
<td>N  V=0</td>
</tr>
<tr>
<td>0024XX</td>
<td>BLT</td>
<td>Branch if Less Than (zero)</td>
<td>N V=0</td>
</tr>
<tr>
<td>0030XX</td>
<td>BGT</td>
<td>Branch if Greater Than (zero)</td>
<td>Z (N V)=0</td>
</tr>
<tr>
<td>0034XX</td>
<td>BLE</td>
<td>Branch if Less than or Equal (to zero)</td>
<td>Z (N V)=1</td>
</tr>
<tr>
<td>1000XX</td>
<td>BPL</td>
<td>Branch if PLus</td>
<td>N=0</td>
</tr>
<tr>
<td>1004XX</td>
<td>BMI</td>
<td>Branch if MINus</td>
<td>N=1</td>
</tr>
<tr>
<td>1010XX</td>
<td>BHI</td>
<td>Branch if HIGher</td>
<td>C  Z=0</td>
</tr>
<tr>
<td>1014XX</td>
<td>BLO</td>
<td>Branch if LOWer or Same</td>
<td>C1Z=1</td>
</tr>
<tr>
<td>1020XX</td>
<td>BVC</td>
<td>Branch if oVerflow Clear</td>
<td>V=0</td>
</tr>
<tr>
<td>1024XX</td>
<td>BVS</td>
<td>Branch if oVerflow Set</td>
<td>V=1</td>
</tr>
<tr>
<td>1030XX</td>
<td>BCC (or BHIS)</td>
<td>Branch if Carry Clear (or Branch if HIGher or Same)</td>
<td>C=0</td>
</tr>
<tr>
<td>1034XX</td>
<td>BCS (or BLO)</td>
<td>Branch if Carry Set (or Branch if LOW)</td>
<td>C=1</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

C.5.7 Register Destination (OP ER,A)

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>004RDD</td>
<td>JSR</td>
<td>Jump to SubRoutine</td>
<td>Push register on the SP stack, put the PC in the register:</td>
<td>N  Z  V  C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DE TEMP (TEMP= temporary storage register internal to processor,)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(SP)=2 + SP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(REG) -(SP)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(PC) + REG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(TEMP) + PC</td>
<td></td>
</tr>
</tbody>
</table>

The following instruction is available only on the 11/35, 11/40, 11/45:

074RDD XOR eXclusive OR (R) 1 (DE) + (DE) * * 0 -

C.5.8 Register-Offset (OP R,E)

The following instruction is available only on the PDP-11/35, 11/40, 11/45:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>N  Z  V  C</th>
</tr>
</thead>
<tbody>
<tr>
<td>077RDD</td>
<td>SOB</td>
<td>Subtract One and Branch</td>
<td>(R)-1→(R)</td>
<td>-  -  -</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(PC)-(2*DE) + (PC)</td>
<td></td>
</tr>
</tbody>
</table>

C.5.9 Subroutine Return (OP ER)

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>N  Z  V  C</th>
</tr>
</thead>
<tbody>
<tr>
<td>00020R</td>
<td>RTS</td>
<td>ReTurn from Subroutine</td>
<td>Put register in PC and pop old contents from SP stack into register.</td>
<td>-  -  -</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

C.5.10 Source-Register (OP A,R)

The following instructions are available on the 11/35, 11/40, 11/45 only:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Floating Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>071RSS</td>
<td>DIV</td>
<td>DIVide</td>
<td>(R),(R11)/(SRC) ÷ (R),(R11)</td>
<td>* * * *</td>
</tr>
<tr>
<td>070RSS</td>
<td>MUL</td>
<td>MULtiply</td>
<td>(R)*(SRC) ÷ (R),(R11)</td>
<td>* * * *</td>
</tr>
<tr>
<td>072RSS</td>
<td>ASH</td>
<td>Arithmetic SHift</td>
<td>R is shifted according to low-order 6-bits of source</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Floating Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>073RSS</td>
<td>ASHC</td>
<td>Arithmetic SHift</td>
<td>R,RVL are shifted according to low-order 6 bits of source</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Combined</td>
<td></td>
<td>or</td>
</tr>
</tbody>
</table>

C.5.11 Floating-Point Source Double Register (OP A,AC)

The following instructions are available on the PDP-11/45 with FP11-B only:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Floating Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>172(AC)ss</td>
<td>ADDD</td>
<td>ADD Double</td>
<td>(FSE) ÷(AC) ÷(AC)</td>
<td>* * * 0</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
<td>AC</td>
<td>FSE</td>
<td>(AC)</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td><code>ADD</code></td>
<td>Floating</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>CMPD</code></td>
<td>Double</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>CMPF</code></td>
<td>Floating</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>DIVD</code></td>
<td>Double</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>DIVF</code></td>
<td>Floating</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>LDCDF</code></td>
<td>(FSE) → (AC)</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>LDCFD</code></td>
<td>(FSE) → (AC)</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>LDD</code></td>
<td>Double</td>
<td>*</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td><code>LDF</code></td>
<td>Floating</td>
<td>*</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td><code>MULD</code></td>
<td>Double</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>MULF</code></td>
<td>Floating</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>SUBD</code></td>
<td>Double</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td><code>SUBF</code></td>
<td>Floating</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

C.5.12 Source-Double Register (OP A,AC)

The following instructions are available on the PDP-11/45 with FP11-B only:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>177(AC)SS</td>
<td>LDCID</td>
<td>Load and Convert Integer to Double</td>
<td>(SE) → (AC)</td>
<td>*  *  * 0</td>
</tr>
<tr>
<td>177(AC)SS</td>
<td>LDCIF</td>
<td>Load and Convert Integer to Floating</td>
<td>(SE) → (AC)</td>
<td>*  *  * 0</td>
</tr>
<tr>
<td>177(AC)SS</td>
<td>LDCLD</td>
<td>Load and Convert Long Integer to Double</td>
<td>(SE) → (AC)</td>
<td>*  *  * 0</td>
</tr>
<tr>
<td>177(AC)SS</td>
<td>LDCLF</td>
<td>Load and Convert Long Integer to Floating</td>
<td>(SE) → (AC)</td>
<td>*  *  * 0</td>
</tr>
<tr>
<td>176(AC+4)SS</td>
<td>LDEXP</td>
<td>Load EXPonent (SE)+200 → (AC EXP)</td>
<td>*  * 0 0</td>
<td></td>
</tr>
</tbody>
</table>

C.5.13 Double Register-Destination (OP AC,A)

The following instructions are available on the PDP-11/45 with FP11-B only:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>176(AC)DD</td>
<td>STCFD</td>
<td>Store, Convert from Floating to Double</td>
<td>(AC) → (FDE)</td>
<td>*  *  * 0</td>
</tr>
<tr>
<td>176(AC)DD</td>
<td>STCDF</td>
<td>Store, Convert from Double to Floating</td>
<td>(AC) → (FDE)</td>
<td>*  *  * 0</td>
</tr>
<tr>
<td>175(AC+4)DD</td>
<td>STCDI(1)</td>
<td>Store, Convert from Double to Integer</td>
<td>(AC) → (FDE)</td>
<td>*  * 0  *</td>
</tr>
</tbody>
</table>

(1)These instructions set both the floating-point and processor condition codes as indicated.
### MACRO Assembler, Instruction, and Character Code Summaries

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
<th>Operation</th>
<th>Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>175(AC+4)DD STCDL(1)</td>
<td>Store, Convert from Double to Long integer</td>
<td>(AC) → (FDE)</td>
<td>* * 0 *</td>
</tr>
<tr>
<td>175(AC+4)DD STCFI(1)</td>
<td>Store, Convert from Floating to Integer</td>
<td>(AC) → (FDE)</td>
<td>* * 0 *</td>
</tr>
<tr>
<td>174(AC+4)DD STCFL(1)</td>
<td>Store, Convert from Floating to Long integer</td>
<td>(AC) → (FDE)</td>
<td>* * 0 *</td>
</tr>
<tr>
<td>174(AC)DD STD</td>
<td>Store Double</td>
<td>(AC) → (FDE)</td>
<td>- - - -</td>
</tr>
<tr>
<td>174(AC)DD STF</td>
<td>Store Floating</td>
<td>(AC) → (FDE)</td>
<td>- - - -</td>
</tr>
<tr>
<td>175(AC)DD STEXP(1)</td>
<td>Store EXPonent</td>
<td>(AC EXP)-200 + (DE)</td>
<td>* * 0 0</td>
</tr>
</tbody>
</table>

#### C.5.14 Number

The following instruction is available on the 11/35, 11/40, 11/45 only:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operation</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0064NN</td>
<td>MARK</td>
<td>MARK</td>
<td>Stack cleanup on return from sub-routine.</td>
<td>- - - -</td>
</tr>
</tbody>
</table>

#### C.5.15 Priority

The following instruction is available on the PDP-11/45 only:

<table>
<thead>
<tr>
<th>Op-Code</th>
<th>Mnemonic</th>
<th>Stands for</th>
<th>Operations</th>
<th>Status Word Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>00023N</td>
<td>SPL</td>
<td>Set Priority Level</td>
<td>(X) + (PS) (bits 7-5)</td>
<td>- - - -</td>
</tr>
</tbody>
</table>

(1) These instructions set both the floating-point and processor condition codes as indicated.

C-18
### C.6 ASSEMBLER DIRECTIVES

<table>
<thead>
<tr>
<th>Form</th>
<th>Described in Manual Section</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>'</td>
<td>5.5.3.3</td>
<td>A single quote character (apostrophe) followed by one ASCII character generates a word containing the 7-bit ASCII representation of the character in the low-order byte and zero in the high-order byte.</td>
</tr>
<tr>
<td>&quot;</td>
<td>5.5.3.3</td>
<td>A double quote character followed by two ASCII characters generates a word containing the 7-bit ASCII representation of the two characters.</td>
</tr>
<tr>
<td>\</td>
<td>5.6.3.3</td>
<td>A backslash preceding an argument causes the number to be treated in the current radix.</td>
</tr>
<tr>
<td>†Bn</td>
<td>5.5.4.2</td>
<td>Temporary radix control; causes the number n to be treated as a binary number.</td>
</tr>
<tr>
<td>†Cn</td>
<td>5.5.6.2</td>
<td>Creates a word containing the one's complement of n.</td>
</tr>
<tr>
<td>†Dn</td>
<td>5.5.4.2</td>
<td>Temporary radix control; causes the number n to be treated as a decimal number.</td>
</tr>
<tr>
<td>†Fn</td>
<td>5.5.6.2</td>
<td>Creates a one-word floating point quantity to represent n.</td>
</tr>
<tr>
<td>†On</td>
<td>5.5.4.2</td>
<td>Temporary radix control; causes the number n to be treated as an octal number.</td>
</tr>
<tr>
<td>.ASCII string</td>
<td>5.5.3.4</td>
<td>Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters) one character per byte.</td>
</tr>
<tr>
<td>.ASCIIZ string</td>
<td>5.5.3.5</td>
<td>Generates a block of data containing the ASCII equivalent of the character string (enclosed in delimiting characters) one character per byte with a zero byte following the specified string.</td>
</tr>
<tr>
<td>.ASECT</td>
<td>5.5.9</td>
<td>Begin or resume absolute section.</td>
</tr>
<tr>
<td>.BLKB exp</td>
<td>5.5.5.3</td>
<td>Reserves a block of storage space exp bytes long.</td>
</tr>
<tr>
<td>.BLKW exp</td>
<td>5.5.5.3</td>
<td>Reserves a block of storage space exp words long.</td>
</tr>
<tr>
<td>MACRO Assembler, Instruction, and Character Code Summaries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.BYTE exp1,</strong> exp2,<strong>...</strong> 5.5.3.1</td>
<td>Generates successive bytes of data containing the octal equivalent of the expression(s) specified.</td>
<td></td>
</tr>
<tr>
<td><strong>.CSECT symbol</strong> 5.5.9</td>
<td>Begins or resumes named or unnamed relocatable section.</td>
<td></td>
</tr>
<tr>
<td><strong>.DSABL arg</strong> 5.5.2</td>
<td>Disables the assembler function specified by the argument.</td>
<td></td>
</tr>
<tr>
<td><strong>.ENABL arg</strong> 5.5.2</td>
<td>Provides the assembler function specified by the argument.</td>
<td></td>
</tr>
<tr>
<td><strong>.END</strong> 5.5.7.1</td>
<td>Indicates the physical end of the source program. An optional argument specifies the transfer address.</td>
<td></td>
</tr>
<tr>
<td><strong>.END exp</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.ENDC</strong> 5.5.11</td>
<td>Indicates the end of a condition block.</td>
<td></td>
</tr>
<tr>
<td><strong>.ENDM</strong> 5.6.1.2</td>
<td>Indicates the end of the current repeat block, indefinite repeat block, or macro. The optional symbol, if used, must be identical to the macro name.</td>
<td></td>
</tr>
<tr>
<td><strong>.ENDM symbol</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>.EOT</strong> 5.5.7.2</td>
<td>Ignored. Indicates End-of-Tape which is detected automatically by the hardware.</td>
<td></td>
</tr>
<tr>
<td><strong>.ERROR exp,string</strong> 5.6.5</td>
<td>Causes a text string to be output to the listing containing the optional expression specified and the indicated text string. The line will be flagged with the &quot;P&quot; error code.</td>
<td></td>
</tr>
<tr>
<td><strong>.EVEN</strong> 5.5.5.1</td>
<td>Ensures that the assembly location counter contains an even address by adding 1 if it is odd.</td>
<td></td>
</tr>
<tr>
<td><strong>.FLT2 arg1,</strong> arg2,<strong>...</strong> 5.5.6.1</td>
<td>Generates successive two-word floating point equivalents for the floating-point numbers specified as arguments.</td>
<td></td>
</tr>
<tr>
<td><strong>.FLT4 arg1,</strong> arg2,<strong>...</strong> 5.5.6.1</td>
<td>Generates successive four-word floating point equivalents for the floating-point numbers specified as arguments.</td>
<td></td>
</tr>
<tr>
<td><strong>.GLOBL sym1,</strong> sym2,<strong>...</strong> 5.5.10</td>
<td>Defines the symbol(s) specified as global symbol(s).</td>
<td></td>
</tr>
<tr>
<td><strong>.IDENT symbol</strong> 5.5.1.5</td>
<td>Provides a means of labeling the object module produced as a result of assembly. This directive is not supported by RT-11, but is included for compatibility with other systems.</td>
<td></td>
</tr>
<tr>
<td><strong>.IF cond,</strong> arguments 5.5.11</td>
<td>Begins a conditional block of source code which is included in the assembly code.</td>
<td></td>
</tr>
<tr>
<td>Macro</td>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.IFF</td>
<td>5.5.11.1</td>
<td>Appears only within a conditional block and indicates the beginning of a section of code to be assembled if the condition tested false.</td>
</tr>
<tr>
<td>.IFT</td>
<td>5.5.11.1</td>
<td>Appears only within a conditional block and indicates the beginning of a section of code to be assembled if the condition tested true.</td>
</tr>
<tr>
<td>.IFTF</td>
<td>5.5.11.1</td>
<td>Appears only within a conditional block and indicates the beginning of a section of code to be unconditionally assembled.</td>
</tr>
<tr>
<td>.IF cond, arg, statement</td>
<td>5.5.11.2</td>
<td>Acts as a one-line conditional block where the condition is tested for the argument specified. The statement is assembled only if the condition tests true.</td>
</tr>
<tr>
<td>.IRP sym, &lt;arg1, arg2,...&gt;</td>
<td>5.6.6</td>
<td>Indicates the beginning of an indefinite repeat block in which the symbol specified is replaced with successive elements of the real argument list (which is enclosed in angle brackets).</td>
</tr>
<tr>
<td>.IRPC sym,string</td>
<td>5.6.6</td>
<td>Indicates the beginning of an indefinite repeat block in which the symbol specified takes on the value of successive characters in the character string.</td>
</tr>
<tr>
<td>.LIMIT</td>
<td>5.5.8</td>
<td>Reserves two words into which the Linker inserts the low and high addresses of the relocated code.</td>
</tr>
<tr>
<td>.LIST</td>
<td>5.5.1.1</td>
<td>Without an argument, .LIST increments the listing level count by 1. With an argument, .LIST does not alter the listing level count but formats the assembly listing according to the argument specified.</td>
</tr>
<tr>
<td>.MACRO sym,arg1,...</td>
<td>5.6.1.1</td>
<td>Indicates the start of a macro with the specified name containing the dummy arguments specified.</td>
</tr>
<tr>
<td>.MCALL</td>
<td>5.6.8</td>
<td>Used to specify the names of all system macro definitions not defined in the current program but required by the program.</td>
</tr>
<tr>
<td>.MEXIT</td>
<td>5.6.1.3</td>
<td>Causes an exit from the current macro or indefinite repeat block.</td>
</tr>
</tbody>
</table>
MACRO Assembler, Instruction, and Character Code Summaries

.NARG symbol 5.6.4 Appears only within a macro definition and equates the specified symbol to the number of arguments in the macro call currently being expanded.

.NCHR sym,<string> 5.6.4 Can appear anywhere in a source program; equates the symbol specified to the number of characters in the string (enclosed in delimiting characters).

.NLIST 5.5.1.1 Without an argument, .NLIST decrements the listing level count by 1. With an argument, .NLIST deletes the portion of the listing indicated by the argument.

.NTYPE symbol,arg 5.6.4 Appears only in a macro definition and equates the low-order six bits of the symbol specified to the six-bit addressing mode of the argument.

.ODD 5.5.5.2 Ensures that the assembly location counter contains an odd address by adding 1 if it is even.

.PAGE 5.5.1.6 Causes the assembly listing to skip to the top of the next page.

.PRINT exp,string 5.6.5 Causes a text string to be output to the listing containing the optional expression specified and the indicated text string.

.RADIX n 5.5.4.1 Alters the current program radix to n, where n can be 2, 4, 8, or 10.

.Rad50 string 5.5.3.6 Generates a block of data containing the Radix-50 equivalent of the character string (enclosed in delimiting characters).

.REPT exp 5.6.7 Begins a repeat block. Causes the section of code up to the next .ENDM or .ENDR to be repeated exp times.

.SBTTL string 5.5.1.4 Causes the string to be printed as part of the assembly listing page header. The string part of each .SBTTL directive is collected into a table of contents at the beginning of the assembly listing.

.TITLE string 5.5.1.3 Assigns the first symbolic name in the string to the object module and causes the string to appear on each page of the assembly listing. One .TITLE directive should be issued per program.
MACRO Assembler, Instruction, and Character Code Summaries

.WORD exp1,
 exp2,...
 5.5.3.2 Generates successive words of data containing the octal equivalent of the expression(s) specified.

C.7 MACRO/CREF SWITCHES

C.7.1 Listing Control Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/L:arg</td>
<td>These switches are used to control listing output. Arguments which are valid for either switch include:</td>
</tr>
<tr>
<td>/N:arg</td>
<td>Source line sequence numbers</td>
</tr>
<tr>
<td>SEQ</td>
<td>Location counter</td>
</tr>
<tr>
<td>LOC</td>
<td>Generated binary code</td>
</tr>
<tr>
<td>BIN</td>
<td>Binary extensions</td>
</tr>
<tr>
<td>BEX</td>
<td>Source code</td>
</tr>
<tr>
<td>SRC</td>
<td>Comments</td>
</tr>
<tr>
<td>COM</td>
<td>Macro definitions and repeat range definitions</td>
</tr>
<tr>
<td>MD</td>
<td>Macro calls and repeat range expansions</td>
</tr>
<tr>
<td>ME</td>
<td>Macro expansions</td>
</tr>
<tr>
<td>MEB</td>
<td>Macro expansion binary code</td>
</tr>
<tr>
<td>CND</td>
<td>Unsatisfied conditions and all .IF and .ENDC statements.</td>
</tr>
<tr>
<td>LD</td>
<td>Listing directives having no arguments</td>
</tr>
<tr>
<td>TOC</td>
<td>Table of contents</td>
</tr>
<tr>
<td>TTM</td>
<td>Listing output format</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbol table</td>
</tr>
<tr>
<td>&lt;no arg&gt; /N with no argument causes only table of contents, symbol table, and error listings to be produced.</td>
<td></td>
</tr>
<tr>
<td>/L with no argument causes .LIST and .NLIST directives without arguments which appear in the source program to be ignored.</td>
<td></td>
</tr>
</tbody>
</table>

C.7.2 Function Control Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/D:arg</td>
<td>These switches are used to enable or disable certain functions in source input files. Valid arguments include:</td>
</tr>
<tr>
<td>/E:arg</td>
<td>Enables or Disables:</td>
</tr>
<tr>
<td>Arg</td>
<td>Absolute binary output</td>
</tr>
<tr>
<td>ABS</td>
<td>Assembly of all absolute addresses as relative addresses</td>
</tr>
<tr>
<td>AMA</td>
<td>Source columns 73 and greater to be treated as comments</td>
</tr>
<tr>
<td>CDR</td>
<td>Floating point truncation</td>
</tr>
<tr>
<td>PPT</td>
<td>Accepts lower case ASCII input</td>
</tr>
<tr>
<td>LC</td>
<td>Local symbol block</td>
</tr>
<tr>
<td>LSB</td>
<td>Binary output</td>
</tr>
</tbody>
</table>

C-23
MACRO Assembler, Instruction, and Character Code Summaries

C.7.3 CREF Switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Produces Cross-Reference of:</th>
</tr>
</thead>
<tbody>
<tr>
<td>/C:S</td>
<td>User-defined symbols</td>
</tr>
<tr>
<td>/C:R</td>
<td>Register symbols</td>
</tr>
<tr>
<td>/C:M</td>
<td>MACRO symbolic names</td>
</tr>
<tr>
<td>/C:P</td>
<td>Permanent symbols</td>
</tr>
<tr>
<td>/C:C</td>
<td>Control sections</td>
</tr>
<tr>
<td>/C:E</td>
<td>Error codes</td>
</tr>
<tr>
<td>/C:&lt;no arg&gt;</td>
<td>Equivalent to /C:S:M:E</td>
</tr>
</tbody>
</table>
## C.8 OCTAL-DECIMAL CONVERSIONS

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>1101</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td>1111</td>
<td>1111</td>
</tr>
</tbody>
</table>

<p>| C-25 | January 1976 |</p>
<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1024</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2777</td>
<td>1535</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20000</td>
<td>1096</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>24576</td>
<td>20480</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>70000</td>
<td>26872</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1024</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2777</td>
<td>1535</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20000</td>
<td>1096</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>24576</td>
<td>20480</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>70000</td>
<td>26872</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>1536</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3777</td>
<td>2047</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>1536</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3777</td>
<td>2047</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### MACRO Assembler, Instruction, and Character Code Summaries

<table>
<thead>
<tr>
<th>Octal</th>
<th>2048 to 4777 (Octal)</th>
<th>20000 to 50000 (Octal)</th>
<th>60000 to 70000 (Octal)</th>
<th>4000 to 2047 (Octal)</th>
<th>0 to 2047 (Octal)</th>
</tr>
</thead>
</table>

#### 0 to 2047 (Octal)

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
<th>Character Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

#### 4000 to 2047 (Octal)

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
<th>Character Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>4096</td>
<td>A</td>
</tr>
<tr>
<td>4001</td>
<td>4097</td>
<td>A</td>
</tr>
<tr>
<td>4002</td>
<td>4098</td>
<td>A</td>
</tr>
<tr>
<td>4003</td>
<td>4099</td>
<td>A</td>
</tr>
<tr>
<td>4004</td>
<td>4100</td>
<td>A</td>
</tr>
<tr>
<td>4005</td>
<td>4101</td>
<td>A</td>
</tr>
<tr>
<td>4006</td>
<td>4102</td>
<td>A</td>
</tr>
<tr>
<td>4007</td>
<td>4103</td>
<td>A</td>
</tr>
</tbody>
</table>

#### 10000 to 4095 (Decimally)

<table>
<thead>
<tr>
<th>Decimally</th>
<th>Octal</th>
<th>Character Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10002</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10003</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10004</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10005</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>10006</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>10007</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

#### 20000 to 50000 (Decimally)

<table>
<thead>
<tr>
<th>Decimally</th>
<th>Octal</th>
<th>Character Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20002</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>20003</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>20004</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>20005</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>20006</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>20007</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

#### 60000 to 70000 (Decimally)

<table>
<thead>
<tr>
<th>Decimally</th>
<th>Octal</th>
<th>Character Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>60000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>60002</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>60003</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>60004</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>60005</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>60006</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>60007</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

---

**Note:** The table above provides character codes for various octal numbers. Each row corresponds to a specific octal value, showing the decimal equivalent and the character code associated with it. The table is structured to facilitate quick reference for coding in the MACRO Assembler.
### MACRO Assembler, Instruction, and Character Code Summaries

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>3072</td>
</tr>
<tr>
<td>6777</td>
<td>3583</td>
</tr>
</tbody>
</table>

### Table 1

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>3072</td>
</tr>
<tr>
<td>6777</td>
<td>3583</td>
</tr>
<tr>
<td>7000</td>
<td>3584</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>3072</td>
</tr>
<tr>
<td>6777</td>
<td>3583</td>
</tr>
<tr>
<td>7000</td>
<td>3584</td>
</tr>
</tbody>
</table>

---

**January 1976**

C-28
APPENDIX D

SYSTEM MACRO FILE

The following is a listing of the system macro library, SYSMAC.SML. This file is stored on the system device, and used by MACRO when it expands the programmed requests discussed in Chapter 9.

SYSMAC.SML--SYSTEM MACRO LIBRARY
FOR RT11 V2C.

DEC-11-ORSYA-E

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 COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.

THE INFORMATION IN THIS SOFTWARE 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.

EF,JD
MACRO .V1.
   .V1=1
   .ENDM

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

MACRO ...CM1 .AREA, .CODE, .CHAN
   .IF NB .AREA
      MOV .AREA, 0
      MOVB #.CODE, 1(0)
   .ENDIF
   .IF NB .CHAN
      .IF IDN <.CHAN>, (0)
         CLR (0)
      .ENDIF
      MOVB .CHAN, (0)
   .ENDIF
   .ENDM

MACRO ...CM2 .ARG, .OFFSET, .INS
   .IF NB <.ARG>,
      MOV #.ARG, .OFFSET(0)
   .ENDIF
   .ENDM

MACRO ...CM3 .CHAN, .CODE
   .IF NB <.CHAN>,
      MOV #.CODE+0400, 0
      BISB .CHAN, 0
      EMT #0374
   .ENDIF
   .ENDM

MACRO ...CM4 .AREA, .CHAN, .BUFF, .WCNT, .BLK, .CRTN, .CODE
   .IF NB <.AREA>, <.CODE>, <.CHAN>
   .IF NB <.BLK>, 2,
   .IF NB <.BUFF>, 4,
   .IF NB <.WCNT>, 6,
   .IF NB <.CRTN>, 8,
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDIF
   .ENDER
MACRO CNTXSW AREA ADD
...CM1 <AREA>,27,#0
...CM2 <ADD>,2,X
ENDM

MACRO CMKT AREA ID TIME
...CM1 <AREA>,19,#0
...CM2 <ID>,2,
IFS TIME

IFF CLR 4,(0)
ENDC MOV .TIME,4,(0)
EMT =0375

MACRO CLOSE CHAN
IFS OF ...V1

IFF EMT =0<160+_CHAN>
...CM3 <_CHAN>,6,
ENDC
ENDM

MACRO CSIGEN DEVSPEC DEFEXT CSTRING
IFS CSTRING

 iff CLR = (6)
 MOV .CSTRING,= (6)
ENDC EMT =0344
ENDM

MACRO CSISP OUTSPEC DEFEXT CSTRING
IFS CSTRING

 iff CLR = (6)
 MOV .CSTRING,= (6)
ENDC EMT =0345
ENDM

MACRO CSTAT AREA CHAN ADD
...CM1 <AREA>,23,<_CHAN>
...CM2 <ADD>,2,X
ENDM

MACRO DATE
MOV #54,%0
MOV =0262(0),%0
ENDM
SYSTEM MACRO FILE

.MACRO DELETE .AREA, .CHAN, .DEVBLK, SPF
.IF OF ..., V1
.IFF NB < .CHAN>
  MOV .CHAN, %0
  EMT %0 < .AREA>
.IFF
... CM1 < .AREA>, 0, < .CHAN>
... CM2 < .DEVBLK>, 2
.IF B, SPF
  CLR 4, (0)
  MOV .SPF, 4, (0)
  EMT %0375
. ENDC
. ENDC
. ENDM

.MACRO DEVICE .AREA, ADD
... CM1 < .AREA>, 12, %0
... CM2 < .ADD>, 2, X
. ENDM

.MACRO DSTATUS, RETSPC, DNAME
.IF NB < .DNAME>
  MOV .DNAME, %0
  MOV .RETSPC, -(6,)
  EMT %0342
. ENDM

.MACRO ENTER .AREA, .CHAN, .DEVBLK, .LEN, SPF
.IF OF ..., V1
.IF B, .DEVBLK
  CLR 6,)
  MOV .DEVBLK, -(6,)
  EMT %0 < 40+, .AREA>
.IFF
... CM1 < .AREA>, 2, < .CHAN>
... CM2 < .DEVBLK>, 2
.IF NB < .LEN
  MOV .LEN, 4, (0)
  EMT %0375
. ENDC
. IF NB < .LEN
  CLR 4, (0)
  EMT %0375
. ENDC
. ENDM

.MACRO EXIT
  EMT %0350
. ENDM

January 1976
D-4
M3acro FETCH .ADD , DNAME
.IF NB < , DNAME>,
  MOV DNAME , %0
  MOV ADD , (6.)
.EMT 0343
.EMDM

M3acro GTIM .AREA .ADD
...CM1 < , AREA>, 17 , # 0
...CM2 < , ADD>, 2 , # X
.EMDM

M3acro GTJB .AREA .ADD
...CM1 < , AREA>, 16 , # 0
...CM2 < , ADD>, 2 , # X
.EMDM

M3acro HERR
...CM3 , 5.
.EMDM

M3acro HRESET
.EMT 0357
.EMDM

M3acro INTEN , PRI0 , PIC
.IF NB , PIC
  MOV #054 , (6.)
  JSR 5 , #(6.)+
.IFF
  JSR 5 , #054
.EMDC
  .WORD C< .PRI0>32 , >224.
.EMDM

M3acro LOCK
.EMT 0346
.EMDM

M3acro LOOKUP .AREA .CHAN , DEVBLK , SPF
.IF DF .V1
...IIF NB < , CHAN>
  MOV CHAN , %0
  EMT 0<20> .AREA>
.IFF
...CM1 < , AREA>, 1 , < , CHAN>
...CM2 < , DEVBLK>, 2.
...IF 8 , SPF
...IIF
  CLR 4 , (0)
  MOV SPF , 4 , (0)
.EMDC
  .EMT 0375
.EMDC
.EMDM

M3acro MFPS
.ADD #054 , (6.)
.ADD #0342 , (6.)
.JSR 7 , #(6.)+
.IF NB < , ADD>
.EMDM

MOV (6.)+ , , ADD
MACRO .MRKT .AREA,.TIME,.CRTN,.ID
  .CM1 <.AREA>,18,#0
  .CM2 <.TIME>,2.
  .CM2 <.CRTN>,4.
  .CM2 <.ID>,6,x
.ENDM

MACRO .MTPS
  .IIF NB <.ADD>, ADD
  .IIF NB <.ADD>, CLR (6)
  .IIF NB <.ADD>, MOV MOVB .ADD,(6)
  .IIF NB <.ADD>, MOV #F054,= (6)
  .IIF NB <.ADD>, ADD #F0360,= (6)
  .IIF NB <.ADD>, JSR 7,#(6)+
.ENDM

MACRO .MWAIT
  .CM3 ,9.
.ENDM

MACRO .PRINT .ADD
  .IIF NB <.ADD>, MOV .ADD,%0
  .IIF NB <.ADD>, EMT #0351
.ENDM

MACRO .PROTECT .AREA,.ADD
  .CM1 <.AREA>,25,#0
  .CM2 <.ADD>,2,x
.ENDM

MACRO .PURGE .CHAN
  .CM3 <.CHAN>,3.
.ENDM

MACRO .QSET .QADD,.QLEN
  .IIF NB <.QLEN>, MOVB .QLEN,%0
  .IIF NB <.QLEN>, MOV #QADD,= (6)
  .IIF NB <.QLEN>, EMT #0353
.ENDM

MACRO .RCTRL0
  .IIF NB <.QLEN>, EMT #0355
.ENDM

MACRO .RCVD .AREA,.BUFF,.WCNT
  .CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,#1,22.
.ENDM

MACRO .RCVDC .AREA,.BUFF,.WCNT,.CRTN
  .CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,<.CRTN>,22.
.ENDM

MACRO .RCVDW .AREA,.BUFF,.WCNT
  .CM4 <.AREA>,#0,<.BUFF>,<.WCNT>,#0,22.
.ENDM
*MACRO .READ .AREA,.CHAN,.BUFF,.WCNT,.BLK
  *IF DF ...V1
    *IFF NB <..WCNT>, 
      MOV ..WCNT,%0 
      MOV #1,=(6,)
      MOV ..BUFF,=(6,)
      MOV ..CHAN,=(6,)
      EMT 0<200+, AREA>
  *ENDC
  *ENDM

*MACRO .READC .AREA,.CHAN,.BUFF,.WCNT,.CRTN,.BLK
  *IF DF ...V1
    *IFF NB <..CRTN>, 
      MOV ..CRTN,%0 
      MOV ..WCNT,=(6,)
      MOV ..BUFF,=(6,)
      MOV ..CHAN,=(6,)
      EMT 0<200+, AREA>
  *ENDC
  *ENDM

*MACRO .READW .AREA,.CHAN,.BUFF,.WCNT,.BLK
  *IF DF ...V1
    *IFF NB <..WCNT>, 
      MOV ..WCNT,%0 
      CLR =(6,)
      MOV ..BUFF,=(6,)
      MOV ..CHAN,=(6,)
      EMT 0<200+, AREA>
  *ENDC
  *ENDM

*MACRO .REGDEF
  R0=00
  R1=11
  R2=22
  R3=33
  R4=44
  R5=55
  SP=66
  PC=77
  *ENDM

*MACRO .RELEASE .DEVBLK
  *IFF NB <..DEVBLK>, 
    MOV ..DEVBLK,%0 
    CLR =(6,)
    EMT 0343
  *ENDM
MACRO .RENAME .AREA,.CHAN,.DEVLK
  IF DF ...,V1
  IIF NB ,CHAN>, MOV $CHAN,X0 
     EMT $0<100+,AREA>
  IFF
  ...CM1 <.AREA>,4,.,CHAN>
  ...CM2 <.DEVLK>,2,.,X
  ENDC
  ENDM

MACRO .REOPEN .AREA,.CHAN,.CBLK
  IF DF ...,V1
  IIF NB ,CHAN>, MOV $CHAN,X0 
     EMT $0<140+,AREA>
  IFF
  ...CM1 <.AREA>,5,.,CHAN>
  ...CM2 <.CBLK>,2,.,X
  ENDC
  ENDM

MACRO .SAVESTAT .AREA,.CHAN,.CBLK
  IF DF ...,V1
  IIF NB ,CHAN>, MOV $CHAN,X0 
     EMT $0<120+,AREA>
  IFF
  ...CM1 <.AREA>,5,.,CHAN>
  ...CM2 <.CBLK>,2,.,X
  ENDC
  ENDM

MACRO .RSUM
  ...CM3 ,2.
  ENDM

MACRO .SDAT .AREA,.BUFF,.WCNT
  ...CM4 <.AREA>,#0,.<BUFF>,.<WCNT>,,#1,21.
  ENDM

MACRO .SDATC .AREA,.BUFF,.WCNT,.CRTN
  ...CM4 <.AREA>,#0,.<BUFF>,.<WCNT>,.<CRTN>,21.
  ENDM

MACRO .SDATW .AREA,.BUFF,.WCNT
  ...CM4 <.AREA>,#0,.<BUFF>,.<WCNT>,,#0,21.
  ENDM

MACRO .SERR
  ...CM3 ,4.
  ENDM

MACRO .SETTOP .ADD
  IIF NR ,ADD>, MOV $ADD,X0 
     EMT $0354
  ENDM

MACRO .SFPA .AREA,.ADD
  ...CM1 <.AREA>,24,.,#0
  ...CM2 <.ADD>,2,.,X
  ENDM
SYSTEM MACRO FILE

.MACRO SPFUN .AREA, .CHAN, .CODE, .BUFF, .WCNT, .BLK, .CRTN
  .CM1 <.AREA>, 26, <.CHAN>
  .CM2 <.BLK>, 2
  .CM2 <.BUFF>, 4
  .CM2 <.WCNT>, 6
  .IF NB <.CODE
      MOVB #F077, 8, (0)
      MOVB .CODE, 9, (0)
  .ENDIF
  .ENDIF
  .IF 0 <.CRTN
  .ENDIF
  .ENDIF
  .ENDIF
  .ENDIF
  .EMT F0375
  .EMT F0352

.MACRO .BRESET
  .EMT F0352
  .ENDM

.MACRO .SPND
  .CM3 1
  .ENDM

.MACRO .SYNCH .AREA
  .IF NB <.AREA>,
      MOV .AREA, %4
      MOV #F054, %5
      JSR 5, #F0324 (5.)
  .ENDIF
  .ENDM

.MACRO .TLOCK
  .CM3 7
  .ENDM

.MACRO .TRPSET .AREA, .ADD
  .CM1 <.AREA>, 3, #0
  .CM2 <.ADD>, 2, X
  .ENDM

.MACRO .TTCN
  .ENDM

.MACRO .TTCN .CHAR
  .EMT F0340
  .BCS F0340
  .BCS .CHAR
  .ENDM

.MACRO .TTOUTR
  .ENDM

.MACRO .TTYOUT .CHAR
  .IF NB <.CHAR>,
      MOVB .CHAR, 0
      EMT F0341
      BCS F0340
  .ENDIF
  .ENDIF
  .ENDIF
  .ENDIF
  .ENDIF
  .ENDM

D-9  January 1976
MACRO .TWAIT .AREA , TIME
CM1 < .AREA > , 28 , 0
CM2 < .TIME > , 2 , X
ENDM

MACRO .UNLOCK
EMT $0347
ENDM

MACRO .WAIT .CHAN
IF DF ... V1
EMT $0<240+.CHAN>
ENDIF
CM3 < .CHAN > , 0
ENDC
ENDM

MACRO .WRITE .AREA , .CHAN , BUFF , WCNT , BLK
IF DF ... V1
IIF NB < .WCNT > ,
MOV .WCNT , %0
MOV #1 , -(6 ,)
MOV .BUFF , -(6 ,)
MOV .CHAN , -(6 ,)
EMT $0<220+.AREA>
ENDIF
CM4 < .AREA > , < .CHAN > , < .BUFF > , < .WCNT > , < .BLK > , #1 , 9.
ENDC
ENDM

MACRO .WRITE .AREA , .CHAN , BUFF , WCNT , BLK
IF DF ... V1
IIF NB < .WCNT > ,
MOV .WCNT , %0
CLR -(6 ,)
MOV .BUFF , -(6 ,)
MOV .CHAN , -(6 ,)
EMT $0<220+.AREA>
ENDIF
CM4 < .AREA > , < .CHAN > , < .BUFF > , < .WCNT > , < .BLK > , #0 , 9.
ENDC
ENDM

MACRO .WRITE .AREA , .CHAN , BUFF , WCNT , CRTN , BLK
IF DF ... V1
IIF NB < .CRTN > ,
MOV .CRTN , %0
MOV .WCNT , -(6 ,)
MOV .BUFF , -(6 ,)
MOV .CHAN , -(6 ,)
EMT $0<220+.AREA>
ENDIF
CM4 < .AREA > , < .CHAN > , < .BUFF > , < .WCNT > , < .BLK > , < .CRTN > , 9.
ENDC
ENDM
APPENDIX E

PROGRAMMED REQUEST SUMMARY

E.1 PARAMETERS

The following parameters are used as arguments in various calls. (Any parameters used which are not mentioned here are particular to a request and the appropriate section in Chapter 9 should be consulted.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.addr</td>
<td>an address, the meaning of which depends on the request being used</td>
</tr>
<tr>
<td>.area</td>
<td>a pointer to the EMT argument list</td>
</tr>
<tr>
<td>.blk</td>
<td>a block number specifying the relative block in a file where an I/O operation is to begin</td>
</tr>
<tr>
<td>.buff</td>
<td>a buffer address specifying a memory location into which or from which an I/O transfer is to be performed</td>
</tr>
<tr>
<td>.chan</td>
<td>a channel number in the range 0-377 (octal)</td>
</tr>
<tr>
<td>.crtin</td>
<td>the entry point of a completion routine</td>
</tr>
<tr>
<td>.count</td>
<td>file number for magtape/cassette operations</td>
</tr>
<tr>
<td>.dbblk</td>
<td>the address of a four-word RAD50 descriptor of the file to be opened</td>
</tr>
<tr>
<td>.num</td>
<td>a number, the value of which depends on the request</td>
</tr>
<tr>
<td>.wcnt</td>
<td>a word count specifying the number of words to be transferred to or from the buffer during an I/O operation</td>
</tr>
</tbody>
</table>

E.2 REQUEST SUMMARY

Refer to Appendix D (SYSMAC.SML) to see how each macro call is expanded in assembly language code.
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Macro Call</th>
<th>Error Codes (Byte 52=)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.CDFN</td>
<td>Increases number of I/O channels to as many as 255(decimal)</td>
<td>.CDFN .area,.addr,.num</td>
<td>0 - attempt to define fewer channels than already exist</td>
</tr>
<tr>
<td>.CHAIN</td>
<td>Allows one background program to transfer control to another without operator intervention</td>
<td>.CHAIN</td>
<td>Can produce any errors which the monitor RUN command can produce</td>
</tr>
<tr>
<td>.CHCOPY</td>
<td>Opens a channel for input, logically connecting it to another job open for either input or output (F/B only)</td>
<td>.CHCOPY .area,.chan,.ochan</td>
<td>0 - other job does not exist, or does not have enough channels defined, or does not have specified channel open</td>
</tr>
<tr>
<td>.CLOSE</td>
<td>Terminates activity on specified channel and frees it for use in another operation; makes tentative files permanent</td>
<td>.CLOSE .chan</td>
<td>1 - channel already open</td>
</tr>
<tr>
<td>.CMKT</td>
<td>Cancels one or more outstanding mark time requests (F/B only)</td>
<td>.CMKT .area,.id,.time</td>
<td>0 - id is not 0 and mark time with that identification number not found</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>.CNTXSW</td>
<td>Specifies locations to be included in the context switch (F/B only)</td>
<td>.CNTXSW .area,.addr</td>
<td>0 - list is in an area into which the USR swaps; or list is modified while the job is running.</td>
</tr>
<tr>
<td>.CSIGEN</td>
<td>Calls the CSI in general mode</td>
<td>.CSIGEN .devspc,.defext,.cstring</td>
<td>0 - illegal command</td>
</tr>
<tr>
<td></td>
<td>Note: if input is taken from TT:, all errors are printed out</td>
<td></td>
<td>1 - device not found</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - unused</td>
</tr>
<tr>
<td>.CSISPC</td>
<td>Calls the CSI in special mode</td>
<td>.CSISPC .outspc,.defext,.cstring</td>
<td>3 - full directory</td>
</tr>
<tr>
<td></td>
<td>Note: if input is taken from TT:, all errors are printed out</td>
<td></td>
<td>4 - input file not found</td>
</tr>
<tr>
<td>.CSTAT</td>
<td>Furnishes information about a channel (F/B only)</td>
<td>.CSTAT .area,.chan,.addr</td>
<td>0 - illegal command line</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - illegal device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - channel not open</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------</td>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.DATE</td>
<td>Moves current date information into R0</td>
<td>.DATE</td>
<td>None</td>
</tr>
<tr>
<td>.DELETE</td>
<td>Deletes named file from indicated device</td>
<td>.DELETE .area,.chan,.dblk,.count</td>
<td>0 - active channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - file not found</td>
</tr>
<tr>
<td>.DEVICE</td>
<td>Sets up list of addresses to be loaded with specified values upon program</td>
<td>.DEVICE .area,.addr</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>termination (F/B only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.DSTATUS</td>
<td>Provides information about a device</td>
<td>.DSTATUS .cblk,.devnam</td>
<td>0 - device not found</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ENTER</td>
<td>Allocates space on specified device and creates tentative entry for</td>
<td>.ENTER .area,.chan,.dblk,.length,.count</td>
<td>0 - channel in use</td>
</tr>
<tr>
<td></td>
<td>named file</td>
<td></td>
<td>1 - no space greater than or equal to the specified length was found</td>
</tr>
<tr>
<td>.EXIT</td>
<td>Terminates user program and returns control to monitor</td>
<td>.EXIT</td>
<td>None</td>
</tr>
<tr>
<td>.FETCH</td>
<td>Loads device handlers into memory from system device</td>
<td>.FETCH .coradd,.devnam</td>
<td>0 - nonexistent device name or no handler for that device</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------------</td>
<td>------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>.GTIM</td>
<td>Allows access to the current time of day</td>
<td>.GTIM .area,.addr</td>
<td>None</td>
</tr>
<tr>
<td>.GTJB</td>
<td>Passes certain job parameters back to user program</td>
<td>.GTJB .area,.addr</td>
<td>None</td>
</tr>
<tr>
<td>.HERR</td>
<td>Disables error interception and allows system to</td>
<td>.HERR</td>
<td>Monitor Error occurs if:</td>
</tr>
<tr>
<td></td>
<td>detect and act on normally fatal errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.HRESET</td>
<td>Resets channels, releases device handlers and stops</td>
<td>.HRESET</td>
<td>1. called USR from completion routine</td>
</tr>
<tr>
<td></td>
<td>all I/O transfers in progress</td>
<td></td>
<td>2. no device handler</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. error doing directory I/O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4. FETCH error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5. Error reading overlay</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6. no room in directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7. illegal address</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8. illegal channel number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9. illegal EMT</td>
</tr>
</tbody>
</table>

None
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Macro Call</th>
<th>Error Codes</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>.INTEN</td>
<td>Notifies monitor that interrupt has occurred, and sets processor priority to correct state</td>
<td>.INTEN .priority, .pic</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.LOCK</td>
<td>Locks the USR in memory</td>
<td>.LOCK</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.LOOKUP</td>
<td>Associates a specified channel with a device and/or file on that device</td>
<td>.LOOKUP .area,.chan,.dblk,.count</td>
<td>0 - channel already open</td>
<td></td>
</tr>
<tr>
<td>.MFP5</td>
<td>Reads priority bits from processor status word</td>
<td>.MFP5 .addr</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.MRKT</td>
<td>Schedules a completion routine to be entered after specified time interval (F/B only)</td>
<td>.MRKT .area,.time,.crtn,.id</td>
<td>0 - no queue element is available</td>
<td></td>
</tr>
<tr>
<td>.MFTS</td>
<td>Sets priority bits, condition codes, and T bit in processor status word</td>
<td>.MFTS .addr</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.MWAIT</td>
<td>Suspends execution until all messages have been transmitted or received (F/B only)</td>
<td>.MWAIT</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.PRINT</td>
<td>Outputs a string to the terminal</td>
<td>.PRINT .addr</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td>.PROTECT</td>
<td>Used to obtain exclusive control of a vector in the range 0-476 (F/B only)</td>
<td>.PROTECT .area,.addr</td>
<td>0 - protect failure; locations already in use</td>
<td></td>
</tr>
<tr>
<td>.PURGE</td>
<td>Deactivates a channel without taking any other action</td>
<td>.PURGE .chan</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.QSET</td>
<td>Enlarges I/O queue for monitor</td>
<td>.QSET .addr,.qleng</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.RCTRLRO</td>
<td>Enables terminal printing</td>
<td>.RCTRLRO</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.RCVD</td>
<td>Posts request to receive message and continues execution (F/B only)</td>
<td>.RCVD .area,.buff,.wcnt</td>
<td>0 - no other job exists in system</td>
<td></td>
</tr>
<tr>
<td>.RCVDC</td>
<td>Posts request to receive message and enters completion routine when message received (F/B only)</td>
<td>.RCVDC .area,.buff,.wcnt,.crtn</td>
<td>0 - no other job exists in system</td>
<td></td>
</tr>
<tr>
<td>.RCVDW</td>
<td>Posts request to receive message and waits (F/B only) until received</td>
<td>.RCVDW .area,.buff,.wcnt</td>
<td>0 - no other job exists in system</td>
<td></td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>.READ</td>
<td>Initiates transfer from specified channel to memory; returns to program immediately</td>
<td>.READ .area,.chan,.buff,.wcnt,.blk</td>
<td>0 - attempt to read past end-of-file</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - hard error on channel</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - channel not open</td>
<td></td>
</tr>
<tr>
<td>.READC</td>
<td>Transfers words from specified channel to memory; returns control to specified routine when complete</td>
<td>.READC .area,.chan,.buff,.wcnt,.crtn,.blk</td>
<td>0 - attempt to read past end-of-file</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - hard error on channel</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - channel not open</td>
<td></td>
</tr>
<tr>
<td>.READW</td>
<td>Transfers words from specified channel to memory; returns control to user program when transfer complete</td>
<td>.READW .area,.chan,.buff,.wcnt,.blk</td>
<td>0 - attempt to read past end-of-file</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - hard error on channel</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - channel not open</td>
<td></td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------</td>
<td>---------------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>.REGDEF</td>
<td>Defines general registers R0-R5 SP, PC</td>
<td>.REGDEF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.RELEAS</td>
<td>Removes device handler from memory</td>
<td>.RELEAS .devnam</td>
<td>0 - handler name is illegal</td>
<td></td>
</tr>
<tr>
<td>.RENAME</td>
<td>Changes file name</td>
<td>.RENAME .area,.chan,.dblk</td>
<td>0 - channel open</td>
<td></td>
</tr>
<tr>
<td>.REOPEN</td>
<td>Reassociates channel with file on which a SAVESTATUS was performed</td>
<td>.REOPEN .area,.chan,.cblk</td>
<td>1 - file not found</td>
<td></td>
</tr>
<tr>
<td>.RSUM</td>
<td>Resumes job after it was suspended via .SPND (P/B only)</td>
<td>.RSUM</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>.SAVESTATUS</td>
<td>Stores five words (containing data concerning file definition) into memory</td>
<td>.SAVESTATUS .area,.chan,.cblk</td>
<td>1 - SAVESTATUS is illegal</td>
<td></td>
</tr>
<tr>
<td>.SDAT</td>
<td>Initiates message transfer; returns control to user program immediately (P/B only)</td>
<td>.SDAT .area,.buff,.wcnt</td>
<td>0 - no other job exists</td>
<td></td>
</tr>
</tbody>
</table>
Programmed Request Summary

<table>
<thead>
<tr>
<th>Error Codes (Byte 52=)</th>
<th>Function</th>
<th>Macro Call</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = no other job exists</td>
<td>Initiates message transfer routine when complete</td>
<td>SDAYC area, buff, wcnt, crtn</td>
<td>SDAYC</td>
</tr>
<tr>
<td>0 = no other job exists</td>
<td>Initiates message transfer to specified control program when complete</td>
<td>SDATW area, buff, wcnt</td>
<td>SDATW</td>
</tr>
<tr>
<td>-1 = called USR routine from completion</td>
<td>Inhibits fatal errors from aborting job</td>
<td>SERR</td>
<td>SERR</td>
</tr>
<tr>
<td>-2 = no device handler</td>
<td>-3 = error doing directory I/O</td>
<td>-5 = error reading overlay</td>
<td>-6 = no more room for files in directory</td>
</tr>
<tr>
<td>-4 = FETCH error</td>
<td>-7 = illegal address</td>
<td>-10 = illegal channel number</td>
<td>-11 = illegal ERR</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>.SETTOP</td>
<td>Requests additional memory for program</td>
<td>.SETTOP .addr</td>
<td>None</td>
</tr>
<tr>
<td>.SFPA</td>
<td>Sets user interrupt address for floating point processor exceptions</td>
<td>.SFPA .area,.addr</td>
<td>None</td>
</tr>
<tr>
<td>.SPFUN</td>
<td>Provides special device functions to magtape and cassette (and diskette).</td>
<td>.SPFUN .area,.chan,.code,.buff,.wcnt,.crtn,.blk</td>
<td>0 - attempt to read past end-of-file</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - hardware error on channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - channel not open</td>
</tr>
<tr>
<td>.SPND</td>
<td>Suspends running job (F/B only)</td>
<td>.SPND</td>
<td>None</td>
</tr>
<tr>
<td>.SRESET</td>
<td>Resets certain areas of memory, dismisses device handlers brought in by FETCH, purges currently open files, resets to 16 I/O channels, queue to one element</td>
<td>.SRESET</td>
<td>None</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>.SYNCH</td>
<td>Enables monitor programmed request from within an interrupt service routine; normal return is to 2nd location following .SYNCH</td>
<td>SYNCH .area</td>
<td>Monitor returns to location following .SYNCH if:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. another .SYNCH specifying same 7-word block is pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. illegal job number was specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. job is not running</td>
</tr>
<tr>
<td>.TLOCK</td>
<td>Attempts to gain ownership of USR; if unsuccessful, control returns with C bit set (F/B only)</td>
<td>.TLOCK</td>
<td>0 - USR is in use by another job</td>
</tr>
<tr>
<td>.TRPSET</td>
<td>Allows user job to intercept traps to 4 and 10</td>
<td>.TRPSET .area,.addr</td>
<td>None</td>
</tr>
<tr>
<td>.TTYIN</td>
<td>Inputs character from terminal and waits until done</td>
<td>.TTYIN .char</td>
<td>None</td>
</tr>
<tr>
<td>.TTINR</td>
<td>Inputs character from terminal</td>
<td>.TTINR</td>
<td>0 - No characters available in ring buffer</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Function</td>
<td>Macro Call</td>
<td>Error Codes (Byte 52=)</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------</td>
<td>------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>.TTOUTR</td>
<td>Outputs character to terminal</td>
<td>.TTOUTR</td>
<td>0 - Output ring buffer full</td>
</tr>
<tr>
<td>.TTYOUT</td>
<td>Outputs character to terminal and waits until done</td>
<td>.TTYOUT .char</td>
<td>None</td>
</tr>
<tr>
<td>.TWAIT</td>
<td>Suspends the running job for a specified amount of time (F/B only)</td>
<td>.TWAIT .area,.time</td>
<td>0 - No queue element was available</td>
</tr>
<tr>
<td>.UNLOCK</td>
<td>Releases USR from memory</td>
<td>.UNLOCK</td>
<td>None</td>
</tr>
<tr>
<td>.V1..</td>
<td>Enables macro expansions to occur in Version 1 format</td>
<td>.V1..</td>
<td>None</td>
</tr>
<tr>
<td>.V2..</td>
<td>Enables macro expansions to occur in Version 2 format</td>
<td>.V2..</td>
<td>None</td>
</tr>
<tr>
<td>.WAIT</td>
<td>Suspends program execution until all channel I/O is complete</td>
<td>.WAIT .chan</td>
<td>0 - channel not open</td>
</tr>
<tr>
<td>.WRITC</td>
<td>Initiates transfer from memory to specified channel and returns to user program; when complete, passes control to specified routine</td>
<td>.WRITC .area,.chan,.buff,.wcnt,.crtn,.blk</td>
<td>1 - hardware error</td>
</tr>
</tbody>
</table>

0 - end-of-file reached
1 - hardware error
2 - channel not open
**Programmed Request Summary**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Macro Call</th>
<th>Error Codes (Byte 52=)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td>Initiates transfer from memory to channel; returns control to user program immediately</td>
<td>WRITE .area,.chan,.buff,.wcnt,.blk</td>
<td>0 - end-of-file reached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - hardware error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 - channel not open</td>
</tr>
</tbody>
</table>

| WRITW    | Transfers words from memory to channel; when complete, returns control to user program | WRITW .area,.chan,.buff,.wcnt,.blk | 0 - end-of-file reached |
|          |                                                  |                                      | 1 - hardware error     |
|          |                                                  |                                      | 2 - channel not open    |
APPENDIX F

BASIC/RT-11 LANGUAGE SUMMARY

BASIC/RT-11 is a single-user conversational BASIC for use under the RT-11 system. While the BASIC language is one of the simplest computer languages to learn, it contains advanced techniques which the experienced programmer can use to perform more intricate manipulations or to express a problem more efficiently.

BASIC/RT-11 interfaces with the RT-11 monitor to provide powerful sequential and random-access file capabilities and allows the user to save and retrieve programs from peripheral devices. BASIC/RT-11 has provision for alphanumeric character string I/O and string variables (12K or larger systems) and allows user-defined functions and assembly language subroutine calls from user BASIC programs.

For details on using the BASIC language, and for instructions on running BASIC/RT-11, refer to the BASIC/RT-11 Language Reference Manual (DEC-11-LBACA-D-D). A summary of the BASIC/RT-11 commands and error messages is included here for quick reference.

F.1 BASIC/RT-11 STATEMENTS

The following summary of BASIC statements defines the general format for the statement and gives a brief explanation of its use.

CALL "function name" [(argument list)] Used to call assembly language user functions from a BASIC program.

CHAIN"file descriptor"[LINE number] Terminates execution of user program, loads and executes the specified program starting at the line number, if included.

CLOSE{VFn} #n Closes the logical file specified. If no file is specified, closes all files which are open.

DATA data list Used in conjunction with READ to input data into an executing program.
BASIC/RT-11 Language Summary

DEF FN letter (argument)=expression
Defines a user function to be used in the program (letter is any alphabetic letter).

DIM variable(n), variable(n,m), variable$(n), variable$(n,m)
Reserves space for lists and tables according to subscripts specified after variable name.

END
Placed at the physical end of the program to terminate program execution.

FOR variable = expression1 TO expression2 STEP expression3
Sets up a loop to be executed the specified number of times.

GOSUB line number
Used to transfer control to a specified line of a subroutine.

GO TO line number
Used to unconditionally transfer control to other than the next sequential line in the program.

IF expression rel.op. expression THEN line number
Used to conditionally transfer control to the specified line of the program.

IF END #n THEN line number
Used to test for end file on sequential input file #n.

INPUT list
Used to input data from the terminal keyboard or papertape reader.

INPUT #expression: list
Inputs from a sequential file or particular device.

[LET] variable=expression
Used to assign a value to the specified variable.

[LET] VFn(i)=expression
Used to set the value of a virtual memory file element.

NEXT variable
Placed at the end of a FOR loop to return control to the FOR statement.

OPEN file FOR INPUT [(b)] AS FILE #n [DOUBLE BUF]
Opens a sequential file for input or output as specified. File may be of the form "dev:filnam.ext" or a scalar string variable. The number of blocks may be specified by b.

OPEN file FOR INPUT [(b)] AS FILE VFnx (dimension)=string length
Opens a virtual memory file for input or output. x represents the type of file: floating point (blank), integer (%), or character strings ($). File may be of the form "dev:fil.ext" or a scalar
BASIC/RT-11 Language Summary

string variable. The number of blocks may be specified by b.

OVERLAY "file descriptor" Used to overlay or merge the program currently in memory with a specified file, and continue execution.

PRINT list Used to output data to the terminal. The list can contain expressions or text strings.

PRINT "text" Used to print a message or a string of characters.

PRINT #expression: expression list Outputs to a particular output device, as specified in an OPEN statement.

PRINT TAB(x); Used to space to the specified column.

RANDOMIZE Causes the random number generator to calculate different random numbers every time the program is run.

READ variable list Used to assign the values listed in a DATA statement to the specified variables.

REM comment Used to insert explanatory comments into a BASIC program.

RESTORE Used to reset data block pointer so the same data can be used again.

RESTORE #n Rewinds the input sequential file #n to the beginning.

RETURN Used to return program control to the statement following the last GOSUB statement.

STOP Used at the logical end of the program to terminate execution.

F.2 BASIC/RT-11 COMMANDS

The following key commands halt program execution, erase characters or delete lines.

<table>
<thead>
<tr>
<th>Key</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTMODE</td>
<td>Deletes the entire current line. Echoes DELETED message (same as CTRL U). On some terminals the ESC key must be used.</td>
</tr>
<tr>
<td>CTRL C</td>
<td>Interrupts execution of a command or program and returns control to the RT-11 monitor. BASIC may be restarted without loss of the current program by using the monitor REENTER command.</td>
</tr>
</tbody>
</table>
BASIC/RT-11 Language Summary

CTRL O  Stops output to the terminal and returns BASIC to the READY message when program or command execution is completed.

CTRL U  Deletes the entire current line. Echoes DELETED message (same as ALTMODE).

<-  (SHIFT 0) Deletes the last character typed and echoes a backarrow (same as RUBOUT). On VT05 or LA30 use the underscore (_) key.

RUBOUT  Deletes the last character typed and echoes a backarrow (same as <-).

The following commands list, punch, erase, execute and save the program currently in memory.

