RSTS/E
System Directives Manual

Order No. AA–D748C–TC

March 1983

This manual contains general information on run-time systems and describes RSTS/E monitor, RSX emulator, and RT11 emulator directives for the assembly-language programmer.

OPERATING SYSTEM AND VERSION:	RSTS/E	V8.0
SOFTWARE VERSION:	RSTS/E	V8.0

digital equipment corporation, maynard, massachusetts

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

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

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1981, 1983 by Digital Equipment Corporation.
All Rights Reserved.

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

The following are trademarks of Digital Equipment Corporation:

digital™	DECwriter	RSTS
	DIBOL	RSX
MASSBUS	UNIBUS	DECmate
PDP	VAX	DECsystem–10
P/OS	VMS	DECSYSTEM–20
Professional	VT	DECUS
DEC	Rainbow	Work Processor

Commercial Engineering Publications typeset this manual using DIGITAL's TMS–11 Text Management System.

Contents

Page

Preface xi

Summary of Technical Changes xiii

Part I Introduction

Chapter 1 Introduction

1.1 Run-Time Systems 1–1
1.1.1 Environments for People 1–2
1.1.2 Environments for Programs 1–2
1.2 Jobs. 1–5

Chapter 2 General RSTS/E Environment

2.1 How RSTS/E Allocates Memory — Physical and Virtual Addressing . . 2–1
2.2 Job Space — High Segment and Low Segment 2–4
2.3 Important Installation Options 2–6
2.3.1 The 'Disappearing' RSX Run-Time System 2–6
2.3.2 Resident Libraries. 2–6
2.4 Low-Segment Details — First 1000 Bytes of the Low Segment 2–7
2.5 High-Segment Details — Pseudo Vectors 2–15
2.5.1 Run-Time System Capability and Default Definitions 2–17
2.5.2 Synchronous System Trap Addresses. 2–21
2.5.3 Asynchronous System Trap Addresses 2–23
2.5.4 Entry Points 2–25

Part II Monitor Directives

Chapter 3 General Monitor Directives

3.1 Introduction. 3–1
3.1.1 Summary of General Monitor Directives 3–1
3.1.2 Prefix File COMMON.MAC 3–3
3.1.2.1 How to Assemble with COMMON.MAC 3–3
3.1.2.2 Macros Provided in COMMON.MAC 3–3
3.1.3 Error Mnemonics: Link-File ERR.STB 3–7
3.1.4 Programming Hints 3–7
3.2 CALFIP — Call the File Processor. 3–10
3.2.1 ASSFQ (Assign a Device) 3–11
3.2.2 CLSFQ (Close a Channel) 3–14
3.2.3 CRBFQ (Create a Binary [Executable] File and Open it on a Channel) 3–17
3.2.4 CREFQ (Create a File and Open It on a Channel) 3–23

3.2.5 CRTFQ (Create and Open a Temporary File) . . . 3–32
3.2.6 DALFQ (Deassign All Devices) . . . 3–38
3.2.7 DEAFQ (Deassign a Device) . . . 3–40
3.2.8 DIRFQ (Get Directory Information) . . . 3–42
3.2.9 DLNFQ (Delete a File) . . . 3–50
3.2.10 ERRFQ (Return Error Message Text) . . . 3–53
3.2.11 LOKFQ (Disk File/Wildcard Lookup) . . . 3–55
3.2.12 OPNFQ (Open a File/Device on a Channel) . . . 3–65
3.2.13 RENFQ (Rename a File) . . . 3–74
3.2.14 RSTFQ (Reset a Channel) . . . 3–77
3.2.15 UUOFQ (Hook to File Processor) . . . 3–80

3.3 .CCL — Check String for CCL Command . . . 3–81
3.4 .CHAIN — Execute Under Same RTS . . . 3–86
3.5 .CLEAR — Clear Keyword Bits . . . 3–87
3.6 .CORE — Change Memory Size . . . 3–89
3.7 .DATE — Return Current Date and Time . . . 3–92
3.8 .ERLOG — Log an Error from RTS . . . 3–94
3.9 .EXIT — Exit to Default Keyboard Monitor . . . 3–95
3.10 .FSS — Check File Specification String . . . 3–96
3.11 .LOGS — Check for Logical Devices . . . 3–109
3.12 .MESAG — Message Send/Receive . . . 3–116

3.12.1 Declare Receiver Subfunction . . . 3–117
3.12.2 Remove Receiver Subfunction . . . 3–119
3.12.3 Send Local Data Message Subfunction . . . 3–120
3.12.4 Receive Subfunction . . . 3–123

3.13 .NAME — Install Program Name with Monitor . . . 3–127
3.14 .PEEK — Look at Monitor Memory . . . 3–129
3.15 .PLAS — Access Resident Library . . . 3–132

3.15.1 ATRFQ (Attach Resident Library) . . . 3–133
3.15.2 CRAFQ (Create Address Window) . . . 3–137
3.15.3 DTRFQ (Detach Resident Library) . . . 3–143
3.15.4 ELAFQ (Eliminate Address Window) . . . 3–146
3.15.5 MAPFQ (Map Address Window) . . . 3–149
3.15.6 UMPFQ (Unmap Address Window) . . . 3–154

3.16 .POSTN — Return Current Horizontal Position . . . 3–156
3.17 .READ — Read Data from File or Device . . . 3–158
3.18 .RSX — Execute Job and Disappear (RSX only) . . . 3–164

3.18.1 No Requests for Extra Space . . . 3–164
3.18.2 Request for Mapping a Window in APR 7 . . . 3–165
3.18.3 Request to Extend the User Job Image . . . 3–166

3.19 .RTS — Pass Control to Run-Time System . . . 3–167
3.20 .RUN — New Program to Run . . . 3–172
3.21 .SET — Set Keyword Bits . . . 3–176
3.22 .SLEEP — Suspend Job . . . 3–178
3.23 .SPEC — Special Functions for I/O . . . 3–180

3.23.1 .SPEC for Disk . . . 3–180
3.23.2 .SPEC for Terminal . . . 3–183

3.23.2.1 All but Private Delimiters . . . 3–183
3.23.2.2 Private Delimiters . . . 3–185

3.23.3 .SPEC for Magnetic Tape . . . 3–195
3.23.4 .SPEC for RX01/RX02 Flexible Diskette . . . 3–200
3.23.5 .SPEC for Pseudo Keyboards . . . 3–202

3.24 .STAT — Return Job Statistics 3–204
3.25 .TIME — Return Timing Information 3–206
3.26 .TTAPE — Enter Tape Mode 3–208
3.27 .TTDDT — Disable Full-Line Buffering 3–209
3.28 .TTECH — Undo .TTAPE or .TTNCH 3–211
3.29 .TTNCH — Stop Echo 3–212
3.30 .TTRST — Restart Output. 3–213
3.31 .ULOG — Assign/Reassign/Deassign Device or Assign/Deassign User Logical 3–214
3.31.1 UU.ASS (Assign/Reassign a Device or Assign User Logical) 3–215
3.31.2 UU.DEA (Deassign a Device or User Logical) 3–222
3.31.3 UU.DAL (Deassign All Devices and User Logicals). 3–226
3.32 .UUO — Execute BASIC–PLUS SYS Call 3–229
3.32.1 UU.ACT (Accounting Information Dump) 3–235
3.32.2 UU.ASS (Assign/Reassign Device). 3–236
3.32.3 UU.ATR (Read/Write File Attributes) 3–238
3.32.4 UU.ATT (Attach/Reattach Job/Swap Console). 3–240
3.32.5 UU.BCK (Change File Statistics) 3–246
3.32.6 UU.BYE (Logout) 3–247
3.32.7 UU.CCL (Add/Delete CCL Command) 3–250
3.32.8 UU.CHE (Enable/Disable Disk Caching) 3–252
3.32.9 UU.CHU (Change Password/Quota, Disable Terminal, Kill Job) 3–254
3.32.10 UU.CNV (Date and Time Conversion) 3–258
3.32.11 UU.DAL (Deassign All Devices) 3–260
3.32.12 UU.DAT (Change System Date/Time) 3–261
3.32.13 UU.DEA (Deassign Device) 3–262
3.32.14 UU.DET (Detach) 3–263
3.32.15 UU.DIE (System Shutdown) 3–264
3.32.16 UU.DIR (Directory Lookup) 3–265
3.32.17 UU.DLU (Delete User Account) 3–270
3.32.18 UU.DMP (Snap Shot Dump) 3–272
3.32.19 UU.ERR (Return Error Messages) 3–273
3.32.20 UU.FCB (Get Open Channel Statistics (WCB/DDB/FCB)) . .3–275
3.32.21 UU.FIL (File Placement and Modification) 3–279
3.32.22 UU.HNG (Hang Up a Dataset) 3–282
3.32.23 UU.JOB (Create Job) 3–283
3.32.24 UU.LIN (Login) 3–290
3.32.25 UU.LOG (Set Number of Logins) 3–292
3.32.26 UU.LOK (Disk Directory Lookup by File Name/Wildcard Lookup) 3–294
3.32.27 UU.MNT (Disk Pack Status) 3–298
3.32.28 UU.NAM (Associate a Run-Time System with a File) 3–301
3.32.29 UU.NLG (Disable Further Logins) 3–302
3.32.30 UU.PAS (Create User Account) 3–304
3.32.31 UU.POK (Poke Memory) 3–307
3.32.32 UU.PPN (Wildcard PPN Lookup) 3–308
3.32.33 UU.PRI (Change Priority/Run Burst/Job Size) 3–310
3.32.34 UU.RAD (Read or Read-and-Reset Accounting Data) 3–312
3.32.35 UU.RTS (Add/Remove/Load/Unload Run-Time System or Resident Library) 3–314
3.32.36 UU.SLN (System Logical Names) 3–326
3.32.37 UU.SPL (Spooling) 3–330