<table>
<thead>
<tr>
<th>Command</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>Sets the array and string buffers to nulls and zeroes.</td>
</tr>
<tr>
<td>LIST</td>
<td>Prints the user program currently in memory on the terminal.</td>
</tr>
<tr>
<td>LIST line number</td>
<td></td>
</tr>
<tr>
<td>LIST -line number</td>
<td></td>
</tr>
<tr>
<td>LIST line number-[END]</td>
<td></td>
</tr>
<tr>
<td>LIST line number-line number</td>
<td>Types out the specified program line(s) on the terminal.</td>
</tr>
<tr>
<td>LISTNH line number</td>
<td></td>
</tr>
<tr>
<td>LISTNH -line number</td>
<td></td>
</tr>
<tr>
<td>LISTNH line number-[END]</td>
<td></td>
</tr>
<tr>
<td>LISTNH line number-line number</td>
<td>Lists the lines associated with the specified numbers but does not print a header line.</td>
</tr>
<tr>
<td>NEW &quot;filnam&quot;</td>
<td>Does a SCRatch and sets the current program name to the one specified.</td>
</tr>
<tr>
<td>OLD &quot;file&quot;</td>
<td>Does a SCRatch and inputs the program from the specified file.</td>
</tr>
<tr>
<td>RENAME &quot;filnam&quot;</td>
<td>Changes the current program name to the one specified.</td>
</tr>
<tr>
<td>REPLACE &quot;dev:filnam.ext&quot;</td>
<td>Replaces the specified file with the current program.</td>
</tr>
<tr>
<td>RUN</td>
<td>Executes the program in memory.</td>
</tr>
<tr>
<td>RUNNH</td>
<td>Executes the program in memory but does not print a header line.</td>
</tr>
<tr>
<td>SAVE &quot;dev:filnam.ext&quot;</td>
<td>Outputs the program in memory as the specified file.</td>
</tr>
</tbody>
</table>
BASIC/RT-11 Language Summary

SCRatch  Erases the entire storage area.

F.3 BASIC/RT-11 FUNCTIONS

The following functions perform standard mathematical operations in BASIC.

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(x)</td>
<td>Returns the absolute value of x.</td>
</tr>
<tr>
<td>ATN(x)</td>
<td>Returns the arctangent of x as an angle in radians in the range + or - pi/2.</td>
</tr>
<tr>
<td>BIN(x$)</td>
<td>Computes the integer value of a string of blanks (ignored), 1's and 0's.</td>
</tr>
<tr>
<td>COS(x)</td>
<td>Returns the cosine of x radians.</td>
</tr>
<tr>
<td>EXP(x)</td>
<td>Returns the value of e^x where e=2.71828.</td>
</tr>
<tr>
<td>INT(x)</td>
<td>Returns the greatest integer less than or equal to x.</td>
</tr>
<tr>
<td>LOG(x)</td>
<td>Returns the natural logarithm of x.</td>
</tr>
<tr>
<td>OCT(x$)</td>
<td>Computes an integer value from a string of blanks (ignored) and digits from 0 to 7.</td>
</tr>
<tr>
<td>RND(x)</td>
<td>Returns a random number between 0 and 1.</td>
</tr>
<tr>
<td>SGN(x)</td>
<td>Returns a value indicating the sign of x.</td>
</tr>
<tr>
<td>SIN(x)</td>
<td>Returns the sine of x radians.</td>
</tr>
<tr>
<td>SQR(x)</td>
<td>Returns the square root of x.</td>
</tr>
<tr>
<td>TAB(x)</td>
<td>Causes the terminal type head to tab to column number x.</td>
</tr>
</tbody>
</table>

The string functions are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC(x$)</td>
<td>Returns as a decimal number the seven-bit internal code for the one-character string (x$).</td>
</tr>
<tr>
<td>CHR$(x)</td>
<td>Generates a one-character string having the ASCII value of x.</td>
</tr>
<tr>
<td>DAT$</td>
<td>Returns the current date in the format 07-MAY-73.</td>
</tr>
<tr>
<td>LEN(x$)</td>
<td>Returns the number of characters in the string (x$).</td>
</tr>
<tr>
<td>POS(x$,y$,z)</td>
<td>Searches for and returns the position of the first occurrence of y$ in x$ starting with the zth position.</td>
</tr>
</tbody>
</table>
BASIC/RT-11 Language Summary

SEG$(x$,y$,z$) Returns the string of characters in positions y through z in x$.

STR$(x$) Returns the string which represents the numeric value of x$.

TRM$(x$) Returns x$ without trailing blanks.

VAL(x$) Returns the number represented by the string (x$).

F.4 BASIC/RT-11 ERROR MESSAGES

The information that formerly appeared in Section F.4 has now been incorporated into the RT-11 System Message Manual, DEC-11-ORMEA-A-D. BASIC/RT-11 error messages are also found in the BASIC/RT-11 Language Reference Manual, DEC-11-LBACA-D-D.
APPENDIX G

FORTRAN LANGUAGE SUMMARY

FORTRAN IV is a problem-solving language designed to allow scientists and engineers to express mathematical operations in a familiar format.

A FORTRAN source program is composed of a series of statements. Statements are usually made up of one- or two-word commands, which, when used in conjunction with data variables and constants, can be used to control program execution, assign values to variables and read and write data. The FORTRAN library contains routines to simplify mathematical functions such as computing sines, cosines, square roots, etc. The source program is translated by the FORTRAN compiler into a machine language program which, when linked with the FORTRAN object time routines, can be executed by the computer.

This appendix describes first how to run a FORTRAN program as a foreground job, and then summarizes briefly the RT-11 FORTRAN IV language, statements, and error messages. For details, refer to the PDP-11 FORTRAN Language Reference Manual and the RT-11/RSTS/E FORTRAN IV User's Guide.

The FORTRAN programmer may also want to read Appendix O for details concerning the FORTRAN System Subroutine Library. SYSLIB allows the programmer to easily call and use the programmed requests described in Chapter 9. A summary of the SYSLIB library calls is in Table 0-1.

G.1 RUNNING A FORTRAN PROGRAM IN THE FOREGROUND

Since the FORTRAN Object Time System needs work areas to perform some of its functions, the FRUN/N!x monitor command must be used to allocate the needed space when running a FORTRAN program as a foreground job. The following formula can be used to calculate the decimal number (x) which must be designated to the /N option:

\[ x = \frac{1}{2} [378 + (29*N) + (R-136) + A*512] \]

where:

\[ N = \text{the value of the } /N \text{ FORTRAN compiler switch option (the compiler defaults this to 6).} \]

\[ R = \text{the value of the } /R \text{ FORTRAN compiler switch option (the compiler defaults this to 210 octal, 136 decimal).} \]
FORTRAN Language Summary

A = the number of I/O buffers in simultaneous use during execution. (Console terminal I/O does not require any buffers. Each open logical unit typically requires one buffer.)

For example, to calculate x for a program (FILENA.REL) which uses only terminal I/O and allows the compiler to assign the standard defaults, set the formula as follows (all values are decimal):

\[ x = \frac{1}{2} [378 + (29 \times 6) + (136 - 136) + (\# \times 512)] \]

276 decimal words are required. The FORTRAN program is executed using the FRUN command as shown:

```
.FRUN FILENA.REL/N!276 <CR>
```

NOTE

An exception can occur if the size of the linked FORTRAN program is less than the size of the USR (2K) and the USR is to be swapped. In this case, additional space must be allocated to allow the USR to successfully swap over the linked FORTRAN program. The additional space needed may be calculated by subtracting the size of the linked FORTRAN program from 10000 octal bytes (the size of the USR) and adding this result to the value of x previously calculated.

G.2 FORTRAN CHARACTER SET

The characters that may appear in a FORTRAN statement are restricted to those listed below.

1. The letters A through Z
2. The numerals 0 through 9
3. The following "special" characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Space or Blank</td>
</tr>
<tr>
<td>=</td>
<td>Equals</td>
</tr>
<tr>
<td>+</td>
<td>Plus</td>
</tr>
<tr>
<td>-</td>
<td>Minus</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk</td>
</tr>
<tr>
<td>/</td>
<td>Slash</td>
</tr>
<tr>
<td>(</td>
<td>Left Parenthesis</td>
</tr>
</tbody>
</table>
FORTRAN Language Summary

) \hspace{3em} \text{Right Parenthesis}
, \hspace{3em} \text{Comma}
. \hspace{3em} \text{Decimal Point}
' \hspace{3em} \text{Apostrophe}
" \hspace{3em} \text{Double Quote}
$ \hspace{3em} \text{Dollar Sign}

Other printable characters may appear in a FORTRAN statement only as part of a Hollerith constant or alphanumeric literal. Any printable character may appear in a comment.

G.3 EXPRESSION OPERATORS

Each group of operators is shown in order of descending precedence during execution.

<table>
<thead>
<tr>
<th>Type</th>
<th>Operator</th>
<th>Operates Upon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
<td>numeric constants, variables and expressions</td>
</tr>
<tr>
<td>* /</td>
<td>multiplication, division</td>
<td></td>
</tr>
<tr>
<td>+ -</td>
<td>addition, subtraction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unary minus</td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.GT.</td>
<td>greater than</td>
<td>variables, constants, and arithmetic expressions (all relational operators have equal priority)</td>
</tr>
<tr>
<td>.GE.</td>
<td>greater than or equal to</td>
<td></td>
</tr>
<tr>
<td>.LT.</td>
<td>less than</td>
<td></td>
</tr>
<tr>
<td>.LE.</td>
<td>less than or equal to</td>
<td></td>
</tr>
<tr>
<td>.EQ.</td>
<td>equal to</td>
<td></td>
</tr>
<tr>
<td>.NE.</td>
<td>not equal to</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.NOT.</td>
<td>.NOT.A is true if and only if A is false, and is false if A is true</td>
<td>logical variables, logical constants, integer variables and constants, logical integers and expressions</td>
</tr>
<tr>
<td>.AND.</td>
<td>A.AND.B is true if and only if A and B are both true and is false if either A or B is false.</td>
<td></td>
</tr>
<tr>
<td>.OR.</td>
<td>A.OR.B is true if and only if either A or</td>
<td></td>
</tr>
</tbody>
</table>
FORTRAN Language Summary

B or both A and B
are true, and false
if both A and B
are false.

_.EQV._
A.EQVB is true if and
only if A and B
are both true or if A
and B are both false.

_.XOR._
A.XORB is true if and only
if A is true and B is
false or if B is true
and A is false.

G.4 SUMMARY OF FORTRAN STATEMENTS

The following is a summary of statements available in the PDP-11
FORTRAN IV language, including the general format for the statement
and its effect.

<table>
<thead>
<tr>
<th>Statement Formats</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic/Logical Assignment</td>
<td>The value of the arithmetic or logical expression is assigned to the variable.</td>
</tr>
<tr>
<td>variable=expression</td>
<td></td>
</tr>
<tr>
<td>Arithmetic Statement Functions</td>
<td></td>
</tr>
<tr>
<td>name(var1, var2, ...) = expression</td>
<td>Creates a user-defined function having the variables (var) as dummy arguments. When referenced, the expression is evaluated using the actual arguments in the function call.</td>
</tr>
<tr>
<td>ACCEPT</td>
<td></td>
</tr>
<tr>
<td>ACCEPT f, list</td>
<td>Reads input from logical unit 5 (TT is default); f is the format statement label and list is an optional data list.</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>Assigns the statement number n to the integer variable ivar.</td>
</tr>
<tr>
<td>ASSIGN n TO ivar</td>
<td></td>
</tr>
</tbody>
</table>
FORTRAN Language Summary

BACKSPACE

BACKSPACE u
The currently open file on logical unit number u is backspaced one record.

BLOCK DATA

BLOCK DATA
Specifies the subprogram which follows as a BLOCK DATA subprogram.

CALL

CALL name
CALL name (arg1, arg2,...)
Calls the SUBROUTINE subprogram with the name specified, passing the actual arguments (arg) to replace the dummy arguments in the SUBROUTINE definition.

COMMON

COMMON/name1/var1, var2,.../name1/var(j),var(k)
Reserves one or more blocks of storage space under the name specified to contain the variables (var) associated with that block name.

CONTINUE

CONTINUE
Causes no processing.

DATA

DATA var1, var2,.../va1, va2,.../
Causes elements in the list of values (val) to be initially stored in the corresponding elements of the list of variable names (var).

DECODE

DECODE (c,f,v)list
Changes the elements in the list of variables from ASCII into the desired internal format; c is the number of characters, f is the format, and v is the name of an array.
FORTRAN Language Summary

DEFINE FILE

DEFINE FILE u(m,n,U,lvar),...

Defines the record structure of a disk or DECtape direct access file where u is the logical unit number, m is the number of fixed length records in the file, n is the length in words of a single record, U is a fixed argument, and lvar is the associated variable pointing to the next record.

DIMENSION

DIMENSION s1,s2,...,sk

Reserves storage space for the specified array(s). Each s consists of an array variable name followed by up to seven integer constants, separated by commas and enclosed in parentheses.

DO

DO n lvar = e1,e2,e3

1. Sets integer variable ivar = to expression (e1, e2, e3 must result in an integer quantity)

2. Executes statements through statement number n

3. Increments lvar = lvar+e3

4. If e3>0 and ivar <=e2, or e3<0 and ivar >=e2, goes back to 2 above

END

END

ENCODE

ENCODE (c,f,v)list

Changes the elements in the list of variables into ASCII format; c is the number of characters in the buffer, f is the format statement number, and v is the name of the array to be used as the resulting buffer.

END FILE

END FILE u

The file currently open on logical unit number u is closed.
FORTRAN Language Summary

EQUIVALENCE

EQUIVALENCE (var1, var2, ...), (varj, vark, ...)

Each of the variables within a set of parentheses is assigned the same storage location.

EXTERNAL

EXTERNAL namel, name2, ...

Informs the system that the names specified are those of FUNCTION or SUBROUTINE subprograms.

FIND

FIND(u'r)

Positions the direct access file on logical unit number u to record r and sets the associated variable to record number r.

FORMAT

n FORMAT(field specification,...)

Describes the format in which one or more records are to be transmitted; the statement number n must be specified (the field specifications may include the field descriptors, I, O, F, E, D, G, L, A, H, X, T, Q, and $, delimited by field separators , and /).

FUNCTION

FUNCTION name
FUNCTION name (var1, var2, ...)
type FUNCTION name (var1, var2, ...)

Begins a FUNCTION subprogram, indicating the program name and any dummy variable names (var). An optional type specification can be included.

GOTO

Unconditional

GOTO n

Transfers control to statement number n.
FORTRAN Language Summary

Computed

GOTO (k1,k2,...,kn), e

Transfers control to the statement number ki where i = value of expression e. If e<1 or e>n no transfer takes place.

Assigned

GOTO i
GOTO i,(k1,k2,...,kn)

Transfers control to the statement most recently associated with i by an ASSIGN statement.

IF

Arithmetic

IF (expression) n1,n2,n3

Transfers control to statement number n depending upon the value of the expression. If the value of the expression is less than zero, transfers to n1; if the value of the expression is equal to zero, transfers to n2, if the value of the expression is greater than zero, transfers to n3.

Logical

IF (expression) statement

Executes the statement if the logical expression tests true.

IMPLICIT

IMPLICIT type (a1,b1-b2,...),...

The elements a and b represent single (or a range of) letter(s) whose presence as the initial letter of a variable specifies the variable to be of that type, if that variable is not explicitly given a type.

PAUSE

PAUSE
PAUSE display

Suspends program execution and prints the octal constant, Hollerith constant, or alphanumeric literal display, if one is specified. Program execution is resumed by typing a carriage return.
FORTRAN Language Summary

PRINT

PRINT f,list

Writes output on logical unit 6 (LP is default); f is the format statement label and list is an optional data list.

READ

Formatted

READ(u,f)
READ(u,f)list
READ f,list

Reads at least one logical record from device u according to format specification f and assigns values to the variables in the list. Logical unit number 1 is used as default in the third form above.

Unformatted

READ(u)
READ(u) list

Reads one logical record from device u, assigning values to the variables in the list.

Direct Access

READ(u'r)list

Reads from logical unit number u, record number r, and assigns values to the variables in the list.

Transfer of Control on Error

END=n
ERR=m
END=n,ERR=m

Optional elements in the READ statement list (e.g., READ(u,f, END=n, ERR=m) allowing control transfer on error conditions. If an end-of-file condition is detected and END=n is specified, execution continues at statement n. If a hardware I/O error occurs and ERR=m is specified, execution continues at statement m.
FORTRAN Language Summary

RETURN

RETURN

Returns control to the calling program from the current subprogram.

REWIND

REWIND u

Logical unit number u is repositioned to the beginning of the currently opened file.

STOP

STOP display

Terminates program execution and prints the octal constant, Hollerith constant, or alphanumeric literal display, if one is specified.

SUBROUTINE

SUBROUTINE name

SUBROUTINE name (var1, var2,...)

Begins a SUBROUTINE subprogram, indicating the program name and any dummy variable names (var).

TYPE

TYPE f,list

Writes output on logical unit 7 (TT: is default); f is the format statement label and list is an optional data list.

Type Declarations

type var1, var2, ..., vark

The variable names, (var), are assigned the specified data type in the program unit. type is one of:

INTEGER(*2,*4)
REAL(*4,*8)
DOUBLE PRECISION
COMPLEX(*8)
LOGICAL(*4,*1)

The optional * indicates that an explicit length is to be set for the variable; the number that follows is the length in bytes. The length must be one of the permitted values above.
FORTRAN Language Summary

WRITE

Formatted

WRITE (u,f)
WRITE (u,f) list

Causes one or more logical records containing the values of the variables in the list to be written onto device u according to the format specification f.

Unformatted

WRITE (u)
WRITE (u)list

Causes one logical record containing the values of the variables in the list to be written onto device u.

Direct Access

WRITE (u'r) list

Causes one logical record containing the values of the variables in the list to be written onto record r of the logical unit number u.

Transfer of Control on Error

END=n
ERR=m
END=n,ERR=m

Optional elements in the WRITE statement list (e.g., WRITE u,f, END=n, ERR=m) allowing control transfer on error conditions. If an end-of-file condition is detected and END=n is specified, execution continues at statement n. If a hardware I/O error occurs and ERR=m is specified, execution continues at statement m.

G.5 COMPILER ERROR DIAGNOSTICS

The information that formerly appeared in Sections G.5 and G.6 has been incorporated into the RT-11 System Message Manual, DEC-11-ORMEA-A-D. RT-11 FORTRAN error messages are also found in the RT-11/RSTS/E FORTRAN IV User's Guide, DEC-11-LRRUA-A-D.
APPENDIX H

F/B PROGRAMMING AND DEVICE HANDLERS

H.1 F/B PROGRAMMING IN RT-11, VERSION 2

Certain programming conventions must be observed in RT-11, Version 2, which were not required in Version 1. These conventions are necessary to permit interrupt routines to function properly while running two jobs in the F/B environment. All Version 2 device handlers follow these conventions; the user is urged to consult the listings of the example handlers (Section H.3) as they illustrate some of the techniques discussed.

NOTE

Device handlers distributed with RT-11, Version 1, will not work properly with Version 2. Also, any user-written device handlers should be re-written to comply with the Version 2 conditions. Instructions for interfacing new handlers to RT-11 are provided in the RT-11 Software Support Manual. (DEC-11-ORPGA-B-D). Section H.2 describes Version 2 device handlers.

The procedures described in this appendix are necessary and must be followed to prevent system failures when jobs are running under RT-11. If at any time a program which services its own interrupts (and does not follow the guidelines described here) is run with another job, the system may malfunction. Therefore, it is required that all programs follow the procedures that are indicated here.

H.1.1 Interrupt Priorities

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

H.1.2 Interrupt Service Routine

If the conventions outlined in Section H.1.1 are followed, when an interrupt occurs, the processor priority will be 7. The first task of the interrupt service routine is to lower the processor priority to the correct value. This can be done using the .INTEN macro call. The call is:

```
.INTEN .priority
```
or

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

The .INTEN call is explained in Chapter 9, Programmed Requests. On return from the .INTEN call, the processor priority is set properly; registers 4 and 5 have been saved and can be used without the necessity of saving them again.

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

```
DEVPRI=5

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

If the contents of the processor status word, loaded from the interrupt vector, is of significance to the interrupt service routine (e.g., the condition bits), the PS should be moved to a memory location (not the stack) before issuing the .INTEN. A device handler's interrupt service routine uses the monitor stack and should avoid excessive use of stack space.

H.1.3 Return from Interrupt Service

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

When a device handler has completed a transfer and is ready to return to the monitor via the internal monitor completion address, R4 must be pointing to the fifth word of the handler. It is no longer necessary to have R0 and R3 on the stack, as it was in Version 1. The user is urged to consult the example handlers at the end of this appendix, which illustrates the operation of device handlers.

H.1.4 Issuing Programmed Requests at the Interrupt Level

Programmed requests from interrupt routines must be preceded by a .SYNCH call. This ensures that the proper job is running when the programmed request is issued. The .SYNCH call assumes that nothing has been pushed onto the stack by the user program between the .INTEN call and the .SYNCH call. On successful completion of a .SYNCH, R0 and R1 have been saved and are free to be used. R4 and R5 are no longer free, and should be saved and restored if they are to be used.

Programmed requests that require USR action should not be called from within interrupt routines.

January 1976
F/B Programming and Device Handlers

H.1.5 Setting Up Interrupt Vectors

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

H.1.6 Using .AECT Directives in Relocatable Image Files

With some exceptions, user programs in REL format should not contain .AECT directives. The Linker does not allow .AECTs above 1000 when the /R switch is used; in general all .AECTs below 1000 are ignored. The exceptions are:

<table>
<thead>
<tr>
<th>.AECT</th>
<th>Words</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34,36</td>
<td>2</td>
<td>TRAP instruction vector</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>program start address</td>
</tr>
<tr>
<td>42</td>
<td>1</td>
<td>stack start address</td>
</tr>
<tr>
<td>44</td>
<td>1</td>
<td>job status word</td>
</tr>
<tr>
<td>46</td>
<td>1</td>
<td>USR swap address</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>program high limit</td>
</tr>
</tbody>
</table>

.AECTs may be used to initialize any of the above locations providing they are not in a program to be linked for the foreground. To initialize any other locations between 0-777, the user should first .PROTECT the appropriate words, and then move the correct values into place.

A job which contains overlays cannot have any .AECTs at all, including those listed above. If overlays are to be used, the program start address should be set up as the argument for the .END statement; the other values can be initialized when the program is initiated.

Since .AECTs into device vectors are not permitted, the user program must initialize vectors at runtime. To do this, execute a .PROTECT for the specified vector. If the .PROTECT is successful, it is safe to move values into the vectors. If the .PROTECT request fails, it is an indication that the vector is already in use by another job and that it is not safe to initialize the vector.

H.1.7 Using .SETTOP

Proper use of .SETTOP is vital to preserve system integrity. Since RT-11 employs no hardware memory protection, the user must at all times observe the value returned in R0 from a .SETTOP request. It must never be assumed that the value requested by the program is the value returned by .SETTOP.

H-3 January 1976
F/B Programming and Device Handlers

The monitor loads foreground programs into a memory partition just large enough for the program plus job header (impure area). No free space is available for a .SETTOP request, which may cause some programs to fail, in particular, any FORTRAN program. Free space may be provided within the program using, for example, the .BLKW directive, or it may be allocated at runtime using the PRUN /N switch. Appendix G, Section G.1, explains how to allocate more space for FORTRAN programs.

H.1.8 Making Device Handlers Resident

Device handlers used by a foreground job must be made permanently resident with a LOAD command. A .FETCH from a foreground job of a non-resident handler will cause a fatal error and abort the job. Therefore, a program written specifically for the foreground, or one that is to be run in either foreground or background partitions, should first execute a .DSTATUS on the device, to determine if the handler is resident, before issuing a .FETCH directive.

H.2 DEVICE HANDLERS

This section deals with the device handlers which are part of the RT-11 System, Version 2. Any device dependent information or general information required by the user is contained here. No mention of a handler implies that no special conditions must be met to use that device (all disks except diskette and DECTape are in this category, and therefore are not covered here).
This page intentionally blank.
Differences Between V1 and V2 Device Handlers

User-written device handlers must, in all cases, conform to the standard practices for Version 2. Since the last word of a device handler is used by the monitor, the user should be sure to include one extra word at the end of his program when indicating the size of the handler.

The differences between Version 1 and Version 2 handlers include the following:

A. Header Words

In Version 1 handlers, the third header word was taken to be the priority at which the device interrupt was taken. In Version 2, this word should be 340, indicating that the interrupt should be taken at priority level 7. (The priority level is set to 7 by the FETCH code regardless of what appears in the third word of the header.)

B. Entry Conditions

It is not necessary to save/restore registers when the handler is first entered, although to do so is not harmful. In Version 1, it was necessary to preserve R5 on first entering the handler.

C. Interrupt Handling

When an interrupt occurs, Version 2 handlers must execute an .INTEN request or its equivalent. (This was not necessary in Version 1.) On return from the .INTEN, R4 and R5 may be used as scratch registers. Device handlers may not do EMT requests without executing a .SYNCH request. (Refer to Chapter 9 for explanations of .INTEN and .SYNCH.)

The handler must return from an interrupt via an RTS PC rather than an RTI, as in Version 1. The .INTEN request essentially reenters the handler via a JSR PC, and thus the RTS PC returns control to the monitor, which executes the RTI when appropriate.

When the transfer is complete, the handler must exit to the monitor to terminate the transfer and/or enter a completion routine. When return is made to the monitor, R4 should point to the fifth word of the handler.

D. Abort Entry Point

All Version 2 device handlers must have an abort entry point to which control is transferred on CTRL C, hard .EXIT, or
F/B Programming and Device Handlers

.HRESET. This abort entry point is located one word before the interrupt service routine entry point and must contain a BR instruction to abort code. The abort code must stop the device operation and then exit through the jump into the completion code in RMON. Some devices, such as moving head disks, cannot be effectively stopped. In this case, it is permissible for the abort code to consist of just an RTS PC instruction. Examples are the RK05 and RPR02/RP03 device drivers. See Section H.3 for examples of device handlers with abort entry code.

The user is urged to refer to the example handlers enclosed. The same general techniques can be used to interface user device handlers.

H.2.1 PR  (High-Speed Paper Tape Reader)

Unlike the PR handler for Version 1, the Version 2 PR handler does not print a † on the terminal when it is entered for the first time. The tape must be in the reader when the command is issued, or an input error occurs. This prohibits any two-pass operations from being done using PR. For example, linking and assembling from PR will not work properly; an input error will occur when the second pass is initiated. The correct (and recommended) procedure is to use PIP to transfer the paper tape to disk or DECTape, and then perform the operation on the transferred file.

H.2.2 TT  (Handler for Console Terminal)

The console terminal may be used as a peripheral device by using the TT handler. Note that:

1. A † is typed when the handler is ready for input.

2. CTRL Z can be used to specify the end of input to TT. No carriage return is required after the CTRL Z. If CTRL Z is not typed, the TT handler accepts characters until the word count of the input request is satisfied.

3. CTRL O, struck while output is directed to TT, causes an entire output buffer (i.e., all characters currently queued) to be ignored. This is somewhat different than the normal action of CTRL O while at the console.

4. A single CTRL C struck while typing input to TT causes a return to the monitor. If output is directed to TT, a double CTRL C is required to return to the monitor if F/B is running. If the S/J monitor is running, only a single CTRL C is required to terminate output.

5. The TT handler can be in use for only one job (foreground or background) at a time, and for only one function (input or output) at a time. The terminal communication for the job not using TT is not affected at all.

6. The user may type ahead to TT; the input ring buffer is emptied before the keyboard is referenced. The terminating CTRL Z may also be typed ahead.

7. If the main line code of a job is using TT for input, and a completion routine does a .TTYIN, typed characters will be passed unpredictably to the .TTYIN and TT. Therefore, this should not be done.

H-5  January 1976
P/B Programming and Device Handlers

8. If a job sends a buffer to TT for output and then does a .TTYOUT or a .PRINT, the output from the latter is delayed until the handler completes its transfer. If a TT output operation is started when the monitor's terminal output ring buffer is not empty (i.e., before the print-ahead is complete), the handler supersedes the ring buffer until its transfer is complete.

H.2.3 CR (Card Readers)

The card reader handler can transfer data either as ASCII characters in DEC 026 or DEC 029 card codes (Section H.4), or as column images controlled by the SET command (see Chapter 2). In ASCII mode (SET CR NOIMAGE), invalid punch combinations are decoded as the error character 134 (octal)--backslash. In IMAGE mode, no punch combination is invalid; each column is read as 12 bits of data right-justified in one word of the input buffer. The handler continues reading until the transfer word count is satisfied or until a standard end-of-file card is encountered (12-11-0-1-6-7-8-9 punch in column 1; the rest of the card is arbitrary). On end-of-file, the buffer is filled with zeroes and the request terminates successfully; the next input request from the card reader gives an end-of-file error. Note that if the transfer count is satisfied at a point which is not a card boundary, the next request continues from the middle of the card with no loss of information. If the input hopper is emptied before the transfer request is complete, the handler will loop until the hopper is reloaded and the "START" button on the reader is pressed again. The transfer will then continue until completion or until another hopper empty condition exists. End-of-file is not reported on the hopper empty condition. The handler will loop if the hopper empties during the transfer regardless of the status of the SET CR HANG/NOHANG option. No special action is required to use the card reader handler with the CM-11 mark sense card reader. The program should be aware of the fact that mark sense cards may contain less than 80 characters. Note also that when the CR handler is set to CRLF or TRIM and is reading in IMAGE mode, unpredictable results may occur.

H.2.4 MT/CT (Magtape (TM11, TU16) and Cassette (TAll))

These devices have a file structure which differs from the common RT-11 structure. Each handler is capable of entirely supporting its own file structure and of allowing RT-11 users full access to the devices without being totally familiar with them.

H.2.4.1 General Characteristics - Both magtape and cassette can operate in two possible modes: "hardware" mode and "software" mode. These names refer to the type of operation which can be performed on the device at a given time. Software mode is the normal mode of operation and the mode used when accessing the device through any of the RT-11 system programs. In software mode, the handler automatically attains to file headers and uses a fixed record length to transfer data (256-word for magtape; 64-word for cassette).

Hardware mode allows the user to read or write any format desired, using any record size. In this mode, the word count is taken as the physical record size.
P/B Programming and Device Handlers

When the handlers are initially loaded by either the .FETCH programmed request or the LOAD command, only software functions are permitted. To switch from software to hardware mode, either a rewind or a nonfile-structured lookup must be performed. (A nonfile-structured lookup is a lookup in which the first word of the filename is null.) Note that if the TM11 handler has been modified (as described in the RT-11 System Generation Manual) to one of the following: 888 BPI non-dump 7-track, 556 BPI 7-track, or 280 BPI 7-track, the handler is used somewhat differently. Refer to the RT-11 System Generation Manual for details.

NOTE

Due to the size of the TM11 and TJU16 magtape handlers, users who have systems with less than 20K, and who have loaded both handlers simultaneously, may experience user memory restrictions.

In software mode, the following functions are permitted:

ENTER - Open new file for output
LOOKUP - Open existing file for input and/or output
DELETE - Delete an existing file on the device
CLOSE - Close a file which was opened with ENTER or LOOKUP
READ/WRITE - Perform data transfer requests

In ENTER, LOOKUP, and DELETE, an optional file count parameter can be specified for use with magtape or cassette. Its meaning is as follows:

<table>
<thead>
<tr>
<th>Count Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=0</td>
<td>A rewind is done before the operation.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>No rewind is done. The value of the count is taken as a limit of how many files to skip before performing the operation (e.g., a count of 2 will skip over, at most, one file. A count of 1 will not skip at all).</td>
</tr>
<tr>
<td>&lt;0</td>
<td>A rewind is done. The absolute value of the switch value is then used as the limit.</td>
</tr>
</tbody>
</table>

If the file indicated in the request is located before the limit is exhausted, the search succeeds at that point.

As an example, consider:

```
.LOOKUP #AREA,#8,#PTR,#5
BCS A1
.

AREA: BLKW 10,
PTR: .RAD50 /MT0/
 .RAD50 /EXAMPLMAC/
```

In this case, the file count argument is +5, indicating that no rewind is to be done and that MT0 is to be searched for the indicated file (EXAMPLMAC). If the file is not found after four files have been skipped, or if an end-of-tape occurs in that space, the search is...
stopped, and the tape is positioned either at the end-of-tape (EOT) or at the start of the fifth file. If the named file is found within the five files, the tape is positioned at its start. If the EOT is encountered first, an error is generated.

As another example:

```
.LOOKUP #AREA,#0,#PTR,#=5
```

This performs a rewind, and then uses a file count of five in the same way the above example does.

H.2.4.2 Handler Functions

1. **LOOKUP**

   If the filename (or the first word of the filename) is null, the operation is considered to be a nonfile-structured LOOKUP. This operation puts the handler into hardware mode. A rewind is automatically done in this case.

   If the filename is not null, the handler tries to find the indicated file. LOOKUP uses the optional file count as illustrated above. Only software functions are allowed.

2. **DELETE**

   DELETE will delete a file of the indicated name from the device. DELETE also uses the file count argument, and can thus do a delete of a numbered file as well as a delete by name. When a file is deleted from MT or CT, an unused space is created there. However, it is not possible to reclaim that space, as it is when the device is random access. The unused spot will remain until the volume is re-initialized and rewritten.

   If a filename is not present, a nonfile-structured DELETE is performed and the tape is zeroed.

3. **ENTER**

   ENTER creates a new file of the indicated name on the device. ENTER uses the optional file count, and can thus ENTER a file by name or by number. If ENTER by name is done, the handler deletes any files of the same name it finds in passing. If ENTER by number is done, the indicated number of files is skipped, and the tape is positioned at the start of the next file.

**NOTE**

Care must be used in performing numbered ENTERs, as it is possible to ENTER a file in the middle of existing files and thus destroy any files from the next file to the end of the tape.

It is also possible to create more than one file with the same name, since ENTER
will only delete files of the same name it sees while passing down the tape. If an ENTER is done with a count greater than 0, no rewind is performed before the file is entered. If a file of the same name is present at an earlier spot on the tape, the handler cannot delete it. A nonfile-structured ENTER performs the same function as a nonfile-structured LOOKUP but does not rewind the tape. Since both functions allow writing to the tape without regard to the tape's file structure, they should be used with care on a file-structured tape.

4. CLOSE

CLOSE terminates operations to a file on magtape or cassette and resets the handler to allow more LOOKUPS, ENTERs, or DELETEs. If a CLOSE is not performed to an ENTERed file, the end-of-tape mark will be missing and no new files can be created on that volume. In this case, the last file on the tape must be rewritten and CLOSED to create a valid volume.

5. READ/WRITE

READ and WRITE are unique in that they can be done either in hardware or software mode. In software mode (file opened with LOOKUP or ENTER), records are written in a fixed size (256 words for magtape, 64 words for cassette). The word count specified in the operation is translated to the correct number of records. On a READ, the user buffer is filled with zeroes if the word count exceeds the amount of data available.

Following is a discussion of how the various parameters for READ/WRITE are used for magtape and cassette.

a. Block Number

For READ operations to magtape, the block number is used for random access (i.e., blocks need not be read sequentially from magtape). WRITE operations, however, disregard the block number and merely write the next sequential block. Block 0 on either READ or WRITE causes the device to rewind to the start of the file. A subsequent WRITE will destroy all previous output to that file.

For cassette, only sequential operations are performed. If the block number is 0, the cassette is rewound to the start of the file. Any other block number is disregarded.
F/B Programming and Device Handlers

b. Word Count

On a magtape READ, if the word count is 0, the block number is ignored and the next sequential record is read, no matter how big it is.

NOTE

Care must be taken when performing a magtape READ, because the READ will be done even if the record to be read is larger than the buffer allocated.

On a magtape WRITE, if the word count is 0, one 256-word block is written.

If the word count for cassette is 0, the following conditions are possible:

If the block number is non-zero, the operation is actually a file-name seek. The block number is interpreted as the file count argument, as discussed in the above example of LOOKUP. The buffer address should point to the RAD50 of the device and file to be located. This feature essentially allows an asynchronous LOOKUP to be performed. The standard LOOKUP request does not return control to the user program until the tape has been positioned properly, whereas this asynchronous version will return control immediately and interrupt when the file is positioned.

The user may then do a synchronous, positively numbered LOOKUP to the file just positioned, thus avoiding a long synchronous search of the tape.

If the block number is 0, a CRC (cyclic redundancy check) error occurs.

Following is a description of the allowed hardware mode functions for the handlers, as well as examples of how to call them. In general, special functions are called by using the .SPFUN request; examples of usage follow each function. The special functions require a channel number as an argument. The channel must initially be opened with a non-file structured LOOKUP which places the handler in hardware mode.

The general form of the .SPFUN request is:

.SPFUN .area,.chan,.code,.buff,.wcnt,.blk,.crtn

where:

.code is the function code to be performed.

The request format is:

| R0 = .area: | 32 | .chan |
|            |    | .blk  |
|            |    | .buff |
|            |    | .wcnt |
|            | .code 377 |
|            | .crtn |

H-10
F/B Programming and Device Handlers

1. Magtape Special Functions

a. Rewind (Code = 373) - Rewinds the tape to the load point. This puts the unit into hardware mode (as does a nonfile-structured LOOKUP) where any of the other functions may be done.

Sample Macro Call:

```
.SPFUN #AREA,#0,#373,#0
```

The above performs a synchronous rewind on channel 0 (i.e., control will not return before the tape is positioned). An asynchronous rewind could be done with:

```
.SPFUN #AREA,#0,#373,#0,,#CRTN;REWIND MT, CHANNEL 0
```

where CRTN is a completion routine to be entered when the operation is finished. The other arguments are not required for this call.

b. Write End-of-File (Code = 377) - This request writes an end-of-file mark, thus terminating a file.

Sample Macro Call:

```
.SPFUN #AREA,#0,#377,#0;THIS IS THE SYNCHRONOUS FORM
```

The asynchronous form is:

```
.SPFUN #AREA,#0,#377,#0,,#CRTN;WRITE EOF ON
```

MT, CHANNEL 0

c. Forward Space (Code = 376) - This function spaces forward the specified number of records and then stops. If the EOT or EOF is discovered before the count is exhausted, the tape stops there. Note that the number of records to space forward is contained in the word count argument, and the number passed should be the positive value of the desired number.

Sample Macro Call:

```
.SPFUN #AREA,#0,#376,#0,#2
```

;SPACE FORWARD 2
;RECORDS, CHANNEL 0

This spaces forward two records.

d. Back Space (Code = 375) - This is similar to Forward Space except the tape is backed up by the indicated number of blocks. Again, the word count must be the positive value of the number to back up.

H-11
F/B Programming and Device Handlers

Sample Macro Call:

```
.SPFUN #AREA,#0,#375,#0,#0
```

This backs the tape up by two records.

e. Write With Extended Gap (Code = 374) - This request is the same as any of the WRITE requests, except that a longer file gap is written between records. This can be used to get past bad spots on the tape.

Sample Macro Call:

```
.SPFUN #AREA,#0,#374,#BUFF,#100,#0
```

This performs a synchronous write, while:

```
.SPFUN #AREA,#0,#374,#BUFF,#100,#1,#CRTN
```

is asynchronous and goes to CRTN when the operation is complete.

f. Offline (Code = 372) - This request sets the drive to an off-line state. The unit can only be set on-line by manual control.

Sample Macro Call:

```
.SPFUN #AREA,#0,#372,#0
```

Since this in an instantaneous function, no asynchronous forms are required.

2. Cassette Special Functions

With exceptions noted, these requests are identical to those involving magtape.

a. Rewind (Code = 373) - This request is identical to the rewind request for magtape, except that unless a completion routine is specified, control does not return to the user until the rewind completes.

b. Last File (Code = 377) - This request rewinds the cassette and positions it immediately before the sentinel file (logical end-of-tape). The macro call is the same as for magtape code 377.

c. Last Block (Code = 376) - This request rewinds one record. See the magtape code 376 for a sample macro call.

d. Next File (Code = 375) - This request spaces the cassette forward to the next file.

e. Next Block (Code = 374) - This request spaces the cassette forward by one record.

H-12
f. Write File Gap (code = 372) - This request terminates a file written by the user program when in hardware mode.

Sample Macro Call:
```
.SPFUN #AREA,#0,#372,#0
```
This writes a file gap synchronously, while:
```
.SPFUN #AREA,#0,#372,#0,,#1
```
or
```
.SPFUN #AREA,#0,#372,#0,,#CRTN
```
performs asynchronous write file gap operations.

H.2.4.3 Magtape and Cassette End-of-File Detection - Since magtape and cassette are sequential devices, the handlers for these devices cannot know in advance the number of blocks in a particular file, and thus cannot determine if a particular read request is attempting to read past the end of the file. User programs can use the following procedures to determine if the handlers have encountered end-of-file in either software or hardware mode.

In software mode, if end-of-file is encountered, magtape and cassette handlers will set the end-of-file bit (bit 13) in the channel status word. The next read attempted on that channel will return with the carry bit set and with the EMT error byte (absolute location 52) set to indicate an attempt to read past end-of-file. To avoid having magtape or cassette files appear one block longer than they really are, user software should check the end-of-file bit in the channel status word after a magtape or cassette read completes. If the bit is set, this indicates that the read just completed encountered end-of-file.

The following example demonstrates the procedure in software mode:
```
.MCALL ,V2,,,REGEF,,GTJB,,LOOKUP,,READW
.REGEF
...V2...
CSWEOF=$8000
ERRMD=52
CHNL=1

...GTJB #LIST,#JOBARG
GET JOB PARAMETERS

...LOOKUP #AREA,#CHNL,#FILNM
LOOKUP MAGTAPE OR CASSETTE FILE

...READW #AREA,#CHNL,#BUFF,#400,DLKNUM
READ A BLOCK
BCS EMTRR
CHECK FOR ERROR
MOV JOBARG+6,R1
GET POINTER TO I/O CHANNEL SPACE
(CHANNEL SPACE IS 5 WORDS PER CHANNEL
BEGINNING WITH CHANNEL 0),
BIT #CSWEOF,CHNL*10,(R1)
IS THE EOF BIT SET FOR THIS CHANNEL?
AND EOF
IF NE = YES - EOF ENCOUNTERED ON THIS READ
```

H-13 January 1976
F/B Programming and Device Handlers

EOF:  IEND OF FILE CODE

FILNAM1: *RAD50 'MT0FILNAMEXT' FOR *RAD50 'CT0FILNAMEXT'
AREA1: *BLKW 10
LIST1: *BLKW 2
JOBARG1: *BLKW 10

/*JOB PARAMETERS STORED HERE*/

In hardware mode, the cassette and magtape handlers do not report end-of-file, as they do in software mode. The best way that user programs may determine if a magtape read has encountered a tape mark or if a cassette read has encountered a file gap is to check the device status registers after each hardware mode read is complete.

Examples are:

For TM11/TMAll Magtape

MTS=172520  I/MT11 STATUS REGISTER
MTC=MTS+2  I/MT11 COMMAND REGISTER
MTSEOF=40000  I/EOF BIT IN MTS
MTSEOT=2000  I/EOT BIT IN MTS

.*READ #AREA,#CHNL,#BUFF,#400,BLKNUM  I/READ A BLOCK FROM MT
BCS EMTRR  I/CHECK ERRORS
TST #=MTC  I/ERROR BIT SET IN COMMAND REGISTER?
BPL NOERR  I/IF PL = NO
BIT #=MTSEOF,##MTS  I/YES = WAS IT EOF (TAPE MARK)?
BNE EOF  I/IF NE = YES = DO END OF FILE PROCESSING

EOF:  I/MT EOF ENCOUNTERED

For Cassette

TACS=172500  I/TAC1 CONTROL AND STATUS REGISTER
TAEOF=4000  I/EOF BIT IN TACS
TAEOT=2000  I/EOT BIT IN TACS

.*READ #AREA,#CHNL,#BUFF,#400,BLKNUM  I/READ FROM CT
BCS EMTRR  I/TEST ERRORS
TST #=TACS  I/ERROR BIT SET IN TACS?
BPL NOERR  I/IF PL = NO
BIT #=TAEOF,##TACS  I/YES = WAS IT END OF FILE?
BNE EOF  I/IF NE = YES

EOF:  I/CASSETTE END OF FILE ENCOUNTERED

January 1976       H-13.1
F/B Programming and Device Handlers

If desired, both the EOF and EOT bits could be checked:

\[ \text{BIT } \#\text{MTSEOF+MTSEOT,##MTS } \text{ IF EOF OR EOT?} \]

or

\[ \text{BIT } \#\text{TAEOF+TAEOT,##TACS } \text{ IF EOT OR EOT?} \]

H.2.5 DX (RX01 Diskette)

The .SPFUN request is used to allow reading and writing of absolute sectors on the RX01 diskette. An RX01 disk contains 77 tracks, each having 26 sectors, for a total of 2002 sectors. Each sector is 64 words long. RT-ll normally reads and writes groups of 4 sectors. Sectors may be accessed individually using the .SPFUN request as follows:

\[ .\text{SPFUN } \text{.area,.chan,.code,.buff,.wcnt,.blk,.crtn} \]

where:

- .code is the function code to be performed (377 for READ physical, 376 for WRITE physical, 375 for WRITE physical with deleted data mark)
- .buff is the location of a 65-word buffer; word 1 is the flag word and is normally set to 0 (if set to 1, a READ on physical sector containing deleted data mark is indicated); words 2-65 are the 64-word area to read (or write) the absolute sector
- .wcnt is the absolute track number, 0 through 76, to be read (or written)
- .blk is the absolute sector number, 1 through 26, to be read (or written)

The diskette should be opened with a nonfile-structured LOOKUP. Note also that the .buff, .wcnt, and .blk arguments have different meanings when used with diskette.

Sample Macro Call:

\[ .\text{SPFUN } \#\text{MLIST},\#1,\#377,\#\text{BUFF},\#0,\#7,\#0 \]

Perform a synchronous sector
Read from track 0, sector 7

RT-ll currently provides only one diskette handler as part of the system. The user may optionally install a second diskette handler by following the procedures outlined in Chapter 4 of the RT-ll System Generation Manual.
"TITLE PP V02-04 11/12/75
RT-11 HIGH SPEED PAPER TAPE (PC11) PUNCH HANDLER
DEC-11 ORTRAP-E
ABC/PG8/DV
MAY 1974
COPYRIGHT (C) 1974, 1975
DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
ON A SINGLE COMPUTER SYSTEM AND MAY NOT BE COPIED 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 SOFTWARE 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."
.CSECT PC11PP

R0=#10
R1=#11
R2=#12
R3=#13
R4=#14
R5=#15
SP=#16
PC=#17

; PAPER TAPE PUNCH CONTROL REGISTERS
PPS  = 177554 ; PUNCH CONTROL REGISTER
PPB  = 177556 ; PUNCH DATA BUFFER
PPVEC = 74   ; PUNCH VECTOR ADDR

; CONSTANTS FOR MONITOR COMMUNICATION
MERR  = 1    ; HARD ERROR BIT
MONLOW = 56  ; MONITOR BASE ADDR
OFFSET = 270 ; POINTER TO Q MANAGER COMPL ENTRY
PS    = 177776 ; PSW
PR7   = 340  ; PRIORITY 7
PR4   = 200  ; PRIORITY 4
1 000000 000074
2 000002 000034
3 000004 000030
4 000006 000000
5 000010 000000
6 000012 016704 177772
7 000016 004364 000006
8 000022 183026
9 000024 052737 000100 177554
10 000032 000207
11 000034 000423
12 000036 000457 000064
13 000042 000140
14 000044 016704 177748
15 000050 005737 177554
16 000054 100411
17 000056 062704 000006
18 000062 005714
19 000064 000100 000207
20 000070 115437 177556
21 000074 005214
22 000076 000207
23 000100 052734 000001
24 000104 005037 177554
25 000110 013704
26 000112 062704 177776
27 000116 013705 000054
28 000122 000175 000270
29 000126 000000
30 000130
31 000001
32
33
34
35
36
37
38
39
40
41
42
43
44
45

J LOAD POINT
LOADPT: .WORD PPVEC
1 ADDR OF INTERRUPT VECTOR
OFFSET TO INTERRUPT SERVICE
PRIORITE G
POINTER TO LAST Q ENTRY
POINTER TO CURRENT Q ENTRY
ENTRY POINT
PP: MOV PPOCQE,R4
R4 POINTS TO CURRENT Q ENTRY
WORD COUNT TO BYTE COUNT
READ REQUEST IS ILLEGAL
CAUSES INTERRUPT, STARTING TRANSFER
BR PPODONE
ABORT BY STOPPING

INTERRUPT SERVICE
PPINT: JSR R5,#GIPTR
WORD #CPPR4&PR7
R4 POINTS TO CURRENT Q ENTRY
ERROR?
YES=PUNCH OUT OF PAPER
POINT R4 TO BYTE COUNT
ANY MORE CHARs TO OUTPUT?
NO=TRANSFER DONE
DECOMATIC BYTE COUNT (IT IS NEGATIVE)
PUNCH CHARACTER
BUMP POINTER
PPERR: BIS #MDBR,#(R4)
SET HARD ERROR BIT
PPDONE: CLR #PPS
CLEAR INTERRUPT ENABLE
MOV PC,R4
ADD #PPCQE,#R4
MOV #MNDW,#R5
JMP #OFFSET(R5)
JUMP TO G MANAGER

PINPTR: .WORD G
POINTS TO COMMON ENTRY CODE

PPSIZE = -LOADPT
END
TITLE PR  V02-03

RT-11 HIGH SPEED PAPER TAPE READER (PC111) HANDLER

DEC-11-OPRM-A-D

ABC/ECR/GR/FF

MAY 1974

COPYRIGHT (C) 1974, 1975

DIGITAL EQUIPMENT CORPORATION

Maynard, Massachusetts 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE:
FOR ANY OTHER PURPOSE, MAY NOT BE DISTRIBUTED 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        ORA#0000
2        .CSECT PR1PR
3        
4        R2=0
5        R1=1
6        R2=2
7        R3=3
8        R4=4
9        R5=5
10       R6=6
11       PC=7
12
13
14        177550
15        177552
16        000070
17
18        PADD = 1
19        PRINT = 101
20        I CONSTANTS FOR MONITOR COMMUNICATION
21        HDRPR = 1
22        MONLOW = 54
23        OFFSET = 270
24
41 000132 001372  BNE  PRED1  J?ORE
42 000134 057744 020000 177772  R18  #20000, #67R4
43 44  
45 000142 0000142  I?PER? COMPLETE
46 000142 047377 004191 177550  PRDNE1: BIC  #PIN,T,#ERRS
47 000142 088744 004200 177550  PRDNE1: BIC  #PIN,T,#ERRS
48 000150 067744  MOV  PC, R4
49 000150 067744 177550  MOV  #PRC4E_.R4
50 000156 017975 004200 177550  MOV  #MONLOW,R4
51 000156 069176 004200 177550  JMP  #OFFSET(R5)
52 000162 009175 004200 177550  JTO MONITOR COMPLETION
53 000166 057744 004001  PREIRR: RIS  #ERRR,R-(#4)
54 000172 004743  PRDNE1: PR  PRDNE1: IAND COMPLETE OPERATION
55 000172 009175 004191  PRDNE1: PR  PRDNE1: IAND COMPLETE OPERATION
56 000172 009175 004191  PREIRR: RIS  #ERRR,R-(#4)
57 000172 009175 004191  PREIRR: RIS  #ERRR,R-(#4)

PR  V82-03  RT-11 MACRO VMP2-19  09144149 PAGE 3+

58 000001  
59 000001  'END

PR  V82-03  RT-11 MACRO VMP2-19  09144149 PAGE 3+

SYMBOL TABLE

HDERR = 000001  MONLOW= 000004  OFFSET= 000270  PC = #000007  PINT = 000101
PR = 000120  02  PRED1 = 000142  02  PRA = 177552  PRC4E = 000100  02  PRDNE = 000142  02
PREDF = 000120  02  PR = 177552  02  PRERR = 000166  02  PRED = 000001  02  PRINT = 000446  02
PAME = 000006  02  PRERR = 000166  02  PRERR = 000166  02  PRSTRT = 000000  02  PRVEC = 000070
R0 = #000000  02  R1 = #000001  02  R2 = #000002  02  R3 = #000003  02  R4 = #000004
R5 = #000005  02  SEFK = 000142  02  SP = #000076
. ABS. = 000000  000  .ABS. = 000001  01
PC11PR  000176  02  ERRORS DETECTED: 0  0  
. REPE  000018  000  . REPE  000100  000  
+LPI=PR/CNITTHIEND
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDERR</td>
<td>3=57</td>
<td>3=54</td>
</tr>
<tr>
<td>MONLOW</td>
<td>2=90</td>
<td>3=51</td>
</tr>
<tr>
<td>OFFSFT</td>
<td>2=32</td>
<td>2=52</td>
</tr>
<tr>
<td>PINT</td>
<td>2=17</td>
<td>3=47</td>
</tr>
<tr>
<td>PR</td>
<td>3=11</td>
<td>3=47</td>
</tr>
<tr>
<td>PRABRT</td>
<td>3=20</td>
<td>3=46</td>
</tr>
<tr>
<td>PAB</td>
<td>2=15</td>
<td>3=31</td>
</tr>
<tr>
<td>PAGG</td>
<td>3=7</td>
<td>3=27</td>
</tr>
<tr>
<td>PRDONE</td>
<td>3=34</td>
<td>3=55</td>
</tr>
<tr>
<td>PREDI</td>
<td>3=38</td>
<td>3=47</td>
</tr>
<tr>
<td>PREDF</td>
<td>3=30</td>
<td>3=49</td>
</tr>
<tr>
<td>PREME</td>
<td>3=13</td>
<td>3=40</td>
</tr>
<tr>
<td>PRINT</td>
<td>3=4</td>
<td>3=29</td>
</tr>
<tr>
<td>PRLOE</td>
<td>3=60</td>
<td></td>
</tr>
<tr>
<td>PRS</td>
<td>2=15</td>
<td>3=15</td>
</tr>
<tr>
<td>PRSIZE</td>
<td>3=57</td>
<td></td>
</tr>
<tr>
<td>PRSTAT</td>
<td>3=3</td>
<td>3=35</td>
</tr>
<tr>
<td>PRVCF</td>
<td>2=16</td>
<td>3=49</td>
</tr>
<tr>
<td>SEEK</td>
<td>3=14</td>
<td></td>
</tr>
</tbody>
</table>
F/B Programming and Device Handlers

H.4 DEC 026/DEC 029 CARD CODE CONVERSION TABLE

Card codes in the following table are broken down by zone punch. Except where noted, all codes apply to both DEC 026 and DEC 029 punched cards.

Table H-1
Card Code Conversions

<table>
<thead>
<tr>
<th>Zone</th>
<th>Digit</th>
<th>Octal</th>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>none</td>
<td>040</td>
<td>1</td>
<td>SPACE</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>061</td>
<td>2</td>
<td>digit 1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>062</td>
<td>3</td>
<td>digit 2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>063</td>
<td>4</td>
<td>digit 3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>064</td>
<td>5</td>
<td>digit 4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>065</td>
<td>6</td>
<td>digit 5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>066</td>
<td>7</td>
<td>digit 6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>067</td>
<td></td>
<td>digit 7</td>
</tr>
<tr>
<td>12</td>
<td>none</td>
<td>046</td>
<td>&amp;</td>
<td>ampersand</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>053</td>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>101</td>
<td>A</td>
<td>upper case A</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>102</td>
<td>B</td>
<td>upper case B</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>103</td>
<td>C</td>
<td>upper case C</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>104</td>
<td>D</td>
<td>upper case D</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>105</td>
<td>E</td>
<td>upper case E</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>106</td>
<td>F</td>
<td>upper case F</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>107</td>
<td>G</td>
<td>upper case G</td>
</tr>
<tr>
<td>11</td>
<td>none</td>
<td>055</td>
<td>-</td>
<td>minus sign</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>122</td>
<td>J</td>
<td>upper case J</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>123</td>
<td>K</td>
<td>upper case K</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>124</td>
<td>L</td>
<td>upper case L</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>125</td>
<td>M</td>
<td>upper case M</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>126</td>
<td>N</td>
<td>upper case N</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>127</td>
<td>O</td>
<td>upper case O</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>130</td>
<td>P</td>
<td>upper case P</td>
</tr>
<tr>
<td>0</td>
<td>none</td>
<td>060</td>
<td>0</td>
<td>digit 0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>057</td>
<td>/</td>
<td>slash</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>123</td>
<td>S</td>
<td>upper case S</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>124</td>
<td>T</td>
<td>upper case T</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>125</td>
<td>U</td>
<td>upper case U</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>126</td>
<td>V</td>
<td>upper case V</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>127</td>
<td>W</td>
<td>upper case W</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>130</td>
<td>X</td>
<td>upper case X</td>
</tr>
<tr>
<td>8</td>
<td>none</td>
<td>70</td>
<td>8</td>
<td>digit 8</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>140</td>
<td>`</td>
<td>accent grave</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>072</td>
<td>:</td>
<td>colon</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>137</td>
<td>+</td>
<td>backarrow</td>
</tr>
<tr>
<td></td>
<td>(DEC 029)</td>
<td>3</td>
<td>#</td>
<td>(underscore)</td>
</tr>
</tbody>
</table>

(continued on next page)
### F/B Programming and Device Handlers

**Table H-1 (Cont.)**
Card Code Conversions

<table>
<thead>
<tr>
<th>Zone</th>
<th>Digit</th>
<th>Octal</th>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(DEC 026)</td>
<td>4</td>
<td>075</td>
<td>=</td>
<td>equal sign</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>136</td>
<td>@</td>
<td>commercial &quot;at&quot;</td>
</tr>
<tr>
<td>(DEC 026)</td>
<td>6</td>
<td>075</td>
<td>`</td>
<td>single quote</td>
</tr>
<tr>
<td>(DEC 026)</td>
<td>7</td>
<td>042</td>
<td>†</td>
<td>uparrow</td>
</tr>
<tr>
<td>(DEC 026)</td>
<td>8</td>
<td>0134</td>
<td></td>
<td>(circumflex)</td>
</tr>
<tr>
<td>(DEC 029)</td>
<td>9</td>
<td>071</td>
<td>9</td>
<td>digit 9</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>026</td>
<td>ctrl - V</td>
<td>SYN</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>004</td>
<td>ctrl - D</td>
<td>EOT</td>
</tr>
<tr>
<td>12-11</td>
<td>none</td>
<td>174</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>152</td>
<td>/</td>
<td>lower-case J</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>153</td>
<td>j</td>
<td>lower-case K</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>154</td>
<td>k</td>
<td>lower-case L</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>155</td>
<td>l</td>
<td>lower-case M</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>156</td>
<td>m</td>
<td>lower-case N</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>157</td>
<td>n</td>
<td>lower-case O</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>160</td>
<td>p</td>
<td>lower-case P</td>
</tr>
<tr>
<td>12-0</td>
<td>none</td>
<td>173</td>
<td>{</td>
<td>open brace</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>141</td>
<td>a</td>
<td>lower-case A</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>142</td>
<td>b</td>
<td>lower-case B</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>143</td>
<td>c</td>
<td>lower-case C</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>144</td>
<td>d</td>
<td>lower-case D</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>145</td>
<td>e</td>
<td>lower-case E</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>146</td>
<td>f</td>
<td>lower-case F</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>147</td>
<td>g</td>
<td>lower-case G</td>
</tr>
<tr>
<td>12-8</td>
<td>none</td>
<td>110</td>
<td>[</td>
<td>upper-case H</td>
</tr>
<tr>
<td>(DEC 029)</td>
<td>2</td>
<td>133</td>
<td>![ ]</td>
<td>open square bracket</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>077</td>
<td>?</td>
<td>question mark</td>
</tr>
<tr>
<td>(DEC 029)</td>
<td>6</td>
<td>074</td>
<td>.</td>
<td>period</td>
</tr>
<tr>
<td>(DEC 026)</td>
<td>4</td>
<td>056</td>
<td>&lt;</td>
<td>open angle bracket</td>
</tr>
<tr>
<td>(DEC 026)</td>
<td>5</td>
<td>051</td>
<td>)</td>
<td>close parenthesis</td>
</tr>
<tr>
<td>(DEC 029)</td>
<td>6</td>
<td>050</td>
<td>(</td>
<td>open parenthesis</td>
</tr>
<tr>
<td>(DEC 026)</td>
<td>6</td>
<td>135</td>
<td>]</td>
<td>close square bracket</td>
</tr>
<tr>
<td>(DEC 029)</td>
<td>7</td>
<td>053</td>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>(DEC 026)</td>
<td>7</td>
<td>074</td>
<td>&lt;</td>
<td>open angle bracket</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>041</td>
<td>!</td>
<td>exclamation mark</td>
</tr>
<tr>
<td>12-9</td>
<td>none</td>
<td>111</td>
<td>I</td>
<td>upper-case I</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>001</td>
<td>ctrl - A</td>
<td>SOH</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>002</td>
<td>ctrl - B</td>
<td>STX</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>003</td>
<td>ctrl - C</td>
<td>ETX</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>011</td>
<td>ctrl - I</td>
<td>HT</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>177</td>
<td></td>
<td>DEL</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Zone</th>
<th>Digit</th>
<th>Octal</th>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11-0</td>
<td>none</td>
<td>175</td>
<td>]</td>
<td>close brace</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>176</td>
<td>~</td>
<td>tilda</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>163</td>
<td>s</td>
<td>lower-case S</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>164</td>
<td>t</td>
<td>lower-case T</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>165</td>
<td>u</td>
<td>lower-case U</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>166</td>
<td>v</td>
<td>lower-case V</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>167</td>
<td>w</td>
<td>lower-case W</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>170</td>
<td>x</td>
<td>lower-case X</td>
</tr>
<tr>
<td>11-8</td>
<td>none</td>
<td>121</td>
<td>Q</td>
<td>upper-case Q</td>
</tr>
<tr>
<td></td>
<td>2 (DEC 029)</td>
<td>135</td>
<td>]</td>
<td>close square bracket</td>
</tr>
<tr>
<td></td>
<td>2 (DEC 026)</td>
<td>072</td>
<td>colon</td>
<td>colon</td>
</tr>
<tr>
<td></td>
<td>3 (DEC 029)</td>
<td>044</td>
<td>$</td>
<td>currency symbol</td>
</tr>
<tr>
<td></td>
<td>3 (DEC 026)</td>
<td>052</td>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td></td>
<td>4 (DEC 029)</td>
<td>051</td>
<td>)</td>
<td>close parenthesis</td>
</tr>
<tr>
<td></td>
<td>4 (DEC 026)</td>
<td>133</td>
<td>[</td>
<td>open square bracket</td>
</tr>
<tr>
<td></td>
<td>5 (DEC 029)</td>
<td>073</td>
<td>;</td>
<td>semi-colon</td>
</tr>
<tr>
<td></td>
<td>5 (DEC 026)</td>
<td>076</td>
<td>&gt;</td>
<td>close angle bracket</td>
</tr>
<tr>
<td></td>
<td>6 (DEC 029)</td>
<td>136</td>
<td>†</td>
<td>uparrow</td>
</tr>
<tr>
<td></td>
<td>6 (DEC 026)</td>
<td>046</td>
<td>(c)</td>
<td>(circumflex)</td>
</tr>
<tr>
<td></td>
<td>7 (DEC 026)</td>
<td>046</td>
<td>&amp;</td>
<td>ampersand</td>
</tr>
<tr>
<td>11-9</td>
<td>none</td>
<td>122</td>
<td>R</td>
<td>upper-case R</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>021</td>
<td>ctrl - Q</td>
<td>DC1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>022</td>
<td>ctrl - R</td>
<td>DC2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>023</td>
<td>ctrl - S</td>
<td>DC3</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>010</td>
<td>ctrl - H</td>
<td>BS</td>
</tr>
<tr>
<td>0-8</td>
<td>null</td>
<td>131</td>
<td>\</td>
<td>upper-case Y</td>
</tr>
<tr>
<td></td>
<td>2 (DEC 029)</td>
<td>134</td>
<td>/</td>
<td>backslash</td>
</tr>
<tr>
<td></td>
<td>2 (DEC 026)</td>
<td>073</td>
<td>;</td>
<td>semi-colon</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>054</td>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td></td>
<td>4 (DEC 029)</td>
<td>045</td>
<td>%</td>
<td>percent sign</td>
</tr>
<tr>
<td></td>
<td>4 (DEC 026)</td>
<td>050</td>
<td>(</td>
<td>open parenthesis</td>
</tr>
<tr>
<td></td>
<td>4 (DEC 029)</td>
<td>137</td>
<td>†</td>
<td>backarrow</td>
</tr>
<tr>
<td></td>
<td>4 (DEC 026)</td>
<td>042</td>
<td>(underscore)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 (DEC 026)</td>
<td>076</td>
<td>&gt;</td>
<td>double quote</td>
</tr>
<tr>
<td></td>
<td>5 (DEC 029)</td>
<td>043</td>
<td>#</td>
<td>close angle bracket</td>
</tr>
<tr>
<td></td>
<td>5 (DEC 026)</td>
<td>077</td>
<td>?</td>
<td>number sign</td>
</tr>
<tr>
<td></td>
<td>6 (DEC 029)</td>
<td>045</td>
<td>%</td>
<td>question mark</td>
</tr>
<tr>
<td></td>
<td>6 (DEC 026)</td>
<td>045</td>
<td>percent sign</td>
<td></td>
</tr>
<tr>
<td>0-9</td>
<td>null</td>
<td>132</td>
<td>Z</td>
<td>upper-case Z</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>012</td>
<td>ctrl - J</td>
<td>LF</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>027</td>
<td>ctrl - W</td>
<td>ETB</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>033</td>
<td>ESC</td>
<td></td>
</tr>
<tr>
<td>9-8</td>
<td>4</td>
<td>024</td>
<td>ctrl - T</td>
<td>DC4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>025</td>
<td>ctrl - U</td>
<td>NAK</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>032</td>
<td>ctrl - Z</td>
<td>SUB</td>
</tr>
</tbody>
</table>

(continued on next page)
# Table H-1 (Cont.)
## Card Code Conversions

<table>
<thead>
<tr>
<th>Zone</th>
<th>Digit</th>
<th>Octal</th>
<th>Character</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-9-8</td>
<td>3</td>
<td>013</td>
<td>ctrl - K</td>
<td>VT</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>014</td>
<td>ctrl - L</td>
<td>FF</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>015</td>
<td>ctrl - M</td>
<td>CR</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>016</td>
<td>ctrl - N</td>
<td>SO</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>017</td>
<td>ctrl - O</td>
<td>SI</td>
</tr>
<tr>
<td>11-9-8</td>
<td>None</td>
<td>030</td>
<td>ctrl - X</td>
<td>CAN</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>031</td>
<td>ctrl - Y</td>
<td>EM</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>034</td>
<td>ctrl - \</td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>035</td>
<td>ctrl - ]</td>
<td>GS</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>036</td>
<td>ctrl - {</td>
<td>RS</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>037</td>
<td>ctrl - _</td>
<td>US</td>
</tr>
<tr>
<td>0-9-8</td>
<td>5</td>
<td>005</td>
<td>ctrl - E</td>
<td>ENQ</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>006</td>
<td>ctrl - F</td>
<td>ACK</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>007</td>
<td>ctrl - G</td>
<td>BEL</td>
</tr>
<tr>
<td>12-0-8</td>
<td>None</td>
<td>150</td>
<td>h</td>
<td>lower-case H</td>
</tr>
<tr>
<td>12-0-9</td>
<td>None</td>
<td>151</td>
<td>i</td>
<td>lower-case I</td>
</tr>
<tr>
<td>12-11-8</td>
<td>None</td>
<td>161</td>
<td>q</td>
<td>lower-case Q</td>
</tr>
<tr>
<td>12-11-9</td>
<td>None</td>
<td>162</td>
<td>r</td>
<td>lower-case R</td>
</tr>
<tr>
<td>11-0-8</td>
<td>None</td>
<td>171</td>
<td>y</td>
<td>lower-case Y</td>
</tr>
<tr>
<td>11-0-9</td>
<td>None</td>
<td>172</td>
<td>z</td>
<td>lower-case Z</td>
</tr>
<tr>
<td>12-11-9-8</td>
<td>1</td>
<td>020</td>
<td>ctrl - P</td>
<td>DLE</td>
</tr>
<tr>
<td>12-0-9-8</td>
<td>1</td>
<td>000</td>
<td>NUL</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX I

DUMP

RT-ll DUMP is a program which outputs to the console or lineprinter all or any part of a file in octal words, octal bytes, ASCII characters and/or RAD50 characters. DUMP is particularly useful for examining data such as directories or files.

I.1 CALLING AND USING DUMP

DUMP is called using the monitor command:

R DUMP

in response to the dot printed by the Keyboard Monitor. The name of the file which is to be output is entered as follows in response to the asterisk printed by the Command String Interpreter:

dev:output=dev:input/s

where:

dev: represents any valid device specification (line printer is default for output if no output file is designated. The default can be changed; refer to Chapter 4 of the RT-ll System Generation Manual.

output represents the filename and extension assigned to the output file. The default extension for file-structured output is .DMP.

input represents the input source filename and extension.

NOTE

Input files residing on magtape and cassette must have been written under the RT-ll system; magtape and cassette files written under other system environments cannot be dumped.

/s represents one or more of the switches listed in Table I-1.

Type CTRL C to halt DUMP at any time and return control to the monitor. To restart DUMP type R DUMP or the REENTER command in response to the monitor's dot.

I-1

January 1976
I.1.1 DUMP Switches

The following switches can appear in the command string for DUMP:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B</td>
<td>Output octal bytes</td>
</tr>
<tr>
<td>/E:n</td>
<td>End output at block number n</td>
</tr>
<tr>
<td>/G</td>
<td>Ignore input errors</td>
</tr>
<tr>
<td>/N</td>
<td>Suppress ASCII output</td>
</tr>
<tr>
<td>/O:n</td>
<td>Output only block number n (same as /E:n, /S:n)</td>
</tr>
<tr>
<td>/S:n</td>
<td>Start output with block number n</td>
</tr>
<tr>
<td>/W</td>
<td>Output octal words</td>
</tr>
<tr>
<td>/X</td>
<td>Output RAD50 characters</td>
</tr>
</tbody>
</table>

If neither /W nor /B is given, /W is assumed. ASCII characters are always dumped unless /N is given. The number n is an octal block number.

If an input filename is given, block numbers are relative to the beginning of the file to which the block belongs. If not, block numbers are absolute block numbers on the device (i.e., the physical block numbers on the corresponding device).

I.1.2 Examples

The following are two examples of DUMP. /B is used in the first example to output octal bytes of the file SQRT.FTN into a file called DIF.DMP on device DT1.

```
R DUMP
*DT1:DIF=SQRT.FTN/B
```

If DIF.DMP is then listed on the line printer (using PIP), it appears as follows:

```
BLOCK NUMBER 0000
000/ 001 000 006 000 001 003 011 261 214 072 150 000 000 000 001 257
     226 817 110 001 000 006 262 252 304 037 110 004 210 000 374 252
     110 001 004 006 003 254 217 163 110 004 000 000 012 001
     226 204 003 001 004 253 100 070 100 004 000 000 123 263 204
     253 150 001 000 000 000 000 000 003 000 032 001 000
```
The second example illustrates the use of /X to output RAD50 and octal values for locations in the file. The numbers in the left column represent the byte displacement. ASCII characters are printed in the far right hand column (dot represents a non-printing character):

```
R DUMP
*RK1:LINKD.FE/X

BLOCK NUMBER 0008

000/ 027111 044524 040124 002085 052122 044514 045515 045140 **TITLE,RTLINK R*
   GNY K/L LMT ASM MSZ K/D LAB MEX

029/ 047517 029124 045703 024564 029040 054840 031480 034655 **DOT CODE X03=1*
   LSW EF6 LSX KCL EEX ND MGS G4/

049/ 028460 034142 051409 026524 030461 046840 047111 024513 **RT=11 LINKE*
   DGY IQ4 MEX GJD U43 LGH LUA KCS

069/ 035252 035412 042048 014150 035455 026461 051117 046114 **R_1 DEC=11-URLR*
   BEJ IQ4 J6 JUU G4/ GII MF1 LML

109/ 026561 026562 04514 005815 020073 004515 020131 032461 **A=9-LA..5 MAY 15*
   GIY GID JG6 AKM EFK JG7 EGA MTU

129/ 028054 02404 024061 024061 024061 024061 024061 024061 **1974...EP/EN*
   LSA LPA MFA AXM EFK LSY EEE LT7

149/ 040130 041516 042105 01404 022313 043512 050510 051605 **HANDED BY JG...**
   JU2 J08 J67 JWK EGA KJP AXM AXM

159/ 026473 035412 041448 051117 051117 043511 052110 024040 **THE COPYRIGHT (**
   B0S IQ4 J/X L3D MGA KIP MRP FP2

209/ 028460 024840 034461 032067 034512 035445 034445 035445 **C(1974)...**
   LSA LPA MFA AXM EFK LSY EEE LT7

229/ 042440 043511 052111 046110 042440 052351 058111 042455 **DIGITAL EQUIPEME*
   JG KPI MSG LHA KJP MIA L33 KCU

249/ 025119 044440 051117 047520 040522 044442 027119 040515 **NT CORPORATION...**
   L90 J/A MF1 L5X JRR K/L LUG JXM

269/ 028073 042159 047313 051119 024104 044440 024515 045150 **MAYNARD, MASSA...**
   EFK JD7 LUML MGR CCL CML LM3 JNC

339/ 044109 051525 052105 051524 039440 033661 038265 010515 **MAYNARD, MASSA...**
   KV5 MM8 KSM JLX G4/ MSH NHE AXM

359/ 040473 035412 052240 042510 044448 043116 051117 040515 **THE INFORMATION...**
   B0S IQ4 MMR KCP K KIB MF1 JG7

349/ 044224 047117 044448 022116 044124 051515 042420 044115 **TION IN THIS DOC...**
   JU2 J08 J67 JWK EGA KJP MIA L33 KCU

359/ 045025 047185 022114 051515 041440 041125 042512 052115 **I.MAY BE SUBJECT...**
   JU2 J08 J67 JWK EGA KJP MIA L33 KCU

409/ 052549 043512 044440 014448 045410 043516 028125 044557 **TO CHANGE W**
   L90 J/A MF1 L5X JRR K/L LUG JXM

459/ 044124 045217 022124 047816 044524 042503 045440 042115 **THERE USED TO BE A...**
   KV5 MY5 KFP LEX K/L LCS JU2 JPP

469/ 041449 049017 044515 046243 047195 022124 047516 042592 041440 **SHOULD NOT BE C...**
   L90 J/A MF1 L5X JRR K/L LUG JXM

469/ 047117 051235 022525 042125 05015 02973 050525 045852 042048 **COMMITMENT BY DE...**
   L90 J/A MF1 L5X JRR K/L LUG JXM

509/ 041449 049017 044515 046243 047195 022124 047516 042592 041440 **COMMITMENT BY DE...**
   L90 J/A MF1 L5X JRR K/L LUG JXM

529/ 043011 049011 022124 05015 043025 046500 047185 022124 **EQUIPMENT**
   KPI JG3 EF6 MN8 LN8 LT7 EF6

549/ 047538 055122 051117 052105 027116 040515 028673 **CORPORATION...**
   L90 LMB MF1 L5X LGP GPN AXM EFK

569/ 042948 022133 031591 025253 042515 028125 047516 051488 **DE A SUES ASSUMES NO R...**
   KCL EFA ML3 MZC LGU LSY M3X

639/ 051059 047520 031596 044511 045111 054511 021131 047506 **RESPONSIBILITY OF**
   L90 LX5 MJF LMI M5Q EGA LSN

659/ 026112 047185 020131 051103 047922 051522 045200 048059 **ANY ERRORS THAT...**
   EF4 LT3 EFA LMB L5Q MNJ MRP JG2

649/ 026524 035412 045448 045481 048448 050128 045855 020122 **T...I MAY AAPP...**
   BEL IDA LMD N51 JP2 L4 JG5 EF4
```

I-4
I.2 DUMP ERROR MESSAGES

The following errors may occur when using DUMP:

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?IN ERR?</td>
<td>A hardware error occurred while reading the input file and /G was not specified in the command line.</td>
</tr>
<tr>
<td>?OUT ERR?</td>
<td>A hardware error occurred while writing an output file, or the output device was full.</td>
</tr>
<tr>
<td>?LP NOT FND?</td>
<td>A line printer handler is not available on the system.</td>
</tr>
</tbody>
</table>
APPENDIX J

FILEX

J.1 FILEX OVERVIEW

FILEX is a general file transfer program used to convert files among devices for various operating systems. Transfers may be initiated between any block-replaceable RT-11 directory-structured device and PDP-11 DOS/BATCH DECtape (as input or output), DOS/BATCH disk (as input only), RSTS DECtape (as input or output), and DECsystem-10 DECtape (as input only).

"Wild-card" names (the *.* construction explained in Chapter 4) are permitted. Magtape and cassette are not supported in any operation.

J.1.1 File Formats

FILEX can transfer files created by three different operating systems—RT-11, DECsystem-10, and DOS/BATCH (PDP-11 Disk Operating System/BATCH). Data formats that may be used in a transfer are ASCII, image, and packed image. Because the file structure and data formats for each system vary somewhat, switches are needed in the command line to indicate the file structures and the data formats involved in the transfer. These switches are defined in Section J.2.1. Devices are assumed to be in RT-11 file structure unless either a /S or /T switch (for DOS/BATCH and RSTS or DECsystem-10 respectively) is indicated. If both input and output devices are RT-11 format (or are not file-structured), FILEX will operate like FIP.

J.2 CALLING AND USING FILEX

FILEX is called from the RT-11 system device by typing:

    R FILEX

in response to the dot printed by the Keyboard Monitor. An asterisk is printed when FILEX is loaded and ready to accept command string input.
Type CTRL C to halt FILEX at any time and return control to the monitor. To restart FILEX, type R FILEX or the REENTER command in response to the monitor's dot.

J.2.1 FILEX Switch Options

Table J-1 lists the switch options which are used for various FILEX operations. /S and /T must appear following the device and filename to which they apply; other switches may appear anywhere in the command line. These switches are explained in more detail following the table.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/A</td>
<td>Indicates a character-by-character ASCII transfer in which rubouts and nulls are deleted; when /T is also used, each PDP-10 word is assumed to contain five 7-bit ASCII bytes. (The transfer terminates upon reaching a $Z$ for compatibility with RSTS; the $Z$ is not transferred.)</td>
</tr>
<tr>
<td>/D</td>
<td>Deletes the named file from the device; valid only for DOS/BATCH and RSTS DECTape.</td>
</tr>
<tr>
<td>/F</td>
<td>Causes a &quot;fast&quot; listing of the device directory by listing filenames only.</td>
</tr>
<tr>
<td>/I</td>
<td>Performs an image mode transfer; if the input is either DOS/BATCH, RSTS or RT-11, this is a word-for-word transfer; if the input is from DECSYSTEM-10, /I indicates that the file resembles a file created on DECSYSTEM-10 by MACY11, MACX11, or LNKX11 with the /I switch; in this case each DECSYSTEM-10 36-bit word contains one PDP-11 8-bit byte in its low-order bits.</td>
</tr>
<tr>
<td>/L</td>
<td>Causes a complete listing of the device directory, including filenames, block lengths, and creation dates, to appear on the console terminal.</td>
</tr>
<tr>
<td>/P</td>
<td>Performs a packed image transfer; if the input is either DOS/BATCH, RSTS or RT-11, this is a word-for-word transfer; if the input is from DECSYSTEM-10, /P indicates that the file resembles a file created on DECSYSTEM-10 by MACY11, MACX11, or LNKX11 with the /P switch, in which case each DECSYSTEM-10 36-bit word contains four PDP-11 8-bit bytes aligned on bits 0, 8, 16, and 24. This mode is assumed if no mode switch (/A, /I) is indicated in a command line.</td>
</tr>
</tbody>
</table>
| /S     | Indicates the device is a legal DOS/BATCH (or RSTS) block-replaceable device; the switch must appear on the
Table J-1 (Cont)
FILEX Switch Options

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>same side of the command line as the device to which it applies.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>Neglecting to include the /S switch in a command line involving DOS/BATCH (or RSTS) causes unpredictable results.</td>
</tr>
<tr>
<td>/T</td>
<td>Indicates the device is a legal DECsystem-10 block-replaceable device; the switch must appear on the same side of the command line as the device to which it applies.</td>
</tr>
<tr>
<td>/V</td>
<td>Types out version number of FILEX.</td>
</tr>
<tr>
<td>/Z</td>
<td>Zeroes the directory of the specified device in the proper format (valid only for DOS/BATCH and RSTS DECTape).</td>
</tr>
</tbody>
</table>

J.2.2 Transferring Files Between RT-11 and DOS/BATCH (or RSTS)

File transfers may be initiated between block-replaceable devices used by RT-11 and the PDP-11 DOS/BATCH system. Input from DOS/BATCH may be either disk or DECTape; both linked and contiguous files are supported as input. If the input device is a DOS/BATCH disk, the user should specify a DOS/BATCH user identification code (called a UIC; refer to the DOS/BATCH Handbook, DEC-11-ODBHA-A-D). This UIC then becomes default for all future transfers. If no UIC is specified, the current default UIC is used (see the description of UIC, following). Output to DOS/BATCH is limited to DECTape only. DECTape used under the RSTS system is also legal as both input and output, since its format is identical to DOS/BATCH DECTape; see the PDP-11 Resource Sharing Time-Sharing System User's Guide, DEC-11-ORSUA-D-D. Any valid RT-11 file storage device may be used for either input or output in the transfer. The RT-11 device DK: is assumed if no device is indicated.

**NOTE**

An RT-11 DECTape can hold more information than a DOS/BATCH or RSTS DECTape. Thus, caution should be observed in copying files from a full RT-11 tape to a DOS DECTape as some information may not transfer. In such a case an error message will be printed and the transfer will not be completed.

When a transfer from an RT-11 device to a DOS DECTape occurs, the block size of the file may increase. However, if the file is later transferred back to an RT-11 device, the block size does not decrease.

To transfer a file from a legal DOS/BATCH block-replaceable device (or RSTS DECTape) to a legal RT-11 device, the command string format is:

J-3 January 1976
FILEX

*dev:filnam.ext=dev:filnam.ext/S/s[UIC]

where:

dev:    = on the output side, any valid RT-ll device (if that device is not file-structured, filnam.ext may be omitted); on the input side, DOS/BATCH DECTape or disk, or RSTS DECTape.

filnam.ext = for output, any valid RT-ll filename and extension; for input, any valid DOS/BATCH (RSTS) filename and extension.

/S = the switch from Table J-1 which designates a DOS/BATCH (RSTS) block-replaceable device. (This switch MUST be included in the command line.)

/s = optionally any other switch (one only) from Table J-1 (in this case, /A is the only meaningful switch which could be chosen); /A indicates an ASCII transfer is to be performed in which nulls and rubouts are deleted; if /A is not used, the transfer will be performed word-for-word.

[UIC] = the DOS/BATCH user identification code in the format [nnn,nnn] where nnn represents 1 to 3 octal digits less than or equal to 377 specifying first the user-group number, and then the individual user number; this code may be placed anywhere on the side of the command line containing the DOS/BATCH device; whenever a UIC is entered, it becomes the default for any further DOS/BATCH transfers; the initial default value is [1,1].

NOTE

A UIC need not be indicated in any command line if accessing only DECTape since individual users do not "own" files on DECTape under DOS; no error will occur if a code is used, however.

To transfer files from an RT-ll storage device to a DOS/BATCH or RSTS DECTape, the command line format is as follows:

*DTn:filnam.ext/S/s=dev:filnam.ext

where:

DTn:    = a valid DOS/BATCH (RSTS) DECTape assignment (only DECTape is legal for output).

filnam.ext = for output, any valid DOS/BATCH (RSTS) filename and extension; for input, any valid RT-ll filename and extension.

/S = the switch from Table J-1 which designates a DOS/BATCH (RSTS) block-replaceable device. (This switch MUST be included in the command line.)
FILEX

/s = optionally any other switch (one only) from Table J-1 (in this case, /A is the only meaningful switch which could be chosen); /A indicates an ASCII transfer is to be performed in which nulls and rubouts are deleted; if /A is not used, the transfer will be performed word-for-word.

dev: = any valid RT-11 device.

Examples:

*DT2:SORT.ABC/S=SORT.ABC

This command instructs FILEX to transfer a file called SORT.ABC from the RT-11 system device to a DOS/BATCH (or RSTS) formatted DECTape on unit DT2.

*PP:=DT2:TYPE:FILE/S/A

This command allows a file to be transferred from DOS/BATCH (or RSTS) DECTape to the papertape punch under RT-11. The transfer is done in ASCII mode.

*DK:*=RK1:MCR1.MAC/S[200,200]

This command causes the file MCR1.MAC from the DOS/BATCH disk on unit 1 which is stored under the UIC [200,200] to be transferred to the RT-11 device DK. [200,200] becomes the default UIC for any further DOS/BATCH operations.

J.2.3 Transferring Files to RT-11 from DECSYSTEM-10

Files may be transferred to RT-11 devices from a DECSYSTEM-10 DECTape whenever a foreground job is not running (this restriction is due to the fact that when reading DECSYSTEM-10 files, FILEX accesses the DECTape control registers directly rather than using the RT-11 DECTape control handler). Output may be to any valid RT-11 device; DECSYSTEM-10 DECTape is the only valid input device. The format of the command line is:

*dev:filnam.ext=DTn:filnam.ext/T/s

where:

dev: = any valid RT-11 device; if that device is not file-structured, the filnam.ext may be omitted.

filnam.ext = for output, any valid RT-11 filename and extension; for input, any valid DECSYSTEM-10 filename and extension (see NOTE below).

DTn: = a valid DECSYSTEM-10 DECTape assignment (only DECTape is legal for input).

/T = the switch from Table J-1 which signifies a DECSYSTEM-10 DECTape. When using /T, especially with /A, the time of day clock will lose time. It should be examined with the monitor TIME command, and reset if necessary.

J-5

January 1976
FILEX

/s = any other switch from Table J-1 which specifies the mode of transfer. Only one additional switch may be indicated per transfer; /P is assumed if no other mode is specified.

NOTE

RT-11 files may be indirectly converted to DECsystem-10 format by running RT-11 FILEX and converting the files to DOS formatted DECTape, and then running DECsystem-10 FILEX to read the DOS DECTape; however, it is currently not possible under RT-11 to convert files directly from RT-11 to DECsystem-10 format.

Examples:

*DT2:STAND.LIS=DT1:STAND.LIS/T/R

The ASCII file STAND.LIS is converted from DECsystem-10 ASCII format to RT-11 ASCII format and stored under RT-11 on DECTape 2 as STAND.LIS.

NOTE

Transfers from DECsystem-10 DECTape to RT-11 DECTape may cause an <UNUSED> block to appear after the file on the RT-11 device. This is a result of the method by which RT-11 handles the increased amount of information on a DECsystem-10 DECTape.

*SV:*NEW=DT0:*LIS/T

This command indicates that all files on DECsystem-10 DECTape 0 with the extension .LIS are to be transferred to the RT-11 system device (disk or DECTape) using the same filename and an extension of .NEW. The /P switch is the assumed transfer mode.

J.2.4 Listing Directories

Device directories of any of the block-replaceable devices used in a FILEX transfer can be listed on the console terminal. The following FILEX command lines are used:

*dev:/L for RT-11
*dev:/L/S for DOS/BATCH (RSTS-11)
*DTn:/L/T for DECsystem-10
FILEX
where:

dev: = disk or DECTape (DK is assumed if no device is specified).

DTn: = DECSystem-10 DECTape.

/L = the switch from Table J-1 which indicates a listing is desired (/F may be substituted if a "fast listing" is preferred).

/S = the switch from Table J-1 which designates a DOS/BATCH or RSTS-11 block-replaceable device.

/T = the switch from Table J-1 which designates a DECSystem-10 block-replaceable device.

Examples:

*RK1:/L/S
BADD .SYS 1 22-JUL-74
MDNLIB.CIL 175C 22-JUL-74
DU11 .PAL 45 24-JUL-74
VERIFY .LDR 67C 22-JUL-74
CILUS .LDR 39 22-JUL-74

This command lists the complete disk directory of the device RK1. The letter "C" following the file size on a DOS/BATCH or RSTS directory listing indicates the file is a contiguous file.

*DT1:*.*.PAL/L/S

This command lists all files with the extension .PAL which are on drive 1.

*DT1:*.*.*/F/T

All files on DECSystem-10 formatted DECTape 1, regardless of filename or extension, are listed; a fast directory is requested (/F) in which only filenames are printed.

J.2.5 Deleting Files from DOS/BATCH (RSTS) DECTapes

FILEX may be used to delete files from, and zero directories of, DOS/BATCH and RSTS formatted DECTapes. The format of these command lines is:

*dev:filnam.ext/S/D to delete a file

*dev:/S/Z to zero a directory

dev:/Z ARE YOU SURE?

where:

dev: = DOS/BATCH or RSTS DECTape.

filnam.ext = a valid DOS/BATCH (RSTS) filename and extension.
FILEX

/S = the switch from Table J-1 which designates that the device is a DOS/BATCH (RSTS) block-replaceable device.

/D = the switch from Table J-1 which designates that a file is to be deleted.

/Z = the switch from Table J-1 which designates that a directory is to be zeroed.

Examples:
*D10:*.PAL/D/S

All files on DECTape 0 with the extension .PAL are deleted.

*D2:TABLE.OBJ/D/S

The file TABLE.OBJ is deleted from the DECTape on unit 2.

*D10:/Z/S

DT0:/Z ARE YOU SURE ?Y

The DECTape on drive 0 is initialized in DOS/BATCH (RSTS) format so that it contains no files.

J.3 FILEX ERROR MESSAGES

Following is a list of error messages that may occur under FILEX:

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?filnam.ext ALREADY EXISTS?</td>
<td>An attempt was made to create the named file (filnam,ext) on a DOS DECTape when a file already existed under the name specified. Use /D to delete the file, and retry the transfer.</td>
</tr>
<tr>
<td>?BAD PPN?</td>
<td>The DOS/BATCH user identification code was not in the form [nnn,nnn], where each nnn is an octal number less than or equal to 377(octal).</td>
</tr>
<tr>
<td>?COR OVR?</td>
<td>There was insufficient main storage for buffers and input list expansion. Try copying files one at a time, without using the &quot;wild-card&quot; (<em>.</em>) construction on input.</td>
</tr>
<tr>
<td>?DEV FULL?</td>
<td>There was no room in the directory for the filename or there was no room on the output device for the file (in which case the filename is not placed in the output device directory).</td>
</tr>
<tr>
<td>?DIR ERR?</td>
<td>An error occurred while reading or looking up the directory of the input device, or the</td>
</tr>
</tbody>
</table>

January 1976

J-8
FILEX

input device does not have the proper file structure.

?FEATURE NOT IMP?

An operation was attempted which FILEX cannot perform (e.g., zeroing an RT-ll device).

?FG ACTIVE?

An attempt was made to use /T when a foreground job was active. The transfer is not allowed until the foreground job is terminated and unloaded via UNLOAD FG.

?FIL NAM?

The input filename was invalid or null.

?FIL NOT FND?

The input file was not found, or the "wild-card" (*.*) construction matched none of the existing files.

?ILL CMD?

The length of the command line exceeded 72 characters; the command line was not in proper CSI format; the UIC exceeded the allowed number of characters or [ was used without ]; a "wild-card" (*.*) construction was used on a sequential-access device; no output or no input file was specified for a copy operation, or more than one filename construction (dev:filnam.ext) was specified on either side of the = (<) sign.

?ILL DEV?

The device handler was not found, an invalid or illegal device name was used, or one of the following was attempted:

- RK or DT was not used for DOS/BATCH (RSTS) input in a copy operation;
- DT was not used for DOS/BATCH (RSTS) output in a zero or delete operation;
- DT was not used for DOS/BATCH (RSTS) output in a copy operation;
- DT was not used for DECsystem-10 input in any operation.

?ILL SWT?

An illegal switch was used in a command line (e.g., a switch not listed in Table J-1).

?IN ERR?

A device error occurred on input.

?NO UFD?

The specified UFD was not found on the DOS input disk.

?OUT ERR?

A device error occurred on output.

?SWT ERR?

An attempt was made to use more than one /S or /T switch in a command line (only one is allowed); an attempt was made to use more than one transfer mode switch (/I, /F, /A) or more than one operation switch (/D, /L, /F, /Z) in a command line (only one of each is allowed).

J-9 January 1976
APPENDIX K

SOURCE COMPARE (SRCCOM)

The RT-11 Source Compare program (SRCCOM) is used to compare two ASCII files and to output any differences to a specified output device. It is particularly useful when the two files are different versions of a single program, in which case SRCCOM prints all the editing changes which transpired between the two versions.

K.1 CALLING AND USING SRCCOM

To run SRCCOM type the command:

```
R SRCCOM
```

followed by a carriage return in response to the dot printed by the Keyboard Monitor; the CSI prints an asterisk. Then enter the names of the files which are to be compared using a command string in the following format:

```
dev:output=dev:input1,dev:input2/s
```

where:

- `dev:` is any valid device specification.
- `output` is the filename and extension assigned to the output file. If no output file is indicated, output is directed to the terminal.
- `input1...` are the input source filenames and extensions to be compared.
- `/s` is one of the switches listed in Table K-1.

Source files are examined line by line for groups of lines which match. When a mismatch occurs, all differences are output until n successive lines in the first file are identical to n lines in the second file. The number (n) is a variable which the user can set with the `/L` switch.
Source Compare

K.1.1 Extensions

No default extension is assigned by SRCCOM to the output file. The default extension for an input file is .MAC, representing a source file in MACRO language.

K.1.2 Switches

Command switches are generally placed at the end of the command string but may follow any filename in the string. The following switches can appear in the command string:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/B</td>
<td>Compare blank lines. Without this switch, blank lines are ignored.</td>
</tr>
<tr>
<td>/C</td>
<td>Ignore comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs). This switch does not cause a line consisting entirely of a comment to become a blank line, and therefore ignored in the line count.</td>
</tr>
<tr>
<td>/F</td>
<td>Include form feeds in the output file. (Form feeds are still compared if /F is not used, but they are not included in the output of differences.)</td>
</tr>
<tr>
<td>/H</td>
<td>Type list of switches available (help text). No I/O device is necessary since /H always prints the help text on the terminal.</td>
</tr>
<tr>
<td>/L:n</td>
<td>Specify the number of lines that determine a match (n is an octal number &lt;=310). All differences occurring before and after a match are output. In addition, the first line of the current match is output after the differences to aid in locating the place within each file at which the differences occurred. The default value for n is 3.</td>
</tr>
<tr>
<td>/S</td>
<td>Ignore spaces and tabs.</td>
</tr>
</tbody>
</table>

K.2 OUTPUT FORMAT

The first line of each file is always output as identification and is also compared. A blank line is then printed, followed by the differences between the files, in the following format:

January 1976

K-2
Source Compare

1)1 FILEA
1) A
****
2)1 FILEB
2) A

************

% FILES ARE DIFFERENT

The different lines are listed followed by a reference line which is the same for both files. Note the example below.

The following example uses SRCCOM to compare an edited file and its backup version. The default value for a match is 3 lines. Blank lines are ignored but all other characters are compared.

Following the example is a coded explanation of the comparison.

.R SRCCOM
*RK1:LINK0.FB, LINK0.BAK
A{1)
  1) TITLE RTLINK ROOT CODE X03-16
  2) TITLE RTLINK ROOT CODE X03-15
B{1)
  1) SEVENK= 31452 ;MINIMUM CORE TO START LINKER
C{1)
  1) .MCALL .CSISP, .CSIGEN, .SETTOP, .LOCK, .UNLOCK
  ****
  2)1 SEVENK= 31500 ;JUST BELOW 8K RESIDENT
B{2)
  2) .MCALL .CSISP, .CSIGEN, .SETTOP, .LOCK, .UNLOCK
C{2)
  2) ************
B{1)2 GLOBL RSWIT, RELPTR, FBTXT, OVLYNUM, RELOVL, RLSTRT
C{1)2 GLOBL RELADR, PNRELO, RELID1, RSIZ1, OVSIZ1, OVLCDE
  ****
B{2)2 GLOBL RSWIT, RELPTR, FBTXT, OVLYNUM, RELOVL
C{2)2 GLOBL RELADR, PNRELO, RELID1, RSIZ1, OVSIZ1, OVLCDE
  ************
B{1)2 RLSTRT: .BLKW ;CURRENT REL BLK OVERLAY NUM
C{1)2 RELPTR: .BLKW ;POINTER TO CURRENT REL BLK LOCATION
  ****
C{2)2 RELPTR: .BLKW ;POINTER TO CURRENT REL BLK LOCATION
  ************
B{1)2 MTITLE: .ASCII /RT-11 LINK X03-16/
C{1)2 .ASCII / LOAD MAP /
  ****
B{2)2 MTITLE: .ASCII /RT-11 LINK X03-15/
C{2)2 .ASCII / LOAD MAP /
  ************
1)12 .IF DF FB
B{1)1 MOV OBLK, RLSTRT ;IND START OF OVL FOR REL BLK
C{1)1 .ENDC
  ****
C{2)12 BR 1$
  ************

XFILES ARE DIFFERENT

K-3
Source Compare

A Headers, consisting of the first line of each file; for identification purposes.

B n)m. A notation where n is the number of the input file, and m is the page number (less than 256 decimal) of the input file on which the text appears. The right column lists the lines in the files which are different.

C Following a section of differences, a line identical to each file is output for reference purposes.

D Indicates that the files are different (this is printed on the system console, not in the output file).

This example uses the /L:n switch and sets the number of lines that determines a match to 2 lines. The first two columns represent the input files:

TEST FILE 1
LINE C
LINE E
LINE C
LINE D
LINE F
LINE H
LINE I
LINE J

TEST FILE 2
LINE C
LINE D
LINE C
LINE E
LINE F
LINE G
LINE H
LINE I
LINE J

The files are compared and differences listed on the line printer.

*LP*:TEST1,TEST2/L:2
1)1 TEST FILE 1
2)1 TEST FILE 2

January 1976

K-4
Source Compare

1)1  LINE E
1)  LINE C
1)  LINE D
1)  LINE F
1)  LINE H
****
2)1  LINE D
2)  LINE C
2)  LINE E
2)  LINE F
2)  LINE G
2)  LINE H

This message prints on the terminal indicating that the files are different.

%FILES ARE DIFFERENT

K.3 SRCCOM ERROR MESSAGES

The following errors may be reported by SRCCOM:

<table>
<thead>
<tr>
<th>Messages</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?COR OVR?</td>
<td>Not enough memory to hold a particular difference section.</td>
</tr>
<tr>
<td>?IN ERR?</td>
<td>A hardware error occurred in reading input.</td>
</tr>
<tr>
<td>?OUT ERR?</td>
<td>A hardware error occurred in writing output file, or output device full.</td>
</tr>
<tr>
<td>?SWITCH ERROR?</td>
<td>An invalid switch was found or a switch other than /L was given a value.</td>
</tr>
<tr>
<td>?TOO MUCH DIFFERENCE?</td>
<td>More than 310 (octal) lines of difference between two files were found.</td>
</tr>
</tbody>
</table>
APPENDIX L

PATCH

The PATCH utility program is used to make code modifications to memory image (.SAV) files, including overlay-structured and monitor files. PATCH, like ODT, can be used to interrogate, and then to change, words or bytes in the file.

PATCH provides eight relocation registers. Before changing a program with PATCH, copy the old file to a backup file with PIP, as the old file is modified when PATCH is used.

L.1 CALLING AND USING PATCH

To run PATCH, type the command:

R PATCH

followed by the RETURN key in response to the dot printed by the monitor. PATCH prints a version number message:

PATCH V01-02

and then prints the message:

FILE NAME --
*

In response to the asterisk, enter the name of the file to be modified, using the following format:

dev:filnam.ext/M/O

where:

dev: represents an optional device specification; if not specified, DK: is assumed.

filnam.ext represents the name of the file which is to be patched, if an extension is not indicated, .SAV is assumed.

/M must be used if the file is an RT-ll monitor file.

/O must be used if the file is an overlay-structured file.
After this information has been entered, the Command String Interpreter prints an asterisk indicating that it is ready to accept a command. Note that /O and /M, if used, must be specified when the file name is typed; they are not legal at any other time.

L.2 PATCH COMMANDS

Table L-1 summarizes the PATCH commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vr;nR</td>
<td>Set relocation register n to value Vr.</td>
</tr>
<tr>
<td>b;B</td>
<td>Set bottom address of overlay file to b.</td>
</tr>
<tr>
<td>[s:]r,o/</td>
<td>Open word location Vr + o in overlay segment s.</td>
</tr>
<tr>
<td>[s:]r,o\</td>
<td>Open byte location Vr + o in overlay segment s.</td>
</tr>
<tr>
<td>&lt;CR&gt;</td>
<td>Close currently open word/byte.</td>
</tr>
<tr>
<td>&lt;LF&gt;</td>
<td>Close currently open word/byte and open the next one.</td>
</tr>
<tr>
<td>t or ^</td>
<td>Close currently open word/byte and open the previous one.</td>
</tr>
<tr>
<td>@</td>
<td>Close the currently open word and open the word addressed by it.</td>
</tr>
<tr>
<td>F</td>
<td>Begin patching a new file.</td>
</tr>
<tr>
<td>E</td>
<td>Exit to RT-11 monitor.</td>
</tr>
</tbody>
</table>

Explanations of each command follow. An example of the use of the commands is provided in Section L.3.

L.2.1 Patch a New File

The F command causes PATCH to close the file being patched, and accept a new file name to be patched.
PATCH

L.2.2 Exit from PATCH

The E command causes PATCH to close the file being patched and return control to the RT-ll monitor.

L.2.3 Examine, Change Locations in the File

For a non-overlay file, a word address may be opened, as with ODT, by typing:

[<relocation register>,]offset/

At this point, PATCH will type out the contents of the location and wait for the user to type in either new location contents (in octal) or another command.

In an overlay file, the format is:

[<segment number>:] [<relocation register>,]offset/

Where <segment number> is the overlay segment number as it is printed on the link map for the file. If it is omitted, the root segment is assumed.

Similarly, to open a byte address in file, the format is:

[<relocation register>,]offset\ for non-overlay files, or

[<segment number>:] [<relocation register>,]offset\ for overlay files.

Once a location has been opened, the user may optionally type in the new contents in the format:

[<relocation register>,]value

followed by one of these control characters:

<carriage return>          Close the current location by changing its contents to the new contents (if specified), and await more control input.

<line feed>                Close the current location, and open the next word/byte.

↑ or ^                     Close the current location, and open the previous word/byte.

@                         Close the current word location, and open the word addressed by it (in the same segment if an overlay file).
L.2.4 Set Bottom Address

To patch an overlay file, PATCH must know the bottom address at which the program was linked if it is different from the initial stack pointer. This is the case if the program sets location 42 in an .ASECT. To set the bottom address, type:

<bottom address> B

Note that the B command must be issued before any locations are opened for modification.

L.2.5 Set Relocation Registers

The relocation registers 0-7 are set, as with ODT, by the R command. The R command has the format:

<relocation value> <relocation register> R

Once one of the eight relocation registers has been set, the expression:

<relocation register> <octal number>

typed as part of a command will have the value:

<relocation value> + <octal number>

L.3 EXAMPLES USING PATCH

The following example shows how to patch a non-overlaid file. Assume the following program (EXAM):

```
.data
MAIN RT=11 MACRO VM02=38 PAGE 1

1         000015  CR=  15
2         000012  LF=  12
3         000000  .CSECT MAIN
4         000000  .MCALL .PRINT,.EXIT
5         000000  .NLIST BEX
6         000000  124 MSG1  ASC11 /THIS IS A SUCCESSFUL PATCH/<CR><LF>
7         000000  BEX
8         00034  000403 START  BR  EXIT
9         00036  .PRINT MSG
10        00044  EXIT  .EXIT
11        00034  .ENU START
```

This program has been assembled with MACRO and linked with LINK; execution causes no output of text:

```
.R EXRM
```
PATCH

To make a line of text print on the terminal, PATCH is used as follows:

```
R PATCH
PATCH V01-02
FILE NAME--
*EXAM.SAV
*1000:OR
*0.34/403240
*E
```

Now when the program is executed:

```
R EXAM
THIS IS A SUCCESSFUL PATCH
```

The next example demonstrates a similar situation, only includes an overlay file. These programs have been assembled and linked; the output of both operations is included:

```
MAIN. RT-11 MACRO VM02*08 3-SEP-74 PAGE 1

1
2
3 000015 CRI#15
4 00012 LFI#12
5 00007 PC#17
6 000001 SECT MAIN
7 .GLOBAL ENTRY,MSG1
8 .MCALL ,PRINT,,EXIT
9 .LIST BEX
10 0000 124 MSG1 ASCLZ /THIS IS A SUCCESSFUL PATCH/<CR><LF>
11 00035 124 MSG1: ASCLZ /THIS IS AN OVERLAY PATCH/
12
13 00066 00403 START: WI EXIT
14 00070 .PRINT MSG2
15 00076 004767 EXIT: JSR PC,ENTRY
0000000G
16 00142 .EXIT
17 000051 .END START
```
Running the program (PTCH) produces no terminal output:

.R PTCH

But by using PATCH to modify the file as follows:
.R PATCH

PATCH V01-02

FILE NAME--
*PTCH.SAV/0
*1230:0R
*1:0.0/ 403 240
*E

the following line results:

.R PATCH
THIS IS AN OVERLAY PATCH

L.4 PATCH ERROR MESSAGES

Error messages which may occur under PATCH follow.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ADDR NOT IN SEG?</td>
<td>The address is not in the specified overlay segment.</td>
</tr>
<tr>
<td>?BAD SWITCH?</td>
<td>Typed a switch other than /O or /M.</td>
</tr>
<tr>
<td>?BOTTOM ADDR WRONG?</td>
<td>The bottom address specified or contained in location 42 of an overlay file is incorrect. Specify the correct one using the b:B command.</td>
</tr>
<tr>
<td>?INCORRECT FILE SPEC?</td>
<td>The response to the &quot;FILE NAME --&quot; message was not of the correct form. Try again.</td>
</tr>
<tr>
<td>?INSUFFICIENT CORE?</td>
<td>PATCH did not have enough memory to hold the file's device handler plus the internal &quot;segment table.&quot; This message should not occur.</td>
</tr>
<tr>
<td>?INVALID RELOC REG?</td>
<td>Tried to reference a relocation register outside the range 0-7.</td>
</tr>
<tr>
<td>?INVALID SEG NO?</td>
<td>The segment number S: does not exist.</td>
</tr>
<tr>
<td>?MUST OPEN WORD?</td>
<td>The @ command was typed when a byte location was open.</td>
</tr>
<tr>
<td>?MUST SPECIFY SEG?</td>
<td>The address referenced is not in the root section; a segment number S: must be used.</td>
</tr>
</tbody>
</table>

L-7
January 1976
<table>
<thead>
<tr>
<th>Question</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?NO ADDR OPEN?</td>
<td>The &lt;line feed&gt;, t or @ command was typed when no location was open.</td>
</tr>
<tr>
<td>?NOT IN PROGR BOUNDS?</td>
<td>Tried to open a location beyond the end of the file.</td>
</tr>
<tr>
<td>?ODD ADDRESS?</td>
<td>Tried to open a word address which was odd.  (Use &quot;&quot;.)</td>
</tr>
<tr>
<td>?ODD BOTTOM ADDR?</td>
<td>The bottom address specified or contained in location 42 of an overlay file is odd.</td>
</tr>
<tr>
<td>?PROG HAS NO SEGS?</td>
<td>The file specified as an overlay file is not.</td>
</tr>
</tbody>
</table>
APPENDIX M

PATCHO

The RT-11 PATCHO program is used to correct and update object modules (files output by the assemblers or by the FORTRAN compiler). It is particularly useful when making corrections to routines that are in .OBJ format for which the source files are not available. PATCHO cannot be used to patch libraries built by LIBR, but it can be used to patch the OBJ modules from which a library is built.

M.1 CALLING AND USING PATCHO

To run PATCHO type the command:

R PATCHO

in response to the dot printed by the Keyboard Monitor. When PATCHO is ready to accept commands, an asterisk is printed. The standard RT-11 command string is not used for PATCHO. Specific commands (described in Section M.2) are typed in response to the asterisk.

Type CTRL C to halt PATCHO at any time and return control to the monitor. To restart PATCHO, type R PATCHO or the REENTER command in response to the monitor's dot.

M.2 PATCHO COMMANDS

There are nine commands and arguments accepted by PATCHO. All arguments to a command must be separated from the command name by one or more spaces (e.g., POINT TOP is acceptable, POINTTOP is not). Commands are terminated by a carriage return.

M.2.1 OPEN Command

The OPEN command sets input and output file names. When the command is given, PATCHO prints:

ENTER INPUT FILE*

M-1
PATCHO

on the console terminal, and waits for a standard RT-ll device and
file specification to be entered (dev:filnam,ext). After the input
specification is given, followed by a carriage return, PATCHO responds
with:

ENTER OUTPUT FILE*

A device and file specification (followed by a carriage return) for
the desired output file is now accepted from the console.

NOTE

There is no default extension.
Therefore, the user must explicitly
specify the filename and extension.

Example:

*OPEN
ENTER INPUT FILE *RK1:OTS.OBJ
ENTER OUTPUT FILE *RK1:NEWOTS.OBJ

M.2.2 POINT Command

The POINT command locates an object module of a given name (used with
concatenated object modules) and prepares it for subsequent WORD,
BYTE, or DUMP operations. POINT takes one argument—the module name
to be located.

Example:

*POINT OTINIT

M.2.3 WORD Command

The WORD command modifies a given word in an object module. There may
be several arguments to the command, entered as illustrated below (the
BYTE command is also included since its arguments are exactly the
same; refer to Section M.2.4):

Address Specifier       Value Specifier

\{ WORD \} \ CSECT + OFFSET \{ NAME \ \%CSECT OR \ \%GLOBAL SYMBOL\}
\{ BYTE \} \ NAME + OFFSET

where in the Address Specifier:

1. CSECT is the name of the CSECT which contains the word to be
   modified. The CSECT must be present in the current module
   (the one specified in the POINT command).
The CSECT NAME is optional; if omitted, the blank CSECT is assumed. To represent a CSECT name which contains embedded blanks (.ABS.), a backarrow (SHIFT 0) or underscore character should be used in place of each embedded blank.

2. OFFSET is the octal location within the CSECT that is to be modified. OFFSET must be present.

3. = is a delimiter and must be present.

In the Value Specifier:

1. If # is used, the absolute value of the OFFSET in the Value Specifier field is placed in the target location (the address indicated by the Address Specifier).
   For example:
   
   *WORD PROG+24=#4
   
   This command patches location 24 in CSECT PROG to contain the value 4.

2. If % is used, displaced relocation is generated on the Value Specifier. This mode is used for PC-relative references. For example:

   *WORD PROG+24=%PROG1+24

   This command patches PROG+24 to contain the value:

   Address (PROG1+24)-Address (PROG+26)

3. If neither # or % are specified, the address of the Value Specifier is placed in the target location. For example:

   *WORD PROG+24=PROG+14

   This causes the word at PROG+24 to contain the address of the word at PROG+14, while:

   *WORD PROG+24=Ssymbol+0

   causes the word at PROG+24 to contain the address of the global symbol "symbol".

4. GLOBAL is optional and is the name of a global symbol whose value (address) is to be used in the word to be modified.

5. The characters + or - are optional and indicate the sign of the following OFFSET. If a global symbol is indicated, a + (or -) and OFFSET must be present.

M.2.4 BYTE Command

The BYTE command modifies a given byte in an object module. The arguments are the same as for WORD, explained in Section M.2.3.
PATCHO

Example:

*BYTE CSECTA+21=#101

The byte in CSECTA whose offset is 21 is patched to contain octal 101 (ASCII "A").

M.2.5 DUMP Command

The DUMP command prints the contents of the object module currently being pointed to and causes an automatic POINT to the next module in the input file, if any. Use the monitor SET LP CR command first for correct output format.

NOTE

LIST and DUMP go to LP: (lineprinter) by default. If the user wants LIST and DUMP to go to the terminal, he must execute an ASSIGN TT:6 command before typing R PATCHO.

Refer to Section M.4 for an example of the DUMP command.

M.2.6 LIST Command

The LIST command lists the names of all the object modules in the input file in the order in which they appear in the file. A POINT command should be given after the LIST command to assure that PATCHO is positioned at the desired module for the next operation. LIST lists the names of all object modules in a file without changing the module being pointed to.

Refer to Section M.4 for an example of the LIST command.

M.2.7 EXIT Command

The EXIT command returns to the operating system terminating a patch session and closing a file. The EXIT command takes no argument. As the EXIT command is executed, ENTER CHECKSUM: -- -- is displayed. The user should type a six-digit octal number corresponding to the appropriate checksum for the patch (if any). If the checksum agrees, PATCHO prints STOP-- and returns control to the monitor. If the checksum does not agree, PATCHO prints PAUSE -- ?BAD PATCH? and waits for user input. The user has two options: typing a CTRL C at this point aborts PATCHO without closing the output file, thus allowing PATCHO to be rerun and the patch to be reentered. For example:

```
; +EXIT
ENTER CHECKSUM: 115570

PAUSE -- ?BAD PATCH?
^C
```

January 1976
Typing a carriage return causes the output file to be closed, despite the bad patch, and returns control to the monitor. Note that in the latter case, if the output and input files were called by the same name and extension, the input file is destroyed and the output file contains an incorrect patch.

M.2.8 DEC Command

The DEC command is used generally by maintainers when creating system patches and when the proper checksum for the patch being made is unknown. If a DEC command is issued during a patch session, the EXIT command is automatically modified to display:

```
ENTER CHECKSUM:
```

followed by the correct checksum for the patch just completed. The checksum computed by PATCHO is derived from all of the commands entered during the session, thereby providing a safeguard against typographical errors. If the DEC command is used, it is the first command given.

```
*DEC
:
*EXIT
ENTER CHECKSUM: 115570
STOP --
```

M.2.9 HELP Command

The HELP command prints an explanation of PATCHO commands. The HELP command takes no arguments.

M.3 PATCHO LIMITATIONS

Note the following limitations on the use of PATCHO:

1. PATCHO only works with object modules or concatenated object modules, not with libraries.

2. PATCHO always copies its input file to its output file as it makes changes.

3. As a consequence of 2 (above) the POINT command can only be used to point to modules that physically follow the current module in the input file.

4. If changes are being made to a file, PATCHO must be given an EXIT command when all the changes are complete in order to assure that the entire input is copied to the output (i.e., CTRL C should not be used).

5. The LIST and DUMP commands will work in an 8K system only if an ASSIGN TT:6 command has been entered prior to running PATCHO. Otherwise 12K is required for these two commands.
M.4 EXAMPLES

The following is an example of the PATCHO command LIST in which a few of the names of object modules in the library file (OTS.OBJ) are listed:

*LIST

OBJECT MODULES:
ERRS
IVEC
IVECP
IPVEC
PVEC
PVECP
PPVEC
DVEC
DVECP
DPVEC
LVEC
LVECP
LPVEC
ERRS
ADTS
OTINIT

The next example is a sample of output produced by the PATCHO DUMP Command. The module is dumped by formatted binary block.

*DUMP

DUMP OF MODULE LPS0
-------------

BLOCK TYPE GSD
GLOBAL USAGE DEFINED RELOC EXTERNAL SIZE/ADRS
LPS0 MOD NAME NO NO NO 0
,ABS,CSECT YES NO NO 0
ERRARG GLOBAL NO NO YES 0
ERRPDL GLOBAL NO NO YES 0
ERRSYN GLOBAL NO NO YES 0

BLOCK TYPE GSD
GLOBAL USAGE DEFINED RELOC EXTERNAL SIZE/ADRS
EVAL GLOBAL NO NO YES 0


BLOCK TYPE GSD
GLOBAL USAGE DEFINED RELOC EXTERNAL SIZE/ADRS
USE GLOBAL YES YES YES 104
,ABS,TRANADR YES NO NO 1

BLOCK TYPE GSD END

BLOCK TYPE RLD
ADDRESS RLD TYPE GLOBAL OFFSET
1772 LCTR DEF 0

January 1976
M-6
M.5 PATCHO ERROR MESSAGES

<table>
<thead>
<tr>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>?BAD CHECKSUM?</td>
<td>A formatted binary block in the input file has a checksum which does not agree with that calculated for its contents.</td>
</tr>
<tr>
<td>?BAD OBJ?</td>
<td>The input file contains information which cannot be interpreted as an object module.</td>
</tr>
<tr>
<td>?DUMP ERROR?</td>
<td>An input/output error occurred while dumping a module.</td>
</tr>
<tr>
<td>?ILLEGAL COMMAND?</td>
<td>A command line was not recognized, or it was not in the proper format for the particular operation.</td>
</tr>
<tr>
<td>?MODULE NOT FOUND?</td>
<td>The module requested in a POINT command was not found in the input file between the position of the file at the time of the POINT and the file's end.</td>
</tr>
<tr>
<td>?MORE THAN 15 CHANGES?</td>
<td>Too many changes have been specified for a particular module. The patch should be broken up into several steps.</td>
</tr>
<tr>
<td>?MORE THAN 5 CSECTS REQUIRE CHANGE?</td>
<td>An attempt has been made to patch locations in too many different CSECTS. The patch should be made in several steps.</td>
</tr>
<tr>
<td>?NO FILE OPEN?</td>
<td>An attempt was made to use a command other than &quot;DEC&quot; or &quot;HELP&quot; before an OPEN command was issued.</td>
</tr>
<tr>
<td>?OFFSET?</td>
<td>The offset supplied in a WORD or BYTE command is not an octal number, or is in improper format.</td>
</tr>
<tr>
<td>Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>?OUTPUT ERROR?</td>
<td>A hardware error (or possibly a write-lock condition) occurred while attempting to write the output file.</td>
</tr>
<tr>
<td>?OUTPUT FILE TOO SMALL?</td>
<td>The space allocated to the output file is too small. This may be corrected by compressing the device with PIP (if enough total space is free on the device), or by using another device for output.</td>
</tr>
<tr>
<td>PAUSE -- ?BAD PATCH?</td>
<td>The checksum entered on exit does not agree with that calculated by PATCHO.</td>
</tr>
</tbody>
</table>

M.5.1 Run-Time Error Messages

Because PATCHO is a FORTRAN program, run-time error messages may occur. To find a complete explanation of each run-time error refer to the RT-11/RTS/E FORTRAN IV User's Guide or the RT-11 System Message Manual.

Listed below are four of the most important run-time error messages which may be encountered.

<table>
<thead>
<tr>
<th>Code</th>
<th>Level</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>FATAL</td>
<td>HARDWARE I/O ERROR</td>
<td>A hardware error has been detected during an I/O operation.</td>
</tr>
<tr>
<td>28</td>
<td>FATAL</td>
<td>OPEN FAILED FOR FILE</td>
<td>A file could not be found, there was no room in the device, or FORTRAN selected a channel already in use.</td>
</tr>
<tr>
<td>29</td>
<td>FATAL</td>
<td>NO ROOM FOR DEVICE HANDLER</td>
<td>There is not enough free memory left to accommodate a specific device handler.</td>
</tr>
<tr>
<td>30</td>
<td>FATAL</td>
<td>NO ROOM FOR BUFFERS</td>
<td>There is not enough free memory left to set up required I/O buffers.</td>
</tr>
</tbody>
</table>
APPENDIX N
DISPLAY FILE HANDLER

This appendix describes the assembly language graphics support provided under RT-11 for the GT40, GT44, and DECLab-11 display hardware systems.

The following manuals are suggested for additional reference:

For GT40 users:


For GT44 users:


For DECLab-11 users:


2. RT-11 BASIC Reference Manual (DEC-11-LBACA-B-D)

N.1 DESCRIPTION

The GT40, GT44, and DECLab-11 have hardware configurations that include a display processor and CRT (cathode ray tube) display. The GT44 has a 17-inch tube; the GT40 and DECLab-11 use a 12-inch tube, both systems are equipped with light pens and hardware character and vector generators, and are capable of high-quality graphics. The Display File Handler supports this graphics hardware at the assembly language level under the RT-11 monitor.

N.1.1 Assembly Language Display Support

The Display File Handler is not an RT-11 device handler, since it does not use the I/O structure of the RT-11 monitor. For example, it is not possible to use PIP to transfer a text file to the display via the Display File Handler. Rather, the Display File Handler provides the graphics programmer the means for the display of graphics files and the easy management of the display processor. Included in its
Display File Handler

capabilities are such services as interrupt handling, light pen support, tracking object, and starting and stopping of the display processor.

The Display File Handler manages the display processor by means of a base segment (called VTBASE) which contains interrupt handlers, an internal display file and some pointers and flags. The display processor cycles through the internal display file; any user graphics files to be displayed are accessed via display subroutine calls from the Handler's display file. In this way, the Display File Handler exerts control over the display processor, relieving the assembly language user of the task.

Through the Display File Handler, the programmer can insert and remove calls to display files from the Handler's internal display file. Up to two user files may be inserted at one time, and that number may be increased by re-assembling the Handler. Any user file inserted for display may be blanked (the subroutine call to it bypassed) and unblanked by macro calls to the Display File Handler.

Since the Handler treats all user display files as graphics subroutines to its internal display file, a display processor subroutine call is required. This is implemented with software, using the display stop instruction, and is available for user programs. This instruction and several other extended instructions implemented with the display stop instruction are described in Section N.3.

The facilities of the Display File Handler are accessed through a library of macros (VTMAC) which generate calls to a set of subroutines in VTLIB. VTMAC is the MACRO library, and its call protocol is similar to that of the RT-11 macros. The expansion of the macros is shown in Section N.6. VTMAC also contains, for convenience in programming, the set of recommended display processor instruction mnemonics and their values. The mnemonics are listed in Section N.7 and are used in the examples throughout this appendix.

VTCALL through VTCAL4 are the set of subroutines which service the VTMAC calls. They include functions for display file and display processor management. These are described in detail in Section N.2. VTCALL through VTCAL4 are currently constructed, along with the base segment VTBASE, as a library of (five) modules, called VTLIB.

N.1.2 Monitor Display Support

The RT-11 monitor, under Version 2, directly supports the display as console device. A keyboard monitor command, GT ON (GT OFF) described in Section 2.7.1, permits the selection of the display as console device. Selection results in the allocation of approximately 1.25K words of memory for text buffer and code. The buffer holds approximately 2000 characters.

The text display includes a blinking cursor to indicate the position in the text where a character is added. The cursor initially appears at the top left corner of the text area. As lines are added to the text the cursor moves down the screen. When the maximum number of lines are on the screen, the top line is deleted from the text buffer when the line feed terminating a new line is received. This causes the appearance of "scrolling", as the text disappears off the top of the display.
Display File Handler

When the maximum number of characters have been inserted in the text buffer, the scroller logic deletes a line from the top of the screen to make room for additional characters. Text may appear to move (scroll) off the top of the screen while the cursor is in the middle of a line.

The Display File Handler can operate simultaneously with the scroller program, permitting graphic displays and monitor dialogue to appear on the screen at the same time. It does this by inserting its internal display file into the display processor loop through the text buffer. However, the following should be noted. Under the Single-Job Monitor, if a program using the display for graphics is running with the scroller in use (that is, GT ON is in effect), and the program does a soft exit (.EXIT with R0 not equal to 0) with the display stopped, the display remains stopped until a CTRL C is typed at the keyboard.

This can be recognized by failure of the monitor to echo on the screen when expected. If the scroller text display disappears after a program exit, always type CTRL C to restore. If CTRL C fails to restore the display, the running program probably has an error.

Four scroller control characters provide the user with the capability of halting the scroller, advancing the scrolling in page sections, and printing hard copy from the scroller. These are described in Chapter 2.

NOTE

The scroller logic does not limit the length of a line, but the length of text lines affects the number of lines which may be displayed, since the text buffer is finite. As text lines become longer, the scroller logic may delete extra lines to make room for new text, temporarily decreasing the number of lines displayed.

N.2 DESCRIPTION OF GRAPHICS MACROS

The facilities of the Display File Handler are accessed through a set of macros, contained in VTMAC, which generate assembly language calls to the Handler at assembly time. The calls take the form of subroutine calls to the subroutines in VTLIB. Arguments are passed to the subroutines through register 0 and, in the case of the .TRACK call, through both register 0 and the stack.

This call convention is similar to Version 1 RT-11 I/O macro calls, except that the subroutine call instruction is used instead of the EMT instruction. If a macro requires an argument but none is specified, it is assumed that the address of the argument has already been placed in register 0. The programmer should not assume that R0 is preserved through the call.

N.2.1 .BLANK

The .BLANK request temporarily blanks the user display file specified in the request. It does this by by-passing the call to the user
Display File Handler

display file, which prevents the display processor from cycling through the user file, effectively blanking it. This effect can later be cancelled by the .RESTR request, which restores the user file. When the call returns, the user is assured the display processor is not in the file that was blanked.

Macro Call: .BLANK .faddr

where: .faddr is the address of the user display file to be blanked.

Errors:

No error is returned. If the file specified was not found in the Handler file or has already been blanked, the request is ignored.

N.2.2 .CLEAR

The .CLEAR request initializes the Display File Handler, clearing out any calls to user display files and resetting all of the internal flags and pointers.

After initialization with .LNKRT (Section N.2.4), the .CLEAR request can be used any time in a program to clear the display and to reset pointers. All calls to user files are deleted and all pointers to status buffers are reset. They must be re-inserted if they are to be used again.

Macro Call: .CLEAR

Errors:

None.

Example:

This example uses a .CLEAR request to initialize the Handler, then later uses the .CLEAR to re-initialize the display. The first .CLEAR is used for the case when a program may be restarted after a CTRL C or other exit.