3.32.38 UU.STL (Stall/Unstall System) 3–333
3.32.39 UU.SWP (Add, Remove, and List System Files) 3–334
3.32.40 UU.SYS (Return Job Status Information) 3–340
3.32.41 UU.TB1 (Get Monitor Tables, Part I) 3–343
3.32.42 UU.TB2 (Get Monitor Tables, Part II) 3–345
3.32.43 UU.TB3 (Get Monitor Tables, Part III) 3–347
3.32.44 UU.TRM (Set Terminal Characteristics) 3–349
3.32.45 UU.YLG (Enable Logins) 3–353
3.32.46 UU.ZER (Zero Device) 3–355
3.33 .WRITE — Write Data to File or Device 3–357

Part III RSX and RT11 Emulator Directives

Chapter 4 RSX Run-Time System Environment

4.1 Introduction 4–1
4.1.1 Advantage: Transportable Code 4–1
4.1.2 General Services 4–2
4.1.3 RSX Directive Emulation Within RSTS/E Monitor 4–2
4.2 System Macro Library 4–3
4.3 Directive Processing 4–3
4.4 Directive Forms — $, $C, $S — and Their Expansions 4–4
4.4.1 $ Form (and DIR$ Directive) 4–5
4.4.2 $C Form 4–8
4.4.3 $S Form 4–9
4.5 First 1000 Bytes of Low Segment for RSX 4–9

Chapter 5 RSX Emulator Directives

5.1 Introduction 5–1
5.1.1 Perform Non-File-Structured Input/Output 5–1
5.1.2 Specify Trap Routines 5–2
5.1.3 Control Program Execution 5–2
5.1.4 Return System Information 5–3
5.1.5 Access Resident Libraries 5–3
5.2 ABRT$ — Abort 5–4
5.3 ALUN$ — Assign Logical Unit Number 5–5
5.4 ASTX$ — AST Service Exit 5–7
5.5 ATRG$ — Attach Resident Library 5–9
5.6 CRAW$ — Create Address Window 5–12
5.7 DTRG$ — Detach Resident Library 5–17
5.8 ELAW$ — Eliminate Address Window 5–19
5.9 EXIT$ — Task Exit 5–21
5.10 EXST$ — Exit with Status 5–22
5.11 EXTK$ — Extend Task 5–24
5.12 GLUN$ — Get LUN Information 5–26
5.13 GMCR$ — Get MCR (CCL) Command Line 5–28
5.14 GPRT$ — Get Partition (Job) Parameters 5–29
5.15 GTIM$ — Get Time Parameters 5–31
5.16 GTSK$ — Get Task (Job) Parameters 5–33

5.17 MAP$ — Map Address Window 5–36
5.18 QIO$ and QIOW$ — Queue I/O Request (and Wait) 5–39
5.19 SCCA$S — Specify Control/C AST 5–46
5.20 SFPA$ — Specify Floating-Point-Processor Exception Address 5–48
5.21 SPND$S — Suspend 5–50
5.22 SVDB$ — Specify SST Vector Table for Debugging Aid 5–51
5.23 SVTK$ — Specify SST Vector Table for Task 5–53
5.24 UMAP$ — Unmap an Address Window 5–55
5.25 WSIG$ — Wait for Significant Event Flag 5–57
5.26 WTSE$ — Wait for Single Event Flag 5–58

Chapter 6 RT11 Run-Time System Environment

6.1 Introduction 6–1
6.1.1 Advantage: Transportable Code 6–1
6.1.2 General Services 6–3
6.2 System Macro Library 6–3
6.3 Directive Processing 6–4
6.4 Call Forms 6–5
6.4.1 Format for Calls Using Argument Blocks 6–5
6.4.2 Format for Calls Not Using Argument Blocks 6–7
6.5 Channel Number and Device Block Arguments 6–7
6.5.1 Channel Number Arguments 6–7
6.5.2 Device Block Arguments 6–8
6.6 Low 1000 Bytes for RT11 Run-Time System 6–9
6.7 'Scratch Pad' Area in User Job Image 6–10