```assembly
BR RSTRT
RSTRT: .UNLK
.LNKRT
.CLEAR
.INSTR #FILE1
TTYIN CMPI #12,R0
BNE 18
.CLEAN
.INSTR #FILE2

FILE:1 POINT 0
500 LONGV 5001 INTX 0
DRET 0
```

January 1976 N-4
Display File Handler

FILE2: POINT 500
0 LONGV 1DRAW A LINE
0 INTX 1TC (500,500)
500 DRET 0
.END EX1

N.2.3 .INSRT

The .INSRT request inserts a call to the user display file specified in the request into the Display File Handler's internal display file. .INSRT causes the display processor to cycle through the user file as a subroutine to the internal file. The handler permits two user files at one time. The call inserted in the handler looks like the following:

DJSR ;DISPLAY SUBROUTINE
+.4 ;RETURN ADDRESS
.faddr ;SUBROUTINE ADDRESS

The call to the user file is removed by replacing its address with the address of a null display file. The user file is blanked by replacing the DJSR with a DJMP instruction, bypassing the user file.

Macro Call: .INSRT .faddr

where: .faddr is the address of the user display file to be inserted.

Errors:

The .INSRT request returns with the C bit set if there was an error in processing the request. An error occurs only when the Handler's display file is full and cannot accept another file. If the user file specified exists, the request is not processed. Two display files with the same starting address cannot be inserted.

Example:

See the examples in Sections N.2.2 and N.2.4.

N.2.4 .LNKRT

The .LNKRT request sets up the display interrupt vectors and possibly links the Display File Handler to the scroll text buffer in the RT-11 monitor. It must be the first call to the Handler, and is used whether or not the RT-11 monitor is using the display for console output (i.e., the KMON command GT ON has been entered).

The .LNKRT request used with the Version 2 RT-11 monitor enables a display application program to determine the environment in which it is operating. Error codes are provided for the situations where there is no display hardware present on the system or the display hardware is already being used by another task (e.g., a foreground job in the Foreground/Background version).
Display File Handler

The existence of the monitor scroller and the size of the Handler's subpicture stack are also returned to the caller. If a previous call to .LNKRT was made without a subsequent .UNLNK, the .LNKRT call is ignored and an error code is returned.

Macro Call: .LNKRT

Errors:

Error codes are returned in R0, with the N condition bit set.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No VT11 display hardware is present on this system.</td>
</tr>
<tr>
<td>-2</td>
<td>VT11 hardware is presently in use.</td>
</tr>
<tr>
<td>-3</td>
<td>Handler has already been linked.</td>
</tr>
</tbody>
</table>

On completion of a successful .LNKRT request, R0 will contain the display subroutine stack size, indicating the depth to which display subroutines may be nested. The N bit will be zero.

If the RT-11 monitor scroll text buffer was not in memory at the time of the .LNKRT, the C bit will be returned set. The KMON commands GT ON and GT OFF cannot be issued while a task is using the display.

Example:

```
START: .LNKRT
8MI  ERROR
BCS  CONT
.SCROL  #8BUF
.INSTRT  #FILE1
TTYIN
CMPB  #12, R0
BNE  15
,UNLNK
,EXIT

8BUF1  ,BYTE  5
,BYTE  7
,WORD  1000

FILE11  POINT
500
500
,CHAR
.ASCII /THIS IS FILE1, TYPE CR TO EXIT/
,EVEN
,RET

ERROR1 Error routine
```

N-6
Display File Handler

N.2.5 .LPEN

The .LPEN request transfers the address of a light pen status data buffer to VTBASE. Once the buffer pointer has been passed to the Handler, the light pen interrupt handler in VTBASE will transfer display processor status data to the buffer, depending on the state of the buffer flag.

The buffer must have seven contiguous words of storage. The first word is the buffer flag, and it is initially cleared (set to zero) by the .LPEN request. When a light pen interrupt occurs, the interrupt handler transfers status data to the buffer and then sets the buffer flag non-zero. The program can loop on the buffer flag when waiting for a light pen hit (although doing this will tie up the processor, and in a foreground/background environment, timed waits would be more desirable). No further data transfers take place, despite the occurrence of numerous light pen interrupts, until the buffer flag is again cleared to zero. This permits the program to process the data before it is destroyed by another interrupt.

The buffer structure looks like this:

Buffer Flag
Name
Subpicture Tag
Display Program Counter (DPC)
Display Status Register (DSR)
X Status Register (XSR)
Y Status Register (YSR)

The Name value is the contents of the software Name Register (described in N.3.5) at the time of interrupt. The Tag value is the tag of the subpicture being displayed at the time of interrupt. The last four data items are the contents of the display processor status registers at the time of interrupt. They are described in detail in Table N-1.

Macro Call: .LPEN .baddr .

where: .baddr is the address of the 7-word light pen status data buffer.

Errors:

None.

If a .LPEN was already issued and a buffer specified, the new buffer address replaces the previous buffer address. Only one light pen buffer can be in use at a time.

Example:

```
INSRT #LFILE       IDISPLAY LFILE
.LPEN #LBUF        #SET UP LPEN BUFFER
LOOP: TST LBUF    #TEST LBUF FLAG, WHICH
       BEQ LOOP    #WILL BE SET NON-ZERO
       ; PROCESS DATA IN LBUF HERE,
       CLR LBUF     #CLEAR THE BUFFER FLAG,
       BR LOOP      #PERMITTING ANOTHER "HIT",
               #GO WAIT FOR IT.
```
Display File Handler

| LBUFF | BLK | 7 | LFILE1 | ISEVEN WORD LPEN BUFFER

Table N-1
Description of Display Status Words

<table>
<thead>
<tr>
<th>Bits</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DISPLAY PROGRAM COUNTER (DPC=172000)</strong></td>
<td></td>
</tr>
<tr>
<td>0-15</td>
<td>Address of display processor program counter at time of interrupt.</td>
</tr>
<tr>
<td><strong>DISPLAY STATUS REGISTER (DSR=172002)</strong></td>
<td></td>
</tr>
<tr>
<td>0-1</td>
<td>Line Type</td>
</tr>
<tr>
<td>2</td>
<td>Spare</td>
</tr>
<tr>
<td>3</td>
<td>Blink</td>
</tr>
<tr>
<td>4</td>
<td>Italics</td>
</tr>
<tr>
<td>5</td>
<td>Edge Indicator</td>
</tr>
<tr>
<td>6</td>
<td>Shift Out</td>
</tr>
<tr>
<td>7</td>
<td>Light Pen Flag</td>
</tr>
<tr>
<td>8-10</td>
<td>Intensity</td>
</tr>
<tr>
<td>11-14</td>
<td>Mode</td>
</tr>
<tr>
<td>15</td>
<td>Stop Flag</td>
</tr>
<tr>
<td><strong>X STATUS REGISTER (XSR=172004)</strong></td>
<td></td>
</tr>
<tr>
<td>0-9</td>
<td>X Position</td>
</tr>
<tr>
<td>10-15</td>
<td>Graphplot Increment</td>
</tr>
<tr>
<td><strong>Y STATUS REGISTER (YSR=172006)</strong></td>
<td></td>
</tr>
<tr>
<td>0-9</td>
<td>Y Position</td>
</tr>
<tr>
<td>10-15</td>
<td>Character Register</td>
</tr>
</tbody>
</table>
Display File Handler

N.2.6 .NAME

The .NAME request has been added to the Version 2 Handler. The contents of the name register are now stacked when a subpicture call is made. When a light pen interrupt occurs, the contents of the name register stack may be recovered if the user program has supplied the address of a buffer through the .NAME request.

The buffer must have a size equal to the stack depth (default is 10) plus one word for the flag. When the .NAME request is entered, the address of the buffer is passed to the Handler and the first word (the flag word) is cleared. When a light pen hit occurs, the stack's contents are transferred and the flag is set non-zero.

Macro Call: .NAME ,baddr

where: .baddr is the address of the name register buffer.

Errors:

None.

If a .NAME request has been previously issued, the new buffer address replaces the previous buffer address.

N.2.7 .REMOV

The .REMOV request removes the call to a user display file previously inserted in the handler's display file by the .INSRT request. All reference to the user file is removed, unlike the .BLANK request, which merely bypasses the call while leaving it intact.

Macro Call: .REMOV ,faddr

where: .faddr is the address of the display file to be removed.

Errors:

No errors are returned. If the file address given cannot be found, the request is ignored.

N.2.8 .RESTR

The .RESTR request restores to view a user display file that was previously blanked by a .BLANK request. It removes the by-pass of the call to the user file, so that the display processor once again cycles through the user file.

Macro Call: .RESTR ,faddr

where: .faddr is the address of the user file that is to be restored to view.

Errors:

No errors are returned. If the file specified cannot be found, the request is ignored.
Display File Handler

N.2.9 .SCROL

This request is used to modify the appearance of the Display Monitor's text display. The .SCROL request permits the programmer to change the maximum line count, intensity and the position of the top line of text of the scroll. The request passes the address of a two-word buffer which contains the parameter specifications. The first byte is the line count, the second byte is the intensity, and the second word is the \$ position. Both line count and \$ position must be octal numbers. The intensity may be a number from 1 to 8, ranging from lowest to highest intensity. If an intensity of 1 is specified, the scroll text will be almost unnoticeable at a BRIGHTNESS knob setting less than one-half. The scroll parameter change is temporary, since an .UNLINK or CTRL C restores the previous values.

Macro Call: .SCROL .baddr

where: .baddr is the address of the two-word scroll parameters buffer.

Errors:

No errors are returned. No checking is done on the values of the parameters. A zero argument is interpreted to mean that the parameter value is not to be changed. A negative argument causes the default parameter value to be restored.

Example:

```
 .SCROL #SCBUF  // ADJUST SCROLL PARAMETERS

 SCBUF1 .BYTE 5    // DECREASE # LINES TO 5,
 .BYTE 0           // LEAVE INTENSITY UNCHANGED,
 .WORD 300         // STOP LINE AT Y = 300.
```

N.2.10 .START

The .START request starts the display processor if it was stopped by a .STOP directive. If the display processor is running, it is stopped first, then restarted. In either case, the subpicture stack is cleared and the display processor is started at the top of the handler's internal display file.

Macro Call: .START

Errors:

None.

N.2.11 .STAT

The .STAT request transfers the address of a seven-word status buffer to the display stop interrupt routine in VTBASE. Once the transfer has been made, display processor status data is transferred to the buffer by the display stop interrupt routine in VTBASE whenever a
Display File Handler

.DSTAT or .DHALT instruction is encountered (see Sections N.3.3 and N.3.4). The transfer is made only when the buffer flag is clear (zero). After the transfer is made, the buffer flag is set non-zero and the .DSTAT or .DHALT instruction is replaced by a .DNOP (Display NOP) instruction.

The status buffer must be a seven-word, contiguous block of memory. Its contents are the same as the light pen status buffer. For a detailed description of the buffer and an explanation of the status words, see section N.2.5 and Table N-1.

Macro Call: .STAT .baddr

where: .baddr is the address of the status buffer receiving the data.

Errors:

No errors are indicated. If a buffer was previously set up, the new buffer address is replaced as the old buffer address.

N.2.12 .STOP

The .STOP request "stops" the display processor. It actually effects a stop by preventing the DPU from cycling through any user display files. It is useful for stopping the display during modification of a display file, a risky task when the display processor is running. However, a .BLANK could be equally useful for this purpose, since the .BLANK request does not return until the display processor has been removed from the user display file being blanked.

Macro Call: .STOP

Errors:

None.

NOTE

Since the display processor must cycle through the text buffer in the Display Monitor in order for console output to be processed, the text buffer remains visible after a .STOP request is processed, but all user files disappear.

N.2.13 .SYNC/.NOSYN

The .SYNC and .NOSYN requests provide program access to the power line synchronization feature of the display processor. The .SYNC request enables synchronization and the .NOSYN request disables it (the default case).

Synchronization is achieved by stopping the display and restarting it when the power line frequency reaches a trigger point, e.g., a peak or zero-crossing. Synchronization has the effect of fixing the display
Display File Handler

refresh time. This may be useful in some cases where small amounts of material are displayed but the amount frequently changes, causing changes in intensity. In most cases, however, using synchronization increases flicker.

Macro Calls: .SYNC
.NOSYN

Errors:
None.

N.2.14 .TRACK

The .TRACK request causes the tracking object to appear on the display CRT at the position specified in the request. The tracking object is a diamond-shaped display figure which is light-pen sensitive. If the light pen is placed over the tracking object and then moved, the tracking object follows the light pen, trying to center itself on the pen.

The tracking object first appears at a position specified in a two-word buffer whose address was supplied with the .TRACK request. As the tracking object moves to keep centered on the light pen, the new center position is returned to the buffer. A new set of X and Y values is returned for each light pen interrupt.

The tracking object cannot be lost by moving it off the visible portion of the display CRT. When the edge flag is set, indicating a side of the tracking object is crossing the edge of the display area, the tracking object stops until moved toward the center. To remove the tracking object from the screen, repeat the .TRACK request without arguments.

The .TRACK request may also include the address of a completion routine as the second argument. If a .TRACK completion routine is specified, the light pen interrupt handler passes control to the completion routine at interrupt level. The completion routine is called as a subroutine and the exit statement must be an RTS PC. The completion routine must also preserve any registers it may use.

Macro Call: .TRACK .baddr, .croutine

where: .baddr is the address of the two-word buffer containing the X and Y position for the track object.
    .croutine is the address of the completion routine.

Errors:
None.

Example:
See Section N.10.
Display File Handler

N.2.15 .UNLNK

The .UNLNK request is used before exiting from a program. In the case where the scroller is present, .UNLNK breaks the link, established by .LNKRT, between the Display File Handler's internal display file and the scroll file in the Display Monitor. The display processor is started cycling in the scroll text buffer, and no further graphics may be done until the link is established again. In the case where no scroller exists, the display processor is simply left stopped.

Macro Call: .UNLNK

Errors:

No errors are returned. An internal link flag is checked to determine if the link exists. If it does not exist, the request is ignored.

N.3 EXTENDED DISPLAY INSTRUCTIONS

The Display File Handler offers the assembly language graphics programmer an extended display processor instruction set, implemented in software through the use of the Load Status Register A (LSRA) instruction. The extended instruction set includes: subroutine call, subroutine return, display status return, display halt, and load name register.

N.3.1 DJSR Subroutine Call Instruction

The DJSR instruction (octal code is 173400) simulates a display subroutine call instruction by using the display stop instruction (LSRA instruction with interrupt bits set). The display stop interrupt handler interprets the non-zero word following the DJSR as the subroutine return address, and the second word following the DJSR as the address of the subroutine to be called. The instruction sequence is:

    DJSR
    Return address
    Subroutine address

Example:

To call a subroutine SQUARE:

    POINT 100
    100 100
    DJSR
    THEN CALL SUBROUTINE SQUARE

The use of the return address preceding the subroutine address offers several advantages. BASIC/GT uses the return address to branch around subpicture tag data stored following the subpicture address. This structure is described in Section N.5.3. In addition, a subroutine may be temporarily bypassed by replacing the DJSR code with a DJMP instruction, without the need to stop the display processor to make the by-pass.
Display File Handler

The address of the return address is stacked by the display stop interrupt handler on an internal subpicture stack. The stack depth is conditionalized and has a default depth of 10. If the stack bottom is reached, the display stop interrupt handler attempts to protect the system by rejecting additional subroutine calls. In that case, the portions of the display exceeding the legal stack depth will not be displayed.

N.3.2 DRET Subroutine Return Instruction

The DRET instruction provides the means for returning from a display file subroutine. It uses the same octal code as DJSR, but with a single argument of zero. The DRET instruction causes the display stop interrupt handler to pop its subpicture stack and fetch the subroutine return address.

Example:

```
SQUARE: LONGV
100|INRX
0
0|INRX
100
100|INRX|MINUSX
0
0|INRX
100|MINUSX
DRET
0
```

N.3.3 DSTAT Display Status Instruction

The DSTAT instruction (octal code is 173420) uses the LSRA instruction to produce a display stop interrupt, causing the display stop interrupt handler to return display status data to a seven-word user status buffer. The status buffer must first have been set up with a .STAT macro call (if not, the DSTAT is ignored and the display is resumed). The first word of the buffer is set non-zero to indicate the transfer has taken place, and the DSTAT is replaced with a DNOP (display NOP). The first word is the buffer flag and the next six words contain name register contents, current subpicture tag, display program counter, display status register, display X register, and display Y register. After transfer of status data, the display is resumed.

N.3.4 DHALT Display Halt Instruction

The DHALT instruction (octal code is 173500) operates similarly to the DSTAT instruction. The difference between the two instructions is that the DHALT instruction leaves the display processor stopped when exiting from the interrupt. A status data transfer takes place provided the buffer was initialized with a .STAT call. If not, the DHALT is ignored.
Display File Handler

Example:

```
STAT
MOV TYPE,STPLOC
INSRT
TB
SEQ

;STAT
;MOVE
;INSERT
;TYPE
;DISPLAY TYPE
;DISPLAY THE PICTURE

SBUF
BUFFER

;STATUS BUFFER
;POSITION NEAR TOP OF 12" TUBE

DFILE
POINT

;DRAW A LINE, MAYBE OVER EDGE
;IF IT IS A 12" SCOPE,

STPLOC
DNOP
DRET

;STATUS WILL BE RETURNED AT THIS POINT
```

N.3.5 DNAME Load Name Register Instruction

The Display File Handler provides a name register capability through the use of the display stop interrupt. When a DNAME instruction (octal code is 173520) is encountered, a display stop interrupt is generated. The display stop handler stores the argument following the DNAME instruction in an internal software register called the "name register". The current name register contents are returned whenever a DSTAT or DHALT is encountered, and more importantly, whenever a light pen interrupt occurs. The use of a "name" (with a valid range from 1 to 77777) enables the programmer to label each element of the display file with a unique name, permitting the easy identification of the particular display element selected by the light pen.

The name register contents are stacked on a subpicture call and restored on return from the subpicture.

Example:

The SQUARE subroutine with "named" sides:

```
SQUARE

NAME IS
10
DRAW A SIDE

100\INTX
0
NAME
11
THIS SIDE IS NAMED
0\INTX
100
NAME
12
STILL IN LONG VECTOR MODE
```

N-15
N.4 USING THE DISPLAY FILE HANDLER

Graphics programs which intend to use the Display File Handler for display processor management can be written in MACRO assembly language. The display code portions of the program may use the mnemonics described in Section N.7. Calls to the Handler should have the format described in Section N.6.

The Display File Handler is supplied in two pieces, a library of MACRO calls and a library containing the Display File Handler modules.

MACRO Library:       VTMAC.MAC

Display File Handler: VTHDLR.OBJ (consisting of:)

VBASE.OBJ
VTCALL1.OBJ
VTCALL2.OBJ
VTCALL3.OBJ
VTCALL4.OBJ

N.4.1 Assembling Graphics Programs

To assemble a graphics program using the display processor mnemonics or the Display Handler macro calls, the file VTMAC.MAC must be assembled with the program, and must precede the program in the assembler command string.

Example:

Assume PICTUR.MAC is a user graphics program to be assembled. An assembler command string would look like this:

*R MACRO
*PICTUR=VTMAC,PICTUR

N.4.2 Linking Graphics Programs

Once assembled with VTMAC, the graphics program must be linked with the Display File Handler, which is supplied as a single concatenated object module, VTHDLR.OBJ. The Handler may optionally be built as a library, following the directions in N.8.5. The advantage of using the library when linking is that the Linker will select from the library only those modules actually used. Linking with VTHDLR.OBJ results in all modules being included in the link.

To link a user program called PICTUR.OBJ using the concatenated object module supplied with RT-ll:

*R LINK
*PICTUR=PICTUR,VTHDLR
*
Display File Handler

To link a program called PICTUR,OBJ using the VTLIB library built by following the directions in N.8.5, be sure to use the Version 2 Linker:

```
.R LINK
*PICTUR=PICTUR,VTLIB
*

VTLIB (Handler Modules):

<table>
<thead>
<tr>
<th>Module</th>
<th>CSECT</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTCA1</td>
<td>$GT1</td>
<td>.CLEAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.START</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.INSRT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.REMOV</td>
</tr>
<tr>
<td>VTCA2</td>
<td>$GT2</td>
<td>.BLANK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.RESTR</td>
</tr>
<tr>
<td>VTCA3</td>
<td>$GT3</td>
<td>.LPEN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.STAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.SYNC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.NOSYN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.TRACK</td>
</tr>
<tr>
<td>VTCA4</td>
<td>$GT4</td>
<td>.LNKRT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.UNLNK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.SCROL</td>
</tr>
<tr>
<td>VTBASE</td>
<td>$GTB</td>
<td>Memory resident base module containing interrupt handlers and internal display file</td>
</tr>
</tbody>
</table>
```

To link a display program using overlays, the modules must be specified individually. The user must acquire the sources and follow the assembly directions in Section N.8. The modules VTCA1 and VTCA2 must be simultaneously resident. For example:

```
.R LINK
*PICTUR=PICTUR,VTBASE/C
*VTCA1,VTCA2,VTCA3/O:1/C
*VTCA4/O:1
```

To link a display program to run as a foreground task:

```
.R LINK
*PICTUR=PICTUR,VTLIB/R
```

N.5 DISPLAY FILE STRUCTURE

The Display File Handler supports a variety of display file structures, takes over the job of display processor management for the programmer, and may be used for both assembly language graphics programming and for systems program development. For example, the
Display File Handler

Handler supports the tagged subpicture file structure used by BASIC/GT, as well as simple file structures. These are discussed in this section.

N.5.1 Subroutine Calls

A subroutine call instruction, with the mnemonic DJSR, is implemented using the display stop (DSTOP) instruction with an interrupt. The display stop interrupt routine in the Display File Handler simulates the DJSR instruction, and this allows great flexibility in choosing the characteristics of the DJSR instruction.

The DJSR instruction stops the display processor and requests an interrupt. The DJSR instruction may be followed by two or more words, and in this implementation the exact number may be varied by the programmer at any time. The basic subroutine call has this form:

```
DJSR
Return Address
Subroutine Address
```

In practice, simple calls to subroutines could look like:

```
DJSR
.WORD .+4
.WORD SUB
```

where SUB is the address of the subroutine. Control will return to the display instruction following the last word of the subroutine call. This structure permits a call to the subroutine to be easily by-passed without stopping the display processor, by replacing the DJSR with a display jump (DJMP) instruction:

```
DJMP
.WORD .+4
.WORD SUB
```

A more complex display file structure is possible if the return address is generalized:

```
.DJSR
.WORD NEXT
.WORD SUB
```

where NEXT is the generalized return address. This is equivalent to the sequence:

```
DJSR
.WORD .+4
.WORD SUB
DJMP
.WORD NEXT
```

It is also possible to store non-graphic data such as tags and pointers in the subroutine call sequence, such as is done in the tagged subpicture display file structure of BASIC/GT. This technique looks like:
Display File Handler

DJSR
    .WORD NEXT
    .WORD SUB

DATA
NEXT:


For simple applications where the flexibility of the DJSR instruction described above is not needed and the resultant overhead not desired, the Display File Handler (VTBASE,MAC and VTCALL,MAC) can be conditionally re-assembled to produce a simple DJSR call. If NOTAG is defined during the assembly, the Handler will be configured to support this simple DJSR call:

DJSR
    .WORD SUB

where SUB is the address of the subroutine. Defining NOTAG will eliminate the subpicture tag capability, and with it the tracking object, which uses the tag feature to identify itself to the light pen interrupt handler.

Whatever the DJSR format used, all subroutines and the user main file must be terminated with a subroutine return instruction. This is implemented as a display stop instruction (given the mnemonic DRET) with an argument of zero. A subroutine then has the form:

SUB: Display Code
    DRET
    .WORD 0

N.5.2 Main File/Subroutine Structure

A common method of structuring display files is to have a main file which calls a series of display subroutines. Each subroutine will produce a picture element and may be called many times to build up a picture, producing economy of code. If the following macros are defined:

.MACRO CALL <ARG>
DJSR
    .WORD +4
    .WORD ARG
    .ENDM
.MACRO RETURN
DRET
    .WORD 0
    .ENDM

then a main file/subroutine file structure would look like:

; MAIN DISPLAY FILE

MAIN: Display Code
CALL SUB1 ;CALL SUBROUTINE 1
Display Code
CALL SUB2 ;CALL SUBROUTINE 2
;ETC
Display File Handler

; RETURN
; DISPLAY SUBROUTINES
; SUB1: Display Code ;SUBROUTINE 1
\RETURN
; SUB2: Display Code ;SUBROUTINE 2
\RETURN
; ;ETC.

N.5.3 BASIC/GT Subroutine Structure

An example of another approach to display file structure is the tagged subpicture structure used by BASIC/GT. The display file is divided into distinguishable elements called subpictures, each of which has its own unique tag.

The subpicture is constructed as a subroutine call followed by the subroutine. It is essentially a merger of the main file/subroutine structure into an in-line sequence of calls and subroutines. As such, it facilitates the construction of display files in real time, one of the important advantages of BASIC/GT.

The following is an example of the subpicture structure. Each subpicture has a call to a subroutine with the return address set to be the address of the next subpicture. The subroutine called may either immediately follow the call, or may be a subroutine defined as part of a subpicture created earlier in the display file. This permits a subroutine to be used by several subpictures without duplication of code. Each subpicture has a tag to identify it and it is this tag which is returned by the light pen interrupt routine. To facilitate finding subpictures in the display file, they are made into a linked list by inserting a forward pointer to the next tag.

SUB1: DJSR ;START OF SUBPICTURE 1
\WORD SUB2 ;NEXT SUBPICTURE
\WORD SUB1+12 ;JUMP TO THIS SUBPICTURE
\WORD 1 ;JTAG = 1
\WORD SUB2+6 ;POINTER TO NEXT TAG

; BODY OF SUBPICTURE 1
DRET ;RETURN FROM
0 ;SUBPICTURE 1

SUB2: DJSR ;START SUBPICTURE 2
\WORD SUB3 ;NEXT SUBPICTURE
\WORD SUB2+12 ;JUMP TO THIS SUBPICTURE
\WORD 2 ;JTAG = 2
\WORD SUB3+6 ;PTR TO NEXT TAG

; BODY OF SUBPICTURE 2
DRET ;RETURN FROM
0 ;SUBPICTURE 2

N-20
Display File Handler

SUB31 DJSR  
.WORD SUB4  
.WORD SUB1+12  
.WORD 3  
.WORD SUB4+6  

SUB41 DJSR  
.
.

N.6 SUMMARY OF GRAPHICS MACRO CALLS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>MACRO Call (See Note 1)</th>
<th>Assembly Language Expansion (see Note 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.BLANK</td>
<td>Temporarily blanks a user display file.</td>
<td>.BLANK .faddr</td>
<td>.GLOBAL $VBLNK .IF NB, .faddr MOV .faddr, %t00 .ENDC JSR %t07, $VBLNK</td>
</tr>
<tr>
<td>.CLEAR</td>
<td>Initializes handler.</td>
<td>.CLEAR</td>
<td>.GLOBAL $VINIT JSR %t07, $VINIT</td>
</tr>
<tr>
<td>.INSRT</td>
<td>Inserts a call to user display file in handler's master display file.</td>
<td>.INSRT .faddr</td>
<td>.GLOBAL $VINSRT .IF NB, .faddr MOV .faddr, %t00 .ENDC JSR %t07, $VINSRT</td>
</tr>
<tr>
<td>.LNKRT</td>
<td>Sets up vectors and links display file handler to RT-ll scroller.</td>
<td>.LNKRT</td>
<td>.GLOBAL $VRLNK JSR %t07, $VRLNK</td>
</tr>
<tr>
<td>.LPEN</td>
<td>Sets up light pen status buffer.</td>
<td>.LPEN .baddr</td>
<td>.GLOBAL $VLPPEN .IF NB, .baddr MOV .baddr, %t00 .ENDC JSR %t07, $VLPPEN</td>
</tr>
<tr>
<td>.NAME</td>
<td>Sets up buffer to receive name register stack contents.</td>
<td>.NAME .baddr</td>
<td>.GLOBAL $VNAME .IF NB, .baddr MOV .BADDR, %t00 .ENDC JSR %t07, $VNAME</td>
</tr>
<tr>
<td>.NOSYN</td>
<td>Disable power line sync.</td>
<td>.NOSYN</td>
<td>.GLOBAL $VNSYN JSR %t07, $VNSYN</td>
</tr>
<tr>
<td>.REMOV</td>
<td>Removes the call to a user display file.</td>
<td>.REMOV .faddr</td>
<td>.GLOBAL $VRMOV .IF NB, .faddr MOV .faddr, %t00 .ENDC JSR %t07, $VRMOV</td>
</tr>
</tbody>
</table>

N-21
Display File Handler

.RESTR Unblanks the user display file.

.RESTR .faddr

.GLOBL $VRSTR
IF NB, .faddr
MOV .faddr, %100
ENDC
JSR %107, $VRSTR

.SCROL Adjusts monitor scroller parameters.

.SCROL .baddr

.GLOBL $VSCRL
IF NB, .baddr
MOV .baddr, %100
ENDC
JSR %107, $VSCRL

.START Starts the display.

.START

.GLOBL $VSTRT
JSR %107, $VSTRT

.STAT Sets up status buffer.

.STAT .baddr

.GLOBL $VSTPM
IF NB, .baddr
MOV .baddr, %100
ENDC
JSR %107, $VSTPM

.STOP Stops the display.

.STOP

.GLOBL $VSTOP
JSR %107, $VSTOP

.SYNC Enables power line sync.

.SYNC

.GLOBL $SYNC
JSR %107, $SYNC

.TRACK Enables the track object.

.TRACK .baddr, .croutine

.GLOBL $VTRAK
IF NB, .baddr
MOV .baddr, %100
ENDC
IF NB, .croutine
MOV .croutine, -(%106)
IFF
CLR-(%106)
ENDC
NARG T
IF EQ, T
CLR %100
ENDC
JSR %107, $VTRAK

.UNLINK Unlinks display handler from RT-ll if linked, otherwise leaves display stopped.

.UNLINK

.GLOBL $VUNLK
JSR %107, $VUNLK

NOTE 1
.baddr Address of data buffer.

.faddr Address of start of user display file.

.croutine Address of .TRACK completion routine.
Display File Handler

NOTE 2

The lines preceded by a dot will not be assembled. The code they enclose may or may not be assembled depending on the conditionals.

N.7 DISPLAY PROCESSOR MNEMONICS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>100000</td>
<td>Character Mode</td>
</tr>
<tr>
<td>SHORTV</td>
<td>104000</td>
<td>Short Vector Mode</td>
</tr>
<tr>
<td>LONGV</td>
<td>110000</td>
<td>Long Vector Mode</td>
</tr>
<tr>
<td>POINT</td>
<td>114000</td>
<td>Point Mode</td>
</tr>
<tr>
<td>GRAPHX</td>
<td>120000</td>
<td>Graphplot X Mode</td>
</tr>
<tr>
<td>GRAPHY</td>
<td>124000</td>
<td>Graphplot Y Mode</td>
</tr>
<tr>
<td>RELATV</td>
<td>130000</td>
<td>Relative Point Mode</td>
</tr>
<tr>
<td>INTO</td>
<td>2000</td>
<td>Intensity 0 (Dimmest)</td>
</tr>
<tr>
<td>INT1</td>
<td>2200</td>
<td>Intensity 1</td>
</tr>
<tr>
<td>INT2</td>
<td>2400</td>
<td>Intensity 2</td>
</tr>
<tr>
<td>INT3</td>
<td>2600</td>
<td>Intensity 3</td>
</tr>
<tr>
<td>INT4</td>
<td>3000</td>
<td>Intensity 4</td>
</tr>
<tr>
<td>INT5</td>
<td>3200</td>
<td>Intensity 5</td>
</tr>
<tr>
<td>INT6</td>
<td>3400</td>
<td>Intensity 6</td>
</tr>
<tr>
<td>INT7</td>
<td>3600</td>
<td>Intensity 7 (Brightest)</td>
</tr>
<tr>
<td>LPOFF</td>
<td>100</td>
<td>Light Pen Off</td>
</tr>
<tr>
<td>LPON</td>
<td>140</td>
<td>Light Pen On</td>
</tr>
<tr>
<td>BLKOFF</td>
<td>20</td>
<td>Blink Off</td>
</tr>
<tr>
<td>BLKON</td>
<td>30</td>
<td>Blink On</td>
</tr>
<tr>
<td>LINE0</td>
<td>4</td>
<td>Solid Line</td>
</tr>
<tr>
<td>LINE1</td>
<td>5</td>
<td>Long Dash</td>
</tr>
<tr>
<td>LINE2</td>
<td>6</td>
<td>Short Dash</td>
</tr>
<tr>
<td>LINE3</td>
<td>7</td>
<td>Dot Dash</td>
</tr>
<tr>
<td>DJMP</td>
<td>160000</td>
<td>Display Jump</td>
</tr>
<tr>
<td>DNOP</td>
<td>164000</td>
<td>Display No Operation</td>
</tr>
<tr>
<td>STATSA</td>
<td>170000</td>
<td>Load Status A Instruction</td>
</tr>
<tr>
<td>LPLITE</td>
<td>200</td>
<td>Light Pen Hit On</td>
</tr>
<tr>
<td>LPDARK</td>
<td>300</td>
<td>Light Pen Hit Off</td>
</tr>
<tr>
<td>ITAL0</td>
<td>40</td>
<td>Italics Off</td>
</tr>
<tr>
<td>ITAL1</td>
<td>60</td>
<td>Italics On</td>
</tr>
<tr>
<td>SYNC</td>
<td>4</td>
<td>Halt and Resume in Sync</td>
</tr>
<tr>
<td>STATSB</td>
<td>174000</td>
<td>Load Status B Instruction</td>
</tr>
</tbody>
</table>
Display File Handler

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCR</td>
<td>100</td>
<td>Graphplot Increment</td>
</tr>
<tr>
<td>INTX</td>
<td>40000</td>
<td>Intensity Vector or Point</td>
</tr>
<tr>
<td>MAXX</td>
<td>1777</td>
<td>Maximum X Component</td>
</tr>
<tr>
<td>MAXY</td>
<td>1377</td>
<td>Maximum Y Component</td>
</tr>
<tr>
<td>MINUSX</td>
<td>20000</td>
<td>Negative X Component</td>
</tr>
<tr>
<td>MINUSY</td>
<td>20000</td>
<td>Negative Y Component</td>
</tr>
<tr>
<td>SHIFTX</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>MAXSX</td>
<td>17600</td>
<td>Maximum X Component</td>
</tr>
<tr>
<td>MAXSY</td>
<td>77</td>
<td>Maximum Y Component</td>
</tr>
<tr>
<td>MISVX</td>
<td>20000</td>
<td>Negative X Component</td>
</tr>
<tr>
<td>MISVY</td>
<td>100</td>
<td>Negative Y Component</td>
</tr>
</tbody>
</table>

N.8 ASSEMBLY INSTRUCTIONS

N.8.1 General Instructions

All programs can be assembled in 16K, using RT-ll MACRO. All assemblies and all links should be error free. The following conventions are assumed:

1. Default extensions are not explicitly typed. These are .MAC for source files, .OBJ for assembler output, and .SAV for Linker output.

2. The default device (DK) is used for all files in the example command strings.

3. Listings and link maps are not generated in the example command strings.

N.8.2 VTBASE

To assemble VTBASE with RT-ll link-up capability:

.R MACRO
*VTBASE=VTBASE
Display File Handler

N.8.3 VTCALL - VTCAL4

To assemble the modules VTCALL through VTCAL4:

```.
.R MACRO
*VTCALL1=VTCALL
*VTCAL2=VTCAL2
*VTCAL3=VTCAL3
*VTCAL4=VTCAL4
.
```

N.8.4 VTHDLR

To create the concatenated handler module:

```.
.R PIP
*VTHDLR.OBJ=VTCALL1.OBJ, VTCAL2.OBJ, VTCAL3.OBJ, VTCAL4.OBJ, VTBASE.OBJ/B
.
```

N.8.5 Building VTLIB.OBJ

To build the VTLIB library:

```.
.R LIBR
*VTLIB=VTHDLR.OBJ
.
```

N.9 VTMAC

```.
.NLIST

; VTMAC
; LIBRARY OF MACRO CALLS AND MNEMONIC DEFINITIONS
; FOR THE VT11 DEVICE SUPPORT PACKAGE

; DEC=11=OVTMA=LA

; COPYRIGHT (C) 1974
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS 01754

; MAY 1974

; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
; DEC ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT
; MAY APPEAR IN THIS DOCUMENT.

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

; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A
; LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND
; CAN BE COPIED (WITH INCLUSION OF DEC'S COPYRIGHT
; NOTICE) ONLY FOR USE IN SUCH A SYSTEM, EXCEPT AS MAY
; OTHERWISE BE PROVIDED IN WRITING BY DEC.

N-25
```
Display File Handler

; VTMAC IS A LIBRARY OF MACRO CALLS WHICH PROVIDE SUPPORT
; OF THE VT11 DISPLAY PROCESSOR. THE MACROS PRODUCE CALLS
; TO THE VT11 DEVICE SUPPORT PACKAGE, USING GLOBAL REFERENCES.

; MACRO TO GENERATE A MACRO WITH ZERO ARGUMENTS.

.Macro Macro Name,Call
  .Macro Name
  .Globl Call
  JsR %07,Call
  .EndM
  .EndM

; MACRO TO GENERATE A MACRO WITH ONE ARGUMENT

.Macro Macro1 Name,Call
  .Macro Name Arg
  .If Nb,Arg
  Mov Arg,%00
  .EndC
  .Globl Call
  JsR %07,Call
  .EndM
  .EndM

; MACRO TO GENERATE A MACRO WITH TWO OPTIONAL ARGUMENTS

.Macro Macro2 Name,Call
  .Macro Name Arg1,Arg2
  .Globl Call
  .If Nb,Arg1
  Mov Arg1,%00
  .EndC
  .If Nb,Arg2
  Mov Arg2,-(%06)
  .If
  Clr %06
  .NArg T
  .If Eq,T
  Clr %00
  .EndC
  .EndC
  JsR %07,Call
  .EndM
  .EndM

N-26
Display File Handler

; MACRO LIBRARY FOR VT11

MAC0  "CLEAR", "SVINIT"
MAC0  "STOP", "SVSTOP"
MAC0  "START", "SVSTRT"
MAC0  "SYNC", "SVSYNC"
MAC0  "NOSYN", "SVNOSYN"
MAC0  "UNLNK", "SVUNLNK"
MAC1  "INSRT", "SVNSRT"
MAC1  "REMOV", "SVRMOV"
MAC1  "BLANK", "SVBLNK"
MAC1  "RESTR", "SVRSTR"
MAC1  "STAT", "SVSTM"
MAC1  "LPEN", "SVL PEN"
MAC1  "SCROL", "SVSCRL"
MAC1  "NAME", "SVNAME"
MAC2  " TRACK", "SVTRAK"
MAC0  "LNKRT", "SVRTLK"

; MNEMONIC DEFINITIONS FOR THE VT11 DISPLAY PROCESSOR

DJMP=160000  ; DISPLAY JUMP
DNOP=164000  ; DISPLAY NOP
DJSR=173400  ; DISPLAY SUBROUTINE CALL
DRET=173400  ; DISPLAY SUBROUTINE RETURN
DNAM=173520  ; SET NAME REGISTER
DSTT=173420  ; RETURN STATUS DATA
DMAL=173500  ; STOP DISPLAY AND RETURN STATUS DATA

CHAR=100000  ; CHARACTER MODE
SHORTV=104000 ; SHORT VECTOR MODE
LONGV=110000 ; LONG VECTOR MODE
POINT=114000 ; POINT MODE
GRAPHX=120000 ; GRAPH X MODE
GRAPHY=124000 ; GRAPH Y MODE
RELATV=130000 ; RELATIVE VECTOR MODE

INT0=2000  ; INTENSITY 0
INT1=2200
INT2=2400
INT3=2600
INT4=3000
INT5=3200
INT6=3400
INT7=3600

LPDOFF=100 ; LIGHT PEN OFF
LPDON=140 ; LIGHT PEN ON
BLKOFF=20 ; BLINK OFF
BLKON=30 ; BLINK ON
LINE0=4 ; SOLID LINE
LINE1=5 ; LONG DASH
LINE2=6 ; SHORT DASH
LINE3=7 ; DOT DASH
Display File Handler

STATSA$=170000  ;LOAD STATUS REG A
LPLITE$=200  ;INTENSIFY ON LPEN HIT
LPDKR$=300  ;DON'T INTENSIFY
ITALO$=40  ;ITALICS OFF
ITALI$=60  ;ITALICS ON
SYNC$=4  ;POWER LINE SYNC

STATSB$=174000  ;LOAD STATUS REG B
INCR$=100  ;GRAPH PLOT INCREMENT
INTX$=40000  ;INTENSIFY VECTOR OR POINT
MAXX$=1777  ;MAXIMUM X INCR. = LONGV
MAXY$=1377  ;MAXIMUM Y INCR. = LONGV
MINUSX$=20000  ;NEGATIVE X INCREMENT
MINUSY$=20000  ;NEGATIVE Y INCREMENT
MAXSX$=17600  ;MAXIMUM X INCR. = SHORTV
MAXSY$=77  ;MAXIMUM Y INCR. = SHORTV
MISX$=28000  ;NEGATIVE X INCR. = SHORTV
MISY$=180  ;NEGATIVE Y INCR. = SHORTV

.LIST

N.10 EXAMPLES USING GTON

EXAMPLE #1  RT=11 MACRO VM22=06  9=aug-74 PAGE 4

 2  *TITLE EXAMPLE #1
 3  /
 4  */ THIS EXAMPLE USES THE *LPEN STATUS BUFFER AND THE
 5  */ NAME REGISTER TO MODIFY A DISPLAY FILE WITH THE LIGHT PEN.
 6  /
 7  00000  00000  R0$=0
 8  00001  00001  R1$=0
 9  00007  00007  FC=7
10  00044  00044  JSW=44
11  /J00 JOB STATUS WORD
12  
13  ;MCALL ;NTINR,;EXIT,;PRINT
14  START  ;LNOKRT   ;LINK TO MONITOR
15  00004  10004  BPL  1S  ;LINK UP ERROR?
16  00006  BPL  #EMSG  ;YES, PRINT MESSAGE
17  00014  BPL  #SBUF  ;AND EXIT.
18  00016  BPL  #MSG  ;ADJUST SCROLL
19  00020  BPL  #FILE  ;INSERT DISPLAY FILE
20  00034  BPL  #LPEN  ;SET UP LPEN BUFFER
21  00044  BPL  #BUF  ;SET JSW FOR TTINR
22  00054  55737  BST  15  ;LIGHT PEN HIT?
23  00070  BNE  1S  ;YES
24  00070  BNE  #TTINR  ;NO, ANY TT INPUT?
25  00070  BCC  EXIT  ;YES, EXIT
26  00074  BRC  LTST  ;NO, LOOP AGAIN
27  00075  16777  1S$  ;RESTORE PREVIOUS CODE
27  00074  MOVL  #12,#IPTR
27  00074  MOV  LBUF+2,R1
28  00104  016701  GET NAME VALUE

N-28
Display File Handler

```
28 00116 003301  DEC  R1  ;SUBTRACT ONE
30 00112 006301  ASL  R1  ;MULTIPLY BY TWO
31 00114 002701  ADD  PC,R1  ;USE TO INDEX
32 00116 002701  ADD  #DTABL=.+,R1  ;OFF TABLE DTABL.
32 000062
33 00122 000107  MOV  (R1),IPT  ;MOVE ADDR INTO IPT
33 000066
34 00126 001677  MOV  I1,.+PPK  ;MODIFY THAT CODE
34 000042
34 000022
35 00134 005007  CLR  LBUF  ;CLEAR BUFFER FLAG TO
35 000016
36 00140 000750  BR  LTST  ;ENABLE ANOTHER LP HIT,
38 00142 002700 EXITI  CMP  #12,K0  ;LOOP AGAIN
38 000012
39 00146 000145  BNE  LTST  ;LINE FEED?
40 00150  .UNLINK
41 00154  .EXIT
42 00158  LBUF1  .BLKW  7  ;ILEGN STATUS BUFFER
43 00174 103370  I11  ;WORD CHARINT51BLKONLPON
44 00176 103160  I12  ;WORD CHARINT41BLKOFFILPON
45 00200 000252 DTABL1  ;WORD D1,D2,D3  ;TABLE OF DISPLAY FILE
45 00202 000272
```

EXAMPLE #1

```
RT=11 MACRO VM02-06  9=AUG-74 PAGE 4+
45 00204 000312  ;LOCATIONS TO BE MODIFIED
46 47 00206 000252  ;PREVIOUS LOCATION MODIFIED
48 00210 00002  ;SCROLL LINE COUNT
49 00212 001000  ;WORD 1000
50 00214 0141  ;ASCIZ /ERROR/  ;ERROR MESSAGE
51 50 00215 105
50 00216 122
50 00217 122
50 00220 117
50 00221 122
50 00222 041
50 00223 000
51 .EVEN
52 00224 105  ;ASCIZ /EXAMPLE #1/  ;II.D. READER MESSAGE
52 00225 130
52 00226 101
52 00227 115
52 00230 120
52 00231 114
52 00232 103
52 00233 040
52 00234 043
52 00235 061
52 00236 000
53 .EVEN
54 55  ;DISPLAY FILE FOR EXAMPLE #1
55 57 00240 114000  ;FILENAME POINT
58 59 00242 000100  ;100
59 00244 000090  ;500
60 00246 173520  ;NAME
61 00250 000001  1
62 00252 103150  D1:  ;CHAR1BLKOFFINT41LPON

N-29
```
Display File Handler

63 00254  117  **ASCII /ONE/**
63 00255  116
63 00256  105
63 00257  056
64 00250  114000  POINT
65 00252  000190  100
65 00254  000300  300
67 00256  173520  DNAME
68 00257  000002  2
69 00252  163160  D21  CHARBLKOFF:INT41LPON
70 00254  124  **ASCII /TWO/**
70 00255  127
70 00256  117
70 00277  055
71 00300  114000  POINT
72 00302  000190  100
73 00304  000190  100
74 00306  173520  DNAME
75 00310  000003  3
76 00312  163160  D31  CHARBLKOFF:INT41LPON
77 00314  124  **ASCII /THREE/**
77 00315  110

EXAMPLE #1  RT-11 MACRO VM02=06  9=AUG-74 PAGE 4+
77 00316  122
77 00317  105
77 00320  105
77 00321  056
78 00322  173400  DRET
79 00324  000000  0
80 0000001  **END START**

EXAMPLE #1  RT-11 MACRO VM02=06  9=AUG-74 PAGE 4+
SYMBOL TABLE
BLKOFF  000020  BLKON  000030  CHAR  100000
DFILE  000240R  DMALT  173500  DJMP  100000
DJSR  173400  DNAME  173520  DNOP  164000
DRET  173400  DSTAT  173420  DTABLE 000200R
D1  000292R  D2  000272R  D3  000312R
DMGR  000214R  DEXIT  000142R  DMGRAPHX= 120000
INT0  002000  JCR  001000  INT7  003600
INT1  002200  INI  002200  INT5  003200
INT2  002400  INT4  003000  INT6  003400
INT5  002400  INT7  003600  INPTR  002050
INT6  003400  ITAL1  000050  I1  001740H
INT5  003000  ITAL0  000040  I2  000176R
LJMP  000044  JSH  000044  LBUE  000156H
LINE2  000005  LINE1  000005  LINE1  000005
LINE3  000007  LONGV  110000  LDPARK  000300
LPLITE  000200  LPOLFF  000100  LCP  000090
LST  000020R  MAXIMX  017600  MAXSY  000077
MAXX  001777  MAXY  001377  MINSX  002000
MINUS  020000  MISVX  020000  MISVY  001000
MS2  000010R  MSG  000000  PC  %000000
RELATV  130000  R0  %000000  RELATV  130000
SCBUP  000210R  SHORTV  104000  RELATV  130000
STATS  174000  START  000000H  S0  000000H
SVLEN  000210R  SVNSHT  001000  SVRLKL  000000H
SVSCH  000210R  SVNLK  001000  SVRNLK  000000H
ABS  000000  000  0000328  001
ERRORS DETECTED 0
FREE CORE: 15117, WORDS

MANex1,LP1=VTMAC,MANex1

N-30
Display File Handler

EXAMPLE #2

RT-11 MACRO VM02-06 9-AUG-74 PAGE 4

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
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

.TITLE EXAMPLE #2

; THIS EXAMPLE USES THE TRACKING OBJECT AND THE TRACK
; COMPLETION ROUTINE TO CAUSE A VECTOR TO FOLLOW
; THE LIGHT PEN FROM A SET POINT AT (500,500).

000000 R0=0
000001 R1=1
000005 SP=6
000007 PC=7

000000 START
000004 BPL 13
000006 .EXIT
000015 1S
00026 .EXIT
00042 JSR PC, WAIT
00046 UNLK
00052 .EXIT
00054 WAIT
00060 CMP #12, M0
00270 BNE WAIT
00373 BEQ INS
00030 TBUF
00032 MOV R0, (SP)
00167 MOV TBUF, R1
00766 SUB DX, R1
00052 BPL 1S
00040 NEG R1
00270 BIS #MINUSX, R1
00200 BIS #INTX, R1
00400 MOv R1,DX
00167 MOV TBUF+2, R1
00740 SUB OY, R1
00024 BPL 2S
00040 NEG R1
00270 BIS #MINUSX, R1
00200 BIS #MINUSX, R1
00167 MOV R1, OY

N-31
Display File Handler

EXAMPLE #2  RT-11 MACRO VM02=86  9=AUG-74 PAGE 4+

47 00152 012001  MOV (SP)+,R1  /RESTORE R1
48 00154 000207  RTS PC  /EXIT FROM COMPLETION ROUTINE
49
50 / DISPLAY FILE FOR EXAMPLE #2
51 /
52 00156 114000 0FILEI  POINT  /SET POINT AT
53 00158 000500 DXI  500
54 00158 000500 DYI  500  /(500,500)
55 00158 113000  LONGVINT4  /DRAW A VECTOR
56 00158 000000 DXI  *NURD 0
57 00170 000000 DYI  *WORD 0  /INITIALLY NOWHERE
58 00172 173400  URET  /DISPLAY FILE END
59 00174 000000 0
60 00176 123 EMSG1  ASCIZ /SORRY, THERE SEEMS TO BE A PROBLEM/
60 00177 117
60 00178 122
60 00179 122
60 00180 131
60 00181 054
60 00182 040
60 00183 124
60 00184 110
60 00185 105
60 00186 122
60 00187 103
60 00188 040
60 00189 123
60 00190 123
60 00191 119
60 00192 119
60 00193 119
60 00194 119
60 00195 119
60 00196 119
60 00197 119
60 00198 119
60 00199 119
60 00200 124
60 00201 117
60 00202 046
60 00203 124
60 00204 117
60 00205 046
60 00206 124
60 00207 124
60 00208 124
60 00209 046
60 00210 124
60 00211 124
60 00212 046
60 00213 124
60 00214 124
60 00215 124
60 00216 124
60 00217 124
60 00218 124
60 00219 046
60 00220 046
60 00221 101
60 00222 046
60 00223 101
60 00224 101
60 00225 101
60 00226 101
60 00227 101
60 00228 101
60 00229 101
60 00230 101
60 00231 101
60 00232 101
60 00233 101
60 00234 101
60 00235 101
60 00236 101
60 00237 101
60 00238 101
60 00239 101
60 00240 000
61 .EVEN

62 .END

START

N-32
Display File Handler

EXAMPLE #2

SYMBOL TABLE

<table>
<thead>
<tr>
<th>NAME</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKOFF</td>
<td>000002</td>
<td>BLKON</td>
<td>000030</td>
<td>CHAR</td>
<td>100000</td>
</tr>
<tr>
<td>DFILE</td>
<td>000156R</td>
<td>UHALT</td>
<td>173500</td>
<td>DJMP</td>
<td>160000</td>
</tr>
<tr>
<td>DJSR</td>
<td>173400</td>
<td>DNAME</td>
<td>173520</td>
<td>DNOP</td>
<td>164000</td>
</tr>
<tr>
<td>DRET</td>
<td>173400</td>
<td>USTAT</td>
<td>173420</td>
<td>DX</td>
<td>000165R</td>
</tr>
<tr>
<td>DY</td>
<td>000170R</td>
<td>EMSG</td>
<td>000176R</td>
<td>GRAPHX</td>
<td>120000</td>
</tr>
<tr>
<td>GRAPHX</td>
<td>124000</td>
<td>INCR</td>
<td>000100</td>
<td>INTX</td>
<td>040000</td>
</tr>
<tr>
<td>INT8</td>
<td>002000</td>
<td>INT1</td>
<td>002200</td>
<td>INT2</td>
<td>002400</td>
</tr>
<tr>
<td>INT3</td>
<td>002600</td>
<td>INT4</td>
<td>003000</td>
<td>INT5</td>
<td>003200</td>
</tr>
<tr>
<td>INT6</td>
<td>003400</td>
<td>INT7</td>
<td>003600</td>
<td>ITAL0</td>
<td>000040</td>
</tr>
<tr>
<td>ITAL1</td>
<td>00000</td>
<td>LINE2</td>
<td>000000</td>
<td>LINE1</td>
<td>000000</td>
</tr>
<tr>
<td>LINE2</td>
<td>000006</td>
<td>LINE3</td>
<td>000007</td>
<td>LINGV</td>
<td>100000</td>
</tr>
<tr>
<td>LDAK</td>
<td>000300</td>
<td>LPLITE</td>
<td>002000</td>
<td>LPFF</td>
<td>004100</td>
</tr>
<tr>
<td>LPON</td>
<td>000140</td>
<td>MAXSX</td>
<td>017500</td>
<td>MAXY</td>
<td>001377</td>
</tr>
<tr>
<td>MAXX</td>
<td>001777</td>
<td>MISVX</td>
<td>020000</td>
<td>MISVY</td>
<td>000100</td>
</tr>
<tr>
<td>MINUSX</td>
<td>020000</td>
<td>OY</td>
<td>000162R</td>
<td>PC</td>
<td>#000007</td>
</tr>
<tr>
<td>DXY</td>
<td>000160R</td>
<td>RELATV</td>
<td>130000</td>
<td>R0</td>
<td>#000000</td>
</tr>
<tr>
<td>R1</td>
<td>#000001</td>
<td>SHORTV</td>
<td>104000</td>
<td>SP</td>
<td>#000006</td>
</tr>
<tr>
<td>START</td>
<td>000000R</td>
<td>STATSA</td>
<td>178000</td>
<td>STATSB</td>
<td>174000</td>
</tr>
<tr>
<td>SYNC</td>
<td>000004</td>
<td>TBUF</td>
<td>000070R</td>
<td>TCUM</td>
<td>#00074R</td>
</tr>
<tr>
<td>WAIT</td>
<td>000004R</td>
<td>SVSRT</td>
<td>#000070R</td>
<td>$VRLK</td>
<td>#000000</td>
</tr>
<tr>
<td>SVTRAK</td>
<td>#00000G</td>
<td>ABS</td>
<td>#00070R</td>
<td>$VUNLK</td>
<td>#000000</td>
</tr>
<tr>
<td>ERRORS</td>
<td>DETECTED</td>
<td>FREE</td>
<td>CORE</td>
<td>15022,</td>
<td>WORDS</td>
</tr>
</tbody>
</table>

MANEX2, LPI=VTMAC, MANEX2
APPENDIX O
SYSTEM SUBROUTINE LIBRARY

0.1 INTRODUCTION

The RT-11 FORTRAN System Subroutine Library (SYSLIB) is a collection of FORTRAN-callable routines which allow a FORTRAN user to utilize various features of RT-11 Foreground/Background (F/B) and Single-Job (SJ) Monitors. SYSLIB also provides various utility functions, a complete character string manipulation package, and 2-word integer support. SYSLIB is provided as a library of object modules to be combined with FORTRAN programs at link-time.

The user of SYSLIB should be familiar with Chapters 1, 2, and 9 of this manual. This appendix assumes that FORTRAN users are familiar with the PDP-11 FORTRAN Language Reference Manual and the RT-11/RSTS/E FORTRAN IV User's Guide.

The following are some of the functions provided by SYSLIB.

- Complete RT-11 I/O facilities, including synchronous, asynchronous, and completion-driven modes of operation. FORTRAN subroutines may be activated upon completion of an input/output operation.
- Timed scheduling of asynchronous subjobs (completion routines) (F/B only).
- Complete facilities for interjob communication between Foreground and Background jobs (F/B only).
- FORTRAN interrupt service routines.
- Complete timer support facilities, including timed suspension of execution (F/B only), conversion of different time formats, and time-of-day information. These timer facilities support either 50- or 60-cycle clocks.
- All auxiliary input/output functions provided by RT-11, including the capabilities of opening, closing, renaming, creating, and deleting files from any device.
- All monitor-level informational functions, such as job partition parameters, device statistics, and input/output channel statistics.
- Access to the RT-11 Command String Interpreter (CSI) for acceptance and parsing of standard RT-11 command strings.
- A character string manipulation package supporting variable-length character strings.
- INTEGER*4 support routines that allow 2-word integer computations.

SYSLIB allows the FORTRAN user to write almost all application programs completely in FORTRAN with no assembly language coding.
System Subroutine Library

Assembly language programs may also utilize SYSLIB routines (see Section 0.1.3).

0.1.1 Conventions and Restrictions

In general, the SYSLIB routines were written for use with RT-ll V2B and FORTRAN IV V1B. The use of this SYSLIB package with prior versions of RT-ll or FORTRAN will lead to unpredictable results.

Programs using IPEEK, IPOKE, and/or ISPY to access FORTRAN, monitor, hardware, or other system specific addresses are not guaranteed to run under future releases or on different configurations. Suitable care should be taken with this type of coding to document precisely the use of these access functions and to check a referenced location's usage against the current documentation.

The following must be considered when coding a FORTRAN program that uses SYSLIB.

1. Various functions in the SYSLIB package return values that are of type integer, real, and double precision. If the user specifies an IMPLICIT statement that changes the defaults for external function typing, he must explicitly declare the type of those SYSLIB functions that return integer or real results. Double precision functions must always be declared to be type DOUBLE PRECISION (or REAL*8). Failure to observe this requirement will lead to unpredictable results.

2. All names of subprograms external to the routine being coded that are being passed to scheduling calls (such as ISCHED, ITIMER, IREADF, etc.) must be specified in an EXTERNAL statement in the FORTRAN program unit issuing the call.

3. Certain arguments (noted as such in the individual routine descriptions) to SYSLIB calls must be located in such a manner as to prohibit the RT-ll USR (User Service Routine) from swapping over them at execution time. This is accomplished by issuing a SET USR NO SWAP command before the job is entered, by using the /U switch at compile time, or by assuring that the routine argument is located in the memory image in such a way as to not be swapped over. The latter may be accomplished by changing the order of the object modules and libraries as specified to the linker. If the USR is swapping, it swaps over the first 2K of the program. The default case, for example, swaps the USR at 1000 octal to 11000 octal; arguments being passed to the USR must be kept out of this area. (See Section 0.1.3 for further information.)

4. Quoted-string literals are useful as arguments of calls to routines in the SYSLIB package, notably the character-string routines. These literals are allowed in subroutine calls (i.e., those invoked by the CALL statement), but they are explicitly prohibited in function calls (i.e., those invoked by the appearance of the function name followed by an argument list in an expression). See Sections 0.1.2 and 0.2.4 for further information.

5. Certain restrictions apply to completion or interrupt routines; see Section 0.2.1 for these restrictions.
System Subroutine Library

0.1.2 Calling SYSLIB Subprograms

SYSLIB subprograms are called in the same manner as user-written subroutines. SYSLIB includes both FUNCTION subprograms and SUBROUTINE subprograms. FUNCTION subprograms receive control by means of a function reference, represented in this appendix as:

\[ \text{i = function name \{(arguments)\}} \]

SUBROUTINE subprograms are invoked by means of a CALL statement, i.e.,

\[ \text{CALL subroutine name \{(arguments)\}} \]

All routines in SYSLIB may be called as FUNCTION subprograms if the return value is desired, or as SUBROUTINE subprograms if no return value is desired. For example, the LOCK subroutine can be referenced as either:

\[ \text{CALL LOCK} \]

or

\[ \text{I = LOCK()} \]

Note that routines that do not explicitly return function results will produce meaningless values if referenced as functions. In the following descriptions, the more common usage (function or subroutine) is shown.

0.1.3 Using SYSLIB with MACRO

The calling sequence is standard for all subroutines, including user-written FORTRAN subprograms and assembly language subprograms. SYSLIB routines may be used with MACRO programs by passing control to the SYSLIB routine with the following instruction:

\[ \text{JSR \ PC,routine} \]

Register five points to an argument list having the following format:
System Subroutine Library

Control is returned to the calling program by use of the instruction:

```
RTS   PC
```

The following is an example of calling a SYSLIB function from an assembly language routine.

```
.GLOBL JMUL ;GLOBAL FOR JMUL

,...

MOV #LIST,R5 ;POINT R5 TO ARG LIST
JSR PC,JMUL ;CALL JMUL
CMP #-2,R0 ;CHECK FOR OVERFLOW
BEQ OVRFL ;BRANCH IF ERROR

,...

LIST: .WORD 3 ;ARG LIST,3 ARGS
.WORD OPR1 ;ADDR OF 1ST ARG
.WORD OPR2 ;ADDR OF 2ND ARG
.WORD RESULT ;ADDR OF 3RD ARG
OPR1: .WORD 100 ;LOW-ORDER VALUE OF 1ST ARG
.WORD 0 ;HIGH-ORDER VALUE OF 1ST ARG
OPR2: .WORD 10 ;LOW-ORDER VALUE OF 2ND ARG
.WORD 10 ;HIGH-ORDER VALUE OF 2ND ARG
RESULT: .BLKW 2 ;2-WORD RESULT (LOW ORDER, HIGH ORDER)
.END
```

The following routines can be used only with FORTRAN:

GETSTR
IASIGN
ICDFN
IFETCH
IGETC
ILUN
INTSET
IQSET
IRCVD
IREADF
ISCHED
ISDATF
ISPFNF
ITIMER
IWRITF
PUTSTR
SECNDS

User-written assembly language programs that call SYSLIB subprograms must preserve any pertinent registers before calling the SYSLIB routine and restore the registers, if necessary, upon return.

Function subprograms return a single result in the registers. The register assignments for returning the different variable types are:

Integer, Logical functions - result in R0

0-4
System Subroutine Library

Real functions - high-order result in R0, low-order result in R1

Double Precision functions - result in R0-R3, lowest order result in R3

Complex functions - high-order real result in R0, low-order real result in R1, high-order imaginary result in R2, low-order imaginary result in R3

User-written assembly language routines which interface to the FORTRAN Object Time System (OTS) must be aware of the location of the RT-11 USR (User Service Routine). If a user routine requests a USR function (e.g., ENTER, LOOKUP), or if the USR is invoked by the FORTRAN OTS, the USR will be swapped into memory if it is nonresident. The FORTRAN OTS is designed so that the USR can swap over it. User routines must be written to allow the USR to swap over them or must be located outside the region of memory into which the USR will swap.

User interrupt service routines and completion routines, because of their asynchronous nature, must be further restricted to be located where the USR will not swap. The USR (if in a swapping state) will always swap over the area of memory that starts at the program initial stack pointer address; the USR occupies 2K words. Interrupt and completion routines (and their data areas) must not be located in this area. The best way to accomplish this is to examine the link map, determine whether the USR will swap over an assembly language or FORTRAN asynchronous routine, and, if so, change the order of object modules and libraries as specified to the linker. Continue this process until a suitable arrangement is obtained.

To remove these restrictions, the user must make the USR resident either by specifying the /U switch to the FORTRAN compiler (when compiling a program to be run in the background of P/B or under S-U) or by issuing the SET USR NOSWAP command before executing the program.

0.1.4 Running a FORTRAN Program in the Foreground

The FRUN monitor command must be modified to include various SYSLIB functions. Section G.1 explains the formula used to allocate the needed space when running a FORTRAN program as a foreground job. This formula:

\[ x = \{1/2\{378+(29*N)+(R-136)+A*512)\}\]  

must be modified for SYSLIB functions as follows:

The IQSET function requires the formula to include additional space for queue elements (qleng) to be added to the queue:

\[ x = \{1/2\{378+(29*N)+(R-136)+A*512)\}\} + \{7*qleng\} \]

The ICDFN function requires the formula to include additional space for the integer number of channels (num) to be allocated:

\[ x = \{1/2\{378+(29*N)+(R-136)+A*512)\}\} + \{6*num\} \]

The INTSET function requires the formula to include additional space for the number of INTSET calls (INTSET) issued in the program.
System Subroutine Library

\[ x = \frac{1}{2} \left( 378 + (29\star N) + (R-136) + A \star 512 \right) + 25 \star \text{INTSET} \]

Any SYSLIB calls, including INTSET, that invoke completion routines must include 64(10) words plus the number of words needed to allocate the second record buffer (default is 68(10) words). The length of the record buffer is controlled by the /R switch to the FORTRAN compiler. If the /R switch is not used, the allocation in the formula must be 132(10) words.

\[ x = \frac{1}{2} \left( 378 + (29\star N) + (R-136) + A \star 512 \right) + (64+R/2) \]

If the /N option does not allocate enough space in the foreground on the initial call to a completion routine, the following message appears:

?ERR 0, NON-FORTRAN ERROR CALL

This message also appears if there is not enough free memory for the background job or if a completion routine in the Single-Job monitor is activated during another completion routine. In the latter case, the job aborts. F/B should be used for multiple active completion routines.

0.1.5 Linking with SYSLIB

SYSLIB is provided on the distribution media as a file of concatenated object modules (SYSF4.OBJ). If this file is linked directly with the FORTRAN program, all SYSLIB modules will be included whether they are used or not. For example:

```
.R LINK
*PROG=PROG,SYSF4/F
```

A library can be created by using the librarian to transform SYSF4 into a library file (SYSLIB) as follows:

```
.R LIBR
*SYSLIB=SYSF4
```

When a library is used, only the modules called will be linked with the program. For example:

```
.R LINK
*PROG=PROG,SYSLIB/F
```

The following example links the object module EXAMPL.OBJ into a single save image file EXAMPL.SAV and produces a load map file on LP:. SYSLIB and the default FORTRAN library (FORLIB.OBJ) are searched for any routines that are not found in other object modules.

```
.R LINK
*EXAMPL,LP:=EXAMPL,SYSLIB/F
```

If the FORTRAN library is explicitly specified in the command string, SYSLIB must precede it, i.e.,

```
.R LINK
*TEST=TEST,SYSLIB,FPOOLIB
```
System Subroutine Library

is an acceptable command string, but the following is not:

\*TEST\*TEST,PPULIB,SYSLIB

0.2 TYPES OF SYSLIB SERVICES

Ten types of services are available to the user through SYSLIB. These are:

1. File-Oriented Operations
2. Data Transfer Operations
3. Channel-Oriented Operations
4. Device and File Specifications
5. Timer Support Operations
6. RT-ll Service Operations
7. INTEGER*4 Support Functions
8. Character String Functions
9. Radix-50 Conversion Operations
10. Miscellaneous Services

Table 0-1 alphabetically summarizes the SYSLIB subprograms in each of these categories. Those marked with an asterisk (*) are allowed only in a Foreground/Background environment.

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-Oriented Operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLOSEC</td>
<td>0.3.3</td>
<td>Closes the specified channel.</td>
</tr>
<tr>
<td>IDELET</td>
<td>0.3.19</td>
<td>Deletes a file from the specified device.</td>
</tr>
<tr>
<td>IENTER</td>
<td>0.3.22</td>
<td>Creates a new file for output.</td>
</tr>
<tr>
<td>IRENAME</td>
<td>0.3.37</td>
<td>Changes the name of the indicated file to a new name.</td>
</tr>
<tr>
<td>LOOKUP</td>
<td>0.3.66</td>
<td>Opens an existing file for input and/or output via the specified channel.</td>
</tr>
<tr>
<td>Data Transfer Functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*IRCVD</td>
<td>0.3.35</td>
<td>Receives data. Allows a job to read messages or data sent by another job in an F/B environment. The four modes correspond to the IREAD, IREADC, IREADF, and IREADW modes.</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 0-1 (cont.)
#### Summary of SYSLIB Subprograms

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Transfer Functions (cont.)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IREAD</strong></td>
<td>0.3.36</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.</td>
</tr>
<tr>
<td><strong>IREADC</strong></td>
<td>0.3.36</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers to the assembly language routine specified in the IREADC function call.</td>
</tr>
<tr>
<td><strong>IREADF</strong></td>
<td>0.3.36</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers to the FORTRAN subroutine specified in the IREADF function call.</td>
</tr>
<tr>
<td><strong>IREADW</strong></td>
<td>0.3.36</td>
<td>Transfers data via the specified channel to a memory buffer and returns control to the program only after the transfer is complete.</td>
</tr>
<tr>
<td><strong>ISDAT</strong></td>
<td>0.3.41</td>
<td>Allows the user to send messages or data to the other job in an F/B environment. The four modes correspond to the IWRITE, IWRITC, IWRITF, and IWRITW modes.</td>
</tr>
<tr>
<td><strong>ISDATC</strong></td>
<td>0.3.41</td>
<td></td>
</tr>
<tr>
<td><strong>ISDATF</strong></td>
<td>0.3.41</td>
<td></td>
</tr>
<tr>
<td><strong>ISDATW</strong></td>
<td>0.3.41</td>
<td></td>
</tr>
<tr>
<td><strong>ITTINR</strong></td>
<td>0.3.47</td>
<td>Inputs one character from the console keyboard.</td>
</tr>
<tr>
<td><strong>ITTOUR</strong></td>
<td>0.3.48</td>
<td>Transfers one character to the console terminal.</td>
</tr>
<tr>
<td><strong>IWAIT</strong></td>
<td>0.3.51</td>
<td>Waits for completion of all I/O on a specified channel. (Commonly used with the IREAD and IWRITE functions.)</td>
</tr>
<tr>
<td><strong>IWRITC</strong></td>
<td>0.3.52</td>
<td>Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers to the assembly language routine specified in the IWRITC function call.</td>
</tr>
</tbody>
</table>

(continued on next page)
### System Subroutine Library

#### Table O-1 (cont.)
Summary of SYSLIB Subprograms

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Transfer Functions (cont.)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWRITE</td>
<td>0.3.52</td>
<td>Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of the I/O.</td>
</tr>
<tr>
<td>IWRITF</td>
<td>0.3.52</td>
<td>Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers to the FORTRAN subroutine specified in the IWRITF function call.</td>
</tr>
<tr>
<td>IWRITW</td>
<td>0.3.52</td>
<td>Transfers data via the specified channel to a device and returns control to the user program only after the transfer is complete.</td>
</tr>
<tr>
<td>*MWAIT</td>
<td>0.3.68</td>
<td>Waits for messages to be processed.</td>
</tr>
<tr>
<td>PRINT</td>
<td>0.3.69</td>
<td>Outputs an ASCII string to the console terminal.</td>
</tr>
<tr>
<td><strong>Channel-Oriented Operations</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICFDN</td>
<td>0.3.14</td>
<td>Defines additional I/O channels.</td>
</tr>
<tr>
<td>*ICHCPY</td>
<td>0.3.15</td>
<td>Allows access to files currently open in the other job's environment.</td>
</tr>
<tr>
<td>*ICSTAT</td>
<td>0.3.18</td>
<td>Returns the status of a specified channel.</td>
</tr>
<tr>
<td>IFREEC</td>
<td>0.3.24</td>
<td>Returns the specified RT-ll channel to the available pool of channels for the FORTRAN I/O system.</td>
</tr>
<tr>
<td>IGETC</td>
<td>0.3.25</td>
<td>Allocates an RT-ll channel and marks it in use to the FORTRAN I/O system.</td>
</tr>
<tr>
<td>ILUN</td>
<td>0.3.27</td>
<td>Returns the RT-ll channel number with which a FORTRAN logical unit is associated.</td>
</tr>
<tr>
<td>IREOPN</td>
<td>0.3.38</td>
<td>Restores the parameters stored via an ISAVES function and reopens the channel for I/O.</td>
</tr>
<tr>
<td>ISAVES</td>
<td>0.3.39</td>
<td>Stores five words of channel status information into a user-specified array.</td>
</tr>
<tr>
<td>PURGE</td>
<td>0.3.70</td>
<td>Deactivates a channel.</td>
</tr>
</tbody>
</table>

(continued on next page)
## Table O-1 (cont.)
Summary of SYSLIB Subprograms

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device and File Specifications</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IASIGN</td>
<td>0.3.13</td>
<td>Sets information in the FORTRAN logical unit table.</td>
</tr>
<tr>
<td>ICSI</td>
<td>0.3.17</td>
<td>Calls the RT-ll CSI in special mode to decode file specifications and switches.</td>
</tr>
<tr>
<td><strong>Timer Support Operations</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVTTIM</td>
<td>0.3.5</td>
<td>Converts a 2-word internal format time to hours, minutes, seconds, and ticks.</td>
</tr>
<tr>
<td>GTIM</td>
<td>0.3.9</td>
<td>Gets time of day.</td>
</tr>
<tr>
<td>*ICMKT</td>
<td>0.3.16</td>
<td>Cancels an unexpired ISCHED, ITIMER, or MRKT request.</td>
</tr>
<tr>
<td>*ISCHED</td>
<td>0.3.40</td>
<td>Schedules the specified FORTRAN subroutine to be entered at the specified time of day as an asynchronous completion routine.</td>
</tr>
<tr>
<td>*ISLEEP</td>
<td>0.3.42</td>
<td>Suspends main program execution of the running job for a specified amount of time; completion routines continue to run.</td>
</tr>
<tr>
<td>*ITIMER</td>
<td>0.3.45</td>
<td>Schedules the specified FORTRAN subroutine to be entered as an asynchronous completion routine when the time interval specified has elapsed.</td>
</tr>
<tr>
<td>*ITWAIT</td>
<td>0.3.49</td>
<td>Suspends the running job for a specified amount of time; completion routines continue to run.</td>
</tr>
<tr>
<td>*IUNTL</td>
<td>0.3.50</td>
<td>Suspends the main program execution of the running job until a specified time-of-day; completion routines continue to run.</td>
</tr>
<tr>
<td>JTIME</td>
<td>0.3.63</td>
<td>Converts hours, minutes, seconds, and ticks into 2-word internal format time.</td>
</tr>
<tr>
<td>*MRKT</td>
<td>0.3.67</td>
<td>Marks time, i.e., schedules an assembly language routine to be activated as an asynchronous completion routine after a specified interval.</td>
</tr>
</tbody>
</table>

(continued on next page)
### System Subroutine Library

#### Table O-1 (cont.)

**Summary of SYSLIB Subprograms**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timer Support Operations (cont.)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECONDS</td>
<td>0.3.80</td>
<td>Returns the current system time in seconds past midnight minus the value of a specified argument.</td>
</tr>
<tr>
<td>TIMASC</td>
<td>0.3.84</td>
<td>Converts a specified 2-word internal format time into an 8-character ASCII string.</td>
</tr>
<tr>
<td>TIME</td>
<td>0.3.85</td>
<td>Returns the current system time-of-day as an 8-character ASCII string.</td>
</tr>
<tr>
<td><strong>RT-11 Services</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAIN</td>
<td>0.3.2</td>
<td>Chains to another program (in the background job only).</td>
</tr>
<tr>
<td>*DEVICE</td>
<td>0.3.6</td>
<td>Specifies actions to be taken on normal or abnormal program termination, such as turning off interrupt enable on foreign devices, etc.</td>
</tr>
<tr>
<td>GTJB</td>
<td>0.3.10</td>
<td>Returns the parameters of this job.</td>
</tr>
<tr>
<td>IDSTAT</td>
<td>0.3.21</td>
<td>Returns the status of the specified device.</td>
</tr>
<tr>
<td>IFETCH</td>
<td>0.3.23</td>
<td>Loads a device handler into memory.</td>
</tr>
<tr>
<td>IQSET</td>
<td>0.3.33</td>
<td>Expands the size of the RT-11 monitor queue from the free space managed by the FORTRAN system.</td>
</tr>
<tr>
<td>ISPFN</td>
<td>0.3.43</td>
<td>Issues special function requests to various handlers, e.g., magtape. The four modes correspond to the IWRITE, IWRITC, IWRTIF, and IWRTNW modes.</td>
</tr>
<tr>
<td>ISPFNC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISPFNF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISPFNW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*ITLOCK</td>
<td>0.3.46</td>
<td>Indicates whether the USR is currently in use by another job and performs a LOCK if the USR is available.</td>
</tr>
<tr>
<td>LOCK</td>
<td>0.3.65</td>
<td>Makes the RT-11 monitor User Service Routine (USR) permanently resident until an UNLOCK function is executed. A portion of the user's program is swapped out to make room for the USR if necessary.</td>
</tr>
<tr>
<td>RCHAIN</td>
<td>0.3.74</td>
<td>Allows a program to access variables passed across a chain.</td>
</tr>
</tbody>
</table>

(continued on next page)
System Subroutine Library

Table 0-1 (cont.)
Summary of SYSLIB Subprograms

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RT-11 Services (cont.)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCTRL0</td>
<td>0.3.75</td>
<td>Enables output to the terminal by cancelling the effect of a previously typed CTRL 0, if any.</td>
</tr>
<tr>
<td>*RESUME</td>
<td>0.3.77</td>
<td>Causes the main program execution of a job to resume where it was suspended by a SUSPND function call.</td>
</tr>
<tr>
<td>*SUSPND</td>
<td>0.3.83</td>
<td>Suspends main program execution of the running job; completion routines continue to execute.</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>0.3.88</td>
<td>Releases the USR if a LOCK was performed; the user program is swapped in if required.</td>
</tr>
<tr>
<td><strong>INTEGER*4 Support Functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AJFLT</td>
<td>0.3.1</td>
<td>Converts a specified INTEGER<em>4 value to REAL</em>4 and returns the result as the function value.</td>
</tr>
<tr>
<td>DJFLT</td>
<td>0.3.7</td>
<td>Converts a specified INTEGER<em>4 value to REAL</em>8 and returns the result as the function value.</td>
</tr>
<tr>
<td>IAJFLT</td>
<td>0.3.12</td>
<td>Converts a specified INTEGER<em>4 value to REAL</em>4 and stores the result.</td>
</tr>
<tr>
<td>IDJFLT</td>
<td>0.3.20</td>
<td>Converts a specified INTEGER<em>4 value to REAL</em>8 and stores the result.</td>
</tr>
<tr>
<td>IJCVT</td>
<td>0.3.26</td>
<td>Converts a specified INTEGER<em>4 value to INTEGER</em>2.</td>
</tr>
<tr>
<td>JADD</td>
<td>0.3.53</td>
<td>Computes the sum of two INTEGER*4 values.</td>
</tr>
<tr>
<td>JAFIX</td>
<td>0.3.54</td>
<td>Converts a REAL<em>4 value to INTEGER</em>4.</td>
</tr>
<tr>
<td>JCMP</td>
<td>0.3.55</td>
<td>Compares two INTEGER<em>4 values and returns an INTEGER</em>2 value that reflects the signed comparison result.</td>
</tr>
<tr>
<td>JDFIX</td>
<td>0.3.56</td>
<td>Converts a REAL<em>8 value to INTEGER</em>4.</td>
</tr>
<tr>
<td>JDIV</td>
<td>0.3.57</td>
<td>Computes the quotient and remainder of two INTEGER*4 values.</td>
</tr>
<tr>
<td>JICVT</td>
<td>0.3.58</td>
<td>Converts an INTEGER<em>2 value to INTEGER</em>4.</td>
</tr>
</tbody>
</table>

(continued on next page)
System Subroutine Library

Table O-1 (cont.)
Summary of SYSLIB Subprograms

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER*4 Support Functions (cont.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JJCVT</td>
<td>0.3.59</td>
<td>Converts 2-word internal time format to INTEGER*4 format, and vice versa.</td>
</tr>
<tr>
<td>JMOV</td>
<td>0.3.60</td>
<td>Assigns an INTEGER*4 value to a variable.</td>
</tr>
<tr>
<td>JMUL</td>
<td>0.3.61</td>
<td>Computes the product of two INTEGER*4 values.</td>
</tr>
<tr>
<td>JSUB</td>
<td>0.3.62</td>
<td>Computes the difference between two INTEGER*4 values.</td>
</tr>
<tr>
<td>Character String Functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONCAT</td>
<td>0.3.4</td>
<td>Concatenates two variable-length strings.</td>
</tr>
<tr>
<td>GETSTR</td>
<td>0.3.8</td>
<td>Reads a character string from a specified FORTRAN logical unit.</td>
</tr>
<tr>
<td>INDEX</td>
<td>0.3.28</td>
<td>Returns the location in one string of the first occurrence of another string.</td>
</tr>
<tr>
<td>INSERT</td>
<td>0.3.29</td>
<td>Replaces a portion of one string with another string.</td>
</tr>
<tr>
<td>ISCOMP</td>
<td>0.3.78</td>
<td>Compares two character strings.</td>
</tr>
<tr>
<td>IVERIF</td>
<td>0.3.89</td>
<td>Indicates whether characters in one string appear in another.</td>
</tr>
<tr>
<td>LEN</td>
<td>0.3.64</td>
<td>Returns the number of characters in a specified string.</td>
</tr>
<tr>
<td>PUTSTR</td>
<td>0.3.71</td>
<td>Writes a variable-length character string on a specified FORTRAN logical unit.</td>
</tr>
<tr>
<td>REPEAT</td>
<td>0.3.76</td>
<td>Concatenates a specified string with itself to provide an indicated number of copies and stores the resultant string.</td>
</tr>
<tr>
<td>SCOMP</td>
<td>0.3.78</td>
<td>Compares two character strings.</td>
</tr>
<tr>
<td>SCOOPY</td>
<td>0.3.79</td>
<td>Copies a character string from one array to another.</td>
</tr>
<tr>
<td>STRPAD</td>
<td>0.3.81</td>
<td>Pads a variable-length string on the right with blanks to create a new string of a specified length.</td>
</tr>
</tbody>
</table>

(concluded on next page)
## System Subroutine Library

### Table O-1 (cont.)
Summary of SYSLIB Subprograms

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Section</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character String Functions (cont.)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBSTR</td>
<td>0.3.82</td>
<td>Copies a substring from a specified string.</td>
</tr>
<tr>
<td>TRANSLN</td>
<td>0.3.86</td>
<td>Replaces one string with another after performing character modification.</td>
</tr>
<tr>
<td>TRIM</td>
<td>0.3.87</td>
<td>Removes trailing blanks from a character string.</td>
</tr>
<tr>
<td>VERIFY</td>
<td>0.3.89</td>
<td>Indicates whether characters in one string appear in another.</td>
</tr>
</tbody>
</table>

| **Radix-50 Conversion Operations** | | |
| IRAD50 | 0.3.34 | Converts ASCII characters to Radix-50, returning the number of characters converted. |
| R50ASC | 0.3.72 | Converts Radix-50 characters to ASCII. |
| RAD50 | 0.3.73 | Converts six ASCII characters, returning a REAL*4 result which is the 2-word Radix-50 value. |

| **Miscellaneous Services** | | |
| IADDR | 0.3.11 | Obtains the memory address of a specified entity. |
| INTSET | 0.3.30 | Establishes a specified FORTRAN subroutine as an interrupt service routine at a specified priority. |
| IPEEK | 0.3.31 | Returns the value of a word located at a specified absolute memory address. |
| IPOKE | 0.3.32 | Stores an integer value in an absolute memory location. |
| ISPY | 0.3.44 | Returns the integer value of the word located at a specified offset from the beginning of the RT-11 resident monitor. |

Routines requiring the USR (see Section 9.2.5) differ between the Single-Job and F/B Monitors. The following functions require the use of the USR:

- CLOSEC
- GETSTR (only if first I/O operation on logical unit)
- ICFN (Single-Job only)
System Subroutine Library

ICS
IDLET
IDSTAT
IENTER
IFETCH
IQSET
IRENAM
ITLOCK (only if USR is not in use by the other job)
LOCK (only if USR is in a swapping state)
LOOKUP
PUTSTR (only if first I/O operation on logical unit)

Certain requests require a queue element taken from the same list as the I/O queue elements. These are:

IRCVD/IRCVD/C/IRCVDF/IRCVDW
IREAD/IREADC/IREADP/IREADW
ISCHED
ISDAT/ISDATC/ISDATP/ISDATW
ISLEEP
ISPPN/ISPFNC/ISPFNF/ISPFNW
ITIMER
ITWAIT
IUNITL
IWRITC/IWRITE/IWRITF/IWRITW
MRKT
MWAIT

0.2.1 Completion Routines

Completion routines are subprograms that execute asynchronously with a main program. A completion routine is scheduled to run as soon as possible after the event for which it has been waiting has completed (e.g., the completion of an I/O transfer, or the lapsing of a specified time interval). All completion routines of the current job have higher priority than other parts of the job; therefore, once a completion routine becomes runnable because of its associated event, it interrupts execution of the job and continues to execute until it relinquishes control.

Completion routines are handled differently in the Single-Job and the F/B versions of RT-11. In the Single-Job version, completion routines are totally asynchronous and can interrupt one another. In F/B, completion routines do not interrupt each other but are queued and made to wait until the correct job is running. (For further information on completion routines, see Sections 9.2.8 and 0.1.4.)

A FORTRAN completion routine can have a maximum of two arguments:

SUBROUTINE crtn ((iarg1,iarg2))

where: iarg1 is equivalent to R0 on entry to an assembly language completion routine.

    iarg2 is equivalent to R1 on entry to an assembly language completion routine.

If an error occurs in a completion routine or in a subroutine at completion level, the error handler will trace back normally through
System Subroutine Library

to the original interruption of the main program. Thus the traceback is shown as though the completion routine were called from the main program and lets the user know where the main program was executing if a fatal error occurs.

Certain restrictions apply to completion routines, i.e., those routines that are activated by the following calls:

- INTSET
- IRCVDC
- IRCVDF
- IREADC
- IREADF
- ISCHED
- ISDATC
- ISDATF
- ISPPNC
- ISPPNF
- ITIMER
- IWRITC
- IWRITF
- MRKT

These restrictions are:

1. The first subroutine call that references a FORTRAN completion routine must be issued from the main program.

2. No channels may be allocated (by calls to IGETC) or freed (by calls to IFREEC) from a completion routine. Channels to be used by completion routines should be allocated and placed in a COMMON block for use by the routine.

3. The completion routine may not perform any call which requires the use of the USR, e.g., LOOKUP and IENTER. See Section 0.2 for a list of SYSLIB functions that call the USR.

4. Files to be operated upon in completion routines must be opened and closed by the main program. There are, however, no restrictions on the input or output operations that may be performed in the completion routine. If many files must be made available to the completion routine, they may be opened by the main program and saved for later use (without tying up RT-ll channels) by the ISAVES call. The completion routine may later make them available by reattaching the file to a channel with an IREOPN call.

5. FORTRAN subprograms are reusable but not reentrant. A given subprogram can be used many times as a completion routine or as a routine in the main program but a subprogram executing as main program code will not work properly if interrupted at the completion level. This restriction applies to all subprograms that can be invoked at the completion level and can be active at the same time in the main program.

6. Only one completion function should be active at any time under the Single-Job Monitor (see Section 0.1.4).

7. Assembly language completion routines must be exited via an RTS PC.
8. FORTRAN completion routines must be exited by execution of a RETURN or END statement in the subroutine.

0.2.2 Channel-Oriented Operations

An RT-ll channel being used for input/output with SYSLIB must be allocated in one of the following two ways:

1. The channel is allocated and marked in use to the FORTRAN I/O system by a call to IGETC and is later freed by a call to IFREEC.

2. An ICDFN call is issued to define more channels (up to 256). All channels numbered greater than 17 (octal) can be freely used by the programmer; the FORTRAN I/O system uses only channels 0 through 17 (octal).

Channels must be allocated in the main program routine or its subprograms, not in routines that are activated as the result of I/O completion events or ISCHED or ITIMER calls.

0.2.3 INTEGER*4 Support Functions

INTEGER*4 variables are allocated two words of storage. INTEGER*4 values are stored in two's complement representation. The first word (lower address) contains the low-order part of the value, and the second word (higher address) contains the sign and the high-order part of the value. The range of numbers supported is $-2^{31} + 1$ to $2^{31} - 1$.

Note that this format differs from the 2-word internal time format which stores the high-order part of the value in the first word and the low-order part in the second. The JJCVT function (Section 0.3.59) is provided for conversion between the two internal formats.

Integer and real arguments to subprograms are indicated in the following manner in this appendix.

\[
\begin{align*}
    i & = \text{INTEGER*2 arguments} \\
    j & = \text{INTEGER*4 arguments} \\
    a & = \text{REAL*4 arguments} \\
    d & = \text{REAL*8 arguments}
\end{align*}
\]

When the DATA statement is used to initialize INTEGER*4 variables, it must specify both the low- and high-order parts, i.e.,

\[
\begin{align*}
    \text{INTEGER*4 J} \\
    \text{DATA J/3/}
\end{align*}
\]

only initializes the first word.

The correct way to initialize an INTEGER*4 variable to a constant (e.g., 3) is shown below:

\[
\begin{align*}
    \text{INTEGER*4 J} \\
    \text{INTEGER*2 I(2)} \\
    \text{EQUIVALENCE (J,I)} \\
    \text{DATA I/3,0/} \\
    \text{!INITIALIZE J TO 3}
\end{align*}
\]
System Subroutine Library

If initializing an INTEGER*4 variable to a negative value (e.g., -4), the high-order (second word) part must be the continuation of the two's complement of the low-order part. For example:

```
INTEGER*4 J
INTEGER*2 I(2)
EQUIVALENCE (J,I)
DATA -4,-1/ !INITIALIZE J TO -4
```

The following form is suitable for INTEGER*4 arguments to subprograms:

```
INTEGER*2 J(2)
DATA J/3,0/ !LOW-ORDER,HIGH-ORDER
```

0.2.4 Character String Functions

The SYSLIB character string functions and routines provide variable-length string support for RT-11 FORTRAN. SYSLIB calls are provided to perform the following character string operations:

- Read character strings from a specified FORTRAN logical unit (GETSTR).
- Write character strings to a specified FORTRAN logical unit (PUTSTR).
- Concatenate variable-length strings (CONCAT).
- Return the position of one string in another (INDEX).
- Insert one string into another (INSERT).
- Return the length of a string (LEN).
- Repeat a character string (REPSET).
- Compare two strings (SCOMP).
- Copy a character string (SCOPY).
- Pad a string with rightmost blanks (STRPAD).
- Copy a substring from a string (SUBSTR).
- Perform character modification (TRANSL).
- Remove trailing blanks (TRIM).
- Verify the presence of characters in a string (VERIFY).

Strings are stored in LOGICAL*1 arrays which are defined and dimensioned by the FORTRAN programmer. Strings are stored in these arrays as one character per array element plus a zero element to indicate the current end of the string (ASCII format).

The length of a string may vary at execution time, ranging from zero characters in length to one less than the size of the array which stores the string. The maximum size of any string is 32767 characters. Strings may contain any of the 7-bit ASCII characters except null (0), as the null character is used to mark the end of the string. Bit 7 of each character must be off (0); therefore, the valid characters are those whose decimal representations range from 1 to 127, inclusive.

The ASCII code used in this string package is the same as that employed by FORTRAN for A-type FORMAT items, ENCODE/DECODE strings, and object-time FORMAT strings. ASCII strings in the form used by these routines are generated by the FORTRAN compiler whenever quoted strings are used as arguments in the CALL statement. Note that a null string (a string containing no characters) may be represented in
System Subroutine Library

FORTRAN by a variable or constant of any type which contains the value zero, or by a LOGICAL variable or constant with the .FALSE. value.

The SYSLIB user should ensure that a string never overflows the array that contains it by being aware of the length of the string result produced by each routine. In many routines where the result string length may vary or is difficult to determine, an optional integer argument may be specified to the subroutine to limit the length. In the sections describing the character string routines, this argument is called "len". The length of an output string is limited to the value specified for "len" plus one (for the null terminator); therefore the array receiving the result must be at least "len" plus one elements in size.

The optional argument "err" may be included when "len" is specified. "Err" is a logical variable which should be initialized by the FORTRAN program to the .FALSE. value. If a string function is given the arguments "len" and "err", and "len" is actually used to limit the length of the string result, then "err" is set to the .TRUE. value. If "len" is not used to truncate the string, "err" is unchanged, i.e., remains .FALSE..

"Len" and "err" are always optional arguments. "Len" may appear alone; however, "len" must appear if "err" is specified.

Several routines use the concept of character position. Each character in a string is assigned a position number which is one greater than the position of the character immediately to its left. The first character in a string is in position one.

0.2.4.1 Allocating Character String Variables -- A 1-dimensional LOGICAL*1 array may be used to contain a single string whose length may vary from zero characters to one less than the dimensioned length of the array. For example:

\texttt{LOGICAL*1 A(45) !ALLOCATE SPACE FOR STRING VARIABLE A}

The preceding example allows array A to be used as a string variable that can contain a string of 44 or less characters. Similarly, a 2-dimensional LOGICAL*1 array may be used to contain a 1-dimensional array of strings. Each string in the array may have a length up to one less than the first dimension of the the LOGICAL*1 array. There may be as many strings as the number specified for the second dimension of the LOGICAL*1 array, e.g.,

\texttt{LOGICAL*1 W(21,10) !ALLOCATE AN ARRAY OF STRINGS}

The preceding example creates a string array W which has 10 string elements, each of which may contain up to 20 characters. String I in array W is referenced in subroutine or function calls as W(I,1).

A 2-dimensional string array may be allocated, e.g.,

\texttt{LOGICAL*1 T(14,5,7) !ALLOCATE A 5 BY 7 STRING ARRAY}

In the preceding example, each string in array T may vary in length to a maximum of 13 characters. String I,J of the array may be referenced as T(I,1,J). Note that T is the same as T(I,1,1). This dimensioning process may be continued to create string arrays of up to six
System Subroutine Library

dimensions (represented by LOGICAL*1 arrays of up to seven dimensions).

0.2.4.2 Passing Strings to Subprograms -- The LOGICAL*1 arrays which contain strings may be placed in a COMMON block and referenced by any or all routines with a similar COMMON declaration. However, care should be taken when a LOGICAL*1 array is placed in a COMMON block, for if such an array has an odd length, it may cause all succeeding variables in the COMMON block to be assigned odd addresses. This situation will be detected by the RT-11 FORTRAN compiler, resulting in a warning message if the /W switch is specified.

A LOGICAL*1 array has an odd length only if the product of its dimensions is odd, e.g.,

    LOGICAL*1 B(10,7)  I(10*7)=70; EVEN LENGTH
    LOGICAL*1 H(21)    I21 IS ODD; ODD LENGTH

If odd length arrays are to be placed in a COMMON block, they should either be placed at the end of the block or they should be paired to result in an effective even length. For example:

    COMMON A1,A2,A3(10),H(21)  IPLACE ODD-SIZED ARRAY AT END

or

    COMMON A1,A2,H(21),H1(7),A3(10)  IPAIR ODD-SIZED ARRAYS H AND H1

Note that these cautions apply only to LOGICAL*1 variables and arrays.

The second method of passing strings to subprograms is through arguments and formal parameters. A single string may be passed by using its array name as an argument. For example:

    LOGICAL*1 A(21)    ISTRING VARIABLE "A", 20 CHARACTERS MAXIMUM
    CALL SUBR(A)      IPASS STRING A TO SUBROUTINE SUBR

If the maximum length of a string argument is unknown in a subroutine or function, or if the routine is used to handle many different length strings, the dummy argument in the routine should be declared as a LOGICAL*1 array with a dimension of one, e.g., LOGICAL*1 ARG(1). In this case, the string routines will correctly determine the length of ARG whenever it is used, but it will not be possible to determine the maximum size string which may be stored in ARG. If a multi-dimensional array of strings is passed to a routine, it must be declared in the called program with the same dimensions as were specified in the calling program.

NOTE

The length argument specified in many of the character string functions refers to the maximum length of the string excluding the necessary null byte terminator. The length of the LOGICAL*1 array to receive the string must be at least one greater than the length argument.

O-20
System Subroutine Library

0.2.4.3 Using Quoted-String Literals -- Quoted-strings may be used as arguments to any of the string routines that are invoked by the CALL statement. They cannot be used for routines invoked as functions. For example:

CALL SCOMP(NAME,'DOE, JOHN',M)

compares the string in the array NAME to the constant string DOE, JOHN and sets the value of the integer variable M accordingly. Although the above form of the routine reference is permitted, the statement:

M=ISCOMP(NAME,'DOE, JOHN')

is not allowed since the routine is not invoked by the CALL statement and includes a quoted-string literal.

0.3 LIBRARY FUNCTIONS AND SUBROUTINES

This section presents all SYSLIB functions and subroutines in alphabetic order. To reference these subprograms by usage, see Table 0-1.

O.3.1 AJFLT

The AJFLT function converts an INTEGER*4 value to a REAL*4 value and returns that result as the function value.

Form:  a = AJFLT (jsrc)

where:  jsrc is the INTEGER*4 variable to be converted.

Function Results:

The function's result is the REAL*4 value that is the result of the operation.

Example:

The following example converts the INTEGER*4 value contained in JVAL to single precision (REAL*4), multiplies it by 3.5, and stores the result in VALUE.

REAL*4 VALUE,AJFLT
INTEGER*4 JVAL

VALUE=AJFLT(JVAL)*3.5
0.3.2 CHAIN

The CHAIN subroutine allows a background program (or any program in the Single-Job system) to transfer control directly to another background program, passing it specified information. CHAIN cannot be called from a completion or interrupt routine. CHAIN does not close any of the FORTRAN logical units. When CHAINing to any other program, the user should explicitly close the opened logical units with calls to the CLOSE routine. Any USEREX routine specified to the FORTRAN run-time system will not be executed if a CHAIN is accomplished.

Form: CALL CHAIN (dblk, var, wcnt)

where:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblk</td>
<td>is the address of a 4-word Radix-50 descriptor of the file specification for the program to be run. (See the FORTRAN Language Reference Manual, Section 2.2.8.)</td>
</tr>
<tr>
<td>var</td>
<td>is the first variable in a sequence of variables with increasing memory addresses to be passed between programs in the chain parameter area (absolute locations 510 up to 700). A single array or a COMMON block (or portion of a COMMON block) is a suitable sequence of variables.</td>
</tr>
<tr>
<td>wcnt</td>
<td>is a word count (up to 60 words) specifying the number of words (beginning at var) to be passed to the called program.</td>
</tr>
</tbody>
</table>

If the size of the chain parameter area is insufficient, it may be increased by specifying the /B (bottom) switch to LINK for both the program executing the CHAIN call and the program receiving control.

The data passed may be accessed through a call to the RCHAIN routine (see Section 0.3.74). For more information on chaining to other programs, see Section 9.4.2.

Errors:

None.
System Subroutine Library

Example:

The following example transfers control from the main program to PROG.SAV, on DT0, passing it variables.

```
REAL*4 PROGNM
COMMON /BLK1/ A,B,C,D
DATA PROGNM/6RDT0PRO,6RG SAV/

CALL CHAIN(PROGNM,A,8)
```

!RAD50 FOR PROGRAM NAME
!DATA TO BE PASSED
!RUN DT0:PROG.SAV

0.3.3 CLOSEC

The CLOSEC subroutine terminates activity on the specified channel and frees it for use in another operation. The handler for the associated device must be in memory. CLOSEC cannot be called from a completion or interrupt routine.

Form: CALL CLOSEC (chan)

where: chan is the channel number to be closed. This argument must be located so that the USR cannot swap over it.

A CLOSEC or PURGE must eventually be issued for any channel opened for either input or output. A CLOSEC call specifying a channel that is not open is ignored.

A CLOSEC performed on a file that was opened via an ENTER causes the device directory to be updated to make that file permanent. If the device associated with the specified channel already contains a file with the same name and extension, the old copy is deleted when the new file is made permanent. A CLOSEC on a file opened via LOOKUP does not require any directory operations.

When an entered file is CLOSECed, its permanent length reflects the highest block of the file written since the file was entered; for example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area which was allocated at ENTER time, the unused blocks are reclaimed as an empty area on the device.

Errors:

CLOSEC does not generate any errors. If the device handler for the operation is not in memory, a fatal monitor error is generated.
System Subroutine Library

Example:

The following example creates and processes a 56-block file.

```plaintext
REAL*4 DBLK(2)
DATA DBLK/6RSY0NEW,6RFILDAT/
DATA ISIZE/56/

ICHAN=IGETC()
IF(ICHAN.LT.0) GOTO 100
IF(IENTER(ICHAN, DBLK, ISIZE)-1) 10,110,120
10   
   
   CALL CLOSEC(ICHAN)
   CALL IFREEC(ICHAN)
   CALL EXIT
100  STOP 'NO AVAILABLE CHANNELS'
110  STOP 'CHANNEL ALREADY IN USE'
120  STOP 'NOT ENOUGH ROOM ON DEVICE'
END
```

---

**CONCAT**

0.3.4 CONCAT

The CONCAT subroutine is used to concatenate character strings.

**Form:** CALL CONCAT (a,b, out[, len[, err]])

**where:**
- **a** is the array containing the left string.
- **b** is the array containing the right string.
- **out** is the array into which the concatenated result is placed. This array must be at least one element longer than the maximum length of the result string (i.e., one greater than the value of len, if specified).
- **len** is the integer number of characters representing the maximum length of the output string. The effect of len is to truncate the output string to a given length, if necessary.
- **err** is the Logical error flag set if the output string is truncated to the length specified by len.

---

0-24
System Subroutine Library

The string in array "a" immediately followed on the right by the string in array "b" and a terminating null character replaces the string in array "out". Any combination of string arguments is allowed so long as "b" and "out" do not specify the same array. Concatenation stops either when a null character is detected in "b" or when the number of characters specified by "len" have been moved.

If either the left or right string is a null string, the other string is copied to "out". If both are null strings, then "out" is set to a null string. The old contents of "out" are lost when this routine is called.

Errors:

Error conditions are indicated by "err", if specified. If "err" is given and the output string would have been longer than "len" characters, then "err" is set to .TRUE.; otherwise, "err" is unchanged.

Example:

The following example concatenates the string in array STR and the string in array IN and stores the resultant string in array OUT. OUT cannot be larger than 29 characters.

    LOGICAL*1 IN(30),OUT(30),STR(7)
    :
    :
    CALL CONCAT(STR,IN,OUT,29)

0.3.5 CVTTIM

The CVTTIM subroutine converts a 2-word internal format time to hours, minutes, seconds, and ticks.

Form: CALL CVTTIM (time,hrs,min,sec,tick)

where:  time is the 2-word internal format time to be converted. If time is considered as a 2-element INTEGER*2 array, then:

    time (1) is the high-order time.
    time (2) is the low-order time.

    hrs is the integer number of hours.
System Subroutine Library

\[
\begin{align*}
\text{min} & \quad \text{is the integer number of minutes.} \\
\text{sec} & \quad \text{is the integer number of seconds.} \\
\text{tick} & \quad \text{is the integer number of ticks (1/60 of a second for 60-cycle clocks; 1/50 of a second for 50-cycle clocks).}
\end{align*}
\]

Errors:
None.

Example:

```
INTEGER*4 ITIME

CALL GTIM(ITIME) !GET CURRENT TIME-OF-DAY
CALL CVTTIM(ITIME,IHRS,IMIN,ISEC,ITCK)
IF(IHRS.GE.12) GOTO 100 !TIME FOR LUNCH
```

\[
\text{DEVICE}
\]

0.3.6 DEVICE (F/B only)

The DEVICE subroutine allows the user to set up a list of addresses to be loaded with specified values when the program is terminated. If a job terminates or is aborted with a CTRL C from the terminal, this list is picked up by the system and the appropriate addresses are set up with the corresponding values.

This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates.

Only one address list may be active at any given time; hence, if multiple DEVICE calls are issued, only the last one has any effect. The list must not be modified by the FORTRAN program after the DEVICE call has been issued, and the list must not be located in an overlay or an area over which the USR will swap.

Form: CALL DEVICE (iList)

where: iList \quad \text{is an integer array containing address/value pairs, terminated by a zero word. On program termination, each value is moved to the corresponding address.}

For more information on loading values into device registers, see the assembly language .DEVICE request, Section 9.4.11.
System Subroutine Library

Errors:
None.

Example:

```
INTEGER*2 IDR11(3)
DATA IDR11(1)/"167770/
DATA IDR11(2)/0/
DATA IDR11(3)/0/
CALL DEVICE(IDR11)
```

0.3.7 DJFLT

The DJFLT function converts an INTEGER*4 value into a REAL*8 (DOUBLE PRECISION) value and returns that result as the function value.

Form:  \( d = \text{DJFLT}(\text{jsrc}) \)

where:  jsrc specifies the INTEGER*4 variable which is to be converted.

Notes:

If DJFLT is used, it must be explicitly defined (REAL*8 DJFLT) or implicitly defined (IMPLICIT REAL*8 (D)) in the FORTRAN program. If this is not done, its type will be assumed as REAL*4 (single precision).

Function Results:

The function result is the REAL*8 value that is the result of the operation.

Example:

```
INTEGER*4 JVAL
REAL*8 DJFLT,D
.
.
D=DJFLT(JVAL)
```
GETSTR

0.3.8 GETSTR

The GETSTR subroutine reads a formatted ASCII record from a specified FORTRAN logical unit into a specified array. The data is truncated (trailing blanks removed) and a null byte is inserted at the end to form a character string.

GETSTR can be used in main program routines or in completion routines but cannot be used in both at the same time. If GETSTR is used in a completion routine, it cannot be the first I/O operation on the specified logical unit.

Form:  CALL GETSTR (lun,out,len{,err})

where:  lun is the integer FORTRAN logical unit number of a formatted sequential file from which the string is to be read.

out is the array which is to receive the string; this array must be one element longer than len.

len is the integer number representing the maximum length of the string to be input.

e err is the Logical error flag that is set to .TRUE. if the string specified by "out" exceeds the value of "len" in length; err must be present if the input string can exceed "len".

Errors:

Error conditions are indicated by "err". If "err" is given and the output string was truncated to the length specified by "len", then "err" is set to .TRUE.; otherwise, "err" remains unchanged.

Example:

The following example reads a string of up to 80 characters from logical unit 5 into the array STRING.

LOGICAL*1 STRING(81),ERR
:  
:  
:  
CALL GETSTR(5,STRING,80,ERR)
0.3.9 GTIM

THE GTIM subroutine allows user programs to access the current time of day. The time is returned in two words and is given in terms of clock ticks past midnight. If the system does not have a line clock, a value of zero is returned. If an RT-l1 monitor TIME command has not been entered, the value returned will be the time elapsed since the system was bootstrapped rather than the time of day.

Form: CALL GTIM (itime)

where: itime is the 2-word area to receive the time of day.

The high-order time is returned in the first word, the low-order time in the second word. The SYSLIB routine CVTTIM (Section 0.3.5) can be used to convert the time into hours, minutes, seconds and ticks. CVTTIM performs the conversion based on the monitor configuration word for 50- or 60-cycle clocks (see Section 9.2.6). Under an F/B Monitor, the time-of-day is automatically reset after 24:00 when a GTIM is executed; under the Single-Job Monitor, it is not.

Errors:

None.

Example:

    INTEGER*4 JTIME
    .
    .
    .
    CALL GTIM(JTIME)

0.3.10 GTJB

The GTJB subroutine passes certain job parameters back to the user program.
System Subroutine Library

Form: CALL GTJB (addr)

where: addr is an 8-word area to receive the job parameters. This area, considered as an 8-element INTEGER*2 array, has the following format:

- addr(1) job number. (0=Background, 2=Foreground)
- addr(2) high memory limit
- addr(3) low memory limit
- addr(4) beginning of I/O channel space
- addr(5)-addr(8) reserved for future use

For more information on passing job parameters, see the assembly language .GTJB request, Section 9.4.17.

Errors:
None.

Example:

```
INTEGER*2 PARAMS(8)
CALL GTJB(PARAMS)
IF(PARAMS(1).EQ.0) TYPE 99
 99 FORMAT (' THIS IS THE BACKGROUND JOB')
```

IADDR

O.3.11 IADDR

The IADDR function returns the 16-bit absolute memory address of its argument as the integer function value.

Form: i = IADDR (arg)

where: arg is the variable, constant, or expression whose memory address is to be obtained.

Errors:
None.

Example:

IADDR can be used to find the address of an assembly language global area. For example:

```
EXTERNAL CAREA
J=IADDR(CAREA)
```
0.3.12 IAJFLT

The IAJFLT function converts an INTEGER*4 value to a REAL*4 value and stores the result.

Form:  \( i = \text{IAJFLT} (\text{jsrc, ares}) \)

where: \( \text{jsrc} \) is the INTEGER*4 variable to be converted.

\( \text{ares} \) is the REAL*4 variable or array element to receive the converted value.

Function Results:

The function result indicates the following:

- \( i = -2 \) Significant digits were lost during the conversion.
- \( i = -1 \) Normal return; the result is negative.
- \( i = 0 \) Normal return; the result is zero.
- \( i = 1 \) Normal return; the result is positive.

Example:

```
INTEGER*4 JVAL
REAL*4 RESULT
.
.
.
IF(IAJFLT(JVAL,RESULT).EQ.-2) TYPE 99
99 FORMAT (' OVERFLOW IN INTEGER*4 TO REAL CONVERSION')
```
0.3.13 IASIGN

The IASIGN function sets information in the FORTRAN logical unit table (overriding the defaults) so that the specified information is used when the FORTRAN Object Time System (OTS) opens the logical unit. This function can be used with ICSI (see Section 0.3.17) to allow a FORTRAN program to accept a standard CSI input specification. IASIGN must be called before the unit is opened, i.e., before any READ, WRITE, PRINT, TYPE, or ACCEPT statements are executed that reference the logical unit.

Form: \( i = \text{IASIGN}(\text{lun}, \text{idev}, \text{ifilex}, \text{isize}, \text{itype}) \)

where:

- \( \text{lun} \) is an INTEGER*2 variable, constant, or expression specifying the FORTRAN logical unit for which information is being specified.
- \( \text{idev} \) is a 1-word Radix-50 device name; this can be the first word of an ICSI input or output file specification.
- \( \text{ifilex} \) is a 3-word Radix-50 filename and extension; this can be words 2 through 4 of an ICSI input or output file specification.
- \( \text{isize} \) is the length (in number of blocks) to allocate for an output file; this can be the fifth word of an ICSI output specification. If zero, the larger of either one-half the largest empty segment or the entire second largest empty segment is allocated (see Section 9.4.13). If the value specified for length is -1, the entire largest empty segment is allocated.
- \( \text{itype} \) is an integer value determining the optional attributes to be assigned to the file. This value is obtained by adding the values that correspond to the desired operations:

1. use double buffering for output
2. open the file as a temporary file
4. perform a lookup on the file (otherwise, the first FORTRAN I/O operation determines how the file is opened)
System Subroutine Library

8  expand carriage control information (see Notes below)
16  do not expand carriage control information
32  file is read-only

Notes:

Expanded carriage control information applies only to formatted output files and means that the first character of each record is used as a carriage control character when processing a write operation to the given logical unit. The first character is removed from the record and converted to the appropriate ASCII characters to simulate the requested carriage control.

If carriage control information is not expanded, the first character of each record is unmodified and the FORTRAN OTS outputs a line feed, followed by the record, followed by a carriage return.

If carriage control is unspecified, the FORTRAN OTS sends expanded carriage control information to the terminal and line printer and sends unexpanded carriage control information to all other devices and files. See the PDP-11 FORTRAN Language Reference Manual, Section 6.3, for further carriage control information.

Function Results:

i = 0  Normal return.
<> 0  The specified logical unit is already in use or there is no space for another logical unit association.

Example:

The following example creates an output file on logical unit 3 using the first output file given to the RT-11 command string interpreter (CSI), sets it up for double buffering, creates an input file on logical unit 4 based on the first input file specification given to the RT-11 CSI, and makes it available for read-only access.

```
INTEGER*2 SPEC(39)
REAL*4 EXT(2)
DATA EXT/6RDATDAT,6RDATDAT/  !DEFAULT EXTENSION IS DAT

10 IF(ICS(EXT,,0).NE.0) GOTO 10
   C C DO NOT ACCEPT ANY SWITCHES
   C CALL IASIGN(3,SPEC(1),SPEC(2),SPEC(5),1)
   C CALL IASIGN(4,SPEC(16),SPEC(17),0,32)
```

0-33
0.3.14 ICFDN

The ICFDN function increases the number of input/output channels. Note that ICFDN defines new channels; the previously-defined channels are not used. Thus, an ICFDN for 20 channels (while the 16 original channels are defined) causes only 20 I/O channels to exist; the space for the original 16 is unused. The space for the new channel area is allocated out of the free space managed by the FORTRAN system.

Form:  i = ICFDN (num)

where:  num  is the integer number of channels to be allocated. The number of channels must be greater than 16 and can be a maximum of 256. SYSLIB can use all new channels greater than 16 without a call to IGETC; the FORTRAN system input/output uses only the first 16 channels. This argument must be positioned so that the USR cannot swap over it.

Notes:

1. ICFDN cannot be issued from a completion or interrupt routine.

2. It is recommended that the ICFDN function be used at the beginning of the main program before any I/O operations are initiated.

3. If ICFDN is executed more than once, a completely new set of channels is created each time ICFDN is called.

4. ICFDN requires that extra memory space be allocated to foreground programs (see Section 0.1.4).

Function Results:

i = 0  Normal return.
     = 1  An attempt was made to allocate fewer channels than already exist.
     = 2  Not enough free space is available for the channel area.

Example:

IF(ICFDN(24).NE.0) STOP 'NOT ENOUGH MEMORY'
0.3.15 ICHCPY (F/B only)

The ICHCPY function opens a channel for input, logically connecting it to a file which is currently open by another job for either input or output. This function may be used by either the foreground or the background. An ICHCPY must be done before the first read or write for the given channel.

Form:  \( i = \text{ICHCPY} \ (\text{chan},\text{ochan}) \)

where:  chan is the channel which the job will use to read the data.

ochan is the channel number of the other job which is to be copied.

Notes:

1. If the other job's channel was opened via an IENTER function or a .ENTER programmed request to create a file, the copier's channel indicates a file which extends to the highest block that the creator of the file had written at the time the ICHCPY was executed.

2. A channel which is open on a sequential-access device should not be copied, because buffer requests may become intermixed.

3. A program can write to a file (which is being created by the other job) on a copied channel just as it could if it were the creator. When the copier's channel is closed, however, no directory update takes place.

Errors:

\( i \)

\( 0 \) Normal return.

\( 1 \) Other job does not exist or does not have the specified channel (ochan) open.

\( 2 \) Channel (chan) is already open.
0.3.16 ICMKT (F/B only)

The ICMKT function causes one or more scheduling requests (made by an ISCHED, ITIMER or MRKT routine) to be cancelled.

Form:  \( i = \text{ICMKT}(\text{id}, \text{time}) \)

where:  \( \text{id} \)  is the identification integer of the request to be cancelled. If \( \text{id} \) is equal to zero, all scheduling requests are cancelled.

\( \text{time} \)  is the name of a 2-word area in which the monitor will return the amount of time remaining in the cancelled request.

For further information on cancelling scheduling requests, see the assembly language .CMKT request, Section 9.4.5.

Errors:

\[
\begin{align*}
\text{i} & = 0 \quad \text{Normal return.} \\
\text{i} & = 1 \quad \text{id was not equal to zero and no schedule request with that identification could be found.}
\end{align*}
\]

Example:

\[
\begin{verbatim}
INTEGER*4 J
.
.
.
CALL ICMKT(0,J) !ABORT ALL TIMER REQUESTS NOW
.
.
.
END
\end{verbatim}
\]
The ICSI function calls the RT-ll command string interpreter in special mode to parse a command string and return file descriptors and switches to the program. In this mode, the CSI does not perform any handler fetches, CLOSEs, ENTERs, or LOOKUPS. ICSI cannot be called from a completion or interrupt routine.

Form: \[ i = \text{ICSI}(\text{filspc}, \text{defext}, \{\text{cstring}\}, \{\text{switch}\}, x) \]

where: \( \text{filspc} \)
is the 39-word area to receive the file specifications. The format of this area (considered as a 39-element INTEGER*2 array) is:

- \( \text{filspc}(1) \) - output file number 1 specification
- \( \text{filspc}(4) \) - output file number 1 length
- \( \text{filspc}(6) \) - output file number 2 specification
- \( \text{filspc}(9) \) - output file number 2 length
- \( \text{filspc}(11) \) - output file number 3 specification
- \( \text{filspc}(14) \) - output file number 3 length
- \( \text{filspc}(16) \) - input file number 1 specification
- \( \text{filspc}(19) \) - input file number 1 length
- \( \text{filspc}(20) \) - input file number 2 specification
- \( \text{filspc}(23) \) - input file number 2 length
- \( \text{filspc}(24) \) - input file number 3 specification
- \( \text{filspc}(27) \) - input file number 3 length
- \( \text{filspc}(28) \) - input file number 4 specification
- \( \text{filspc}(31) \) - input file number 4 length
- \( \text{filspc}(32) \) - input file number 5 specification
- \( \text{filspc}(35) \) - input file number 5 length
- \( \text{filspc}(36) \) - input file number 6 specification
- \( \text{filspc}(39) \) - input file number 6 length

\( \text{defext} \)
is the table of Radix-50 default extensions to be assumed when a file is specified without an extension.
System Subroutine Library

defext(1) is the default for all input file extensions.
defext(2) is the default extension for output file number 1.
defext(3) is the default extension for output file number 2.
defext(4) is the default extension for output file number 3.

cstring is the area which contains the ASCII command string to be interpreted; the string must end in a zero byte. If this argument is omitted, the system prints the prompt character (*) at the terminal and accepts a command string.

switch is the name of an INTEGER*2 array dimensioned (4,n) where n represents the number of switches which are defined to the program. This argument must be present if the value specified for "x" is non-zero. This array has the following format for the nth switch described by the array.

switch(1,n) is the 1-character ASCII name of the switch.
switch(2,n) is set by the routine to 0, if the switch did not occur; to 1, if the switch occurred without a value; to 2, if the switch occurred with a value.
switch(3,n) is set to the file number on which the switch is specified.
switch(4,n) is set to the specified value if switch(2,n) is equal to 2.

x is the number of switches defined in the array "switch".

Notes:

The array "switch" must be set up to contain the names of the valid switches. For example, use the following to set up names for five switches:

```
INTEGER*2 SW(4,5)
DATA SW(1,1)='S',SW(1,2)='M',SW(1,3)='I'
DATA SW(1,4)='L',SW(1,5)='E'
```

Multiple occurrences of the same switch are supported by allocating an entry in the switch array for each occurrence of the switch. Each time the switch occurs in the switch array, the next unused entry for the named switch will be used.

The arguments of ICSI must be positioned so that the USR cannot swap over them.

For more information on calling the Command String Interpreter, see the assembly language .CSISPC request, Section 9.4.8.
System Subroutine Library

Errors:

\[ i = 0 \quad \text{Normal return.} \]
\[ i = 1 \quad \text{Illegal command line; no data was returned.} \]
\[ i = 2 \quad \text{An illegal device specification occurred in the string.} \]
\[ i = 3 \quad \text{An illegal switch was specified, or a given switch was specified more times than allowed for in the switch array.} \]

Example:

The following example causes the program to loop until a valid command is typed at the console terminal.

\[
\text{INTEGER*2 SPEC(39)} \\
\text{REAL*4 EXT(2)} \\
\text{DATA EXT/6RDATDAT,6RDATDAT/} \\
\]

\[
10 \quad \text{TYPE 99} \\
99 \quad \text{FORMAT (' ENTER VALID CSI STRING WITH NO SWITCHES')} \\
\]

\[
\text{IF(ICSII(SPEC,EXT,,0).NE.0) GOTO 10} \\
\]

O.3.18 ICSTAT (F/B only)

The ICSTAT function furnishes the user with information about a channel. It is supported only in the F/B environment; no information is returned when operating under the Single-Job Monitor.

Form: \[ i = \text{ICSTAT (chan, addr)} \]

where: \[ \text{chan} \]

is the channel whose status is desired.

\[ \text{addr} \]

is a 6-word area to receive the status information. The area, as a 6-element INTEGER*2 array, has the following format.

\[
\text{addr(1) channel status word (see Section 9.4.34)} \\
\text{addr(2) starting block number of file on this channel} \\
\text{addr(3) length of file} \\
\text{addr(4) highest block number written since file was opened (see Section 9.4.9)} \\
\]

O-39
System Subroutine Library

addr(5) unit number of device with which this channel is associated
addr(6) Radix-50 of device name with which the channel is associated

Errors:

i = 0 Normal return.
i = 1 Channel specified is not open.

Example:

The following example obtains channel status information about channel 1.

INTEGER*2 AREA(6)
I=7
IF(ICSTAT(I,AREA).NE.0) TYPE 99,I
99 FORMAT(1X,'CHANNEL',I4,'IS NOT OPEN')

IDELET

0.3.19 IDELET

The IDELET function deletes a named file from an indicated device.
IDELET cannot be issued from a completion or interrupt routine.

Form: i = IDELET (chan,dblk{,count})

where: chan is the channel to be used for the delete operation.

dblk is the 4-word Radix-50 specification (dev:filnam.ext) for the file to be deleted.

count is used by magtape/cassette only to prevent a rewind before the operation. (Refer to Appendix H for more information concerning the magtape and cassette handlers.)

NOTE

The arguments of IDELET must be located so that the USR cannot swap over them.

O-40
System Subroutine Library

The specified channel is left inactive when the IDELET is complete. IDELET requires that the handler to be used be resident (via an IPETCH call) at the time the IDELET is issued. If it is not, a monitor error occurs.

For further information on deleting files, see the assembly language .DELETE request, Section 9.4.10.

Errors:

i = 0 Normal return.
    = 1 Channel specified is already open.
    = 2 File specified was not found.

Example:

The following example deletes a file named FTN5.DAT from SY0.

REAL*4 FILNAM(2)
DATA FILNAM/6RSY0FTN,6R5 DAT/
.
.
I=IGETC()
IF(I.LT.0) STOP 'NO CHANNEL'
CALL IDELET(I,FILNAM)
CALL IFREEC(I)

IDJFLT

O.3.20 IDJFLT

The IDJFLT function converts an INTEGER*4 value into a REAL*8 (DOUBLE PRECISION) value and stores the result.

Form: i = IDJFLT (jsrc,dres)

where: jsr specifies the INTEGER*4 variable that is to be converted.

  dres specifies the REAL*8 (or DOUBLE PRECISION) variable to receive the converted value.

Function Results:

The function result indicates the following:

i = -1 Normal return; the result is negative.
    = 0 Normal return; the result is zero.
    = 1 Normal return; the result is positive.
System Subroutine Library

Example:

    INTEGER*4 JJ
    REAL*8 DJ
    ...
    IF(IDJFLT(JJ,DJ).LE.0) TYPE '99
    99 FORMAT ('VALUE IS NOT POSITIVE')

IDSTAT

0.3.21 IDSTAT

The IDSTAT function is used to obtain information about a particular device. IDSTAT cannot be issued from a completion or interrupt routine.

Form:  i = IDSTAT (devnam,cblk)

where: devnam is the Radix-50 device name.

      cblk is the 4-word area used to store the status information. The area, as a 4-element INTEGER*2 array, has the following format:

      cblk(1) device status word (see Section 9.4.12)
      cblk(2) size of handler in bytes
      cblk(3) entry point of handler (non-zero implies that the handler is in memory)
      cblk(4) size of the device (in 256-word blocks) for block-replaceable devices; zero for sequential-access devices

NOTE

The arguments of IDSTAT must be positioned so that the USR cannot swap over them.

IDSTAT looks for the device specified by devnam and, if found, returns four words of status in cblk.
System Subroutine Library

Errors:

\[ i = 0 \quad \text{Normal return.} \]
\[ i = 1 \quad \text{Device not found in monitor tables.} \]

Example:

The following example determines whether the line printer handler is in memory. If it is not, the program stops and prints a message to indicate that the handler must be loaded.