Chapter 7 RT11 Emulator Directives

7.1 Introduction 7–1
7.2 .CHAIN — Pass Control to Another Program Under RT11 7–7
7.3 .CLOSE — Close a Channel 7–9
7.4 .CLRFQB — Clear the FIRQB 7–10
7.5 .CLRXRB — Clear the XRB 7–11
7.6 .CSIGEN — Examine String for RT Command, Open Files 7–12
7.7 .CSISPC — Examine String for RT Command, Create Devblk 7–16
7.8 .DATE — Return Current Date to R0 7–19
7.9 .DATTIM — Return Date or Time 7–21
7.10 .DELETE — Delete File from Disk or DECtape 7–23
7.11 .DOCCL — Do a RSTS/E .CCL 7–24
7.12 .DOFSS — Do a RSTS/E .FSS 7–25
7.13 .DORUN — Chain to Non–RT11 RTS Program 7–26
7.14 .DSTATUS — Return Device Status 7–28
7.15 .ENTER — Open File for Output 7–30
7.16 .ERRPRT — Print RSTS/E Error Message 7–32
7.17 .EXIT — Program Exit 7–33
7.18 .FETCH — Check for Device Available 7–34
7.19 .GETCOR — Change Job Image Size 7–35
7.20 .GTIM — Return Time-of-Day 7–36
7.21 .GTLIN — Get Line from Job's Terminal 7–37
7.22 .GTJB — Return Job High Limit 7–38

7.23 .GVAL — Get Value from Scratch Pad 7–39
7.24 .HRESET — Hardware Reset 7–40
7.25 .LOOKUP — Open File for Input 7–41
7.26 .PRINT — Display String on Job's Terminal 7–43
7.27 .PURGE — Release Channel 7–44
7.28 .RCTRLO — Reverse CTRL/O 7–45
7.29 .READ/.READW/.READC — Read Data 7–46
7.30 .RENAME — Rename a File 7–48
7.31 .REOPEN — Reopen File Closed with .SAVESTATUS 7–49
7.32 .SAVESTATUS — Save Status of File for Later .REOPEN 7–50
7.33 .SCCA — Pass CTRL/Z to User Program 7–51
7.34 .SETCC — Process CTRL/C 7–52
7.35 .SETFQB — Set Up FIRQB 7–53
7.36 .SETTOP — Expand to Start of Scratch Pad 7–54
7.37 .SFPA — Set Floating-Point Error Address 7–55
7.38 .SPFUN — Special Functions for I/O 7–56
7.39 .SRESET — Software Reset 7–58
7.40 .TRPSET — Intercept Traps to 4 and 10 7–59
7.41 .TTYIN/.TTINR — One-Character Read from Terminal 7–60
7.42 .TTYOUT/.TTOUTR — Transfer One Character to Job's Terminal . . 7–62
7.43 .TWAIT — Timed Wait 7–63
7.44 .WAIT — Check for Channel Open 7–64
7.45 .WRITE/.WRITW/.WRITC — Write Data 7–65
7.46 ..V1../..V2.. — Use Version 1/Version 2 Expansion 7–67

Appendix A Full List of Errors

Appendix B Device Information

B.1 Disks B–2
B.1.1 MODE Values B–2
B.1.2 Disk Device Sizes B–3
B.2 Flexible Diskettes B–4
B.3 Magnetic Tape B–5
B.3.1 File-Structured Processing B–5
B.3.2 Non-File-Structured Processing B–6
B.4 Line Printers B–7
B.5 Terminals B–8
B.5.1 Terminal MODE and RECORD Values B–8
B.5.2 Echo Control Mode B–10
B.5.3 Escape Sequences B–12
B.6 Pseudo Keyboards B–14

Appendix C Supplementary RSX Directives for Resident Libraries

C.1 RDB Directives C–1
C.2 WDB Directives C–2

Index

Figures

2–1 How a Physical Address Is Formed 2–2
2–2 Memory Mapping with the APRs 2–3
2–3 Job Area in Virtual Memory 2–5
2–4 First 1000 Bytes of Low Segment 2–8
2–5 General FIRQB Format 2–11
2–6 General XRB Format 2–12
2–7 Format of Pseudo Vector Region of High Segment 2–16
4–1 General Form of the Directive Parameter Block (DPB) 4–3
4–2 Example of RSX Directive Forms 4–5
4–3 First 1000 Bytes of Low Segment for RSX 4–10

Tables

3–1 Summary of General Monitor Calls 3–2
3–2 Fixed Monitor Locations 3–130
3–3 Data Input with .READ 3–158
3–4 Private Delimiter Masks 3–190
3–5 Special Functions for Magnetic Tape 3–197
3–6 Value Returned by .SPEC for Magnetic Tape 3–198
3–7 .UUO Subfunctions — Calls to the File Processor (FIP) 3–231
3–8 Data Output with .WRITE 3–358
5–1 Vertical Format Control Characters 5–42
6–1 EMT Instructions Recognized by the RT11 Run-Time System 6–5
7–1 RT–11 Calls Not Functional on RSTS/E 7–2
7–2 RT11 Run-Time System Directives 7–2
B–1 MODE Values for File-Structured Disk Access (FIRQB + FQMODE) . . B–2
B–2 MODE Values for Non-File-Structured Disk Access (FIRQB + FQMODE) B–2
B–3 Disk Device Sizes B–3
B–4 Flexible Diskette MODE Values (FIRQB + FQMODE) B–4
B–5 Flexible Diskette RECORD Values (XRB + XRBLK) B–4
B–6 MODE Values for File-Structured Magnetic Tape (FIRQB + FQMODE) B–5
B–7 CLUSTERSIZE Values for ANSI Magnetic Tape Files B–6
B–8 Line Printer MODE Values (FIRQB + FQMODE) B–7
B–9 Line Printer RECORD Values (FIRQB + FQMODE) B–7
B–10 Terminal MODE Values (FIRQB + FQMODE) B–8
B–11 RECORD Values for Terminal Input (XRB + XRMOD) B–9
B–12 RECORD Values for Terminal Output (XRB + XRMOD) B–9
B–13 Echo Control Mode Character Set B–10
B–14 VT100 ANSI–Compatible Escape Sequences for Screen Control . . . B–12
B–15 Pseudo Keyboard MODE Values (FIRQB + FQMODE) B–14
B–16 RECORD Option Bit Values for Pseudo Keyboard Output (XRB + XRMOD) B–14
B–17 Possible Errors on Pseudo Keyboard Output Request B–14

Preface

This manual describes directives to the RSTS/E monitor, the RSX emulator, and the RT11 emulator that can be used in MACRO programs. To use these directives, you should be familiar with the MACRO–11 assembly language. MACRO is the standard assembler for Digital Equipment Corporation PDP–11 computers and is available under various operating systems for the PDP–11. The syntax is basically the same for all operating systems.

Associated Documents

For information about the syntax of MACRO assembly language, see the *PDP–11 MACRO–11 Language Reference Manual.*

For more information about the RSTS/E system, refer to:

RSTS/E System User's Guide
BASIC–PLUS Language Manual
RSTS/E Programming Manual
RSTS/E Programmer's Utilities Manual
RSTS/E Task Builder Reference Manual

This manual cross-references these manuals where appropriate.

The system manager sets certain parameters that affect the monitor and, consequently, the monitor directives. Therefore, the *RSTS/E System Generation Manual* and the *RSTS/E System Manager's Guide* are referenced here.

Document Structure

This manual contains seven chapters and three appendixes:

Chapter 1	Gives an overview of run-time systems and jobs as they relate to the system directives.
Chapter 2	Describes the RSTS/E environment—memory allocation and job space—for the general monitor directives.
Chapter 3	Serves as a reference guide for the general monitor directives that can be used in programs compiled under either the RSX or RT11 run-time systems.
Chapter 4	Describes the RSX environment for the RSX directives.
Chapter 5	Serves as a reference guide for the directives processed by the RSX emulator in the RSX run-time system.
Chapter 6	Describes the RT11 environment for the RT11 directives.
Chapter 7	Serves as a reference guide for the directives processed by the RT11 emulator in the RT11 run-time system.

Appendix A Lists the RSTS/E errors you can get during directive processing.

Appendix B Summarizes MODE and RECORD values and other useful information for disks, flexible diskettes, magnetic tape, line printers, terminals, and pseudo keyboards.

Appendix C Describes supplementary directives to the RSX emulator, useful if you are working with resident libraries.

Document Conventions

The following conventions are used in this manual:

→ The arrow means "points to," as when the stack pointer register points to, or contains the address of, the first item in the stack. For example:

```
SP→  item at top of stack
     item one word down from top of stack
```

( ) Parentheses mean "the contents of" the item that the parentheses surround. For example, the contents of the program counter would be represented as:

(PC)

[ ] Brackets around an item in a line showing the general form of a directive indicate that the item is optional. For example:

QIO$ param1 [, param2]

{ } Braces around two or more items in a line showing the general form of a directive indicate that you must choose one of the enclosed items. For example:

```
{QIO$  }
{QIOW$ }
```

⇒ The double arrow means "implies." For example:

Bit 0 = 1 ⇒ Error condition

< > Angle brackets around two or more items in a line tell the MACRO assembler that the items make up a list. For example:

GLOBAL <name1[,name2,...]>

You must type the angle brackets.