```fortran
REAL*4 IDNAM
INTEGER*2 CBLK(4)
DATA IDNAM/3RLP /
DATA CBLK/4*0/
CALL IDSTAT(IDNAM,CBLK)
IF(CBLK(3).EQ.0) STOP 'LOAD THE LP HANDLER AND RERUN'
```

0.3.22 IENTER

The IENTER function allocates space on the specified device and creates a tentative directory entry for the named file. If a file of the same name already exists on the specified device, it is not deleted until the tentative entry is made permanent by CLOSEC. The file is attached to the channel number specified.

Form: \( i = \text{IENTER} (\text{chan,dblk,length[,count])} \)

where:

- **chan** is the integer specification for the RT-11 channel to be associated with the file.
- **dblk** is the 4-word Radix-50 descriptor of the file to be operated upon.
- **length** is the integer number of blocks to be allocated for the file. If zero, the larger of either one-half the largest empty segment or the entire second largest empty segment is allocated (see Section 9.4.13). If the value specified for length is \(-1\), the entire largest empty segment is allocated.
- **count** is used by magtape/cassette only to prevent a rewind before the operation (see Appendix H).
System Subroutine Library

Notes:

1. IENTER cannot be issued from a completion or interrupt routine.

2. IENTER requires that the appropriate device handler be in memory.

3. The arguments of IENTER must be positioned so that the USR does not swap over them.

For further information on creating tentative directory entries, see the assembly language .ENTER request, Section 9.4.13.

Errors:

\[ i = n \]
Normal return; number of blocks actually allocated (\( n = 0 \) for nonfile-structured IENTER).

\[ i = -1 \]
Channel (chan) is already in use.

\[ i = -2 \]
In a fixed-length request, no space greater than or equal to length was found.

Example:

The following example allocates a channel for file TEMP.TMP on Y0. If no channel is available, the program prints a message and halts.

```fortran
REAL*4 DBLK(2)
DATA DBLK/6RSY0TEM,6RP TMP/
ICHAN=IGETC()
IF(ICHAN.LT.0) STOP 'NO AVAILABLE CHANNEL'
C
C CREATE TEMPORARY WORK FILE
C
IF(IENTER(ICHAN,DBLK,20).LT.0) STOP 'ENTER FAILURE'
.:.
CALL PURGE(ICHAN)
CALL IFREEC(ICHAN)
```

\[ IFETCH \]

0.3.23 IFETCH

The IFETCH function loads a device handler into memory from the system device, making the device available for input/output operations. The handler is loaded into the free area managed by the FORTRAN system. Once the handler is loaded, it cannot be released and the memory in which it resides cannot be reclaimed. IFETCH cannot be issued from a completion or interrupt routine.
System Subroutine Library

Form: i = IFETCH (devnam)

where: devnam is the 1-word Radix-50 name of the device for which the handler is desired. This argument can be the first word of an ICSI input or output file specification. This argument must be positioned so that the USR cannot swap over it.

For further information on loading device handlers into memory, see the assembly language .FETCH request, Section 9.4.15.

Errors:

i = 0 Normal return.
i = 1 Device name specified does not exist.
i = 2 Not enough room exists to load the handler.
i = 3 No handler for the specified device exists on the system device.

Example:

The following example requests the DX1 handler to be loaded into memory; execution stops if the handler cannot be loaded.

    REAL*4 IDNAM
    DATA IDNAM/3RDX1/
    
    IF (IFETCH(IDNAM).NE.0) STOP 'FATAL ERROR FETCHING HANDLER'

0.3.24 IFREEC

The IFREEC function returns a specified RT-11 channel to the available pool of channels. Before IFREEC is called, the specified channel must be closed or deactivated with a CLOSEC (see Section 0.3.3) or a PURGE (see Section 0.3.70) call. IFREEC cannot be called from a completion or interrupt routine. IFREEC calls must be issued only for channels that have been successfully allocated by IGETC calls; otherwise, unpredictable results will occur.

Form: i = IFREEC (chan)

where: chan is the integer number of the channel to be freed.
System Subroutine Library

Errors:

\[ i = 0 \quad \text{Normal return.} \]
\[ i = 1 \quad \text{Specified channel is not currently allocated.} \]

Example:

See the example under IGETC, (Section 0.3.25).

IGETC

0.3.25 IGETC

The IGETC function allocates an RT-ll channel (in the range 0-17 octal) and marks it in use for the FORTRAN I/O system. IGETC cannot be issued from a completion or interrupt routine.

Form: \( i = \text{IGETC}() \)

Function Results:

\[ i = -1 \quad \text{No channels are available.} \]
\[ i = n \quad \text{Channel n has been allocated.} \]

Example:

\[
\text{ICHAN}=\text{IGETC}() \quad \text{!ALLOCATE CHANNEL}
\text{IF(ICHAN.LT.0) STOP 'CANNOT ALLOCATE CHANNEL'}
\text{.}
\text{.}
\text{CALL IFREEC(ICHAN) \quad !FREE IT WHEN THROUGH}
\text{.}
\text{.}
\text{END}
\]
0.3.26 IJCVT

The IJCVT function converts an INTEGER*4 value to INTEGER*2 format. If ires is not specified, the result returned is the INTEGER*2 value of jsr. If ires is specified, the result is stored there.

Form:  i = IJCVT (jsr{},ires)

where:  jsr specifies the INTEGER*4 variable or array element whose value is to be converted.
ires specifies the INTEGER*2 entity to receive the conversion result.

Function results if ires is specified:

i = -2  An overflow occurred during conversion.
   = -1  Normal return; the result is negative.
   = 0   Normal return; the result is zero.
   = 1   Normal return; the result is positive.

Example:

INTEGER*4 JVAL
INTEGER*2 IVAL
.
.
.
IF(IJCVT(JVAL,IVAL).EQ.-2) TYPE 99
99 FORMAT(' NUMBER TOO LARGE IN IJCVT CONVERSION')

0.3.27 ILUN

The ILUN function returns the RT-11 channel number with which a FORTRAN logical unit is associated.
System Subroutine Library

Form: $i = \text{ILUN} \ (\text{lun})$

where: $\text{lun}$ is an integer expression whose value is a
FORTRAN logical unit number in the range 1-99.

Function Results:

- $i = -1$ Logical unit is not open.
- $i = -2$ Logical unit is opened to console terminal.
- $i = +n$ RT-ll channel number $n$ is associated with lun.

Example:

```
PRINT 99
99 FORMAT(' PRINT DEFAULTS TO LOGICAL UNIT 6, WHICH FURTHER DEFAULTS TO LP: '
LUNRT=ILUN(6) !WHICH RT-ll CHANNEL IS RECEIVING I/O?
```

INDEX

0.3.28 INDEX

The INDEX subroutine searches a string for the occurrence of another
string and returns the character position of the first occurrence of
that string.

Form: CALL INDEX ($a$, $\text{pattn}$, {i}, m)

or

$m = \text{INDEX} \ (a$, $\text{pattn}[,i])$

where: $a$ is the array containing the string to be searched.

$\text{pattn}$ is the string being sought.

i is the integer starting character position of the search in $a$. If $i$ is omitted, $a$ is
searched beginning at the first character position.

m is the integer result of the search; $m$
equals the starting character position of
$\text{pattn}$ in $a$, if found; otherwise it is zero.

Errors:

None.

January 1976          0-48
System Subroutine Library

Example:

The following example searches the array STRING for the first occurrence of strings EFG and XYZ and searches the string ABCABCABC for the occurrence of string ABC after position 5.

CALL SCOPY('ABCDERFGHI',STRING) !INITIALIZE STRING
CALL INDEX(STRING,'EFG',M) !M=5
CALL INDEX(STRING,'XYZ',N) !N=0
CALL INDEX('ABCABCABC','ABC',5,L) !L=7

0.3.29 INSERT

The INSERT subroutine replaces a portion of one string with another string.

Form: CALL INSERT (in,out,i{,m})
  where: in is the array containing the string being inserted.
         out is the array containing the string being modified.
         i is the integer specifying the character position in "out" at which the insertion begins.
         m is the integer maximum number of characters to be inserted.

If the maximum number of characters (m) is not specified, all characters to the right of the specified character position (i) in the string being modified are replaced by the string being inserted. The insert string (in) and the string being modified (out) may be in the same array only if the maximum number of characters (m) is specified and is less than or equal to the difference between the position of the insert (i) and the maximum string length of the array.

Errors:

None.

Example:

CALL SCOPY('ABCDERFGHIJ',S1) !INITIALIZE STRING 1
CALL SCOPY(S1,S2) !INITIALIZE STRING 2
CALL INSERT('123',S1,6,3) !S1 = 'ABCD123IJ'
CALL INSERT('123',S2,4) !S2 = 'ABC123'
INTSET

0.3.30 INTSET

The INTSET function establishes a FORTRAN subroutine as an interrupt service routine, assigns it a priority, and attaches it to a vector. INTSET requires that extra memory be allocated to foreground programs that use it (see Section 0.1.4).

Form: \( i = \text{INTSET} (\text{vect}, \text{pri}, \text{id}, \text{crtn}) \)

where:

\begin{align*}
\text{vect} & \quad \text{is the integer specifying the address of the interrupt vector to which the subroutine is to be attached.} \\
\text{pri} & \quad \text{is the integer specifying the actual priority level (4-7) at which the device interrupts.} \\
\text{id} & \quad \text{is the identification integer to be passed as the single argument to the FORTRAN routine when an interrupt occurs. This allows a single crtn to be associated with several INTSET calls.} \\
\text{crtn} & \quad \text{is a FORTRAN subroutine to be established as the interrupt routine. This name should be specified in an EXTERNAL statement in the FORTRAN program that calls INTSET. The subroutine has one argument:} \\
& \quad \text{SUBROUTINE crtn(id)} \\
& \quad \text{INTEGER id}
\end{align*}

When the routine is entered, the value of the integer argument will be the value specified for id in the appropriate INTSET call.

Notes:

1. The "id" argument may be used to distinguish between interrupts from different vectors if the routine to be activated services multiple devices.

2. When using INTSET in F/B, the SYSLIB call DEVICE must be used in almost all cases to prevent interrupts from interrupting beyond program termination.

3. If the interrupt routine (crtn) has control for a period of
time longer than the time in which two more interrupts using the same vector occur, interrupt overrun is considered to have occurred. The error message:

?SYSLIB- FATAL INTERRUPT OVERRUN

is printed and the job is aborted. Tasks requiring very fast interrupt response may not be viable using FORTRAN as FORTRAN overhead lowers RT-ll's interrupt response rate.

4. The interrupt routine (crtm) is actually run as a completion routine via use of the RT-ll .SYNCH request (Section 9.3.1.4). The "pri" argument is used for the RT-ll .INTEN request (Section 9.3.1.2).

5. A .PROTECT request is issued for the vector, but no attempt is made to report an error if the vector is already protected; furthermore, the vector will be taken over unconditionally. See Section 9.4.25 for more information.

6. The FORTRAN interrupt service subroutine (crtm) cannot call the USR.

7. INTSET cannot be called from a completion or interrupt routine.

8. Interrupt enable should not be set on the associated device until the INTSET call has been successfully executed.

Errors:

<table>
<thead>
<tr>
<th>i</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal return.</td>
</tr>
<tr>
<td>1</td>
<td>Invalid vector specification.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>3</td>
<td>No space is available for the linkage setup.</td>
</tr>
</tbody>
</table>

Example:

```
EXTERNAL CLKSUB !SUBR TO HANDLE KW11-P CLOCK
.
.
I=INTSET("104,6,0,CLKSUB") !ATTACH ROUTINE
IF (I.NE.0) GOTO 100 !BRANCH IF ERROR
.
.
END
SUBROUTINE CLKSUB(ID)
.
.
END
```
IPEEK

0.3.31 IPEEK

The IPEEK function returns the contents of the word located at a specified absolute 16-bit memory address. This function may be used to examine device registers or any location in memory.

Form:  \( i = \text{IPEEK}(\text{iaddr}) \)

where:  \( \text{iaddr} \) is the integer specification of the absolute address to be examined. If this argument is not an even value, a trap will result.

Function Result:

The function result (\( i \)) is set to the value of the word examined.

Example:

\[
\text{ISWIT} = \text{IPEEK}("177570") \quad \text{!GET VALUE OF CONSOLE SWITCHES}
\]

IPOKE

0.3.32 IPOKE

The IPOKE subroutine stores a specified 16-bit integer value into a specified absolute memory location. This subroutine may be used to store values in device registers.

Form:  CALL IPOKE (iaddr,ivalue)

where:  \( \text{iaddr} \) is the integer specification of the absolute address to be modified. If this argument is not an even value, a trap will result.
System Subroutine Library

ivalue is the integer value to be stored in the given address (iaddr).

Errors:
None.

Example:
The following example displays the value of IVAL in the console display register.

CALL IPOKE("177570,IVAL)

To set bit 12 in the JSW without zeroing any other bits in the JSW, use the following procedure.

CALL IPOKE("44,"10000.OR.IPEEK("44))

0.3.33 IQSET

The IQSET function is used to make the RT-ll queue larger (i.e., add available elements to the queue). These elements are allocated out of the free space managed by the FORTRAN system. IQSET cannot be called from a completion or interrupt routine.

Form: i = IQSET (gleng)

where: gleng is the integer number of elements to be added to the queue. This argument must be positioned so that the USR does not swap over it.

All RT-ll I/O transfers are done through a centralized queue management system. If I/O traffic is very heavy and not enough queue elements are available, the program issuing the I/O requests may be suspended until a queue element becomes available. In an F/B system, the other job runs while the first program waits for the element. When IQSET is used in a program to be run in the foreground, the FRUN command must be modified to allocate space for the queue elements (see Section 0.1.4).

A general rule to follow is that each program should contain one or more queue element than the total number of I/O and timer requests that will be active simultaneously. Timing functions such as ITWAIT and MKRT also cause elements to be used and must be considered when allocating queue elements for a program. Note that if synchronous I/O
System Subroutine Library

is done (i.e. IREADW/IWRITW, etc.) and no timing functions are done, no additional queue elements need be allocated. See Section 0.2 for a list of SYSLIB calls that use queue elements.

For further information on adding elements to the queue, see the assembly language .QSET request, Section 9.4.27.

Function Results:

i = 0
    Normal return.

i = 1
    Not enough free space is available for the number of queue elements to be added; no allocation was made.

Example:

IF(IQSET(5).NE.0) STOP 'NOT ENOUGH FREE SPACE FOR QUEUE ELEMENTS'

IRAD50

0.3.34 IRAD50

The IRAD50 function converts a specified number of ASCII characters to Radix-50 and returns the number of characters converted. Conversion stops on the first non-Radix-50 character encountered in the input or when the specified number of ASCII characters have been converted.

Form: n = IRAD50 (icnt,input,output)

where: icnt is the number of ASCII characters to be converted.

      input is the area from which input characters are taken.

      output is the area into which Radix-50 words are stored.

Three characters of text are packed into each word of output. The number of output words modified is computed by the expression (in integer words):

      (icnt+2)/3

Thus, if a count of 4 is specified, two words of output are written even if only a 1-character input string is given as an argument.
System Subroutine Library

Function Results:
The integer number of input characters actually converted (n) is returned as the function result.

Example:

    REAL*8 FSPEC
    CALL IRAD50(12,'SY0TEMP DAT',FSPEC)

IRCVD/IRCVDC/IRCVDF/IRCVDW

0.3.35 IRCVD/IRCVDC/IRCVDF/IRCVDW(F/B only)

There are four forms of the receive data function; these are used in conjunction with the ISDAT (send data) functions to allow a general data/message transfer system. The receive data functions issue RT-11 receive data programmed requests (see Section 9.4.29). These functions require a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

IRCVD

The IRCVD function is used to receive data and continue execution. The operation is queued and the issuing job continues execution. At some point when the job must receive the transmitted message, an MWAIT should be executed. This causes the job to be suspended until the message has been received.

Form:  i = IRCVD (buff,wcnt)

where:  buff is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVD is complete.

        wcnt is the maximum integer number of words that can be received.

Errors:

    i = 0    Normal return.
    = 1    No foreground job exists in the system.
System Subroutine Library

Example:

```
INTEGER*2 MSG(41)
.
.
CALL IRCVD(MSG,40)
.
.
CALL MWAIT
```

IRCVDC

The IRCVDC function receives data and enters an assembly language completion routine when the message is received. The IRCVDC is queued and program execution stays with the issuing job. When the other job sends a message, the completion routine specified is entered.

Form: \( i = \text{IRCVDC}(\text{buff},wcnt,crtn) \)

where: \( \text{buff} \) is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVDC is complete.

\( \text{wcnt} \) is the maximum integer number of words to be received.

\( \text{crtn} \) is the assembly language completion routine to be entered. This name must be specified in a FORTRAN EXTERNAL statement in the routine that issues the IRCVDC call.

Errors:

- \( i = 0 \) Normal return.
- \( i = 1 \) No foreground job exists in the system.

IRCVDF

The IRCVDF function receives data and enters a FORTRAN completion subroutine (see Section 0.2.1) when the message is received. The IRCVDF is queued and program execution continues with the issuing job. When the other job sends a message, the FORTRAN completion routine specified is entered.

Form: \( i = \text{IRCVDF}(\text{buff},wcnt,area,crtn) \)

where: \( \text{buff} \) is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVDF is complete.
System Subroutine Library

**wcnt** is the maximum integer number of words to be received.

**area** is a 4-word area to be set aside for linkage information. This area must not be modified by the FORTRAN program and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion routines when **crttn** has been entered.

**crttn** is the FORTRAN completion routine to be entered. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IRCVDF call.

**Errors:**

\[ i = 0 \quad \text{Normal return.} \]
\[ i = 1 \quad \text{No foreground job exists in the system.} \]

**Example:**

```
INTEGER*2 MSG(41),AREA(4)
EXTERNAL RMSGRT

CALL IRCVDF(MSG,40,AREA,RMSGRT)
```

**IRCVDF**

The IRCVDF function is used to receive data and wait. This function queues a message request and suspends the job issuing the request until the other job sends a message. When execution of the issuing job resumes, the message has been received, and the first word of the buffer indicates the number of words which were transmitted.

**Form:** \( i = \text{IRCVDF} \left( \text{buff,wcnt} \right) \)

**where:**

- **buff** is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word will contain the integer number of words actually transmitted when IRCVDF is complete.

- **wcnt** is the maximum integer number of words to be received.

**Errors:**

\[ i = 0 \quad \text{Normal return.} \]
\[ i = 1 \quad \text{No foreground job exists in the system.} \]

**Example:**

```
INTEGER*2 MSG(41)
IF(IRCVDF(MSG,40).NE.0) STOP 'UNEXPECTED ERROR'
```
0.3.36 IREAD/IREADC/IREADF/IREADW

SYSLIB provides four modes of I/O: IREAD/IWRITE, IREADC/IWRITC, IREADF/IWRITF, and IREADW/IWRITW. Section 0.3.52 explains the output operations. These functions require a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

IREAD

The IREAD function transfers a specified number of words from the file associated with the indicated channel into memory. Control returns to the user program immediately after the IREAD function is initiated. No special action is taken when the transfer is completed.

Form:  \( i = \text{IREAD}(\text{wcnt}, \text{buff}, \text{blk}, \text{chan}) \)

where:

- \( \text{wcnt} \) is the integer number of words to be transferred.
- \( \text{buff} \) is the array to be used as the buffer. This array must contain at least \( \text{wcnt} \) words.
- \( \text{blk} \) is the integer block number of the file to be read. The user program normally updates \( \text{blk} \) before it is used again.
- \( \text{chan} \) is the integer specification for the RT-11 channel to be used.

NOTE

The "blk" argument must be updated, if necessary, by the user program. For example, if reading two blocks at a time, update \( \text{blk} \) by 2.

When the user program needs to access the data read on the specified channel, an IWAIT function should be issued. This ensures that the IREAD operation has been completed. If an error occurred during the transfer, the IWAIT function indicates the error.
System Subroutine Library

Errors:

\[ i = n \]

Normal return; \( n \) equals the number of words read
(0 for nonfile-structured read, multiple of 256
for file-structured read). For example:

If wcnt is a multiple of 256 and less than
that number of words remain in the file, \( n \) is
shortened to the number of words that remain
in the file; e.g., if wcnt is 512 and only
256 words remain, \( i = 256 \).

If wcnt is not a multiple of 256 and more
than wcnt words remain in the file, \( n \) is
rounded up to the next block; e.g., if wcnt
is 312 and more than 312 words remain, \( i =
512 \), but only 312 are read.

If wcnt is not a multiple of 256 and less
than wcnt words remain in the file, \( n \) equals
a multiple of 256 that is the actual number
of words being read.

\[ = -1 \]

Attempt to read past end-of-file; no words remain
in the file.

\[ = -2 \]

Hardware error occurred on channel.

\[ = -3 \]

Specified channel is not open.

Example:

```
INTEGER*2 BUFFER(256),RCODE,BLK
.
.
RCODE = IREAD(256,BUFFER,BLK,ICHAN)
IF(RCODE+1) 1010,1000,10
C IF NO ERROR, START HERE
10   .
.
.
IF(IWAIT(ICHAN).NE.0) GOTO 1010
.
.
1000 CONTINUE
C END OF FILE PROCESSING
.
.
CALL EXIT    !NORMAL END OF PROGRAM
1010 STOP 'FATAL READ'
END
```
System Subroutine Library

IREADC

The IREADC function transfers a specified number of words from the indicated channel into memory. Control returns to the user program immediately after the IREADC function is initiated. When the operation is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form: \[ i = \text{IREADC} \left( wcnt, \text{buff}, \text{blk}, \text{chan}, \text{crtn} \right) \]

where: \( wcnt \) is the integer number of words to be transferred.

\( \text{buff} \) is the array to be used as the buffer. This array must contain at least \( wcnt \) words.

\( \text{blk} \) is the integer block number of the file to be read. The user program normally updates \( \text{blk} \) before it is used again.

\( \text{chan} \) is the integer specification for the RT-11 channel to be used.

\( \text{crtn} \) is the assembly language routine to be activated when the transfer is complete. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IREADC call.

Errors:

See Errors under IREAD.

Example:

\[
\begin{align*}
\text{INTEGER*2} & \quad \text{IBUF}(256), \text{RCODE}, \text{IBLK} \\
\text{EXTERNAL} & \quad \text{RDCMP} \\
\vdots & \quad \vdots \\
\text{RCODE} &= \text{IREADC}(256, \text{IBUF}, \text{IBLK}, \text{ICHAN}, \text{RDCMP})
\end{align*}
\]

IREADF

The IREADF function transfers a specified number of words from the indicated channel into memory. Control returns to the user program immediately after the IREADF function is initiated. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine (see Section 0.2.1).

Form: \[ i = \text{IREADF} \left( wcnt, \text{buff}, \text{blk}, \text{chan}, \text{area}, \text{crtn} \right) \]

where: \( wcnt \) is the integer number of words to be transferred.

\( \text{buff} \) is the array to be used as the buffer. This array must contain at least \( wcnt \) words.
System Subroutine Library

blk
is the integer block number of the file to be used. The user program normally updates blk before it is used again.

chan
is the integer specification for the RT-11 channel to be used.

area
is a 4-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program or swapped over by the USR. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.

crtn
is the FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the routine that issues the IREADF call. The subroutine has two arguments:

SUBROUTINE crtn (iarg1,iarg2)

iarg1
is the channel status word (see Section 9.4.34) for the operation just completed. If bit 0 is set, a hardware error occurred during the transfer.

iarg2
is the octal channel number used for the operation just completed.

Errors:

See Errors under IREAD.

Example:

INTEGER*2 DBLK(4),BUFFER(256),BLKNO
DATA DBLK/3RDX0,3RINP,3RUT,3RDAT/,BLKNO/0/
EXTERNAL RCMLT

ICHAN=IGETC()
IF(ICHAN.LT.0) STOP 'NO CHANNEL AVAILABLE'
IF(IFETCH(DBLK).NE.0) STOP 'BAD FETCH'
IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'

20 IF(IREADF(256,BUFFER,BLKNO,ICHAN,DBLK,RCMLT).LT.0) GOTO 100
C PERFORM OVERLAP PROCESSING

C SYNCHRONIZER
CALL IWAIT(ICHAN) !WAIT FOR COMPLETION ROUTINE TO RUN
BLKNO=BLKNO+1 !UPDATE BLOCK NUMBER
GOTO 20

. . .
System Subroutine Library

C END OF FILE PROCESSING
100 CALL CLOSEC(ICHAN)
    CALL IFREEC(ICHAN)
    .
    .
    CALL EXIT
END
SUBROUTINE RCMPLT(I,J)
C THIS IS THE COMPLETION ROUTINE
    .
    .
RETURN
END

IREADW

The IREADW function transfers a specified number of words from the indicated channel into memory. Control returns to the user program when the transfer is complete or when an error is detected.

Form: i = IREADW (wcnt,buff,blk,chan)

where: wcnt is the integer number of words to be transferred.

buff is the array to be used as the buffer. This array must contain at least wcnt words.

blk is the integer block number of the file to be read. The user program normally updates blk before it is used again.

chan is the integer specification for the RT-11 channel to be used.

Errors:

See Errors under IREAD.

Example:

INTEGER*2 IBUF(1024)
.
.
ICODE=IREADW(1024,IBUF,IBLK,ICHAN)
IF(ICODE.EQ.-1) GOTO 100 ! END OF FILE PROCESSING AT 100
IF(ICODE.LT.-1) GOTO 200 ! ERROR PROCESSING AT 200
C MODIFY BLOCKS
C.
C WRITE THEM OUT
C ICODE=IWRITW(1024,IBUF,IBLK,ICHAN)
0.3.37 IRENAM

The IRENAM function causes an immediate change of the name of a specified file. An error occurs if the channel specified is already open.

Form: \( i = \text{IRENAM} \ (\text{chan}, \text{dblk}) \)

where: \( \text{chan} \) is the integer specification for the RT-11 channel to be used for the operation.

\( \text{dblk} \) is the 8-word area specifying the name of the existing file and the new name to be assigned. If considered as an 8-element INTEGER*2 array, \( \text{dblk} \) has the form:

\[
\text{dblk}(1) - \text{dblk}(4) \quad \text{specify the Radix-50 file descriptor for the old file name.}
\]

\[
\text{dblk}(5) - \text{dblk}(8) \quad \text{specify the Radix-50 file descriptor for the new file name.}
\]

NOTE

The arguments of IRENAM must be positioned so that the USR does not swap over them.

If a file with the same name as the new file name specified already exists on the indicated device, it is deleted. The specified channel is left closed when the IRENAM is complete. IRENAM requires that the handler to be used be resident at the time the IRENAM is issued. If it is not, a monitor error occurs. The device names specified in the file descriptors must be the same.

For more information on renaming files, see the assembly language .RENAME request, Section 9.4.32.

Errors:

\[
i = 0 \quad \text{Normal return.}
\]

\[
i = 1 \quad \text{Specified channel is already open.}
\]

\[
i = 2 \quad \text{Specified file was not found.}
\]
Example:

REAL*8 NAME(2)
DATA NAME/12RDK0PTN2 DAT,12RDK0PTN2 OLD/
.
.
ICHAN=IGETC()
IF(ICHAN.LT.0) STOP 'NO CHANNEL'
CALL IRENAME (ICHAN, NAME) IPRESERVE OLD DATA FILE
CALL IFREEC (ICHAN)

IREOPN

0.3.38 IREOPN

The IREOPN function reassociates a specified channel with a file on which an ISAVES was performed (see Section 0.3.39). The ISAVES/IREOPN combination is useful when a large number of files must be operated on at one time. As many files as are needed can be opened with LOOKUP and their status preserved with ISAVES. When data is required from a file, an IREOPN enables the program to read from the file. The IREOPN need not be done on the same channel as the original LOOKUP and ISAVES.

Form:  \( i = \text{IREOPN}(\text{chan}, \text{cblk}) \)

where:  chan  is the integer specification for the RT-11 channel to be associated with the reopened file. This channel must be initially in an inactive state.

cblk  is the 5-word block where the channel status information was stored by a previous ISAVES. This block, considered as a 5-element INTEGER*2 array, has the following format:

\[
\begin{align*}
\text{cblk}(1) & : \text{Channel status word (see Section 9.4.33).} \\
\text{cblk}(2) & : \text{Starting block number of the file; zero for nonfile-structured devices.} \\
\text{cblk}(3) & : \text{Length of file (in 256-word blocks).} \\
\text{cblk}(4) & : \text{(Reserved for future use.)} \\
\text{cblk}(5) & : \text{Two information bytes. Even byte: I/O count of the number of requests made on this channel. Odd byte: unit number of the device associated with the channel.}
\end{align*}
\]
System Subroutine Library

Errors:

\[
i = 0 \quad \text{Normal return.}
\]
\[
i = 1 \quad \text{Specified channel is already in use.}
\]

Example:

\[
\text{INTEGER*2 SAVES(5,10)}
\]
\[
\text{DATA ISVPTR/1/}
\]
\[
\vdots
\]
\[
\text{CALL ISAVES(ICHAN,SAVES(1,ISVPTR))}
\]
\[
\vdots
\]
\[
\text{CALL IREOPN(ICHAN,SAVES(1,ISVPTR))}
\]

0.3.39 ISAVES

The ISAVES function stores five words of channel status information into a user-specified array. These words contain all the information that RT-11 requires to completely define a file. When an ISAVES is finished, the data words are placed in memory and the specified channel is closed and is again available for use. When the saved channel data is required, the IREOPN function (Section 0.3.38) is used.

ISAVES can be used only if a file was opened with a LOOKUP call (see Section 0.3.66). If IENTER was used, ISAVES is illegal and returns an error. Note that ISAVES is not legal on magtape or cassette files.

Form: \( i = \text{ISAVES} \,(\text{chan}, \text{cblk}) \)

where: \( \text{chan} \) is the integer specification for the RT-11 channel whose status is to be saved.

\( \text{cblk} \) is a 5-word block into which the channel status information describing the open file is stored. See Section 0.3.38 for the format of this block.

The ISAVES/IREOPN combination is very useful, but care must be observed when using it. In particular, the following cases should be avoided:

1. If an ISAVES is performed on a file and the same file is then deleted before it is reopened, the space occupied by the file becomes available as an empty space which could then be used
System Subroutine Library

by the IENTER function. If this sequence occurs, the
contents of the file whose status was supposedly saved will
change.

2. Although the handler for the required peripheral need not be
in memory for execution of an IREOPEN, a fatal error is
generated if the handler is not in memory when an IREAD or
IWRITE is executed.

Errors:

i = 0 Normal return.
i = 1 The specified channel is not currently associated
with any file.
i = 2 The file was opened with an IENTER call; an
ISAVES is illegal.

Example:

INTEGER*2 BLK(5)
.
.
.
IF(ISAVES(ICHAN,BLK).NE.0) STOP 'ISAVES ERROR'

0.3.40 ISCHED (F/B Only)

The ISCHED function schedules a specified FORTRAN subroutine to be run
as an asynchronous completion routine at a specified time-of-day.

Form:  i = ISCHED (hrs,min,sec,tick,area,id,crtn)

where: hrs is the integer number of hours.

min is the integer number of minutes.

sec is the integer number of seconds.

tick is the integer number of ticks (1/60 of a
second on 60-cycle clocks; 1/50 of a second
on 50-cycle clocks).

area is a 4-word-area which must be provided for
linkage information; this area must never be
modified by the FORTRAN program, and the USR
must not swap over it. This area may be
reclaimed by other FORTRAN completion
functions when crtn has been activated.
id is the identification integer to be passed to the routine being scheduled.

crtm is the name of the FORTRAN subroutine to be entered at the time-of-day specified. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISCHED call. The subroutine has one argument, e.g.,

SUBROUTINE crtn(id)
    INTEGER id

When the routine is entered, the value of the integer argument will be the value specified for id in the appropriate ISCHED call.

Notes:

1. The scheduling request made by this function may be cancelled at a later time by an ICMKT function call.

2. If the system is busy, the actual time-of-day that the completion routine is run may be greater than the requested time-of-day.

3. A FORTRAN subroutine may periodically reschedule itself by issuing its own ISCHED or ITIMER calls from within the routine.

4. ISCHED requires a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

Errors:

i = 0 Normal return.
    = 1 No queue elements available; unable to schedule request.

Example:

INTEGER*2 LINK(4)  !LINKAGE AREA
EXTERNAL NOON  !NAME OF ROUTINE TO RUN
.
.
I=ISCHED(12,0,0,0,LINK,0,NOON)  !RUN SUBR NOON AT 12 PM
.
.
(rest of main program)
.
END

SUBROUTINE NOON(id)

C THIS ROUTINE WILL TERMINATE EXECUTION AT LUNCHTIME,
C IF THE JOB HAS NOT COMPLETED BY THAT TIME.
C STOP 'ABORT JOB -- LUNCHTIME'
END
ISDAT/ISDATC/ISDATF/ISDATW

0.3.41 ISDAT/ISDATC/ISDATF/ISDATW (P/B only)

These functions are used with the IRCVD/IRCVD/C/IRCVDW calls to allow message transfers under the P/B Monitor. Note that the buffer containing the message should not be modified or reused until the message has been received by the other job. These functions require a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

ISDAT

The ISDAT function transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued. This call is used with the MWAIT routine (see Section 0.3.68).

Form:  \( i = \text{ISDAT}(\text{buff}, \text{wcnt}) \)

where:  \( \text{buff} \) is the array containing the data to be transferred.

\( \text{wcnt} \) is the integer number of data words to be transferred.

Errors:

\( i = 0 \)  Normal return.
\( i = 1 \)  No foreground job currently exists in the system.

Example:

```
INTEGER*2 MSG(40)
.
.
CALL ISDAT(MSG,40)
.
.
CALL MWAIT
C  PUT NEW MESSAGE IN BUFFER
```
System Subroutine Library

ISDATC

The ISDATC function transfers a specified number of words from one job to another. Control returns to the user program immediately after the transfer is queued. When the other job accepts the message through a receive data request, the specified assembly language routine (crtn) is activated as an asynchronous completion routine.

Form: \( i = \text{ISDATC}(\text{buff}, \text{wcnt}, \text{crtn}) \)

where: \( \text{buff} \) is the array containing the data to be transferred.
\( \text{wcnt} \) is the integer number of data words to be transferred.
\( \text{crtn} \) is the name of an assembly language routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATC call.

Errors:

\( i = 0 \) Normal return.
\( i = 1 \) No foreground job currently exists in the system.

Example:

\begin{verbatim}
INTEGER*2 MSG(40)
EXTERNAL RTN

call ISDATC(MSG,40,RTN)
\end{verbatim}

ISDATF

The ISDATF function transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued and execution continues. When the other job accepts the message through a receive data request, the specified FORTRAN subprogram (crtn) is activated as an asynchronous completion routine (see Section 0.2.1).

Form: \( i = \text{ISDATF}(\text{buff}, \text{wcnt}, \text{area}, \text{crtn}) \)

where: \( \text{buff} \) is the array containing the data to be transferred.
\( \text{wcnt} \) is the integer number of data words to be transferred.
\( \text{area} \) is a 4-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.
System Subroutine Library

crtn is the name of a FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATF call.

Errors:

\[
i = 0 \quad \text{Normal return.} \\
i = 1 \quad \text{No foreground job currently exists in the system.}
\]

Example:

\[
\begin{align*}
\text{INTEGER*2 } & \text{MSG}(40),\text{SPOT}(4) \\
\text{EXTERNAL RTN} \\
\cdot \\
\cdot \\
\cdot \\
\text{CALL ISDATF(MSG,40,SPOT,RTN)}
\end{align*}
\]

ISDATW

The ISDATW function transfers a specified number of words from one job to the other. Control returns to the user program when the other job has accepted the data through a receive data request.

Form: \( i = \text{ISDATW}(\text{buff},\text{wcnt}) \)

where: \( \text{buff} \) is the array containing the data to be transferred.
\( \text{wcnt} \) is the integer number of data words to be transferred.

Errors:

\[
i = 0 \quad \text{Normal return.} \\
i = 1 \quad \text{No foreground job currently exists in the system.}
\]

Example:

\[
\begin{align*}
\text{INTEGER*2 } & \text{MSG}(40) \\
\cdot \\
\cdot \\
\cdot \\
\text{IF (ISDATW(MSG,40).NE.0) STOP 'FOREGROUND JOB NOT RUNNING'}
\end{align*}
\]
0.3.42 ISLEEP (F/B Only)

The ISLEEP function suspends the main program execution of a job for a specified amount of time. The specified time is the sum of hours, minutes, seconds, and ticks specified in the ISLEEP call. All completion routines continue to execute.

Form:  \( i = \text{ISLEEP}(\text{hrs}, \text{min}, \text{sec}, \text{tick}) \)

where:  
- \( \text{hrs} \) is the integer number of hours.
- \( \text{min} \) is the integer number of minutes.
- \( \text{sec} \) is the integer number of seconds.
- \( \text{tick} \) is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).

Notes:

1. ISLEEP requires a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

2. If the system is busy, the time that execution is suspended may be greater than that specified.

Errors:

\( i = 0 \)  Normal return.
\( i = 1 \)  No queue element available.

Example:

```

CALL IQSET(2)

CALL ISLEEP(0,0,0,4)  !GIVE BACKGROUND JOB SOME TIME
```

0-71
ISP FN/ISP F NC/ISP FN F/ISP FN W

0.3.43 ISP FN/ISP F NC/ISP FN F/ISP FN W

These functions are used in conjunction with special functions to various handlers (cassette, diskette, and magtape). They provide a means of doing device-dependent functions, such as rewind and backspace, to those devices. If ISP FN function calls are made to any other devices, the function call is ignored.

To use these functions, the handler must be in memory and a channel associated with a file via a non-file structured LOOKUP call. These functions require a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed. Refer to Appendix H for details of CT, DX, and MT handlers.

ISP FN

The ISP FN function queues the specified operation and immediately returns control to the user program. The IWAIT function may be used to ensure completion of the operation.

Form:  \( i = \text{ISP FN}(\text{code, chan\{,\}, wcnt, buff, blk}) \)

where:  
\begin{align*}
\text{code} & \text{ is the integer numeric code of the function to be performed (see Table 0-2).} \\
\text{chan} & \text{ is the integer specification for the RT-11 channel to be used for the operation.} \\
\text{wcnt} & \text{ is the integer number of data words in the operation.}^1 \\
\text{buff} & \text{ is the array to be used as the data buffer.}^1 \\
\text{blk} & \text{ is the integer block number of the file to be operated upon.}^1 \\
\end{align*}

\(^1\text{These optional parameters are required only when doing a write with extended record gap to MT or a read or write to DX. If specified and not required, these arguments must be 0.}\)
System Subroutine Library

Table 0-2
Special Function Codes (Octal)

<table>
<thead>
<tr>
<th>Function</th>
<th>MT</th>
<th>CT</th>
<th>DX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read absolute</td>
<td></td>
<td>377</td>
<td></td>
</tr>
<tr>
<td>Write absolute</td>
<td></td>
<td>376</td>
<td></td>
</tr>
<tr>
<td>Write absolute with deleted data</td>
<td></td>
<td>375</td>
<td></td>
</tr>
<tr>
<td>Backspace to last file</td>
<td>377</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backspace to last block</td>
<td>376</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forward to next file</td>
<td>375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forward to next block</td>
<td>374</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rewind to load point</td>
<td>373</td>
<td>373</td>
<td></td>
</tr>
<tr>
<td>Write file gap</td>
<td></td>
<td>372</td>
<td></td>
</tr>
<tr>
<td>Write EOF</td>
<td>377</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forward 1 record</td>
<td>376</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backspace 1 record</td>
<td>375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write with extended record gap</td>
<td>374</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offline</td>
<td>372</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Errors:

i = 0 Normal return.
= 1 Attempt to read or write past end-.f-file.
= 2 Hardware error occurred on channel.
= 3 Channel specified is not open.

Example:

CALL ISPFN("373,ICHAN) !REWIND

ISPFNC

The ISPFNC function queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form: i = ISPFNC (code,chan,wcnt,buff,blk,crtn)

where: code is the integer numeric code of the function to be performed (see Table 0-2).

chan is the integer specification for the RT-ll channel to be used for the operation.

wcnt is the integer number of data words in the operation. This argument must be 0 if not required.

buff is the array to be used as the data buffer. This argument must be 0 if not required.

blk is the integer block number of the file to be operated upon. This argument must be 0 if not required.
System Subroutine Library

crtnt is the name of an assembly language routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNC call.

Errors:

\[ i = 0 \quad \text{Normal return.} \]
\[ i = 1 \quad \text{Attempt to read or write past end-of-file.} \]
\[ i = 2 \quad \text{Hardware error occurred on channel.} \]
\[ i = 3 \quad \text{Channel specified is not open.} \]

ISPFNF

The ISPFNF function queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified FORTRAN subprogram (crtnt) is entered as an asynchronous completion routine.

Form: \[ i = \text{ISPFNF} \left( \text{code,chan,wcnt,buff,blk,area,crtnt} \right) \]

where: code is the integer numeric code of the function to be performed (see Table 0-2).

chan is the integer specification for the RT-ll channel to be used for the operation.

wcnt is the integer number of data words in the operation. This argument must be 0 if not required.

buff is the array to be used as the data buffer. This argument must be 0 if not required.

blk is the integer block number of the file to be operated upon. This argument must be 0 if not required.

area is a 4-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area may be reclaimed by other FORTRAN completion functions when crtnt has been activated.

crtnt is the name of a FORTRAN routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNF call. The subroutine has two arguments:

\[ \text{SUBROUTINE} \text{crtnt } \left( \text{iarg1,iarg2} \right) \]

iarg1 is the channel status word (see Section 9.4.34) for the operation just completed. If bit 0 is set, a hardware error occurred during the transfer.

iarg2 is the channel number used for the operation just completed.
System Subroutine Library

Errors:

i = 0    Normal return.
i = 1    Attempt to read or write past end-of-file.
i = 2    Hardware error occurred on channel.
i = 3    Channel specified is not open.

Example:

REAL*4 MTNAME(2),AREA(2)
DATA MTNAME/3RMT0,0./
EXTERN DONSUB
.
.
I=IGETC()       !ALLOCATE CHANNEL
CALL IFETCH(MTNAME)    !FETCH MT HANDLER
CALL LOOKUP(I,MTNAME)    !NON-FILE STRUCTURED LOOKUP ON MT0
IER=ISPFNW("373,I,0,0,0,AREA,DONSUB)  !REWIND NAGTAPE
.
.
END
SUBROUTINE DONSUB
.
.
C
C    RUNS WHEN MT0 HAS BEEN REWOUND
C
.
.
END

ISPFNW

The ISPFNW function queues the specified operation and returns control
to the user program when the operation is complete.

Form:  i = ISPFNW (code,chan{,wcnt,buff,blk})

where:  code    is the integer numeric code of the function
to be performed (see Table 0-2).

chan    is the integer specification for the RT-11
channel to be used for the operation.

wcnt    is the integer number of data words in the
operation.¹

buff    is the array to be used as the data
buffer.¹

blk    is the integer block number of the file to be
operated upon.¹

¹These optional parameters are required only when doing a write
with extended record gap to MT or a read or write to DX. If specified
and not required, these arguments must be 0.
ISPY

0.3.44 ISPY

The ISPY function returns the integer value of the word at a specified offset from the RT-11 resident monitor. (See Section 9.2.6 for information on fixed offset references.)

Form:  \( i = \text{ISPY}(\text{ioff}) \)

where:  \( \text{ioff} \)  is the offset (from the base of RMON) to be examined.

Function Result:

The function result (i) is set to the value of the word examined.

Example:

C C BRANCH TO 200 IF RUNNING UNDER F/B MONITOR
C C IF(ISPY("300").AND.1) GOTO 200
C C WORD AT OCTAL 300 FROM RMON IS
C C THE CONFIGURATION WORD.
0.3.45 ITIMER (F/B Only)

The ITIMER function schedules a specified FORTRAN subroutine to be run as an asynchronous completion routine after a specified time interval has elapsed.

Form: \( i = \text{ITIMER} (\text{hrs}, \text{min}, \text{sec}, \text{tick}, \text{area}, \text{id}, \text{crtnt}) \)

where: 

- **hrs** is the integer number of hours.
- **min** is the integer number of minutes.
- **sec** is the integer number of seconds.
- **tick** is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).
- **area** is a 4-word area which must be provided for linkage information; this area must never be modified by the FORTRAN program, and the USR must never swap over it. This area may be reclaimed by other FORTRAN completion functions when crtn has been activated.
- **id** is the identification integer to be passed to the routine being scheduled.
- **crtnt** is the name of the FORTRAN subroutine to be entered when the specified time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that references ITIMER. The subroutine has one argument, e.g.,

  ```fortran
  SUBROUTINE crtn(id)
  INTEGER id
  ``

  When the routine is entered, the value of the integer argument will be the value specified for id in the appropriate ITIMER call.

---

Notes:

1. This function may be cancelled at a later time by an ICMKT function call.
System Subroutine Library

2. If the system is busy, the actual time interval at which the completion routine is run may be greater than the time interval requested.

3. FORTRAN subroutines can periodically re-schedule themselves by issuing ISCHED or ITIMER calls.

4. ITIMER requires a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

For more information on scheduling completion routines, see Section 0.2.1 and the assembly language .MRKT request, Section 9.4.22.

Errors:

\[
i = 0 \quad \text{Normal return}
\]
\[
i = 1 \quad \text{No queue elements available; unable to schedule request.}
\]

Example:

```
INTEGER*2 AREA(4)
EXTERNAL WATCHD
.
.
C IF THE CODE FOLLOWING ITIMER DOES NOT REACH THE ICMKT CALL
C IN 12 MINUTES, WATCH DOG COMPLETION ROUTINE WILL BE
C ENTERED WITH ID OF 3
C
CALL ITIMER(0,12,0,0,AREA,3,WATCHD)
.
.
CALL ICMKT(3,AREA)
.
.
END
SUBROUTINE WATCHD(ID)
C
C THIS IS CALLED AFTER 12 MINUTES
.
.
RETURN
END
```
0.3.46 ITLOCK (F/B Only)

The ITLOCK function is used in an F/B system to attempt to gain ownership of the USR. It is similar to LOCK (Section 0.3.65) in that if successful, the user job returns with the USR in memory. However, if a job attempts to LOCK the USR while the other job is using it, the requesting job is suspended until the USR is free. With ITLOCK, if the USR is not available, control returns immediately and the lock failure is indicated. ITLOCK cannot be called from a completion or interrupt routine.

Form: \( i = \text{ITLOCK}() \)

For further information on gaining ownership of the USR, see the assembly language .TLOCK request, Section 9.4.41.

Errors:

\( i = 0 \) Normal return.
\( i = 1 \) USR is already in use by another job.

Example:

\[
\text{IF(ITLOCK().NE.0) GOTO 100} \quad \text{!GOTO 100 IF USR BUSY}
\]

0.3.47 ITTINR

The ITTINR function transfers a character from the console terminal to the user program. If no characters are available, return is made with an error flag set.

Form: \( i = \text{ITTINR}() \)
System Subroutine Library

If the function result (i) is less than zero when execution of the ITTINR function is complete, it indicates that no character was available; the user has not yet typed a valid line. Under the F/B Monitor, ITTINR does not return a result of less than zero unless bit 6 of the Job Status Word was on when the request was issued (see below).

There are two modes of doing console terminal input. The mode is governed by bit 12 of the Job Status Word (JSW). The JSW is at octal location 44. If bit 12 equals 0, normal I/O is performed. In this mode, the following conditions apply:

1. The monitor echoes all characters typed; lower case characters are converted to upper case unless JSW bit 14 is set, in which case lower case characters are not translated.

2. CTRL U ("U") AND RUBOUT perform line deletion and character deletion, respectively.

3. A carriage return, line feed, CTRL Z, or CTRL C must be struck before characters on the current line are available to the program. When one of these is typed, characters on the line typed are passed one-by-one to the user program. Both carriage return and line feed are passed to the program.

4. ALTMODEs (octal codes 175 and 176) are converted to ESCAPEs (octal 33).

If bit 12 equals 1, the console is in special mode. The effects are:

1. The monitor does not echo characters typed except for CTRL C and CTRL O.

2. CTRL U and RUBOUT do not perform special functions.

3. Characters are immediately available to the program.

4. No ALTMODE conversion is done.

In special mode, the user program must echo the characters desired. However, CTRL C and CTRL O are acted on by the monitor in the usual way. Bits 12 and 14 in the JSW must be set by the user program if special console mode or lower case characters are desired (see the example under Section 0.3.32). These bits are cleared when control returns to RT-11.

NOTE

To set and/or clear bits in the JSW, do an IPEEK and then an IPOKE. In special terminal mode (JSW bit 12 set), normal FORTRAN formatted I/O from the console is undefined.

In the F/B Monitor, CTRL F and CTRL B are not affected by the setting of bit 12. The monitor always acts on these characters if the SET TTY FB command was issued.
System Subroutine Library

Under the F/B Monitor, if a terminal input request is made and no character is available, job execution is suspended until a character is ready. If a program really requires execution to continue and ITTINR to return a result of less than zero, it must turn on bit 6 of the JSW before the ITTINR. Bit 6 is cleared when a program terminates.

Function Results:

- $i > 0$ Normal return; character read.
- $i < 0$ Error return; no character available.

Example:

```
ICHAR=ITTINR()
IF(ICHAR.LT.0) GOTO 100  ! CHARACTER NOT AVAILABLE
```

0.3.48 ITTOUR

The ITTOUR function transfers a character from the user program to the console terminal if there is room for the character in the monitor buffer. If it is not currently possible to output a character, an error flag is returned.

Form: $i = \text{ITTOUR}(\text{char})$

where: $\text{char}$ is the character to be output, right-justified in the integer (may be \text{LOGICAL}*1 entity if desired).

If the function result ($i$) is 0 when execution of the ITTOUR function is complete, it indicates that there is no room in the buffer and that no character was output. Under the F/B Monitor, ITTOUR normally does not return a result of 0. Instead, the job is blocked until room is available in the output buffer. If a job really requires execution to continue and a result of 0 to be returned, it must turn on bit 6 of the JSW (location 44) before issuing the request.

The ITTINR and ITTOUR have been supplied as a help to those users who do not wish to suspend program execution until a console operation is complete. With these modes of I/O, if a no-character or no-room condition occurs, the user program can continue processing and try the operation again at a later time.

Errors:

- $i = 0$ Normal return; character was output.
- $i = 1$ Error return; ring buffer is full.

0-81
System Subroutine Library

Example:

```
DO 20 I=1,5
  IF(ITTOUR("007").NE.0) GOTO 10  !RING BELL 5 TIMES
10 CONTINUE
20  
```

### ITWAIT

0.3.49  ITWAIT (F/B Only)

The ITWAIT function suspends the main program execution of the current job for a specified time interval. All completion routines continue to execute.

Form:  \[ i = \text{ITWAIT}(\text{itime}) \]

where:  \[ \text{itime} \]  is the 2-word internal format time interval.

- \( \text{itime} (1) \) is the high-order time.
- \( \text{itime} (2) \) is the low-order time.

Notes:

1. ITWAIT requires a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

2. If the system is busy, the actual time interval that execution is suspended may be greater than the time interval specified.

Errors:

- **i = 0**  Normal return.
- **i = 1**  No queue element available.

Example:

```
INTEGER*2 TIME(2)
  :
  :
CALL ITWAIT(TIME)  !WAIT FOR TIME TIME
```
O.3.50 IUNTIL (F/B Only)

The IUNTIL function suspends main program execution of the job until the time-of-day specified. All completion routines continue to run.

Form: i = IUNTIL (hrs,min,sec,tick)

where: hrs is the integer number of hours.
      min is the integer number of minutes.
      sec is the integer number of seconds.
      tick is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).

Notes:

1. IUNTIL requires a queue element; this should be considered when the IQSET function (Section 0.3.33) is executed.

2. If the system is busy, the actual time-of-day that the program resumes execution may be later than that requested.

Errors:

i = 0 Normal return.
    = 1 No queue element available.

Example:

C    TAKE A LUNCH BREAK
CALL IUNTIL(13,0,0,0)   !START UP AGAIN AT 1 P.M.
**IWAIT**

0.3.51 IWAIT

The IWAIT function suspends execution of the main program until all input/output operations on the specified channel are complete. This function is used with IREAD, IWRITE, and ISPFN calls. Completion routines continue to execute.

Form: \[ i = \text{IWAIT}(\text{chan}) \]

where: \( \text{chan} \) is the integer specification for the RT-11 channel to be used.

For further information on suspending execution of the main program, see the assembly language .WAIT request, Section 9.4.46.

Errors:

- \( i = 0 \) Normal return.
- \( i = 1 \) Channel specified is not open.
- \( i = 2 \) Hardware error occurred during the previous I/O operation on this channel.

Example:

\[ \text{IF(IWAIT(ICHAN).NE.0) CALL IOERR(4)} \]

---

**IWRITC/IWRITE/IWRITF/IWRITW**

0.3.52 IWRITC/IWRITE/IWRITF/IWRITW

These functions transfer a specified number of words from memory to the specified channel. The IWRITE functions require queue elements; this should be considered when the IQSET function (Section 0.3.33) is executed.
System Subroutine Library

IWRITC

The IWRITC function transfers a specified number of words from memory to the specified channel. The request is queued and control returns to the user program. When the transfer is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form:  \( i = \text{IWRITC} (\text{wcnt}, \text{buff}, \text{blk}, \text{chan}, \text{crtn}) \)

where:

<table>
<thead>
<tr>
<th>wcnt</th>
<th>is the integer number of words to be transferred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>buff</td>
<td>is the array to be used as the output buffer.</td>
</tr>
<tr>
<td>blk</td>
<td>is the integer block number of the file to be written. The user program normally updates blk before it is used again.</td>
</tr>
<tr>
<td>chan</td>
<td>is the integer specification for the RT-11 channel to be used.</td>
</tr>
<tr>
<td>crtn</td>
<td>is the name of the assembly language routine to be activated upon completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IWRITC call.</td>
</tr>
</tbody>
</table>

NOTE

The "blk" argument must be updated, if necessary, by the user program.
For example, if writing two blocks at a time, update blk by 2.

Errors:

\( i = n \)  

Normal return; \( n \) equals the number of words written, rounded to a multiple of 256 (0 for nonfile-structured writes).

NOTE

If the word count returned is less than that requested, an implied end-of-file has occurred although the normal return is indicated.

\( i = -1 \)  

Attempt to write past end-of-file; no more space is available in the file.

\( i = -2 \)  

Hardware error occurred.

\( i = -3 \)  

Channel specified is not open.
System Subroutine Library

Example:

```
INTEGER*2 IBUF(256)
EXTERNAL CRTN
.
.
ICODE=IWRITC(256,IBUF,IBLK,ICHAN,CRTN)
```

IWRITE

The IWRITE function transfers a specified number of words from memory to the specified channel. Control returns to the user program immediately after the request is queued. No special action is taken upon completion of the operation.

Form: \( i = \text{IWRITE} (\text{wcnt}, \text{buff}, \text{blk}, \text{chan}) \)

where: \( \text{wcnt} \) is the integer number of words to be transferred.

\( \text{buff} \) is the array to be used as the output buffer.

\( \text{blk} \) is the integer block number of the file to be written. The user program normally updates blk before it is used again.

\( \text{chan} \) is the integer specification for the RT-11 channel to be used.

Errors:

See Errors under IWRITC.

Example:

See the example under IREAD, Section 0.3.36.

IWRITF

The IWRITF function transfers a number of words from memory to the specified channel. The transfer request is queued and control returns to the user program. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine (see Section 0.2.1).

Form: \( i = \text{IWRITF} (\text{wcnt}, \text{buff}, \text{blk}, \text{chan}, \text{area}, \text{crtn}) \)

where: \( \text{wcnt} \) is the integer number of words to be transferred.

\( \text{buff} \) is the array to be used as the output buffer.

\( \text{blk} \) is the integer block number of the file to be written. The user program normally updates blk before it is used again.

\( \text{chan} \) is the integer specification for the RT-11 channel to be used.

JADD

O.3.53 JADD

The JADD function computes the sum of two INTEGER*4 values.

Form: \( i = \text{JADD}(\text{jopr1, jopr2, jres}) \)

where: \( \text{jopr1} \) is an INTEGER*4 variable.
\( \text{jopr2} \) is an INTEGER*4 variable.
\( \text{jres} \) is an INTEGER*4 variable that receives the sum of \( \text{jopr1} \) and \( \text{jopr2} \).

Function Results:

\begin{align*}
& i = -2 & \text{An overflow occurred while computing the result.} \\
& i = -1 & \text{Normal return; the result is negative.} \\
& i = 0 & \text{Normal return; the result is zero.} \\
& i = 1 & \text{Normal return; the result is positive.}
\end{align*}

Example:

\begin{verbatim}
INTEGER*4 JOP1,JOP2,JRES
.
.
IF(JADD(JOP1,JOP2,JRES).EQ.-2) GOTO 100
\end{verbatim}

JAFIX

O.3.54 JAFIX

The JAFIX function converts a REAL*4 value to INTEGER*4.

Form: \( i = \text{JAFIX}(\text{asrc, jres}) \)
System Subroutine Library

where:  
    asrc  
        is a REAL*4 variable, constant, or expression
to be converted to INTEGER*4.

    jres  
        is an INTEGER*4 variable that is to contain
the result of the conversion.

Function Results:

i = -2  An overflow occurred while computing the result.
    = -1  Normal return; the result is negative.
    = 0   Normal return; the result is zero.
    = 1   Normal return; the result is positive.

Example:

    INTEGER*4 JOPL
    C
    READ A LARGE INTEGER FROM THE TERMINAL
    ACCEPT 99,A
    99 FORMAT (F15.0)
    IF(JAFIX(A,JOPL).EQ.-2) GOTO 100
    .
    .