Summary of Technical Changes

General Monitor Directives

Several RSTS/E monitor directives have expanded functions for V8.0, and one new directive (UU.STL) has been added. The following is a summary:

CALFIP (Call File Processor)

The CALFIP OPNFQ function returns the error ?Protection violation if a nonprivileged user tries to open a disk for non-file-structured access. See Section 3.2.12.

.MESAG (Message Send/Receive)

The Declare Receiver function of .MESAG is available to nonprivileged users, with certain restrictions. In addition, three new parameters have been added to the data passed: outbound link maximum, packet maximum, and packets per message. The last two parameters are used only in an EMT logging program. See Section 3.12.1.

UU.ASS (Assign/Reassign Device)

A nonprivileged user can reassign a device to a job running under the current account. See Sections 3.31.1 and 3.32.2.

UU.ATT (Attach/Reattach/Swap Console)

UU.ATT allows a nonprivileged user to attach to a job running under the current account. No password is required. In addition, UU.ATT includes a "swap console" function, which allows two jobs running under the same account to exchange ownership of a terminal. One job must be attached and the other detached. See Section 3.32.4.

UU.BYE (Logout)

UU.BYE allows a privileged user to log out without closing files, deassigning devices, checking disk quotas, or performing other "clean-up" functions. In addition, UU.BYE returns information about log-out status as well as disk and detached job quotas.

A nonprivileged user can now use UU.BYE to kill the current job. See Section 3.32.6.

UU.CLN (Clean Disk Pack)

The UU.CLN directive is obsolete, and information on this directive has been removed from the manual. Use the ONLCLN program to rebuild a disk.

UU.DET (Detach)

The description of UU.DET includes changes made to the "close flag" in V7.2. See Section 3.32.14.

UU.FCB (Get Open Channel Statistics)

UU.FCB returns information about the file system's FCB (file control block) as well as the WCB (window control block) and the DDB (device data block). See Section 3.32.20.

UU.JOB (Create Job)

UU.JOB creates logged-in jobs as well as logged-out jobs. You can create both attached and detached logged-in jobs, and you can specify that the job either run a program or enter a keyboard monitor at the P.NEW entry point. In addition, some functions of UU.JOB are available to nonprivileged users. See Section 3.32.23.

UU.MNT (Disk Pack Status)

UU.MNT has two new functions:

- Mount disk read/write even if initialized read-only
- Mount disk for use by single user (/NOSHARE)

See Section 3.32.27.

UU.PAS (Create Account)

UU.PAS lets you position and preextend the User File Directory (UFD) when you create a user account. See Section 3.32.30.

UU.RAD (Read or Read and Reset Accounting Data)

UU.RAD lets you specify a wildcard project-programmer number. See Section 3.32.34.

UU.SPL (Spooling)

UU.SPL spools files to both the micro–RSTS and the standard RSTS/E spooling packages. See Section 3.32.37.

UU.STL (Stall/Unstall System)

UU.STL is a new directive that allows a privileged user to suspend all currently active jobs on the system except for the calling job. See Section 3.32.38.

UU.TB3 (Get Monitor Tables – Part III)

UU.TB3 returns addresses of four additional monitor tables: SATEND, UNTLVL, MFDPTR, and MAGLBL. The directive also returns the number of jobs currently on the system. See Section 3.32.43.

UU.ZER (Zero Device)

For disks, UU.ZER allows a nonprivileged user to delete all files in the current account regardless of their protection codes. Privileged users can delete files in any account regardless of their protection codes as well as deallocate UFD clusters. See Section 3.32.46.

.WRITE (Write Data to File or Device)

The .WRITE directive returns a new value at XRB + XRBC: the number of bytes still to be written. This value is generally 0 except for line printers and terminals.

In addition, a "no stall" modifier is available for line printer and terminal output. This modifier causes the monitor to return control to your program if an output stall is to occur on the device. See Section 3.33.

Other Documentation Changes

A program running under the RSX run-time system can expand to (32K–32) words if the monitor contains RSX directive emulation code. The manual is updated as necessary to include this information. No other changes have been made to RSX or RT11 emulator directives for V8.0, other than correcting documentation errors.

Appendix B is expanded to include MODE and RECORD values for disks, flexible diskettes, magnetic tape, line printers, terminals, and pseudo keyboards. Values are given in both octal and decimal, and FIRQB and XRB offsets are listed. This appendix also contains other device-related information, such as VT100 ANSI–compatible escape sequences and disk sizes.

PART I
Introduction

Chapter 1
Introduction

As an assembly language programmer, you should know that two MACRO assemblers are available on RSTS/E: one for jobs running under the RSX run-time system and one for jobs running under the RT11 run-time system. You will use one of these two run-time systems to assemble and, in most cases, run your programs. In addition to user programs, you can also write or modify run-time systems, which run under direct control of the RSTS/E monitor.

This manual describes the three types of system directives available to RSTS/E assembly language programmers: general monitor directives, RSX emulator directives, and RT11 emulator directives. Before you start using these directives, however, it is helpful to understand some basic concepts about RSTS/E run-time systems and jobs.

1.1 Run-Time Systems

There are several ways to look at run-time systems. From the system design viewpoint, a run-time system is a way to implement code that, when it is resident in memory, can be shared by many users. In a time-sharing system such as RSTS/E, this is an important consideration. Run-time systems are normally implemented as "pure code"; that is, as a series of instructions and fixed data only, containing no variable data. Such reentrant code saves space, since many jobs can use it, and time, since run-time systems need not be copied to and from disk in the way that user programs are swapped in and out of memory. At least one run-time system is permanently resident. It is called the primary run-time system, because it is the first (and only) run-time system at system start-up. Other run-time systems are loaded when requested, remain in memory as long as necessary, and may be removed when they are no longer in use. Because they contain no variable data, they need not be swapped out to disk; they are simply reloaded when they are needed again. To a user, run-time systems provide various services. Basically, they provide an environment for people or an environment for programs, and sometimes both.

1.1.1 Environments for People

The DCL, BASIC–PLUS, RSX, and RT11 run-time systems all provide what is called a "keyboard monitor." That is, they accept, analyze, and act on commands you type at a terminal keyboard. The *RSTS/E System User's Guide* gives an overview of the command environments these run-time systems provide.

1.1.2 Environments for Programs

As a MACRO programmer, you are concerned with the environment for programs provided by the RSX and RT11 run-time systems. Both run-time systems include:

- A loader. This part of each run-time system loads a program from disk into memory and starts its execution.
- An emulator. The RSX and RT11 run-time systems include code that emulates directives handled by DIGITAL's RSX–11M and RT–11 operating systems for the PDP–11 computer.

A run-time system usually takes up space in the 32K–word area called the user job area. Therefore, a run-time system limits the size of your program to less than 32K words. Both the RSX and RT11 run-time systems take 4K words of virtual memory. However, an installation option allows the RSX run-time system to "disappear" in certain situations. In this case, an executing program can use the space normally taken by the run-time system. Space requirements are explained in greater detail in Chapter 2.

Should you program under the RSX or RT11 run-time system? RSX is usually a better choice, but your decision depends on:

- Whether you are coding MACRO subroutines for use in a high-level language program
- Which set of program development tools better satisfies your needs
- Whether you want to use resident libraries

High-Level Languages

When you write MACRO subroutines for use in high-level language programs, the high-level language dictates which run-time system you must use. BASIC–PLUS–2, COBOL–81, PDP–11 COBOL, DIBOL, and FORTRAN–77 all run under the RSX run-time system, while FORTRAN–IV runs under the RT11 run-time system. You must compile, link, and run all the modules in your program under the same run-time system, whether your program is written in MACRO or a high-level language.

Program Development Tools

RSTS/E provides a set of program development tools for the RSX environment and a set for the RT11 environment. While the tools for each environment perform similar functions, they differ in their speed and capabilities:

- Assemblers. RSTS/E supports two MACRO–11 assemblers, the RSX–based MAC assembler and the RT11–based MACRO assembler. The two assemblers are nearly identical in function and produce similar output.
- Linkers. RSTS/E supports two linkers: the Task Builder (TKB) for RSX–based programs and LINK for RT11–based programs. While LINK is faster than the Task Builder, the Task Builder is more powerful. It can link much larger and more complex overlay structures than LINK, including co-trees. And, unlike LINK, the Task Builder has options for linking to resident libraries.
- Librarians. RSTS/E provides LBR for RSX–based programs and LIBR for RT11–based programs. You can create object and macro libraries with either utility. LBR also lets you create universal libraries, which can contain any type of file, including text files.
- Object module patch utilities. RSTS/E provides a PAT utility for each environment. Both let you update code in a relocatable binary object module.

For details on these program development tools, see:

- *RSTS/E Task Builder Reference Manual* — Describes the Task Builder.
- *RSTS/E Programmer's Utilities Manual* — Describes the RSX–based MACRO assembler, librarian, and object module patch utilities.
- *RSTS/E RT11 Utilities Manual* — Describes the RT11–based MACRO assembler, librarian, linker, and object module patch utilities.