0.3.55 JCMP

The JCMP function compares two INTEGER*4 values and returns an
INTEGER*2 value that reflects the signed comparison result.

Form:  
i = JCMP (jopr1,jopr2)

    where:  
        jopr1  
            is the INTEGER*4 variable or array element
            that is the first operand in the comparison.
        jopr2  
            is the INTEGER*4 variable or array element
            that is the second operand in the comparison.

Function Result:

i = -1  If jopr1 is less than jopr2
    = 0   If jopr1 is equal to jopr2
    = 1   If jopr1 is greater than jopr2

Example:

    INTEGER*4 JOPX,JOPY
    .
    .
    IF(JCMP(JOPX,JOPY)) 10,20,30
System Subroutine Library

**JDFIX**

0.3.56 JDFIX

The JDFIX function converts a REAL*8 (DOUBLE PRECISION) value to INTEGER*4.

Form: \( i = \text{JDFIX}(\text{dsr}, \text{jres}) \)

where: \( \text{dsr} \) is a REAL*8 variable, constant, or expression to be converted to INTEGER*4.

\( \text{jres} \) is an INTEGER*4 variable to contain the conversion result.

Function Results:

\[
\begin{array}{ll}
  i = -2 & \text{An overflow occurred while computing the result.} \\
  = -1 & \text{Normal return; the result is negative.} \\
  = 0 & \text{Normal return; the result is zero.} \\
  = 1 & \text{Normal return; the result is positive.}
\end{array}
\]

Example:

```
INTEGER*4 JNUM
REAL*8 DPNUM
.
.
20 TYPE 98
98 FORMAT(' ENTER POSITIVE INTEGER')
ACCEPT 99,DPNUM
99 FORMAT(F20.0)
IF(JDFIX(DPNUM,JNUM).LT.0) GOTO 20
.
.
```

0-90
0.3.57 JDIV

The JDIV function computes the quotient of two INTEGER*4 values.

Form:  
\[ i = \text{JDIV}(\text{jopr1}, \text{jopr2}, \text{jres}, \text{jrem}) \]

where:  
- \text{jopr1} is an INTEGER*4 variable that is the dividend of the operation.
- \text{jopr2} is an INTEGER*4 variable that is divisor of \text{jopr1}.
- \text{jres} is an INTEGER*4 variable that receives the quotient of the operation (i.e., \text{jres} = \text{jopr1}/\text{jopr2}).
- \text{jrem} is an INTEGER*4 variable that receives the remainder of the operation. The sign is the same as that for \text{jopr1}.

Function Results:

\begin{align*}
\text{i} &= -3 & \text{An attempt was made to divide by zero.} \\
&= -2 & \text{(not used)} \\
&= -1 & \text{Normal return; the quotient is negative.} \\
&= 0 & \text{Normal return; the quotient is zero.} \\
&= 1 & \text{Normal return; the quotient is positive.}
\end{align*}

Example:

\begin{verbatim}
INTEGER*4 JN1,JN2,JQUO

: : :
CALL JDIV(JN1,JN2,JQUO)
: : :
\end{verbatim}
JICVT

0.3.58  JICVT

The JICVT function converts a specified INTEGER*2 value to INTEGER*4.

Form:  \( i = \text{JICVT} (\text{isrc}, \text{jres}) \)

where:  \( \text{isrc} \) is the INTEGER*2 quantity to be converted.
        \( \text{jres} \) is the INTEGER*4 variable or array element
        which will receive the result.

Function Results:

\( i = -1 \)
Normal return; the result is negative.

\( i = 0 \)
Normal return; the result is zero.

\( i = 1 \)
Normal return; the result is positive.

Example:

INTEGER*4 JVAL
CALL JICVT(478,JVAL)  !FORM A 32-BIT CONSTANT

JJCVT

0.3.59  JJCVT

The JJCVT function interchanges words of an INTEGER*4 value to form an
internal format time or vice versa. This is necessary when the
INTEGER*4 variable is to be used as an argument in a timer-support
function such as ITWAIT. When a 2-word internal format time is
specified to a function such as ITWAIT, it must have the high-order
time as the first word and the low-order time as the second word.

Form:  CALL JJCVT (jsrc)
System Subroutine Library

where: jsrc is the INTEGER*4 variable whose contents are to be interchanged.

Errors:
None.

Example:

INTEGER*4 TIME
:
.
CALL GTIM(TIME) !GET TIME OF DAY
CALL JJCVT(TIME) !TURN IT INTO INTEGER*4 FORMAT

0.3.60 JMOV

The JMOV function assigns the value of an INTEGER*4 variable to another INTEGER*4 variable and returns the sign of the value moved.

Form: i = JMOV (jsrc,jdest)

where: jsrc is the INTEGER*4 variable whose contents are to be moved.

jdest is the INTEGER*4 variable which is the target of the assignment.

Function Result:

The value of the function is an INTEGER*2 value which represents the sign of the result as follows:

i = -1 Normal return; the result is negative.
    = 0 Normal return; the result is zero.
    = 1 Normal return; the result is positive.

Example:

The JMOV function allows an INTEGER*4 quantity to be compared with zero by using it in a logical IF statement, e.g.,

INTEGER*4 INT1
:
.
.
IF(JMOV(INT1,INT1)) 300,100,300 !GO TO STMT 300 IF INT1 IS NOT 0

0-93
0.3.61 JMUL

The JMUL function computes the product of two INTEGER*4 values.

Form: \( i = \text{JMUL}(jopr1, jopr2, jres) \)

where

- \( jopr1 \) is an INTEGER*4 variable that is the multiplicand.
- \( jopr2 \) is an INTEGER*4 variable that is the multiplier.
- \( jres \) is an INTEGER*4 variable that receives the product of the operation.

Function Results:

- \( i = 2 \) An overflow occurred while computing the result.
- \( i = 1 \) Normal return; the product is negative.
- \( i = 0 \) Normal return; the product is zero.
- \( i = 1 \) Normal return; the product is positive.

Example:

\[
\text{INTEGER*4 } J1, J2, JRES \\
\text{...} \\
\text{IF}(\text{JMUL}(J1, J2, JRES)+1) \text{ 100, 10, 20} \\
\text{C GOTO 100 IF OVERFLOW} \\
\text{C GOTO 10 IF RESULT IS NEGATIVE} \\
\text{C GOTO 20 IF RESULT IS POSITIVE OR ZERO}
\]
System Subroutine Library

0.3.62 JSUB

The JSUB function computes the difference between two INTEGER*4 values.

Form: i = JSUB (jopr1,jopr2,jres)

where: jopr1 is an INTEGER*4 variable that is the minuend of the operation.

jopr2 is an INTEGER*4 variable that is the subtrahend of the operation.

jres is an INTEGER*4 variable that is to receive the difference between jopr1 and jopr2 (i.e., jres = jopr1 - jopr2).

Function Results:

i = -2 An overflow occurred while computing the result.
= -1 Normal return; the result is negative.
= 0 Normal return; the result is zero.
= 1 Normal return; the result is positive.

Example:

INTEGER*4 JOP1,JOP2,J3
::
::
CALL JSUB(JOP1,JOP2,J3)
JTIME

0.3.63 JTIME

The JTIME subroutine converts the time specified to the internal 2-word format time.

Form: CALL JTIME (hrs, min, sec, tick, time)

where: hrs is the integer number of hours.

min is the integer number of minutes.

sec is the integer number of seconds.

tick is the integer number of ticks (1/60 of a second for 60-cycle clocks; 1/50 of a second for 50-cycle clocks).

time is the 2-word area to receive the internal format time: time(1) is the high-order time. time(2) is the low-order time.

Errors:

None.

Example:

INTEGER*4 J1
.
.
.
C CONVERT 3 HRS, 7 MIN, 23 SECONDS TO INTEGER *4 VALUE
CALL JTIME(3,7,23,0,J1)
CALL JJCVT(J1)
0.3.64  LEN

The LEN function returns the number of characters currently in the string contained in a specified array. This number is computed as the number of characters preceding the first null byte encountered. If the specified array contains a null string, a value of zero is returned.

Form:  \( i = \text{LEN}(a) \)

where:  \( a \) specifies the array containing the string; \( a \)
must not be a quoted-string literal.

Errors:
None.

Example:

```
LOGICAL*1 STRING(73)
  :
  :
  TYPE 99,(STRING(I),I=1,LEN(STRING))
  99 FORMAT('0',132A1)
```

0.3.65  LOCK

The LOCK subroutine is issued to "lock" the USR in memory for a series of operations. The USR (User Service Routine) is the section of the RT-11 system which performs various file management functions.
System Subroutine Library

If all the conditions which cause swapping are satisfied, a portion of the user program is written into scratch blocks and the USR is loaded. Otherwise, the USR which is in memory is used, and no swapping occurs. The USR is not released until an UNLOCK (see Section 0.3.88) is given. (Note that in an F/B System, calling the CSI may also perform an implicit UNLOCK.) A program which has many USR requests to make can LOCK the USR in memory, make all the requests, and then UNLOCK the USR; no time is spent doing unnecessary swapping.

In an F/B environment, a LOCK inhibits the other job from using the USR. Thus, the USR should be locked only as long as necessary.

Form: CALL LOCK

Note that the LOCK routine reduces time spent in file handling by eliminating the swapping of the USR. If the USR is currently resident, LOCK involves no I/O. After a LOCK has been executed, the UNLOCK routine must be executed to release the USR from memory. The LOCK/UNLOCK routines are complementary and must be matched. That is, if three LOCKs are issued, at least three UNLOCKS must be done, otherwise the USR will not be released. More UNLOCKS than LOCKs may occur without error; the extra UNLOCKS are ignored.

Notes:

1. It is vital that the LOCK call not come from within the area into which the USR will be swapped. If this should occur, the return from the USR request would not be to the user program, but to the USR itself, since the LOCK function causes part of the user program to be saved on disk and replaced in memory by the USR. Furthermore, subroutines, variables, and arrays in the area where the USR is swapping should not be referenced while the USR is LOCKed in memory.

2. Once a LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an UNLOCK is done.

3. LOCK cannot be called from a completion or interrupt routine.

4. If a SET USR NO SWAP command has been issued, LOCK and UNLOCK will not cause the USR to swap but, in F/B, LOCK will still inhibit the other job from using the USR and UNLOCK will allow the other job access to the USR.

5. Argument lists, such as device file name specifications, being passed to the USR must not be in the area into which the USR has just been locked.

Errors:

None.
0.3.66 LOOKUP

The LOOKUP function associates a specified channel with a device and/or file for the purpose of performing I/O operations. The channel used is then busy until one of the following functions is executed.

CLOSEC
ISAVES
PURGE

Form: i = LOOKUP (chan,dblk{,count})

where: chan is the integer specification for the RT-11 channel to be associated with the file.

dblk is the 4-word area specifying the Radix-50 file descriptor. Note that unpredictable results will occur if the USR swaps over this 4-word area.

count is an optional argument used for the magtape and cassette handlers (See Appendix H).

NOTE

The arguments of LOOKUP must be positioned so that the USR does not swap over them.

The handler for the selected device must be in memory for a LOOKUP. If the first word of the file name in dblk is zero and the device is a file-structured device, absolute block 0 of the device is designated as the beginning of the "file". This technique, called a nonfile-structured lookup, allows I/O to any physical block on the device. If a file name is specified for a device which is not file-structured (i.e. LP:FILE.EXT), the name is ignored.

NOTE

Since nonfile-structured lookups allow I/O to any physical block on the device, the user must be aware that, in this mode, it is possible to overwrite the RT-11 device directory, thus destroying all file information on the device.
System Subroutine Library

Errors:

\[ \begin{align*}
  i = n & \quad \text{Normal return; } n \text{ equals the number of blocks in the file (0 for nonfile-structured lookups).} \\
  i = -1 & \quad \text{Channel specified is already open.} \\
  i = -2 & \quad \text{File specified was not found on the device.}
\end{align*} \]

Example:

\[
\begin{align*}
  \text{INTEGER*2 DBLK}(4) \\
  \text{DATA DBLK/3RDK0,3RFNT,3R44,3RDAT/} \\
  \quad \vdots \\
  \quad \vdots \\
  \quad \text{ICHAN=IGETC()} \\
  \quad \text{IF(ICHAN.LT.0) STOP 'NO CHANNEL'} \\
  \quad \text{IF(IFETCH(DBLK).NE.0) STOP 'BAD FETCH'} \\
  \quad \text{IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'} \\
  \quad \vdots \\
  \quad \text{CALL CLOSEC(ICHAN)} \\
  \quad \text{CALL IFREEC(ICHAN)} \\
  \quad \vdots
\end{align*}
\]

---

**MRKT**

0.3.67 MRKT (F/B only)

The MRKT function schedules an assembly language completion routine to be entered after a specified time interval has elapsed.

Form: \( i = \text{MRKT} \) (id, crtn, time)

where:

- \( \text{id} \) is an integer identification number to be passed to the routine being scheduled.
- \( \text{crtn} \) is the name of the assembly language routine to be entered when the time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the MRKT call.
- \( \text{time} \) is the 2-word internal format time interval; when this interval elapses, the routine will be entered. If considered as a 2-element INTEGER*2 array:

O-100
System Subroutine Library

    time (1) is the high-order
time.
    time (2) is the low-order
time.

Notes:

    1. MRKT requires a queue element; this should be considered
       when the IQSET function (Section 0.3.33) is executed.
    2. If the system is busy, the time interval that elapses before
       the completion routine is run may be greater than that
       requested.

For further information on scheduling completion routines, see the
assembly language .MRKT request, Section 9.4.22.

Errors:

    i = 0  Normal return
    = 1  No queue element was available; unable
       to schedule request.

Example:

INTEG&R*2 TINT(2)
EXTERNAL ARTN
:
.
CALL MRKT(4,ARTN,TINT)

MWAIT

0.3.68 MWAIT (F/B Only)

The MWAIT subroutine suspends main program execution of the current
job until all messages sent to or from the other job have been
transmitted or received. It provides a means for ensuring that a
required message has been processed. MWAIT is used primarily in
conjunction with the IRCVD and ISDAT calls, where no action is taken
when a message transmission is completed. This subroutine requires a
queue element; this should be considered when the IQSET function
(Section 0.3.33) is executed.

Form: CALL MWAIT

O-101
System Subroutine Library

Errors:
None.

Example:
See the example under ISDAT, Section 0.3.41.

PRINT

0.3.69 PRINT

The PRINT subroutine causes output (from a specified string) to be printed at the console terminal. This routine can be used to print messages from completion routines without using the FORTRAN formatted I/O system. Control returns to the user program after all characters have been placed in the output buffer.

The string to be printed may be terminated with either a null (0) byte or a 200 (octal) byte. If the null (ASCII) format is used, the output is automatically followed by a <CR><LF>. If a 200 byte terminates the string, no <CR><LF> is generated.

In the F/B Monitor, a change in the job that is controlling terminal output is indicated by a B> or F>. Any text following the message has been printed by the job indicated (foreground or background) until another B> or F> is printed. When PRINT is used by the foreground job, the message appears immediately, regardless of the state of the background job. Thus, for urgent messages, PRINT should be used rather than ITTOUR.

Form: CALL PRINT (string)

where: string is the string to be printed. Note that all quoted literals used in FORTRAN subroutine calls are in ASCII format as are all strings produced by the SYSLIB string handling package.

Errors:
None.

Example:

CALL PRINT ('THE COFFEE IS READY')
System Subroutine Library

0.3.70 PURGE

The PURGE subroutine is used to deactivate a channel without performing an ISAVES or a CLOSEC. Any tentative file currently associated with the channel is not made permanent. This subroutine is useful for keeping ENTERed (IENTER or .ENTER) files from becoming permanent directory entries.

Form: CALL PURGE (chan)

where: chan is the integer specification for the RT-ll channel to be deactivated.

Errors:
None.

Example:
See the example under IENTER, Section 0.3.22.

0.3.71 PUTSTR

The PUTSTR subroutine writes a variable-length character string to a specified FORTRAN logical unit. PUTSTR may be used in main program routines or in completion routines but not in both in the same program at the same time. If PUTSTR is used in a completion routine, it must not be the first I/O operation on the specified logical unit.

Form: CALL PUTSTR (lun,in,char)

where: lun is the integer specification of the FORTRAN logical unit number to which the string is to be written.
System Subroutine Library

in is the array containing the string to be written.

char is an ASCII character that will be appended to the beginning of the string before it is output. If 0, the first character of "in" will be the first character of the record. This character is used primarily for carriage control purposes (see Section 0.3.13).

Errors:
None.

Example:

LOGICAL*1 STRNG(81)
:
:
CALL PUTSTR(7,STRNG,'0') !OUTPUT STRING WITH DOUBLE SPACING

R50ASC

0.3.72 R50ASC

The R50ASC subroutine converts a specified number of Radix-50 characters to ASCII.

Form: CALL R50ASC (icnt,input,output)

where: icnt is the integer number of ASCII characters to be produced.

input is the area from which words of Radix-50 values to be converted are taken. Note that (icnt+2)/3 words are read for conversion.

output is the area into which the ASCII characters are stored.

Errors:

If an input word contains illegal Radix-50 codes, i.e., if the input word is greater (unsigned) than 174777(octal), the routine outputs question marks for the value.
System Subroutine Library

Example:

REAL*8 NAME
LOGICAL*1 OUTP(12)
.
.
CALL R50ASC(12,NAM,OUTP)

0.3.73 RAD50

The RAD50 function provides a method of encoding RT-11 file descriptors in Radix-50 notation. The RAD50 function converts six ASCII characters from the specified area, returning a REAL*4 result which is the 2-word Radix-50 value.

Form:  a = RAD50 (input)

where:  input is the area from which the ASCII input characters are taken.

The RAD50 call:

A = RAD50 (LINE)

is exactly equivalent to the IRAD50 call:

CALL IRAD50 (6,LIN,E,A)

Function Results:
The 2-word Radix-50 value is returned as the function result.

0.3.74 RCHAIN

The RCHAIN subroutine allows a program to determine whether it has been chained to and to access variables passed across a chain. If RCHAIN is used, it must be used in the first executable FORTRAN statement in a program. RCHAIN cannot be called from a completion or interrupt routine.
System Subroutine Library

Form: CALL RCHAIN (flag, var, wcnt)

where: flag is an integer variable which will be set to
-1 if the program has been chained to;
otherwise, it is 0.

var is the first variable in a sequence of
variables with increasing memory addresses to
receive the information passed across the
chain (see Section 0.3.2).

wcnt is the number of words to be moved from the
chain parameter area to the area specified by
var. RCHAIN moves "wcnt" words into the area
beginning at "var".

Errors:
None.

Example:

INTEGER*2 PARMS(50)
CALL RCHAIN(IFLAG,PARMS,50)
IF(IFLAG) GOTO 10  !GOTO 10 IF CHAINED TO
:  
:

---

0.3.75 RCTRLO

The RCTRLO subroutine resets the effect of any console terminal CTRL O
(\) command which was typed. After an RCTRLO call, any output
directed to the console terminal will print until another CTRL O is
typed.

Form: CALL RCTRLO

Errors:
None.

Example:

CALL RCTRLO
CALL PRINT ('THE REACTOR IS ABOUT TO BLOW UP')

O-106
0.3.76 **REPEAT**

The **REPEAT** subroutine concatenates a specified string with itself to produce the indicated number of copies. **REPEAT** places the resulting string in a specified array.

**Form:** CALL REPEAT (in,out,i[,len[,err]])

- **in** is the array containing the string to be repeated.
- **out** is the array into which the resultant string is placed. This array must be at least one element longer than the value of **len**, if specified.
- **i** is the integer number of times to repeat the string.
- **len** is the integer number representing the maximum length of the output string.
- **err** is the Logical error flag set if the output string is truncated to the length specified by **len**.

Input and output strings may only specify the same array if the repeat count (**i**) is 1 or 0. When the repeat count is 1, this routine is the equivalent of **SCOPY**; when the repeat count is 0, "out" is replaced by a null string. The old contents of "out" are lost when this routine is called.

**Errors:**

Error conditions are indicated by "err", if specified. If "err" is given and the output string would have been longer than "len" characters, then "err" is set to .TRUE.; otherwise, "err" is unchanged.

**Example:**

```plaintext
LOGICAL*1 SIN(21),SOUT(101)
  ...
  ...
CALL REPEAT(SIN,SOUT,5)
```
RESUME

0.3.77 RESUME (F/B Only)

The RESUME subroutine allows a job to resume execution of the main program. A RESUME call is normally issued from an asynchronous FORTRAN routine entered on I/O completion or because of a schedule request. (See Section 0.3.83 for more information.)

Form: CALL RESUME

Errors:

None.

Example:

See the example under SUSPEND, Section 0.3.83.

SCOMP

0.3.78 SCOMP

The SCOMP routine compares two character strings and returns the integer result of the comparison.

Form: CALL SCOMP (a,b,i)

or

i = ISCOMP (a,b)

where:

a is the array containing the first string.

b is the array containing the second string.

i is the integer variable that receives the result of the comparison.
System Subroutine Library

The strings are compared left-to-right, one character at a time, using the collating sequence specified by the ASCII codes for each character. If the two strings are not equal, the absolute value of variable i (or the result of the function ISCOMP) is the character position of the first inequality found in scanning left-to-right.

If the strings are not the same length, the shorter one is treated as if it were padded on the right with blanks to the length of the other string. A null string argument is equivalent to a string containing only blanks.

Notes:

References to ISCOMP may not have quoted-string literals as arguments.

Function Result:

\[
\begin{align*}
i &< 0 & \text{if } a \text{ is less than } b \\
= &0 & \text{if } a \text{ is equal to } b \\
> &0 & \text{if } a \text{ is greater than } b
\end{align*}
\]

Example:

```fortran
LOGICAL*1 INSTR(81)
  :
  :
CALL GETSTR(5,INSTR,80)
CALL SCOMP('YES',INSTR,IVAL)
IF(IVAL) GOTO 10  !IF INPUT STRING IS NOT YES GOTO 10
```

0.3.79 SCOPY

The SCOPY routine copies a character string from one array to another. Copying stops either when a null (zero) character is encountered or when a specified number of characters have been moved.

Form: CALL SCOPY (in,out,len,err))

where: in is the array containing the string to be copied.
       out is the array to receive the copied string. This array must be at least one element longer than the value of len, if specified.
System Subroutine Library

len is the integer number representing the maximum length of the output string. The effect of len is to truncate the output string to a given length, if necessary.

err is a Logical variable that receives the error indication if the output string was truncated to the length specified by len.

The input (in) and output (out) arguments may specify the same array. The string previously contained in the output array is lost when this subroutine is called.

Errors:

Error conditions are indicated by "err", if specified. If "err" is given and the output string was truncated to the length specified by "len", then "err" is set to .TRUE.; otherwise, "err" is unchanged.

Example:

SCOPY is useful for initializing strings to a constant value, for example:

LOGICAL*1 STRING(80)
CALL SCOPY('THIS IS THE INITIAL VALUE',STRING)

0.3.80 SECNDS

The SECNDS function returns the current system time, in seconds past midnight, minus the value of a specified argument. Thus, SECNDS can be used to calculate elapsed time. The value returned is single-precision floating point (REAL*4).

Form:  \( a = \text{SECNDS} \ (\text{atime}) \)

where:  \( \text{atime} \) is a REAL*4 variable, constant, or expression whose value is subtracted from the current time-of-day to form the result.

Notes:

This function does floating-point arithmetic. Elapsed times may also be calculated by using the GTIM call and the INTEGER*4 support functions.

Function Result:

The function result (a) is the REAL*4 value returned.
System Subroutine Library

Example:

```
C START OF TIMED SEQUENCE
T1=SECND5(0.)
C CODE TO BE TIMED GOES HERE
C DELTA=SECND5(T1) !DELTA IS ELAPSED TIME
```

0.3.81 STRPAD

The STRPAD routine pads a character string with rightmost blanks until that string is a specified length. This padding is done in place; the result string is contained in its original array. If the present length of the string is greater than or equal to the specified length, no padding occurs.

Form:   CALL STRPAD (a,i{,err})

where:  
a      is the string to be padded.

  i      is the integer length of the desired result string.

  err    is the Logical error flag that is set to .TRUE. if the string specified by a exceeds the value of i in length.

Errors:

Error conditions are indicated by "err", if specified. If "err" is given and the string indicated is longer than "i" characters, "err" is set to .TRUE.; otherwise, the value of "err" is unchanged.

Example:

This routine is especially useful for preparing strings to be output in A-type FORMAT fields. For example:

```
LOGICAL*1 STR(81)
  .
  .
  CALL STRPAD(STR,80) !ASSURE 80 VALID CHARACTERS
PRINT 100,(STR(I),I=1,80) !PRINT STRING OF 80 CHARACTERS
100 FORMAT(80A1)
```

O-111
0.3.82 SUBSTR

The SUBSTR routine copies a substring from a specified position in a character string. If desired, the substring may then be placed in the same array as the string from which it was taken.

Form: CALL SUBSTR (in, out, i[, len])

where:  in is the array from which the substring is taken.

out is the array to contain the substring result. This array must be one element longer than "len", if specified.

i is the integer character position in the input string of the first character of the desired substring.

len is the integer number of characters representing the maximum length of the substring.

If a maximum length (len) is not given, the substring contains all characters to the right of character position "i" in array "in". If "len" is equal to zero, "out" is replaced by the null string. The old contents of array "out" are lost when this routine is called.

Errors:

None.
0.3.83 SUSPND (F/B Only)

The SUSPND subroutine suspends main program execution of the current job and allows only completion routines (for I/O and scheduling requests) to run.

Form: CALL SUSPND

Notes:

1. The monitor maintains a suspension counter for each job. This count is decremented by SUSPND and incremented by RESUME (see Section 0.3.77). A job will actually be suspended only if this counter is negative. Thus, if a RESUME is issued before a SUSPND, the latter function will return immediately.

2. A program must issue an equal number of SUSPNDs and RESUMEs.

3. A RESUME subroutine call from the main program or from a completion routine increments the suspension counter.

4. A SUSPND subroutine call from a completion routine decrements the suspension counter but does not suspend the main program. If a completion routine does a SUSPND, the main program continues until it also issues a SUSPND, at which time it is suspended and will require two RESUMEs to proceed.

5. Because SUSPND and RESUME are used to simulate an ITWAIT (see Section 0.3.49) in the monitor, a RESUME issued from a completion routine and not matched by a previously executed SUSPND may cause the main program execution to continue past a timed wait before the entire time interval has elapsed.

For further information on suspending main program execution of the current job, see the assembly language .SPND request, Section 9.4.39.

Errors:

None.
System Subroutine Library

Example:

```
INTEGER IAREA(4)
COMMON /RDBLK/ IBUF(256)
EXTERNAL RDFIN

IF(IREADF(256,IBUF,IBLK,ICHAN,IAREA,RDFIN).NE.0) GOTO 1000 C
GOTO 1000 FOR ANY TYPE OF ERROR
C
C DO OVERLAPPED PROCESSING
C
CALL SUSPND  !SYNCHRONIZE WITH COMPLETION ROUTINE
C
END
SUBROUTINE RDFIN(IARG1,IARG2)
COMMON /RDBLK/ IBUF(256)

CALL RESUME  !CONTINUE MAIN PROGRAM
C
C END
```

**TIMASC**

0.3.84 TIMASC

The TIMASC subroutine converts a 2-word internal format time into an ASCII string of the form:

```
hh:mm:ss
```

where:
- `hh` is the 2-digit hours indication
- `mm` is the 2-digit minutes indication
- `ss` is the 2-digit seconds indication

Form: CALL TIMASC (itime,string)
System Subroutine Library

where: itime is the 2-word internal format time to be converted. itime (1) is the high-order time. itime (2) is the low-order time.

strng is the 8-element array to contain the ASCII time.

Errors:
None.

Example:
The following example determines the amount of time until 5 p.m. and prints it.

```
INTEGER*4 J1,J2,J3
LOGICAL*1 STRNG(8)
.
.
CALL JTIME(17,0,0,0,J1)
CALL GTIM(J2)
CALL JJCVT(J1)
CALL JJCVT(J2)
CALL JSUB(J1,J2,J3)
CALL JJCVT(J3)
CALL TIMASC(J3,STRNG)
TYPE 99,(STRNG(I),I=1,8)
99 FORMAT(' IT IS ',8A1,' TILL 5 P.M.')
.
.
```

0.3.85 TIME

The TIME subroutine returns the current system time-of-day as an 8-character ASCII string of the form:

```
hh:mm:ss
```

where: hh is the 2-digit hours indication

mm is the 2-digit minutes indication

ss is the 2-digit seconds indication
System Subroutine Library

Form: CALL TIME (strng)

where: strng is the 8-element array to receive the ASCII time.

Notes:
A 24-hour clock is used, e.g., 1:00 p.m. is represented as 13:00:00.

Errors:
None.

Example:

LOGICAL*1 STRING(8)

CALL TIME(STRING)
TYPE 99,(STRING(I),I=1,8)
99 FORMAT (' IT IS NOW ',8A1)

0.3.86 TRANSL

The TRANSL routine performs character translation on a specified string. The TRANSL routine requires approximately 64 words on the R6 stack for its execution. This space should be considered when allocating stack space.

Form: CALL TRANSL (in, out, r{"p})

where: in is the array containing the input string.
out is the array to receive the translated string.
r is the array containing the replacement string.
p is the array containing the characters in "in" to be translated.

The string specified by array "out" is replaced by the string specified by array "in", modified by the character translation process specified by arrays "r" and "p". If any character position in "in" contains a character which appears in the string specified by "p", it is replaced in "out" by the corresponding character from string "r".
System Subroutine Library

If the array "p" is omitted, it is assumed to be the 127 7-bit ASCII characters arranged in ascending order, beginning with the character whose ASCII code is 001. If strings "r" and "p" are given and differ in length, the longer string is truncated to the length of the shorter. If a character appears more than once in string "p", only the last occurrence is significant. A character may appear any number of times in string "r".

Errors:
None.

Examples:
The following example causes the string in array A to be copied to array B. All periods within A become minus signs, and all question marks become exclamation points.

CALL TRANSL(A,B,'-!','.?)

The following is an example of TRANSL being used to format character data.

```c
LOGICAL*1 STRING(27),RESULT(27),PATRN(27)
C SET UP THE STRING TO BE REFORMATTED
C CALL SCOPY('THE HORN BLOWS AT MIDNIGHT',STRING)
C 0000000001111111112222222
C 12345678901234567890123456
C THE HORN BLOWS AT MIDNIGHT
C NOW SET UP PATRN TO CONTAIN THE FOLLOWING PATTERN:
C 16,17,18,19,20,21,22,23,24,25,26,15,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0
C DO 10 I=16,26
  10 PATRN(I-15)=I
    PATRN(12)=15
    DO 20 I=1,14
      PATRN(I+12)=I
    PATRN(27)=0
C THE FOLLOWING CALL TO TRANSL REARRANGES THE CHARACTERS OF
C THE INPUT STRING TO THE ORDER SPECIFIED BY PATRN:
C CALL TRANSL(PATRN,RESULT,STRING)
C RESULT NOW CONTAINS THE STRING 'AT MIDNIGHT THE HORN BLOWS'
C IN GENERAL, THIS METHOD CAN BE USED TO FORMAT INPUT STRINGS
C OF UP TO 127 CHARACTERS. THE RESULTANT STRING WILL BE
C AS LONG AS THE PATTERN STRING (AS IN THE ABOVE EXAMPLE).
```
TRIM

0.3.87 TRIM

The TRIM routine shortens a specified character string by removing all trailing blanks. A trailing blank is a blank that has no non-blanks to its right. If the specified string contains all blank characters, it is replaced by the null string. If the specified string has no trailing blanks, it is unchanged.

Form: CALL TRIM (a)

where: a is the array containing the string to be trimmed.

Errors:

None.

Example:

```
LOGICAL*1 STRING(81)
ACCEPT 100,(STRING(I),I=1,80)
100 FORMAT(80A1)
CALL SCOPY(STRING,STRING,80) !MAKE ASCIZ
CALL TRIM(STRING) !TRIM TRAILING BLANKS
```
System Subroutine Library

Form: CALL UNLOCK

Notes:

1. It is important that at least as many UNLOCKS are given as LOCKs. If more LOCKs were done, the USR remains locked in memory. It is not harmful to give more UNLOCKS than are required; those that are extra are ignored.

2. When running two jobs in the P/B system, use the LOCK/UNLOCK pairs only when absolutely necessary. If one job LOCKs the USR, the other job cannot use the USR until it is UNLOCKed. Thus, the USR should not be LOCKed unnecessarily, as this may degrade performance in some cases.

3. In an P/B System, calling the CSI (ICSI) with input coming from the console terminal performs an implicit UNLOCK.

For further information on releasing the USR from memory, see the assembly language .LOCK/.UNLOCK requests, Section 9.4.20.

Errors:

None.

Example:

```

C GET READY TO DO MANY USR OPERATIONS
CALL LOCK 1DISABLE USR SWAPPING
C PERFORM THE USR CALLS

C FREE THE USR
CALL UNLOCK
```

0.3.89 VERIFY

The VERIFY routine determines whether each character of a specified string occurs anywhere in another string. If a character does not exist in the string being examined, VERIFY returns its character position in string "b". If all characters exist, VERIFY returns a zero.
System Subroutine Library

Form: CALL VERIFY (a,b,i)
       or
       i = IVERIF (a,b)

where:  a       is the array containing the string to be
        scanned.

        b       is the array containing the string of
                characters to be accepted in a.

        i       is the integer result of the verification.

Notes:

Quoted-string literals must not be used with the IVERIF function
subprogram. They may be used with the VERIFY subroutine subprogram.

Function Result:

       i = 0       if all characters of "a" exist in "b"; also if
                "a" is a null string.
       = n       where n is the character position of the first
                character in "a" that does not appear in "b"; if
                "b" is a null string and "a" is not, i equals 1.

Example:

The following example accepts a 1- to 5-digit unsigned decimal number
and returns its value.

LOGICAL*1 INSTR(81)
.
.
CALL VERIFY(INSTR(IPOS),'0123456789',I)
IF(I.EQ.1) STOP 'NUMBER MISSING'
IF(I.EQ.0) I=LEN(INSTR)-IPOS+1
IF(I.GT.5) STOP 'TOO MANY DIGITS'
NUM=IVALUE(INSTR(IPOS),I)
.
.
END
FUNCTION IVALUE(ARRAY,I)
LOGICAL*1 ARRAY(I)
DECODE(I,99,ARRAY) IVALUE
99 FORMAT(I5)
END
APPENDIX P

ERROR MESSAGE SUMMARY

The information that formerly appeared in this appendix has now been incorporated into the RT-11 System Message Manual, DEC-11-ORMEA-A-D.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute address</td>
<td>A binary number that is assigned as the address of a fixed memory storage location.</td>
</tr>
<tr>
<td>Absolute block numbers</td>
<td>Any blocks which use block 0 of a physical device as a base. Relative blocks use the start of a file as a base.</td>
</tr>
<tr>
<td>Absolute section</td>
<td>That portion of a program in which the programmer specifies the physical (absolute) locations of data items, using the .ASECT assembler directive.</td>
</tr>
<tr>
<td>Addressing mode</td>
<td>The portion of a PDP-11 machine instruction which specifies how the argument is to be referenced.</td>
</tr>
<tr>
<td>Allocation</td>
<td>Assigning a resource to a job.</td>
</tr>
<tr>
<td>Alphanumeric character string</td>
<td>A command string containing any of the 26 alphabetic characters A-Z, the numeric characters 0-9, space and certain special characters.</td>
</tr>
<tr>
<td>Argument</td>
<td>A variable or constant which is given in the call of a subroutine as information to it; a variable upon whose value the value of a function depends; the known reference factor necessary to find an item in a table or array (i.e., the index).</td>
</tr>
<tr>
<td>Argument block</td>
<td>A block of memory used to transmit programmed request arguments to the monitor.</td>
</tr>
<tr>
<td>Array</td>
<td>A set or list of elements, usually subscripted variables or data.</td>
</tr>
<tr>
<td>Assembler</td>
<td>A program that translated symbolic opcodes into machine language and assigns memory locations for variables and constants.</td>
</tr>
<tr>
<td>Assembler directives</td>
<td>The mnemonics used in the assembly language programs to control or direct the assembly process.</td>
</tr>
</tbody>
</table>
GLOSSARY

Assembly language  A symbolic language that translates directly into machine language instructions. Usually there is a one-to-one relation between assembly language instructions and machine language instructions.

Asynchronous  1. Pertaining to the scheduling of hardware operations by ready and done signals rather than by time intervals.
   2. Pertaining to the method of data transmission in which each character is sent with its own synchronizing information.

Autoincrement mode  Mode of operation wherein the contents of the register are incremented immediately after being used as the address of the operand.

Background program  A program operating automatically, under low priority, when higher priority (foreground) programs are not using system resources.

Backup file  A copy of a file created for protection in case the primary file is inadvertently destroyed.

Bad blocks  A defective block of storage on any type of magnetic storage medium that produces a hardware error when attempting to read or write in the block.

Base address  A given address from which an absolute address is derived by combination with a relative address. An address constant.

Base segment  The portion of an overlayed program that is always memory resident; same as root segment.

Baud  A unit of signaling speed. In a code in which all characters have the same length, one baud corresponds to a rate of one signal element per second, usually one bit per second.

Binary  Pertaining to the number system having a base or radix of two. In this system numbers are represented by 1s and 0s. Counting to ten in binary: 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010.
<table>
<thead>
<tr>
<th>Glossary Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Contraction of &quot;Binary digit&quot;, a bit is the smallest unit of information in a binary system of notation. It is the choice between two possible states, usually designated one (1) and zero (0).</td>
</tr>
<tr>
<td>Blocks</td>
<td>A set of consecutive machine words, characters or digits handled as a unit, particularly in reference to I/O. Each type of mass storage has its own block size, its own smallest unit of storage (e.g., PDP-11 DECTape has 256 (decimal) 16-bit words per block). See Data Block.</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>A program whose purpose is to load and (usually) start a complex system of programs.</td>
</tr>
<tr>
<td>Bottom address</td>
<td>The lowest memory address in which a program is located.</td>
</tr>
<tr>
<td>Bounds</td>
<td>The limits a program may operate within.</td>
</tr>
<tr>
<td>Breakpoint</td>
<td>A preset point in a user program where control passes to a debugging routine.</td>
</tr>
<tr>
<td>Buffer</td>
<td>A temporary storage area which may be a special register or an area of storage. Buffers are often used to hold data being passed between processes or devices which operate at different speeds or different times.</td>
</tr>
<tr>
<td>Byte</td>
<td>A group of binary digits usually operated upon as a unit; 8 bits is usually a byte (1/2 a PDP-11 word).</td>
</tr>
<tr>
<td>Carry bit</td>
<td>A bit in the program status word indicating a carry from the most significant bit in the operation; also a common method of indicating a program request failure.</td>
</tr>
<tr>
<td>Central processing unit</td>
<td>The part of the computer containing the Arithmetic and Logical Unit, the Instruction Control Unit, timing generators and I/O interfaces of the basic system.</td>
</tr>
<tr>
<td>Chaining</td>
<td>A programming technique which involves dividing a routine into sections with each section terminated by a call to the next section.</td>
</tr>
<tr>
<td>Channel number</td>
<td>Logical identifier in the range 0 to 255 (decimal) assigned to a file used by RT-11 monitor. Data blocks in an open file may be referred to by channel number.</td>
</tr>
<tr>
<td><strong>GLOSSARY</strong></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td><strong>Channel status word</strong></td>
<td>A word associated with each I/O channel in RT-11 that preserves the status of that channel.</td>
</tr>
<tr>
<td><strong>Character-oriented</strong></td>
<td>Referring to editing operations on a single character. See also Line-oriented.</td>
</tr>
<tr>
<td><strong>Checksum</strong></td>
<td>A value representing the sum of all bytes in a program. When the program is loaded, the sum of the bytes can be compared with the checksum to make sure that the entire program has been loaded correctly.</td>
</tr>
<tr>
<td><strong>Clock</strong></td>
<td>A time-keeping or frequency-measuring device within the computer system.</td>
</tr>
<tr>
<td><strong>Closed location</strong></td>
<td>A location whose contents are not available for examination and change.</td>
</tr>
<tr>
<td><strong>Co-resident overlay routines</strong></td>
<td>Overlay routines that are simultaneously resident in memory.</td>
</tr>
<tr>
<td><strong>Command string</strong></td>
<td>A series of characters which specify the input/output devices, files, and switches.</td>
</tr>
<tr>
<td><strong>Command String Interpreter</strong></td>
<td>The portion of the RT-11 system software which accepts an ASCII string input and interprets the string as input and output files and switches.</td>
</tr>
<tr>
<td><strong>Completion routine</strong></td>
<td>An optional user-supplied routine that is called at the completion of an operation and is generally used to allow the user to handle asynchronous events.</td>
</tr>
<tr>
<td><strong>Compress</strong></td>
<td>To collect into one area all the free (unused) blocks that are interspersed in the directory and files on a specified device.</td>
</tr>
<tr>
<td><strong>Concatenate</strong></td>
<td>To combine many files into one file.</td>
</tr>
<tr>
<td><strong>Condition codes</strong></td>
<td>The four bits of the Program Status Word in the PDP-11 Processor that preserve the results (negative, zero, carry, overflow) of the instruction just completed.</td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td>A particular selection of computer, peripherals, and interfacing equipment that are functioning together. A list of the devices of a computer system.</td>
</tr>
<tr>
<td><strong>Console device</strong></td>
<td>The interface, or communication device, between the operator and the computer, that contains indicator lights, switches, knobs and sometimes a keyboard to permit manual operation of the device.</td>
</tr>
<tr>
<td><strong>Constant register</strong></td>
<td>A logical register in ODT or PATCH that is used to store an often-used constant.</td>
</tr>
</tbody>
</table>
GLOSSARY

Context switching: The saving of key registers and other memory areas prior to switching between jobs, as in timesharing or multiprogramming.

Contiguous: Code that resides in memory or on a peripheral device immediately adjacent to other sections of code or data.

Control Section (CSECT): A named, contiguous part of a program. CSECTs are denoted by the directives "CSECT" and "ASECT" in the MACRO Assembler Language. CSECTs are collected and assigned addresses by the Linker.

Control statement: A job statement that is used to issue commands to the system through the input stream.

Core memory: The main high-speed storage of a computer in which binary data is represented by the switching polarity of magnetic cores. Semi-conductor memory is higher speed than core memory, but semi-conductor memory is more expensive and volatile.

Cross reference table (CREF): A list of all or a subset of symbols in source program and statements where they were defined or used.

Cursor: On a display device, a symbol that appears to indicate the location of the pointer.

Data block: A logical grouping of data, usually associated with input or output. Typical data block involved in RT-11 are 256 words long.

Data format: The form or structure of information, particularly in an I/O file. RT-11 has several standard data formats such as ASCII and formatted binary.

Data image: A job statement that is used to define data for system programs and user programs through the input stream.

Debug: To detect, locate, and correct mistakes in a program.

Default: A specification assigned by a system program when user specification is omitted.

Delimiter: A character that separates, terminates and organizes elements of a statement or program.

Device block: A section of code that specifies a physical device and filename for an RT-11 programmed request.
## GLOSSARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device directory listing</td>
<td>A list of all files on a specified device. The list contains all files with associated creation dates, total free blocks on the device, and number of blocks used by files. Magtape and cassette directories omit some information. See Directory.</td>
</tr>
<tr>
<td>Device handlers</td>
<td>Routines that perform I/O for specific storage devices and translate logical block numbers to physical disk, tape or drum addresses. These routines also handle error recovery and provide device independence in conjunction with operating systems.</td>
</tr>
<tr>
<td>Direct assignment</td>
<td>User definition of a symbol and its associated value.</td>
</tr>
<tr>
<td>Directive</td>
<td>See Assembler directives.</td>
</tr>
<tr>
<td>Directory</td>
<td>An area of a mass storage device that describes the layout of the data on that device in terms of file names, length and location.</td>
</tr>
<tr>
<td>Disk</td>
<td>A mass storage device. Basic unit is an electromagnetic platter on which data is magnetically recorded. Features random access and faster access time than magnetic tape.</td>
</tr>
<tr>
<td>Display unit</td>
<td>A device that provides a visual representation of data.</td>
</tr>
<tr>
<td>Double buffered I/O</td>
<td>An input (or output) operation using two buffers to achieve more efficient usage of computer time. While one buffer is being used by the CPU program, the other is being read from (or written to) by an I/O device.</td>
</tr>
<tr>
<td>Dummy file</td>
<td>A file used only to hold space on a storage device.</td>
</tr>
<tr>
<td>Editor</td>
<td>A program which interacts with the programmer to enter new programs into the computer and edit them as well as modify existing programs. Editors are language independent and will edit anything in alphanumeric representation.</td>
</tr>
<tr>
<td>EMT</td>
<td>A PDP-11 machine instruction (operation codes from 104000-1043777). EMTs are most often used for monitor communication.</td>
</tr>
<tr>
<td>Entry point</td>
<td>A point in a subroutine to which control is transferred when the subroutine is called.</td>
</tr>
</tbody>
</table>
GLOSSARY

Entry point table  A table, kept by the Librarian program, of the names and locations of the routines available in the library.

Entry symbol  A global symbol defined in an object module that can be referred to by other object module.

Error flag  Condition indicating an error has occurred. If the C bit is set on return from an RT-ll call, an error occurred. Thus, the C bit set is the RT-ll error flag.

Exception  An unusual condition (e.g., a floating point exception is an overflow).

Expressions  A combination of variables, constants, and operators (as in a mathematical expression).

External symbol  A symbolic name which is defined in one assembly or compilation and can be referenced in another. The .GLOBL directive is used to indicate external symbols to the MACRO assembler.

Fatal error  An error which makes continued processing impossible (e.g., running out of symbol table space is usually a fatal error).

File allocation scheme  The method used to store data on I/O devices. RT-ll uses a contiguous structure.

File gap  A fixed length of blank tape separating files on a recording medium.

File-structured device  A device on which data is given names and arranged in files (e.g., disk, DECtape, magtape, or cassette).

Filler characters  Null characters output to a device to give it time to perform unusually long operations (such as return the carriage) without explicitly waiting.

Flag  A variable or register used to record the status of a program or device. In the latter case it is sometimes called a device flag.

Floating-point  A number system in which the position of the radix point is indicated by one part (the exponent part), and another part represents the significant digits, the fractional part (e.g., 5.39X10(8) for Decimal; 137.3X8(4) for Octal; 101.10X2(13) for Binary).
GLOSSARY

Foreground program  A program of high priority that utilizes machine facilities when and as needed, but allows less critical background work to be performed in otherwise unused time.

Formatted binary block  A data structure used to hold binary information, usually to be read or written to a data file. For example, an object module produced by the MACRO assembler consists of formatted binary blocks.

Fragmented  Having many empty entries scattered over a device.

Free blocks  Areas of a file structured device which are unused.

General registers  A set of eight general purpose registers available for use as accumulators, as auto index registers or as pointers. General registers 6 and 7 serve as the hardware stack pointer and program counter respectively.

Global symbol  Any symbol accessible to other programs. Linkage must be supplied by a linker.

Global symbol directory  The portion of an object module which describes all external symbols and CSECT names which are defined or referenced by the object module.

Handler  See Device handler

Hardware mode  For magnetic tapes and cassettes. Allows full user control over position and record size (as opposed to software mode.)

High level language  A language in which single statements may result in more than one machine language instruction, e.g., BASIC, FORTRAN, COBOL.

Image mode  A mode of data transfer in PIP in which a file is copied without change of any kind.

Implied argument  An argument which is assumed by the program, whether or not it is explicitly stated by the user.

Indexing  Using a variable index register as an offset into a table.

Initialize  To set counters, switches, addresses and variables to zero or other starting values.
GLOSSARY

Input stream
The sequence of control statements and data images submitted to the operating system on an input device especially dedicated for this purpose.

Internal symbol
A symbolic name which is known only within the assembly or compilation where it is defined. Symbols are internal by default.

Interrupt service routine
Routine entered when an external interrupt occurs.

Interrupt
1. To break the normal operation of the routine being executed and pass control to a specific location, generally accompanied by saving the state of the current routine so that control can return later.
2. The signal which causes the break.

Iterations
Repetitions of a portion of a program.

Job
The largest definable collection of elements describing the work to be performed. The purpose of the job is to collect and describe the logical work which is to be performed. A job is composed of control statements and data images.

Job status word
A word in the RT-ll communications region containing bit flags indicating the status of the program currently in memory.

Keyboard monitor (KMON)
The program that provides communication between the user at the console and the RT-ll system.

Keyword
A symbol that identifies a parameter.

Label
One or more characters used to identify a source language statement or line.

LDA format file
See Load image file.

Library
A collection of standard routines which can be incorporated into other programs.

Library header
Section of code that contains the current status of the library, including version, date and time of creation or update, relative starting address of entry point table, number of EPT entries and placing of next module to be inserted into the library file.

Light pen
A device resembling a pencil or stylus which can detect a fluorescing spot on a CRT screen. Used to input information to a CRT display system.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line-oriented</td>
<td>Referring to editing operations on an entire line of text. See also Character-oriented.</td>
</tr>
<tr>
<td>Load image file</td>
<td>A program that can be executed in stand-alone environment without the aid of relocation.</td>
</tr>
<tr>
<td>Local symbol</td>
<td>A symbol used only within the program in which it is defined (all non-global symbols).</td>
</tr>
<tr>
<td>Location counter</td>
<td>A counter kept by an assembler to determine the address assigned to an instruction or constant being assembled.</td>
</tr>
<tr>
<td>Logical name</td>
<td>A user-defined name assigned as an alternate name for a physical device. Useful for redirecting I/O when device originally specified is unavailable.</td>
</tr>
<tr>
<td>Loop</td>
<td>A sequence of instructions that is executed repeatedly until a terminal condition occurs. Also, used as a verb meaning to execute this circle of instructions while waiting for the ending condition.</td>
</tr>
<tr>
<td>Macro</td>
<td>An instruction in a source language that is equivalent to a specified sequence of machine instructions, or a command in a command language that is equivalent to a specified sequence of commands.</td>
</tr>
<tr>
<td>Macro symbol</td>
<td>Symbol used as macro name in operator field.</td>
</tr>
<tr>
<td>Mainstream code</td>
<td>Any code which is not executing as a result of a completion routine.</td>
</tr>
<tr>
<td>Mask word</td>
<td>A combination of bits that is used to access selected portions of any word, character, byte, or register while retaining other parts for use. Also, to clear these selected locations with a mask.</td>
</tr>
<tr>
<td>Memory address</td>
<td>An address in memory used to store data.</td>
</tr>
<tr>
<td>Memory image</td>
<td>An exact copy of a portion of memory. RT-ll save image files (.SAV) are memory images.</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Alphabetic representation of a function or octal machine instruction.</td>
</tr>
<tr>
<td>Mode</td>
<td>A state or method of program operation. Command mode causes user input to be interpreted as a command; text mode causes user input to be interpreted as alphanumeric data, etc.</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>A routine which handles a particular function.</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td><strong>Monitor</strong></td>
<td>The collection of routines which schedules resources, I/O, system programs, and user programs, and obeys keyboard commands in a Monitor System.</td>
</tr>
<tr>
<td><strong>Monitor System</strong></td>
<td>Editors, assemblers, compilers, loaders, interpreters, data management programs and other utility programs all automated for the user by a monitor.</td>
</tr>
<tr>
<td><strong>Multi-region</strong></td>
<td>A term describing the RT-ll overlay structure in which there may exist multiple independent areas of high-speed memory (regions) in which overlays may occur.</td>
</tr>
</tbody>
</table>
| **Nesting** | 1. Including a program loop inside another program loop, or other similar occurrences within one another.  
2. Algebraic nesting such as (A+B+(C+D)) where execution proceeds from the innermost level to the outermost level. |
<p>| <strong>Nonfile-structured device</strong> | A device that is used only for input or output, and not storage, of a file (e.g., line printer, card reader, terminal, or paper tape). |
| <strong>.LOOKUP (or .ENTER)</strong> | A .LOOKUP (or .ENTER) with a null file-name, that permits access to a device by reference to absolute block numbers, rather than block numbers relative to a file. |
| <strong>NonRT-ll directory-structured</strong> | A device having no standard RT-ll directory at its beginning. This includes magtape and cassette, and any devices for which user-defined directories have been written. |
| <strong>Null</strong> | Characters with ASCII code 000. |
| <strong>Object code</strong> | The result after assembling or compiling source code. Machine code. |
| <strong>Off-line</strong> | Pertaining to equipment, devices or events which are not under direct control of the computer. |
| <strong>Offset</strong> | The difference between a location of interest and some known base location. |
| <strong>On-line</strong> | Pertaining to equipment, devices and events which are in direct communication with the CPU and thereby under its control in some way. |
| <strong>Open location</strong> | A location whose contents have been printed for examination; contents can be changed. |</p>
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operand</td>
<td>1. A quantity which is affected, manipulated or operated upon. 2. The contents of the field following the operator of an assembler instruction.</td>
</tr>
<tr>
<td>Operating system</td>
<td>A program for automating a programmer's use of software. A Monitor System.</td>
</tr>
<tr>
<td>Operator</td>
<td>That symbol or code which indicates an action or operation to be performed.</td>
</tr>
<tr>
<td>Output stream</td>
<td>Output data issued by the operating system on the processing program to an output device especially designated for this purpose.</td>
</tr>
<tr>
<td>Overlay</td>
<td>The technique of repeatedly using the same area of memory during different stages of a program. When one routine is no longer needed in memory, another routine can replace all or part of it. Overlaying replaces parts of programs; chaining replaces the whole program.</td>
</tr>
<tr>
<td>Overlay segment</td>
<td>A section of code handled as a unit. This segment of code can overlay code already in memory or can be overlaid by another overlay segment.</td>
</tr>
<tr>
<td>Overlay-structured</td>
<td>A term describing a program which does not entirely reside in high-speed memory at any instant. Portions of the program are brought into memory when needed.</td>
</tr>
<tr>
<td>Page of text</td>
<td>That portion of a file delimited by form feed characters, generally 50-60 lines long: corresponds approximately to a physical page of program listing.</td>
</tr>
<tr>
<td>Parameter</td>
<td>A variable or an arbitrary constant appearing in a mathematical expression, each value of which restricts or determines the specific form of the expression.</td>
</tr>
<tr>
<td>Patch</td>
<td>To modify a routine in a rough or expedient way, usually by modifying the binary code rather than reassembling it.</td>
</tr>
<tr>
<td>Peripheral devices</td>
<td>In a data processing system, any device distinct from the central processing unit, which may provide the system with outside storage or communication.</td>
</tr>
<tr>
<td>Permanent file</td>
<td>An output file that is stored in memory for later use.</td>
</tr>
<tr>
<td>Permanent symbol</td>
<td>Instruction mnemonics, assembler directives, and macro directives incorporated in the assembler.</td>
</tr>
</tbody>
</table>
GLOSSARY

Physical device
An input, output, or storage device associated with the Central Processing Unit.

Physical name
A 2- or 3-character name identifying a physical device. The first two characters are alphabetic and identify the device; the third character is numeric and identifies the unit.

PIC code
Abbreviation for Position Independent Code. (Code which can operate properly wherever it may be loaded in memory).

Pointer
1. A location containing the address to another location.
2. In the EDIT program, a movable reference pointer normally located between the character most recently operated upon and the next character in the buffer; represents the current position of the Editor in the text.

Positional parameter
A parameter that must appear in a specified position in an ordered group of parameters.

Proceed count
In a program loop, the number of times the breakpoint is to be encountered before suspension of program execution.

Processor status register
A register that indicates the current priority of the processor, the condition of the previous operation, and other basic control items.

Program counter
A register in the CPU that holds the address of the current instruction being executed plus two; in other words, it holds the address of the next instruction unless the current instruction causes a jump.

Program sections
See Control section.

Programmed requests
Machine language instruction which is used to invoke a monitor service for the issuing program.

Purge
Deactivate a channel without taking any other action.

Radix
The base of a number system; the number of digit symbols required by a number system.

Random-access
A means of accessing data in a random order (independently of the data's physical or relative position on the device).

GLOSSARY-13
January 1976
GLOSSARY

Real time 1. Pertaining to the actual time during which a physical process takes place.

2. Pertaining to the performance of computer activity which occurs fast enough to influence the related physical process.

Real time system A system in which computation is performed while a related physical process is occurring so that the results of the computation can be used in guiding the physical process.

Record A collection of related items of data treated as a unit. Example: A line of source code.

Reentry address The Start address 2. Allows the user program to reset itself internally, and resume operation.

Region number A number which is used to identify a portion of memory to the Linker for the purpose of describing an overlay structure.

Relative address A number specifying the difference between an absolute address and a base address.

Relocatable code Code written so that it can be located and executed in any part of memory, once it has been linked by Linker.

Relocatable object modules A set of instructions which are written so that the module can be loaded and executed in any memory area.

Relocatable symbols Symbolic names whose associated value is an offset from the base (beginning) of a control section. Such a symbol's value depends on the address of the control section and must be relocated each time the control section is assigned an address by the Linker.

Relocation To move a routine from one portion of storage to another and to adjust the necessary address references so that the routine can be executed in the new location.

Relocation directory A portion of an object module which describes and identifies all occurrences of relocatable symbols in the object module.

Repeat block Block of code to be repeated a defined number of times.

Repeat count The number of times a block of code is to be repeated.
GLOSSARY

Resident
In memory, as opposed to being stored externally.

Resident monitor (RMON)
The permanent memory-resident part of RT-ll. Contains console terminal service, the error processor, the system device handler, the EMT processor, and system tables.

Root segment
See Base segment.

RT-ll directory-structured
A device having a standard RT-ll directory at its beginning that is updated as files on the device are manipulated (for example, disk and DECTape).

Scratch area
Any memory or registers used for temporary storage of partial results.

Scrolling
On a display screen, when the maximum number of lines are on the screen, the top line is deleted, and all text moves up one line, permitting one new line of display at the bottom of the screen. The visual effect is similar to the rolling up of a scroll.

Sentinel file
The last file on cassette tape; contains only a header record and represents logical end-of-tape.

Sequence number
The number in a cassette directory which designates where a file is placed (in sequence), when that file has been continued on more than one cassette.

Sequential-access
A means of accessing data in which records or blocks are read one after another from the device (as opposed to random-access).

Software mode
For magnetic tapes and cassettes, the mode of operation when device access is through any RT-ll system program.

Source code
Text, usually in the form of an ASCII formatted file, which represents a program. Such a file can be processed by an appropriate system program (MACRO or FORTRAN) to produce an object module.

Source file
A file to be used as input to a translating program such as MACRO or BASIC.

Stack
An area of memory set aside by the programmer for temporary storage or subroutine interrupt service linkage. The stack uses the "last-in, first-out" concept. The stack starts at the highest location reserved for it and expands linearly downward as items are added to the stack.

GLOSSARY-15
January 1976
GLOSSARY

Stack pointer
A general register used to keep track of the last location where data is entered into the stack.

Storage device
A general term for any device capable of retaining information.

Subconditionals
Directives that indicate:
1. assembly of an alternate body of code;
2. assembly of a non-contiguous body of code within a conditional block;
3. unconditional assembly of a body of code within a conditional block.

Subpicture address
Address of a set of display processor instructions which display text on a GT40 (42,44).

Suspend
To temporarily halt execution of a task while another task of different priority runs.

Swap
The process of saving user memory in system device scratch blocks, reading in and executing a USR function, and then restoring the user code.

Switch register
A location in the CPU which can be loaded with a value by the operator by his setting switches on the computer console for each bit he wants to enter.

Switches
1. in a command string to the CSI a switch is a slash followed by an ASCII character which can be given a value by typing a colon after the character followed by an octal number or from 1-3 ASCII characters.
2. Two or three position mechanisms used for operating computer or devices.

Symbols
Names which can be assigned values or which can be used to indicate specific locations in a program.

Symbol table
An array which contains all defined symbols and the binary value associated with each one. Mnemonic operators, labels and user defined symbols are all placed in the symbol table. (Mnemonic operators stay in the table permanently.)

System configuration
See Configuration.

System device
A peripheral file-structured storage device on which the monitor resides.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>System programs</td>
<td>DEC-supplied programs which come in the basic software packages. These include editors, PIP, assemblers, compilers, loaders, etc.</td>
</tr>
<tr>
<td>Time-critical job</td>
<td>A job which demands response within a fixed time period.</td>
</tr>
<tr>
<td>Transfer address</td>
<td>A program entry point.</td>
</tr>
</tbody>
</table>
| Trap                        | 1. An automatic transfer of control to a prespecified routine that can be caused by software or hardware. The trap instruction is an example of a hardware implementation.  
                              | 2. A conditional jump to a known location performed automatically by hardware as a side effect to executing a processor instruction. The address location from which the jump is made is recorded. It is distinguished from an interrupt which is caused by an external event. |
| Truncation                  | The reduction of precision by ignoring one or more of the digits (not rounding off).                                                      |
| Two's complement            | A number used to represent the negative of a given value in many computers. This number is formed from the given binary value by changing all 1s to 0s and all 0s to 1s, and then adding 1. |
| Type-ahead                  | The ability to type information at the console terminal and have it remembered by the system for later use.                               |
| Unary operation             | Operation affecting a single operand. Examples: negation, radix indicator.                                                                |
| User-defined symbol        | 1. A label;  
<pre><code>                          | 2. A symbol defined by direct assignment.                                                                                                 |
</code></pre>
<p>| Utility                     | Any program which performs useful functions, i.e., PIP.                                                                                   |
| Vector                      | Two words describing 1) where to go when external interrupt occurs, and 2) the contents of the processor status word when the interrupt is acknowledged. |
| Wild card operation         | A shorthand method of transferring all files with the same name, extension, or both.                                                    |
| Word-for-word transfer      | A transfer in which no alteration of data is performed.                                                                                  |</p>
<table>
<thead>
<tr>
<th>Words</th>
<th>In the PDP-11 a 16-bit unit of data which may be stored in an addressable location.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write-lock condition</td>
<td>The condition of a device that is protected against any transfers which would write information to it.</td>
</tr>
<tr>
<td>Zero</td>
<td>To initialize a device (e.g., DECtape, cassette) so that it contains no information.</td>
</tr>
</tbody>
</table>
INDEX

Abbreviating command names (BATCH), Arguments (cont.), to Macro calls and definitions, 5-62
Abort entry point, 9-46, H-4
Absolute, 5-3
and relocatable program sections, 6-4
block numbers, 4-21, I-2
expression, 5-18, C-5
load address, 6-1
load module, 6-5
mode, 5-24
quantities, 5-17
section, 6-4
starting block, 4-17
Absolute Loader, 6-5
Absolute sectors, reading and writing, 9-85, H-13.2
Accessing,
general registers, 8-9
internal registers, 8-10
Add two INTEGER*4 values, 0-88
Additional and reference material, xxiii
Address mode syntax, C-5
Address specifier, M-2
Addressed location, 8-8
Addresses, vector, 9-6
Addressing modes, 5-20
Advance command, 3-17
AJFLFT function, 0-21
Allocating,
blocks for files, 4-11
character string variables, 0-19
extra words, 4-18
memory for a queue, 9-66
system resources, 2-16
words, 2-36
Alphabetize switch, 6-18
Alphabetized load map, 6-19
Alphanumeric representation, 1-2
ALTMODE, 3-2
Argument, 3-4
block, 9-3
CSECT, M-3
dummy, 5-67
iteration, 3-8
list, 9-5
list pointer, 9-2
negative line, 3-7
numeric, 3-6
numerical, 9-4
positive, 3-7
substitution, 5-72
Arguments, 9-2
EDIT, 3-4
indefinite repeat, 5-72
missing, 5-66
number of, 5-66
real, 5-63
symbolic, 5-37
ASCII, character set, C-1, C-4
code, 0-18
conversion of one or two characters, 5-40
files, 3-1
format, 2-3
input and output, 8-20
to Radix-50 conversion, 0-105
.ASCII directive, 5-41
.ASCII2 directive, 5-42
.ASECT directive, 5-53, 6-4, 6-6
.ASECTS, H-3
ASEMBL (8K assembler), 1-4, 11-1
calling and using, 11-1
error messages, 11-7
file specifications, 11-2
Assembler, 1-2, 8-4
ASEMBL, 11-1
MACRO, 5-1, 12-29
output, 5-19
Assembler directives, 5-3, 5-27, 7-1
ASCII, 5-41
.ASCII2, 5-42
.ASECT, 5-53
.BLKB, 5-47
.BLKW, 5-47
.SPILE, 5-48
.GLOBAL, 5-54
.IDENT, 5-36
.IFF, 5-57
.IFT, 5-57
.IPTF, 5-57
.NARG, 5-68
.NCHR, 5-68
.NLIST, 5-27
.NTYPE, 5-69
.ODD, 5-46
INDEX-1 January 1976
Assembler directives (cont.),
   .PAGE, 5-36
   .PRINT, 5-71
   .RADI, 5-44
   .RAD50, 5-43
   .REPT, 5-73
   .SBTTL, 5-34
   .TITLE, 5-34
   .WORD, 5-39
Assembling graphics programs, N-16
Assembly instructions, N-24
Assembly,
   language completion routine, 0-100
   language display support, N-1
   language routines, O-5
   language statement, 5-2
   listing, 8-1
   listing table of contents, 5-35
   location counter, 5-14
   pass, 5-14
   source listing showing local
   symbol blocks, 5-15
   ASSIGN command, 2-18
Asterisk,
   data line indicator, 12-38
   wild-card, 4-1, 12-7
Asynchronous completion routines, 9-13
  Autodecrement mode, 5-23
  Autodecrement deferred mode, 5-23
  Autoincrement mode, 5-21
  Autoincrement deferred mode, 5-22
  Automatic generation of local
  symbols, 5-13
  Automatic relocation facility, 8-4
  Automatically created symbols
  within user-defined Macros, 5-66
Backslash, 8-7
Backup storage device, 6-10
Back-arrow, 8-8
Back space, H-11, 0-72
Bad block files, 4-2
Bad block scan, 4-21
Bad entry, 8-2, 8-26
Base address, 5-25, 8-4
Base command, 2-29
Base segment, N-2
$BASIC command, 12-13
BASIC/CT subroutine structure, N-20
BASIC/CT-11, 1-1, 1-5
   commands, P-3
   error messages, F-6
   function errors, F-9
   functions, F-5
   language, F-1
   language summary, F-1
   statements, F-1
   string functions, F-5
BATCH, 1-5, 12-1
   character set, 12-8
   command field switches, 12-3
   command fields, 12-2
   command summary, B-17
   commands, 12-12, 12-36
   comment fields, 12-8
   control statement format, 12-2
   device names, 12-6
   differences between RT-11 BATCH
   and RSX-11D BATCH, 12-52
   error messages, 12-53
   example, 12-36
   filename extensions, 12-7
   file specifications, 12-6
   files, 12-10
   hardware requirements, 12-1
   operating procedures, 12-45
   operator directives, 12-50
   punched card programs, 12-44
   RT-11 mode, 12-38
   rules and conventions, 12-11
   software requirements, 12-2
   specification field switches, 12-8
   specification fields, 12-5
   switch summary, B-14
   wild card construction, 12-7
BATCH commands,
   $BASIC, 12-13
   $CALL, 12-14
   $CHAIN, 12-15
   $COPY, 12-16
   $CREATE, 12-18
   $DATA, 12-19
   $DELETE, 12-20
   $DIRECTORY, 12-20
   $DISMOUNT, 12-21
   $EOD, 12-22
   $EOJ, 12-23
   $FORTRAN, 12-23
   $JOB, 12-25
   $LIBRARY, 12-27
   $LINK, 12-27
   $MACRO, 12-29
   $MESSAGE, 12-31
   $MOUNT, 12-32
   $PRINT, 12-34
   $RTll, 12-35
   $RUN, 12-35
   $SEQUENCE, 12-36
   Beginning command, 3-16
   Binary, 1-2
   code, 1-2
   object module, 8-4
   operators, 5-8, 5-18
   output, 1-2
   radix, 5-17, 5-45
   Bit patterns, 1-3, 8-24
   search, 8-10
   .BLANK, N-3
   Blank COMMON, 6-14

January 1976

INDEX-2
Calls or branches to overlay segments, 6-13
Cancel scheduling requests, 0-36
Card,
debck, 12-45
input, 12-46
Card codes, 2-24, H-6
conversion table, H-25
Card reader handler, H-6, 12-46
Carriage control, 0-33
Carry bit, 9-13
Cassette,
handler functions, O-72
rewind button, 4-7
sequence number, 4-7
Cassette special functions, H-12
last block, H-12
last file, H-12
next block, H-12
next file, H-12
rewind, H-12
write file gap, H-13
Cassettes and magtapes,
end-of-file detection, H-13
initializing, 4-13
legal operations, 4-5
Cathode ray tube, N-1
.CDFN request, 9-26
Centralized queue management system, 9-65
CHAIN bit, 9-8
$CHAIN command, 12-15
.CHAIN request, 9-27
CHAIN subroutine, 0-22
Chaining, 0-105
Change command, 3-22
Changing,
device handler characteristics, 2-23
monitors, 2-2.1
stack size, 2-36
Channel, 9-41
allocation, 0-34, 0-46
assignment, 0-99
data, 9-78
deactivation, 0-103
number, 9-5, 0-47
returned to pool, 0-45
status, 0-39, 0-65
status word, 9-13, 9-41, 9-78
Channel-oriented operations, 0-9, 0-17
Channels, 9-35
Chapter summary, xx1
Character,
deletion, 9-94
substitution, 5-72
transfer, 9-93
Character- and line-oriented command properties, 3-6
Character-oriented commands, 3-6

INDEX-3 January 1976
Character set, 5-5, 12-8, 12-9
ASCII, C-4
Radix-50, C-3
Character string,
comparison, 0-108
functions, 0-13, 0-18
shortening, 0-18
Character transfer,
console terminal to user program, 0-79
user program to console terminal, 0-81, 0-102
Character translation, 0-116
Character verification, 0-119
Characters,
illegal, 5-7
legal, 5-5
operator, 5-8
optional, 2-14
prompting, 2-4
special, 5-64
upper/lower-case, 3-27, 0-80
.CHCOPY request, 9-28
Checking channel status, 9-99
Checksum, 4-9, M-4, M-5
.CLEAR, N-4
Clearing breakpoints, restarting
ODT, 8-2
Clock, 2-17
frequency, 9-52
rate, 2-17, 9-52
ticks, 9-51
CLOSE command, 2-20
CLOSE handler function, H-9
.CLOSE request, 9-30
CLOSE subroutine, 0-23
Closed location, 8-6
Closing a channel, 0-23
.CMNT request, 9-31
.CNTXSW request, 9-32
Code,
binary, 1-2
modifications, L-1
object, 1-2
source, 1-2
Combining
files, 4-9
library switch functions, 7-11
Command and Switch Summaries, B-1
Command,
arguments (EDIT), 3-5, B-5
continuation switch, 7-3
decoder, 8-20
execution routine, 8-21
field, 12-2
field switches, 12-3, 12-6
interpretation services, 9-1
mode, 3-2
repetition, 3-8
string format, 2-10
string interpreter (SYSLIB), 0-37
strings, 3-5, 6-2
structure (EDIT), 3-3
Command (cont.),
switches, 4-1, K-2
syntax (LIBR), 7-2
Command String Interpreter, 2-7,
2-10, 6-3, 9-33, L-2
Command summary,
BATCH, B-17
EDIT, B-5
Keyboard monitor, B-1
ODT, B-12
PATCH, B-20
PATCHO, B-21
Commands,
and functions, 8-5
BASIC/RT-11, F-3
BATCH, see BATCH commands
class-oriented, 3-6
classification, 3-2
display editor, 3-4
edit control, 3-2
input/output, 3-3, 3-10
keyboard, 2-14
line-oriented, 3-6
PATCH, L-2
pointer relocation, 3-16
search, 3-18
text modification, 3-4, 3-20
utility, 3-4, 3-24
Commands to allocate system
resources, 2-16
ASSIGN, 2-18
CLOSE, 2-20
DATE, 2-16
INITIALIZE, 2-18
LOAD, 2-20
SET, 2-23
TIME, 2-17
UNLOAD, 2-23
Commands to control terminal I/O,
2-15
GT OFF, 2-16
GT ON, 2-15
Commands to manipulate memory
images, 2-28
Base, 2-29
Deposit, 2-30
Examine, 2-30
GET, 2-28
SAVE, 2-31
Commands used only in a F/B
environment, 2-35
FRUN, 2-36,
RESUME, 2-38
SUSPEND, 2-37
Command field, 5-2, 5-4
Comments, 12-6, 12-43
Common blocks, 6-14
Comparing
char strings, 0-108
two INTEGER*4 values, 0-89
Compiler, BATCH, 12-1, 12-47
Completion functions, 9-14

January 1976
INDEX-4
Completion routines, 2-37, 2-38, 9-13, 9-49, 9-60, 9-90, 0-15, 0-16
Component sizes, 2-9
Components,
  system hardware, 1-5
  system software, 1-3
Compress operation, 4-19
Compressing
directories, 4-19
  files, 4-19
CONCAT subroutine, 0-24
Concatenation, 5-67
Concatenating specified strings, 0-107
Condition codes, 8-10, 9-21.1
  Conditional,
    assembly directives, 5-55
    block, 5-55
    transfers of control, 12-40
Configuration word, 9-11, 9-52, 9-85
Confirming file transfers, 4-10
Console,
  normal mode, 9-94
  special mode, 9-94
  terminal, 3-1, 3-27
  terminal control and status
  registers, 9-12
  terminal input modes, 0-80
Constant register, 8-5, 8-10, 8-16
Context switch, 9-32
Contiguous
  area, 9-13
  file, 7-12, 9-13, J-7
Continuation character, 12-2
Continuation lines, 5-2
Continue switch, 6-20
Control,
  block, 6-1
  characters in RT-11 mode, 12-42
  commands, 3-2
  parameters, 6-6
  section names, 6-1
  statement format, 12-2
Control section,
  named, 6-4
  unnamed, 6-4
Controlling terminal I/O, 2-15
Conventions and restrictions, 0-2
Conventions and rules, 12-11
Conventions, system, 2-3
Conversion (SYSLIB),
  ASCII to Radix-50, 0-105
  INTEGER*2 to INTEGER*4, 0-92
  INTEGER*4 to INTEGER*2, 0-47
  INTEGER*4 to REAL*4, 0-21, 0-31
  INTEGER*4 to REAL*8, 0-27, 0-41
  Radix-50 to ASCII, 0-109
  REAL*4 to INTEGER*4, 0-88
  REAL*8 to INTEGER*4, 0-90
Conversion, Octal-Decimal, C-25
Converting V1 Macro calls to V2, 9-108
Copy,
  routine, 0-109
  substring, 0-112
$COPY command, 12-16
Copy operations, 4-9
  errors, 4-9.1
  multiple, 4-11
Copying,
  files with the current date, 4-10
  multiple cassette files, 4-8
  system files, 4-10
Co-resident overlay routines, 6-22
Core memory, 1-1
Correct and incorrect macro calls, 9-3
Correcting and updating object
  modules, M-1
CR (card reader), H-6
$CREATE command, 12-18
Creating,
  a library file, 7-4
  a new file, 4-6
RT-11 mode BATCH programs, 12-39
CREP, 1-4
  error messages, 5-86
  listing output, 5-81, 5-82,
  5-83
  specification switches, 5-76
  switches, 5-78, C-24
Cross reference,
  control sections, 5-78
  errors, 5-78
  listings, 1-4, 12-3
  MACRO symbols, 5-78
  permanent symbols, 5-78
  register-equate symbols, 5-78
  table generation, 5-78
CSECT argument, M-3
  .CSECT directive, 5-53, 6-4
CSECT, 7-13
CSI,
  error messages, 9-35
  general mode, 9-34
  special mode, 9-36
  switch separators, 9-38
  .CSIGEN request, 9-33
  .CSISP C request, 9-36
  .CSTAT request, 9-41
CTRL A, 2-12
CTRL B, 2-12
CTRL C, 2-12, 3-2, 8-3, H-5
CTRL E, 2-12
CTRL F, 2-12
CTRL O, 2-13, 3-3, H-5
CTRL O reset, 0-106
CTRL Q, 2-13
CTRL S, 2-13
CTRL U, 2-13, 3-3, 8-4
CTRL X, 3-3
CTRL Z, 2-13, H-5
Current,
  location counter, 5-3, 5-14
  location pointer, 3-6
  subpicture tag, N-14
Data,
format, 2-3, J-1
input, 12-19
length, 9-78
storage directives, 5-37
transfer functions, 0-7, 0-8, 0-9
transfer requests, 9-15
$DATA command, 12-19
DATE command, 2-16
.DATE request, 9-20
De-activating a channel, 9-65
Debugger, 1-2
Debugging,
process, 1-2
tool, 1-3
DEC command, M-5
DEC 026/029 card conversion table, H-23
Decimal,
number, 5-17
radix, 5-17, 5-45
DECSystem-10,
DECtape, J-1
file-structured device, J-3
DECTape,
DECSystem-10, J-1
PDP-11 DOS/BATCH, J-1
RSTS-11, J-1
Default,
devices, 12-6
extensions, 9-34
FORTRAN library switch, 6-20
stack, 9-7
switches, BATCH command field, 12-3
Defining NOTAG, N-19
Delete
command, 3-21
global switch, 7-7
input files, 12-3
operation, 4-13
switch, 7-6
$DELETE command, 12-20
DELETE handler function, H-8
.DELETE request, 9-42
Deleted files,
on cassette, 4-6
on magtape, 4-6
Deleting files, 4-13
from DOS/BATCH (RSTS) DECTapes, J-7
Delimiting characters, 3-25
separating and, 5-6
Deposit command, 2-30
Description of graphics macros, N-3
Destination device, 4-9
Device,
and file specifications, O-10
assignment, 12-6
block, 9-5
block replaceable, 2-7
designation, 3-12
dismounting, 12-21
file-structured, 2-7, J-1
mounting, 12-32
name, logical, 2-5
name, physical, 2-19, 9-5
nonfile-structured, 2-7, 6-21
nonRT-11 directory-structured, 2-7
ownership, 2-21
random access, 2-7, 4-8.1, 6-1
RT-11 directory-structured, 2-7
sequential-access, 2-7
size, 9-46
specification, 2-11
Device-dependent functions, O-72
status, O-42
Device Handlers, 2-8, 9-50, H-3.1
BATCH, 12-46
differences between V1 and V2, H-4
foreground programming and, H-1
loading, 9-50
loading SYSLIB, O-44
making resident, H-3.1
user-written, H-4
Device-dependent functions, 9-85
Device names,
logical, 2-5, 12-6
permanent, 2-5
physical, 2-4, 12-6
.DEVICE request, 9-44
Device structures, 2-7
DEVICE subroutine, O-26
DHALT display halt instructions, N-14
Differences between RT-11 and
RSX-11D BATCH, 12-52, 12-53
Differences between V1 and V2 de-
vice handlers, H-4
abort entry point, H-4
entry conditions, H-4
header words, H-4
interrupt handling, H-4
Differences between versions, xxii
Direct assignment statement, 5-10
5-14
conventions, 5-11
format, 5-10
Directives,
assembler, see assembler direc-
tives
BATCH, 12-49, 12-50
conditional assembly, 5-55
data storage, 5-37
immediate conditional, 5-58
listing control, 5-27

January 1976
INDEX-6
Directives (cont.),
Macro, 5-60
not available in ASEMBl, 11-2
PAL-11R, 5-59
PAL-11S, 5-59
program boundaries, 5-51
program section, 5-51
terminating, 5-50
Directory,
access motion and code consolidation, 9-79
initialization operation, 4-18
list operations, 4-15
listing of magtapes, 4-7
segments, 4-18, 4-19
Directories, compressing, 4-19
Disk, DOS/BATCH, J-1
Diskette, H-13.2
handler function, O-72
Display,
extended instructions, N-13
file handler, N-1, N-2
file structure, N-17
handler macro calls, N-16
hardware, 3-29
interrupt vectors, N-5
processor, N-1
program counter, N-14
screen, 3-28
status register, N-14
status words, N-8
stop instruction, N-2, N-18
stop interrupt handler, N-14
X register, N-14
Y register, N-14
Display editor, 3-28
format, 3-29
using the, 3-29
Display processor, N-1
management, N-2
mnemonics, N-16, N-23
Divide two INTEGER*4 values, O-91
DJFR function, O-27
DJSR subroutine call instruction, N-13
DNAME load name register instruction, N-15
Documentation conventions, xxiv
Dollar sign,
in BATCH control statement, 12-2
in data, 12-4
DOS/BATCH disk, J-1
Double ALTMODE, 3-29
Double-buffered I/O, 9-105
Double Operand Instructions, C-8
Double precision values, 0-27, 0-41
Double Register-Destination, C-17
DRET subroutine return instruction, N-14
.DSABL directive, 5-36
DSTAT display status instruction, N-14
.DSTATUS request, 9-45

Dummy,
argument, 5-67
argument list, 5-60
file, 4-6
names, 4-14
DUMP, I-5, I-1
calling and using, I-1
error messages, I-5
switches, I-2, B-18
DUMP command, M-4

E command, L-3
Edge flag, N-12
Edit Backup command, 3-11
Edit Console command, 3-29
Edit Display command, 3-30
Edit Lower command, 3-27
Edit Read command, 3-10
Edit Upper command, 3-27
Edit Version command, 3-27
Edit Write command, 3-11
Editor (EDIT), 1-2, 3-1
arguments, 3-4, 3-5, B-5
calling and using, 3-1
command structure, 3-3
control commands, 3-2
display, 3-28
editing commands, 3-10
error messages, 3-33example, 3-32
immediate mode commands, 3-31,
B-8
input/output commands, 3-10, B-5
key commands, 3-2, B-8
modes of operation, 3-2
pointer relocation commands,
3-16, B-6
search commands, 3-18, B-6
text modification, 3-20
utility commands, 3-24, B-7
Effective address search, 8-15
Empty entry, 9-13, 9-43
EMT,
and TRAP addressing, 5-27
argument blocks, 9-5
error code, 9-8
instruction, 9-2
.ENABLE directive, 5-36
.END directive, 5-50
End File command, 3-15
End of job (BATCH), 12-23
.EMDM directive, 5-60
.EMDM statement, 5-74
.EMDR statement, 5-74
ENTER handler function, H-8
.ENTER request, 9-47
Entering command information, 2-10
Entry conditions, H-4
Entry point, 5-10, 5-55, 6-7,
6-13, 9-46
.table, 7-5, 7-12, 7-13
Entry symbol, 6-5

INDEX-7 January 1976
Extensions, 3-12, K-2
and filenames, 2-5
filename (BATCH), 12-7
External,
expression, 5-18
symbols, 5-10, 6-5

F command, L-2
Facilities,
available only in RT-11 F/B, 1-8
for input and output operations, 9-1
Fast listing, J-7
Fatal monitor error messages, 2-39
Fatal system boot error messages, 2-38
F/B programming and device han-
dlers, H-1
F/B programming in RT-11, Version
2, H-1
F/B system, background area, 2-9
.FETCH request, 9-50
Field,
comment, 5-2, 5-4
label, 5-2, 5-3
operand, 5-4

File,
allocation scheme, 4-10
ASCII, 3-1
compressing, 4-19
concatenation, 12-16
deletion, 12-20, 0-40
descriptor blocks, 9-37
dummy, 4-6
formats, J-1
gap, 9-86, H-12
length, 9-78
linking, 12-28
manipulation requests, 9-15
manipulation services, 9-1
memory image, 2-3, 4-9, L-1
names and extensions, 2-5
non-overlay, L-3
overlay, L-1
permanent, 9-13
printing, 12-34
relocatable image, 2-3
replacement, 4-8.1
sentinel, 4-4
specifications (ASEMBl), 11-2, (BATCH), 12-6
structure, 9-13, H-6
temporary, 4-23, 12-7, 12-10, 12-15
tentative, 9-13
transfer, 4-1
File-formatted devices, J-1
File-oriented operations, 0-7
File-structured RT-11 device, 2-7, J-1
Filename, 3-12
blank extensions, 2-7
extensions, 2-6, 12-7

January 1976 INDEX-8
Filename (cont.),
  input, 4-1
  output, 4-1
FILEX, 1-4, J-1
  calling and using, J-1
  error messages, J-8
  overview, J-1
  switch options, B-18, J-2
Fill characters, 9-9
Fill count, 9-9
Find command, 3-19
Fixed record length, H-6
Floating point,
  exception, 9-84
  hardware, 5-47, 9-84
  numbers, 5-17, 5-48
  source double register, C-15
  .FLT2 directive, 5-48
  .FLT4 directive, 5-48
Foreground/Background, 1-1
  terminal I/O, 2-13
Foreground links, 6-6
Format,
  ASCII, 2-3
  control, 5-5
  load image, 2-3, 6-2, 6-21
  memory image, 2-3
  object, 2-3
  of Entry Point Table, 7-13
  of library files, 7-12
  of programmed requests, 9-2
  relocatable image, 2-3
  register, 8-6
  statement, 5-2
Formats, data, 2-3
Formatted binary copy switch, 4-9
Formatting,
  horizontal, 5-5
  vertical, 5-5
Form feed character, 3-1, 3-12
Forms of relocatable expressions, 8-5
$FORTRAN command, 12-23
FORTRAN IV, 1-1, 1-5, G-1
  character set, G-2
  compiler error diagnostics, G-11
  expression operators, G-3
  running a FORTRAN program in the foreground, G-1
  statements, G-4
FORTRAN System Subroutine Library, see SYSLIB
Forward space, H-11
Fragmented device, 9-13
Free area, 4-11
Free memory list, 2-9, 2-22
FRUN command, 2-36
FRUN /N option, O-6
FRUN processor, 9-7
Function,
  code, 9-6
  control switches, C-23
Function (cont.),
  switches, 5-76, 5-77
  types, O-2
FUNCTION subprograms, O-3
Functions, BASIC/RT-11, F-5

General,
  address specification, 5-21
  file transfer program, J-1
  library file format, 7-12
  memory layout, 2-8
  registers, 5-11, 8-9
GET command, 2-28
Get command, 3-18
GETSTR subroutine, O-28
Global symbol directory, 7-14
Global symbol table, 6-1, 6-5
Global symbols, 5-10, 5-55, 6-1, 6-5, M-3
Globals, unresolved, 6-1
  .GLOBL directive, 5-54
GOTO statement, 12-41
Graphics macro calls, summary, N-21
Graphics programs,
  assembling and linking, N-16
  GT OFF, 2-15, 2-16, N-2
  GT ON, 2-15, 3-29, N-2
  GTIM subroutine, O-29
  .GTIM request, 9-51
  GTJB subroutine, O-29
  .GTJB request, 9-52
HALT instructions, 2-41
Handler functions, H-8
  CLOSE, H-9
  DELETE, H-8
  ENTER, H-8
  LOOKUP, H-8
  READ/WRITE, H-9
Handler size, 9-46
Handlers, 2-36
  device, 2-8, 12-46, H-1, H-3.1
  display file, N-2
  functions, O-72
  internal display file, N-2
  removing from memory, 9-74
  unloading, 12-48
Hardware bootstrap, 4-20
Hardware,
  configuration, 2-1
  display, 3-29
  memory protection, 9-6
  mode, H-6, H-13.1
  requirements, 12-1
Header words, H-4
HELP command, M-5
  .HERR request, 9-53
High, address, 2-32
level languages, 1-3
memory address, 9-8
order time of day, 9-52
Horizontal formatting, 5-5
.HRESET request, 9-55

IADDR function, 0-30
IAFLT function, 0-31
IASCIGN function, 0-32
ICDPN function, 0-34
ICHCPY function, 0-35
ICMKT function, 0-36
ICSI function, 0-37
ICSTAT function, 0-39
IDELET function, 0-40
.IDENT directive, 5-36
Identification messages, 2-2
IDJFLT function, 0-41
IDSTAT function, 0-42
IENTER function, 0-43
IF statement, 12-40
IFETCH function, 0-44
.IPF Directive, 5-57
IFFREEC function, 0-45
.IPT Directive, 5-57
.IPTF Directive, 5-57
IGETC function, 0-46
IJCVT function, 0-47
Illegal characters, 5-7, 8-26
ILUN function, 0-47
Image mode transfer, 4-9
Immediate conditional directive, 5-58
Immediate mode, 1-3, 3-2, 3-4,
3-30, 5-24, 9-3
commands, 3-31, B-8
Important memory areas, 9-6
Include switch, 6-20
Inclusive directory, 4-5
Incrementing values of variable, 12-40
Indefinite repeat,
arguments, 5-72
block, 5-71, 5-72
Index Mode, 5-23, 5-25
Index Deferred Mode, 5-23
INDEX subroutine, 0-48
Individual module name, 7-2
Inhibit TT wait bit, 9-8
INITIALIZE command, 2-18
Initializing cassettes and magtapes, 4-13
Input and output, 3-3, 6-5
commands (Editor), 3-10, B-5
Input,
filenames, 4-1
list, 2-11
ring buffer, H-6
source filename, I-1
SYSLIB, 0-58
to RT-11 BATCH, 12-2

Insert command, 3-20
INSERT subroutine, 0-49
Inserting modules into a library, 7-5
.INSRT, N-5
Instruction,
EMT, 9-2
mnemonic, 5-3
offset, 8-17
Instructions, C-6
branch, C-13
double operand, C-8
operate, C-11
rotate/shift, C-9
single operand, C-8
trap, C-12
INTEGER*4 support functions, 0-12,
O-13, O-17
Integer value of the word, O-76
.INTEN request, 9-21, H-2, H-4
Interchange words of INTEGER*4
value, O-92
Internal,
buffers, 3-1
Macro buffer, 3-26
registers, 8-10
subpicture stack, N-14
symbol directory, 7-14
symbolic names, 5-53
symbols, 5-10
tables, 10-1
Interrupt,
enable bit, 9-44
handling, H-4, N-2
level (issuing programmed
requests), H-2
priorities, H-1
priority level, 8-10
Service Routine, H-2, H-3, H-5,
O-50
vector, H-1, H-3
INTSET function, 0-50
Invalid punch combinations, H-6
I/O control at terminal (BATCH), 12-42
I/O count, 9-78
exit routine, 9-11
vector, 9-33
IPEEK function, 0-52
IPOKE function, 0-52
IQSET function, 0-53
IRAD50 function, 0-54
IRCVD functions, 0-55
IREAD functions, 0-58
IRENAM function, 0-63
IREOPN function, 0-64
.IRP and .IRPC example, 5-73
.IRP directive, 5-71
.IRPC directive, 5-72
ISAVES function, 0-65
ISCHED function, 0-66
ISCOMP function, 0-108
ISDAT functions, 0-68
ISLEEP function, 0-71

January 1976
INDEX-10
ISPFN functions, 0-72
ISPY function, 0-76
Issuing programmed requests at the
interrupt level, H-2
Iteration argument, 3-8
loops, 3-9
ITIMER function, 0-77
ITLOCK function, 0-79
ITINIR function, 0-79
ITTOUR function, 0-81, 0-102
ITWAIT function, 0-82
IUNITL function, 0-83
IVERIF function, 0-120
IWAIT function, 0-84
IWRITE functions, 0-84

JADD function, 0-88
JAFIX function, 0-88
JCMP function, 0-89
JDFIX function, 0-90
JDIV function, 0-91
JICVT function, 0-92
JJCVT function, 0-92
JMMOV function, 0-93
JMUL function, 0-94
$JOB command, 12-25, 12-38
Job
execution resumed, 0-108
header, 12-3
parameters, 0-29
JSUB function, 0-95
JTIME subroutine, 0-96
Jump command, 3-17
Jumps, 6-13

Key commands, Editor, 3-2, B-8
Key, LINE FEED, 8-7
Keyboard,
commands, 2-14
communication (KMON), 2-11
error messages, 9-40
monitor (KMON), 2-7.1
Keyboard Monitor (KMON), 2-7.1
command summary, B-1
special function keys, B-3
Kill command, 3-22

Label field, 5-2, 5-3
Labels in RT-11 mode, 12-39
Language summary, BASIC/RT-11, F-1
Last block, H-12
Last file, H-12
LDA format, 6-2
switch, 6-21
Legal
argument delimiters, 5-63
characters, 5-5
Legal (cont.)
operations involving cassette
or magtape, 4-5
separating characters, 5-6, 5-67
wild card, 4-2
LEN function, 0-97
Length, string, 0-97
LET statement (BATCH), 12-40
Librarian (LIBR), 1-2, 1-3, 1-4,
7-1
calling and using, 7-1
error messages, 7-14
switch commands, 7-2, 7-3, B-12
Library
BATCH default, 12-4
directory, 7-1
deck trailer, 7-14
header, 7-1, 7-12, 7-13
processing, 7-14
searches, 6-17
$LIBRARY command, 12-27
Library files, 6-8
creation, 7-4
directory listing, 7-9
entry point table, 7-7
format, 7-12
inserting modules into, 7-5
merging, 7-10
SYSLIB, 0-6
Library functions and subroutines,
0-21 - 0-120
Light pen, N-1
status buffer, N-11
.LIMIT directive, 5-51
Limitations, PATCHO, M-5
Line- and character-oriented
command properties, 3-6
Line continuation character, 12-2
Line deletion, 9-94
formatting, 5-5
oriented commands, 3-6
printer overstriking capability, 2-23
LINE FEED key, 8-7
Link and execute programs, 12-5
$LINK command, 12-27
Linkage map, temporary, 12-4
Linker (LINK), 1-2, 1-4, 6-1, 7-14
calling and using, 6-2
error handling and messages, 6-24
input and output, 6-5
load map, 8-2
load map for background job, 6-9
switches, 6-3, B-11
Linked
list, 9-66
program, 6-1
Linking graphics programs, N-16
Linking, relocation and, 5-19
Linking with SYSLIB, 0-6
List command, 3-14
LIST command, M-4
.LIST directive, 5-27
Listing,
  assembly, 8-1
  control directives, 5-27
  control switches, 5-76, C-23
  cross-reference, 1-4
directories, J-6
file, temporary, 12-4
level count, 5-28
the directory of a library file, 7-9
.LNKRT, N-5
Load address, 2-36
  absolute, 6-1
LOAD command, 2-20
Load image format (.LDA), 2-3
Load
  map, 1-2, 6-1, 6-7
  module, 1-2, 6-1, 6-5, 6-21
Loading
  BATCH, 12-45
device handlers, 9-50, 0-44
device registers, 9-44
memory image files, 2-33
ODT with user program, 8-2
root segment, 2-28
Local symbol block, 5-13
Local symbols, 5-12, 5-66
  automatic generation, 5-13
Location,
  addressed, 8-8
closed, 8-6
counter arithmetic, 6-4.1
counter control, 5-46
open, 8-6
Locations, opening, changing, and
  closing, 8-6
..LOCK request, 9-56
LOCK subroutine, O-97, 0-119
Locking USR in memory, 9-56
Log device, 12-1
Logical device name, 2-5
BATCH, 12-6
  end-of-tape, 4-5
  identifier, 9-5
  names, 2-5, 2-18
LOGICAL*1 array, O-19
LOOKUP function, O-99
LOOKUP handler function, H-8
.LOOKUP request, 9-58
Low and high addresses, 5-51
  -order priority, 8-19
  priority, 8-19
Lower-case bit, 9-7
Lower-case characters, O-80
Lowering the processor priority, H-2
.BASE, N-7
L-shaped cursor, 3-29
LSRA instruction, N-14
Machine language, 1-2
Macro,
  arguments, 5-7
  buffer, 3-10
call, 5-3, 9-2
call operator, 5-3
definition, 5-60
definition formatting, 5-61
expansion, 9-24
free source code, 10-1
libraries, 5-74
nesting, 5-63
recursive, 10-6
references, 10-1
Macro assembler, 1-1, 1-2, 5-1
calling and using, 5-74
counter code, C-1
directives, 5-60
error messages, 5-84
features, 5-1
instructions, C-1
program section capabilities, 5-10
source code, 5-80, 5-81
source statements, 5-2
special characters, C-5
switches, 5-76, C-23
symbol table, 5-9
symbols, 5-9
Macro calls,
  requiring no conversion, 9-108
  which may be converted, 9-108
Macro command, 3-10, 3-25
$MACRO command, 12-29
MACRO used with SYSLIB, O-3
.MACRO directive, 5-60
Magtape handler function, O-72
Magtape special functions, H-11
  back space, H-11
  forward space, H-11
  off line, H-12
  rewind, H-11
  write end-of-file, H-11
  write with extended gap, H-12
Main file/subroutine structure, N-19
Main line code, H-5
Mainstream code, 9-87
Making patches permanent, 2-28
Manipulating memory images, 2-28
Manual bootstraps, 2-2
Mark time, 1-8
  requests, 9-31
Mask limit, 8-15
Masking, 8-10
Matching areas, K-1
Maximum file size, 9-12
.MCALL directives, 5-74
Memory block initialization, 8-16
Memory
  allocation (SYSLIB), O-43
  BATCH, 12-1
core, 1-1
diagram, BASIC link with overlay
  regions, 6-11
Memory (cont.),
   examination, 0-52
   maps, 2-8, 8-4
   solid state, 1-1
   storage, 0-52
   usage, 3-9
   usage map, 6-6
Memory image, L-1
   files, 2-3, 4-9, L-1
   format, 2-3
Merging library files, 7-10
$MESSAGE command, 12-31
Message transfers under P/B, 0-68
.MEXIT directive, 5-61
.MEPS request, 9-21.1
Minimum memory configurations, 5-1
Miscellaneous operations, 0-14
Miscellaneous services, 9-16
Missing arguments, 5-66
Mnemonics, 1-2

Mode,
   absolute, 5-24
   autodecrement, 5-23
   autodecrement deferred, 5-23
   autoincrement, 5-21
   autoincrement deferred, 5-22
   command, 3-2
   general, 9-34
   hardware, H-6
   immediate, 1-3, 3-2, 3-4, 3-30, 5-24
   index, 5-23, 5-25
   index deferred, 5-23
   instruction, 8-17
   Radix-50, 8-10
   register, 5-21
   register deferred, 5-21
   relative, 5-24
   relative branch, 8-9
   relative deferred, 5-25
   single instruction, 8-14, 8-17
   software, H-6
   source operand, 5-25
   special, 9-36
   text, 3-2, 3-20
   type-ahead, 2-14

Modes, addressing, 5-20
   of operation (EDIT), 3-2

Modify stack address, 6-21

Modules, 1-2, 6-5
   absolute load, 6-5
   load, 6-1, 6-5, 6-21, 7-1
   object, 6-5, 6-16, 7-1, 7-13, 7-14
Monitor, 1-1, 2-1
   error messages, 2-38
   F/B, 1-6, 9-7
   HALTe, 2-41
   keyboard (KMON), 2-7.1
   memory protection map, 6-6
   Resident (RMON), 2-7.1, 9-7
   single-job, 1-6, 3-30, 9-7

Monitor (cont.),
   software components, 2-7.1
   start procedure, 2-1
   version number, 9-11
   $MOUNT command, 12-32
   MOV instruction, 9-3
   Move INTEGER*4 variable, O-93
   MRKT function, O-100
   .MRKT request, 9-60
   MT/CT (Magtape (TU10/TM11) and
   Cassette (TA11), H-6
   general characteristics, H-6
   .MTPS request, 9-21.1
   Multiple,
   command lines, 2-14
   copy operations, 4-8, 4-11
   delimiters, 5-4
   GETs, 2-28
   labels, 5-3
   operands, 5-4, 5-38
   .QSET requests, 9-66
   Multiply-defined symbols, 5-13
   Multiply two INTEGER*4 values, O-94
   .MWAIT request, 9-62
   MWAIT subroutine, O-101

[n] construction, 4-11
   .NAME, N-9
Name register,
   contents, N-14
   internal software register, N-15
   .NAME request, N-9
Name value, N-7
Names of devices,
   default (BATCH), 12-6
Named
   COMMON, 6-14
   control sections, 6-4
   relocatable program sections,
   5-53, 5-54
   .NARG directive, 5-68
   .NCH directive, 5-68
Negative,
   line arguments, 3-7
   numbers, 5-17
Nesting level, 5-63

Next
   block, H-12
   file, H-12
   Next command, 3-14
   .NLIST, 5-27
Nonfile-structured,
   delete, 4-5
   devices, 2-7, 6-21
   enter, 9-41
   lookup, 9-41, H-7, H-13.2
Non-overlay file, L-3
Non-time-critical job, 1-8
Nonexistent symbol, 6-21
NonRT-11 directory-structured
   devices, 2-7, 9-45

INDEX-13           January 1976
.NTYPE directive, 5-69
Null specification, 11-2
Number, C-18
  channel, 9-5
decimal, 5-17
monitor version, 9-11
of arguments, 5-66
update, 9-11
Numbers,
  floating-point, 5-17, 5-48
MACRO assembler, 5-17
negative, 5-17
octal, 5-17
positive, 5-17
software identification, xxii
Numeric arguments, 3-6, 9-4
  passed as symbols, 5-64
Numeric control, 5-47

Object
code, 1-2
files, 6-5
files, temporary, 12-4, 12-15
format, 2-3
modules, 5-19, 6-5, 6-16, 7-1, 7-13, 7-14
modules, relocatable, 8-1
modules, starting point, 7-13
output, 1-2
.OBJ format, M-1
Object Time System (OTS), 0-32
Octal,
  channel number, 9-13
  numbers, 5-17
  radix, 5-17, 5-45
Octal-Decimal conversions, C-25
.ODD directive, 5-46
Odd (high-order) byte, 9-6

ODT
(On-line debugging technique), 1-5
break routine, 8-23
calling and using, 8-1
command summary, B-12
error detection, 8-25
functional organization, 8-20
linking ODT with the user
  program, 8-2
priority bit, 8-2
priority level, 8-19
Offline, H-12
Offset, 5-13, 5-25
  relative branch, 8-9
  relative branch instruction, 8-17
Offset words, 9-11
On-line debugging technique
  (ODT) - see ODT
OPEN command, M-1
Open file, 2-20
Opening a byte address, L-3
Opening, changing, and closing
  locations, 8-6
Operand field, 5-4

Operands, multiple, 5-38
Operate Instructions, C-11
Operating procedures (BATCH), 12-45
Operation,
  foreground/background, 1-1
  single-job, 1-1
Operations on files, 4-2
  magtape and cassette, 4-4
Operator
  action, wait for, 12-5
  characters, 5-8
  communication (directives), 12-49
  directives, 12-50
  field, 5-3
  message, 12-31, 12-42
Operators,
  binary, 5-8, 5-18
  unary, 5-8, 5-45, 5-50
Optional characters, 2-14
Output,
  filenames, 4-1
  format, K-2
  list, 2-11
  ring buffer, H-6

Packed image transfer, J-2
Padding routine, O-111
Page, 3-1
eject, 5-36, 5-61
headings, 5-34
.PAGE directive, 5-36
.PAL-11R directive, 5-59
PAL-11S conditional assembly di-
  rective, 5-59
Parameter list, 2-31
Parameters used as arguments, E-1
Passing strings to subprograms, 0-20
Passing switch information, 9-35, 9-38
PATCH utility program, 1-4, L-1
  commands, L-2
  command summary, B-20
  error messages, L-7
  examples, L-4
PATCH,
  calling and using, L-1
Patching,
  libraries, M-1
  new files, L-2
  OBJ files, M-1
PATCHO, 1-4
  commands, M-1
  command summary, B-21
  error messages, M-7
  examples, M-6
  limitations, M-5
  run-time error messages, M-8
PATCHO,
  calling and using, M-1
PDP-11 DOS/BATCH DECTape, J-1
Percent (%) character, 5-12, 12-40
Peripheral Interchange Program
  (PIP), see PIP

January 1976
INDEX-14
Permanent, device names, 2-5
files, 2-20, 9-13
Permanent Symbol Table, 5-9
Physical block numbers, I-2
Physical device, 2-19, 9-5
names, 2-4, 12-6
PIC (position independent code), 8-18, 9-21
PIP (Peripheral Interchange Program), 1-4, 4-1
copy operation, 4-9
error messages, 4-24
magtape or cassette operations, 4-4
switches, 4-2, 4-3
switch summary, B-9
warning messages, 4-25
PIP, calling and using, 4-1
POINT command, M-2
Pointer location, 3-3
Pointer relocation commands, 3-16
for Editor, B-6
Position command, 3-20
Position independent code (PIC), 8-18, 9-21
Positive, arguments, 3-7, 4-5
numbers, 5-17
Power line synchronization feature, N-11
PR (High-Speed Paper Tape Reader), H-5
$PRINT command, 12-34
.PRINT directive, 5-71
.PRINT request, 9-63
PRINT subroutine, O-102
Printout formats, 8-5
Priorities, Interrupt, H-1
Priority, 9-21.1, C-18
Proceed command, 9-12
Proceed count, 8-13
Processor Status, 8-1, 9-21.1, H-2
Program Boundaries directive, 5-51
Program,
counter, 5-20, 8-8
development aids, 1-2
dexample (BATCH), 12-36
execution, 8-12
runaway, 8-23
section directives, 5-51
section names, 5-53
sections, absolute and relocatable, 6-4
Program starting commands, 2-33
R, 2-34
REENTER, 2-35
RUN, 2-33
START, 2-34
Programmed requests, 2-7.1, 9-1
format, 9-2
summary, E-1
usage, 9-25
Programming
considerations, 8-20
conventions, H-1
errors, 1-3
Prompting characters, 2-4
.PROTECT request, 9-64, H-3
Punched cards input, 12-2, 12-44
.PURGE request, 9-65
PURGE subroutine, O-103
Purging an inactive channel, 9-65
PUTSTR subroutine, O-103
.QSET request, 9-65
Quantities, absolute, 5-17
Queue element, 9-61
Queue size, O-53
Quoted-string literals, O-21
R50ASC subroutine, O-104
RAD50 function, O-105
.RADIX directive, 5-44
Radix,
binary, 5-17, 5-45
c control, 5-44
decimal, 5-17, 5-45
octal, 5-17, 5-45
specification characters, 5-45
Radix-50,
character set, C-3
conversion operations, 0-14,
0-54, O-104
equivalents, C-4
mode, 8-10
notation, 5-36
terminators, 8-11
.RAD50 directive, 5-43
Random-access,
devices, 2-7, 4-8.1, 6-1, 9-45
file capabilities, F-1
R Command, 2-34
RCHAIN subroutine, O-105
.RCTRLO request, 9-67
RCTRLO subroutine, O-106
.RCV request, 9-68
.RCVD request, 9-68
.RCVDW request, 9-69
READ command, 3-12
Read functions, 0-58
.READ request, 9-71
.READC request, 9-72
Reading an ASCII record, 0-28
.READW request, 9-73
READ/WRITE handler function, H-9
Real arguments, 5-63
Re-assembling, 1-3
Rebooting the system, 4-19

INDEX-15 January 1976
Receive data functions, 0-55
Receiving data, 9-68
Reclaiming memory, 2-22
Reconfiguration, 1-7
Record, H-9
Recovering files, 4-14
Recovery from Bad Blocks, 4-21
Recovery procedures during output operations, 4-8
Recurring coding sequence, 5-60
Recursive macros, 10-2
Reducing disk fragmentation, 4-12
Re-editing, 1-3
Reenter bit, 2-35, 9-7
REENTER command, 2-35, 3-2
Reference line, K-3
-REGDEF, macro call, 9-22
Region number, 6-13
Region, overlay, 6-10, 6-14
Register Deferred Mode, 5-21
Register,
  destination, C-14
  expression, 5-20, C-5
  mnemonic, 9-3
  mode, 5-21
  symbols, 5-11, 5-12
Register-Offset, C-14
Registers,
  console terminal control and status, 9-12
  constant, 8-16
  relocation, L-1
Reinitializing monitor tables, 4-20
Relative,
  branch, 8-15
  branch offset, 8-9
  deferred mode, 5-25
  mode, 5-24
-RELEASE request, 9-74
Releasing USR from memory, 9-57
REL format, 6-1
  output file, 6-2
  switch, 6-23
Relocatable, 1-2, 5-3
  code, 6-4.1
  expressions, 5-18, 8-4
  image file, 2-3
  image format (.REL), 2-3
  object module, 8-1
  values, 8-18
Relocation, 8-4
  base, 2-29
  bias, 8-4, 8-17
  calculators, 8-18
  constant, 5-3
  directory, 7-14
  register commands, 8-17, 8-18
  registers, 8-6, L-1
Relocation and Linking, 5-19
  .REMOV, N-9

Removing,
  handlers from memory, 9-74
  logical assignments, 2-19
  terminated jobs, 2-22
Rename function, 0-63
Rename operation, 4-15
-RENAME request, 9-75
Reopen a channel, 0-64
-REOPEN request, 9-77
Repeat block, 5-73
Repeat counts, 8-23
Repeat subroutine, 0-107
Replace switch, 7-5
Replacement file, 4-8.1
-REPT directive, 5-73
Requests
  for data transfer, 9-14
  for file manipulation, 9-14
  for miscellaneous services, 9-14
  requiring the USR, 9-19
Requests, programmed, 9-1
Requests
  .CDPN, 9-26
  .CHAIN, 9-27
  .CHCOPY, 9-28
  .CLOSE, 9-30
  .CMKT, 9-31
  .CNTXSW, 9-32
  .CSIGEN, 9-33
  .CSISP, 9-36
  .CSTAT, 9-41
  .DELETE, 9-42
  .DEVICE, 9-44
  .DSTATUS, 9-45
  .ENTER, 9-47
  .EXIT, 9-49
  .FETCH, 9-50
  .GETM, 9-51
  .GETJ, 9-52
  .HERR, 9-53
  .HRESET, 9-55
  .LOCL, 9-56
  .LOOKUP, 9-58
  .MPFS, 9-21.1
  .MRKT, 9-60
  .MTPS, 9-21.1
  .MWAIT, 9-62
  .PRINT, 9-63
  .PROTECT, 9-64
  .PURGE, 9-65
  .QSET, 9-65
  .RCTRL, 9-67
  .RCVD, 9-68
  .RCVD, 9-69
  .RCVDW, 9-69
  .READ, 9-71
  .READC, 9-72
  .READW, 9-73
  .RELEAS, 9-74
  .RENAME, 9-75
  .REOPEN, 9-77
  .RSUM, 9-87

January 1976
INDEX-16
Requests (cont.),
  .SAVSTATUS, 9-77
  .SDAT, 9-80
  .SDATC, 9-81
  .SDATW, 9-81
  .SERR, 9-53
  .SETTOP, 9-82
  .SPFA, 9-84
  .SPFUN, 9-85, H-13
  .SPND, 9-87
  .SRESET, 9-90
  .TLOCK, 9-91
  .TRPSET, 9-92
  .TTINR, 9-94
  .TTYIN, 9-93
  .TTYOUT, 9-95
  .TTOUTR, 9-95
  .TWAIT, 9-98
  .UNLOCK, 9-57
  .WAIT, 9-71, 9-99
  .WRITC, 9-101
  .WRITE, 9-100
  .WRITW, 9-102
Reserving storage area, 5-16
Resident monitor (RMON), 2-7.1, 9-7
Resident overlay handler, 6-15
Restarting ODT, clearing breakpoints, 8-2
Restarting PIP, 4-1
  .RESTR, N-9
Restrictions on expanding macros, 10-1
RESUME subroutine, 0-108, 0-113
  Return
  from Interrupt Service, H-2
  to previous sequence, 8-9
  to monitor, CTRL C, 8-3
  Return path, 6-13
  Reusing bad blocks, 4-19
  Rewind, H-11, H-12, 0-72
  Ring buffer I/O, H-6
  Root segment, 2-28, 6-10, 6-14, L-3
  Rotate/Shift Instructions, C-9
  Rounding numbers, 5-48
  Routine, Service Interrupt, H-2
  Routines, 1-3
  Routines, completion, 2-37, 2-38
  Routines used with FORTRAN, 0-4
  RSTS-11 DCTape, J-1
  .RSUM request, 9-87
  RSUM Command, 2-38
  RT-11 BATCH and RSX-11D BATCH
differences, 12-52, 12-53
  $RT-11 command, 12-35, 12-38
  RT-11 directory-structured
deVICES, 2-7
  RT-11 input, 12-2
  RT-11 mode, 12-38
  examples, 12-43
  switch, 12-4
  RT-11 services, 0-11, 0-12
  RT-11 System, 1-1
  data formats, file transfers, 4-1
  DECtape control handler, J-5
  Foreground/Background monitor, 1-7
  I/O transfers, 9-65
  librarian, 7-1
  memory maps 2-8, 2-9
  operating environments, 1-1
  Single-Job monitor, 1-7
  RUBOUT, 2-13, 3-3
  Rules and conventions, 12-11
  Rules for user-defined and macro
  symbols, 5-9
  $RUN command, 12-35
  RUN command, 2-33
Running,
  a FORTRAN program in the fore-
ground, O-5
  BATCH, 12-47
  RT-11 system programs, 12-39
  the program (ODT), 8-12
  Run-time area of memory, 6-10
  Run-time handler, 12-1, 12-45, 12-50
  Run-time overlay handler, 6-12
  Run-time overlay handlers and
tables, 6-1
  Run-time variables, 12-16
Save buffer, 3-10
SAVE command,
  (Monitor), 2-31
  (Editor), 3-10, 3-24
Save image, 6-1, 6-2
  file (SAV), 6-5, 12-23
  format, 2-31, 6-21
  SAVSTATUS request, 9-77
  SBTTL directive, 5-34
  Schedule function, 0-66
  Scheduling requests, cancelling, 0-36
  Scheduling subroutines, 0-77
  SCOMP routine, 0-108
  SCOPY routine, 0-109
  .SCROL, N-10
  Scroller, 2-15, 3-29, N-2
  buffer, N-3
  logic, N-3
  .SDAT request, 9-80
  .SDATC request, 9-81
  .SDATW request, 9-81
  Search,
  algorithm (ODT), 8-24
  commands (EDIT), 3-4, 3-18, B-6
  effective address, 8-15
  for string, 0-48
  limit (ODT), 8-15
  word, 8-15
  Searches, 8-14, 8-24

INDEX-17
January 1976
SECNDS function, 0-110
Segment boundaries, 4-18
Send Data/Receive Data, 1-8
Sentinel file, 4-4
Separating and delimiting
characters, 5-6
Separator, 2-11
$SEQUENCE command, 12-36
Sequence numbers, 4-7
Sequential-access devices, 2-7, 9-45, H-13
Sequential operations, H-9
.SERR request, 9-53
Services, 0-7
Set Bottom Address, L-4
SET command, 2-23
options, 2-23
Set Relocation Registers, L-4
Setting the Editor to Immediate
mode, 3-30
Setting up interrupt vectors, H-3
.SETTOP request, 9-82
.SET USR NOSWAP command, 9-84
Seven-word status buffer, N-10
.SFFA request, 9-84
Sharing a location, 9-32
Sharing system resources, 1-1
Single absolute section, 5-54
Single instruction,
address, 8-14
mode, 8-14
Single-job
monitor, 2-18, 3-30, 9-7
operation, 1-1
Single load module, 7-1
Single operand instructions, C-8
Size specification, 9-37
Slash, 8-6
Soft error recovery, 9-54
Software,
components, monitor, 2-7.1
mode, H-6, H-13
name register, H-7
requirements (BATCH), 12-2
Software Identification Numbers,
xxii
Solid state memory, 1-1
Source compare (SRCCOM), 1-4, K-1
error messages, K-5
switches, K-2, B-19
Source-Double Register, C-17
Source,
code, 1-2
code, macro-free, 10-1
field, 9-3
lines, 5-2
external file
operand mode, 5-25
program, 1-2, 5-2
program format, 5-2
register, C-15
Spacing to the end of the tape,
4-6
Special characters, 5-64
Special key commands, EDIT, 3-2
Special function keys, 2-12, B-3
CTRL A, 2-12
CTRL B, 2-12
CTRL C, 2-12
CTRL E, 2-12
CTRL F, 2-12
CTRL O, 2-13
CTRL Q, 2-13
CTRL S, 2-13
CTRL U, 2-13
CTRL Z, 2-13
RUBOUT, 2-13
Special functions, O-72
Special mode TT bit, 9-8
Specification fields, 12-5
switches, 12-7
Specifier,
address, M-2
value, M-2, M-3
Specifying,
directory segments, 4-19
extra words per directory
entry, 4-18
.SFUN request, 9-85, H-13.2
.SPND request, 9-87
SRCCOM, see Source Compare
.SRESET request, 9-90
Stack, 9-50
address, 6-21
pointer, 6-21, 9-7
size, changing, 2-36
Standard end-of-file card, H-6
.START, N-10
Start address, 6-7, 9-7, H-3
START Command, 2-34
Start procedure, 2-1
Starting and stopping the display
processor, N-2
Starting block number, 9-78
.STAT, N-10
Statement,
format, 5-2
terminator, 5-2
Statements,
assembly language, 5-2
direct assignment, 5-10, 5-14
BASIC/RT-11, F-1
.ENDM, 5-74
.ENDOR, 5-74
.MACRO, 5-2
Status word, 9-45, H-1
.STOP request, N-11
Stopping points, 1-3
Storage device, 3-1
String, BASIC/RT-11, F-5
Strings (SYSLIB), 0-18
concatenation, 0-24
length, 0-97
STRPAD routine, 0-111
Subconditionals, 5-57

January 1976
INDEX-18
System software components (cont.)
SRCCOM, 1-4, K-1
SYSLIB, 1-5, 0-1
System subroutine library, see
SYSLIB

TAB character, 3-3, 5-2
Table of
breakpoints, 8-12, 8-13
mode forms and codes, 5-25
proceed command repeat counts, 8-13
Tag value, N-7
Tagged subpicture file structure, N-18, N-20
Temporary files, 4-23, 12-15
listing, 12-4
object, 12-4, 12-28
Temporary linkage map, 12-4
Temporary numeric control, 5-49
Temporary radix control, 5-45
Tentative entry, 9-47
Tentative file, 9-13, 9-65
Terminal,
input request, 2-14
interrupt, 8-24
I/O control, 12-42
Terminate search, 8-4
Terminating BATCH, 12-52
control statement, 12-2
jobs on cards, 12-45
Terminating directives, 5-50
Terms, 5-17
Testing patches, 2-28
Text, 1-2
blocks, 7-14
buffer, 3-2, 3-21
Editor, 3-1
mode, 3-2, 3-20
registration, 3-4
registration commands, 3-20
TIMASC subroutine, 0-114
TIME command, 2-17
Time conversion, internal to
ASCII, 0-114
Time-critical job, 1-7
Time format, 0-17
Time of day, 12-5, 0-29, 0-115
Time of day, access to, 9-51
Time interval, 9-60
TIME subroutine, 0-115
Timed Wait, 1-8
Timer support functions, 0-10,
0-11, 0-96, 0-110, 0-115
Timing requests, 9-66
.TITLE directive, 5-34
.TLOCK request, 9-91
Trace trap instruction, 8-21
.TRACK completion routine, N-12
.TRACK request, N-12
Tracking object, N-2, N-12, N-19
Trailing blank, 0-118
Trailing delimiter, 5-43
Transfer address, 6-7, 6-24
Transfer,
image mode, 4-9
packed image, J-2
word-for-word, J-2
Transferring characters, 9-93
Transferring files, 4-8, 4-11
between RT-11 and DOS/BATCH
(or RST), J-3
to RT-11 from DECsystem-10, J-5
Transferring memory, 2-31
TRANSL routine, 0-116
Transmitting data, 9-68
TRAP addressing, 5-27
Trap instructions, 8-14, C-12
Trap interception, 9-92
TRIM routine, 0-118
.TRP SET request, 9-92
TT
handler for console terminal, H-5
printer interrupt, 8-25
.TTYIN request, 9-93
.TTYINR request, 9-94
.TTYOUT request, 9-95
.TTYOUTR request, 9-95
Turning off user error interception, 9-54
.TWAIT request, 9-98
Two-volume compress, 4-19
Type ahead, 2-14, H-5
Types of programmed requests, 9-14

UIC, User Identification Code, J-3
default value, J-4
Unary operators, 5-8, 5-45, 5-50
Unconditional transfer of program
control, 12-41
Unit number (system device), 9-11
Unique spelling of switches and
keynames, 12-5
.UNLINK, N-13
UNLOAD command, 2-21
.UNLOCK request, 9-57
UNLOCK subroutine, 0-118
Unnamed control section, 6-4
Unrecoverable hardware/software
error, 7-15
Unresolved globals, 6-1
Unsafe command, 3-25
Unused areas, 4-16
Up-arrow construction, 5-63
Up-arrow, 8-8
prompt, H-5
Update,
number, 9-11
switch, 7-9
Updating cassette sequence number,
4-7
Upper-case characters, 0-80

January 1976 INDEX-20
Upper/lower-case, commands, 3-27
mode, 3-27
terminal, 3-27
Urgent messages, 9-63
User command string, 3-1
User-defined symbol, 5-3, 5-9
table, 5-3
User library searches, 6-16
User program, 5-3
protection, 9-34
User Service Routine (USR), 2-7.1
address, 2-32
swapping, 2-27
User switch commands and functions (LIBR), 7-2
User-written device handlers, H-4
User symbol table, 5-9
Using,
.ASECT Directives, H-3
display editor, 3-29
libraries, 6-15
ODT with P/B jobs, 8-3
overlays, 6-10
.SETTOP, H-3
the system macro library, 9-14
the display file handler, N-16
the wild-card construction, 4-1
USR area, 9-11
load address, 9-8
USR
access to (SYSLIB), O-79
ownership acquisition, 9-91
swap bit, 9-7
swapping, 9-9, 9-79
unlocking, O-118
Utility commands, Editor, 3-4,
3-24, B-7
Utility program, PATCH, L-1
Utility routines, 8-20

.V1. macro call, 9-24
.V2. macro call, 9-24
Value specifier, M-2, M-3
Variables,
character string, O-19
INTEGER*4, O-17
FT-11 mode, 12-40
run-time, 12-16
Vector addresses, 9-6
Verify command, 3-15
VERIFY routine, O-119
Version,
history, xxii
number message, 4-21, L-1
switch, 4-21
Vertical formatting, 5-5
VT-11 Display processor, 1-3, 2-8,
2-15
VTBASE, N-24

VTHDLR, N-25
VTLIB library, N-26
VTMAC, N-27

Wait for operator action, 12-5
.WAIT request, 9-71, 9-99
Warning messages, PIP, 4-25
Wild card,
construction, 4-10, 12-7, 12-16
expansion, 4-10
names, J-1
Word,
address, L-3
count, H-9
-for-word transfer, J-2
search, 8-15, 8-24
status, H-1
WORD command, M-2
Word count, variable number, 9-68
.WORD directive, 5-39
Words, allocating, 2-36
.WRITC request, 9-101
Write,
character string, O-103
data-file, H-11
file gap, H-13
with extended gap, H-12
Write command, 3-13
.WRITE request, 9-100
WRITE-ENABLE, 12-5
Write function, O-84
WRITE-LOCK, 12-5
.WRITW request, 9-102

INDEX-21

January 1976
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. ☐