Resident Libraries

When you program under RSX, you can easily use DIGITAL–supplied resident libraries (such as RMS–11 and FMS–11) as well as create your own resident libraries. In addition, the Task Builder's cluster library feature lets up to five resident libraries share the same virtual address space in your program.

You can also use resident libraries under the RT11 emulator, but the coding is much more difficult. Unlike RSX, you must use .PLAS directives to map and create address windows inside your task. Coding these directives can be quite complex. The Task Builder, on the other hand, has options that build tables describing your task and the window to map, and automatically includes the code to perform the necessary .PLAS directives for you. Thus, RSX is a more practical choice than RT11 if you plan to use resident libraries.

Directives for Each Programming Environment

RSTS/E has three types of directives: monitor directives, RSX emulator directives, and RT11 emulator directives. Monitor directives, described in Chapter 3, are processed directly by the RSTS/E monitor. You can assemble monitor directives using either the RSX–based or the RT11–based MACRO assembler, and you can use these directives in both user programs and run-time systems. (When you write a program to run under the RT11 run-time system, you must precede all monitor directives with a special "prefix EMT"; see Chapter 6 for details.)

RSX emulator directives are processed by the RSX emulator, which is part of the RSX run-time system. These directives, which have basically the same form and function as a subset of the RSX–11M operating system monitor directives, perform non-file-structured I/O and trap handling. You must use the RSX–based MAC assembler to assemble the RSX emulator directives, and you can use them only in a user program that will run under the RSX run-time system. Chapters 4 and 5 of this manual describe the RSX run-time system environment and emulator directives in detail.

RT11 emulator directives are processed by the RT11 emulator, which is part of the RT11 run-time system. These directives provide most of the "single-job" programmed requests available to MACRO programmers using the RT–11 operating system. In addition, the RT11 run-time system also provides directives for the RSTS/E environment not available under the RT–11 operating system. You must use the RT11–based MACRO assembler to assemble RT11 emulator directives, and you can use them only in a user program that will run under the RT11 run-time system. Chapters 6 and 7 of this manual describe the RT11 run-time system environment and emulator directives in detail.

Writing or Modifying a Run-Time System

If you want to modify an existing run-time system or code your own run-time system, you can use either MACRO assembler. You may find the RT11–based programming tools easier to use for this purpose than the RSX–based programming tools, mainly because it is easier to link run-time systems with the LINK program than with the Task Builder. Run-time systems always have a specific address for their top (highest) address. When you use LINK, you can specify the top address the first time you link the run-time system. But when you use the Task Builder, you have to link your run-time system twice — once to find its size, and again to adjust its top address to the value you want.

Unlike a program, a run-time system can contain monitor directives only, not RSX or RT11 emulator directives. In addition, you must store the run-time system file (the product of assembling and linking) on the system disk in "save image library" format. To create a save image library file, use MAKSIL for run-time systems assembled with MAC and linked with the Task Builder; use SILUS for run-time systems assembled with MACRO and linked with LINK.

1.2 Jobs

Like run-time systems, "jobs" can be viewed from several angles. To the RSTS/E monitor, a job is a unit of work generally associated with activity at a terminal on the system. Suppose, for example, that a user types a line at a logged-out terminal. The monitor creates a job, assigning a job number and allocating internal tables for bookkeeping. The monitor then passes control to a "new-user" entry point in the primary run-time system. The primary run-time system has code at this entry point that causes the LOGIN program to be loaded from the system disk and executed. LOGIN analyzes what was typed and performs the normal login dialogue. When LOGIN exits for a valid login, control passes to what the system manager has defined as the default keyboard monitor, which waits for further input from the terminal. The monitor regards the execution of the primary run-time system, LOGIN, the default keyboard monitor, and whatever else occurs at the terminal until it is logged out as the same "job." (If the login sequence was not valid, LOGIN exits with the job still logged out. The monitor destroys "the job" and releases the job number.)

As a MACRO programmer, your awareness of the concept of "job" will probably center around the amount of memory RSTS/E provides for work space for a job, and the fact that the run-time system can take part of this work space. The allocation of work space is described in Chapter 2.

Chapter 2
General RSTS/E Environment

To understand how and why one copy of a run-time system, shared by many users, can still take up space in each user's work area, we must go into some background on memory accessing in the PDP–11 (Section 2.1) and how RSTS/E uses it to define a job space, or work area in memory, for each user to run programs (Section 2.2). Section 2.3 briefly describes resident libraries and the special-case "disappearing" RSX run-time system. Sections 2.4 and 2.5 give specifics on certain areas in the job space that are used by the monitor, the run-time system, and the user program.

2.1 How RSTS/E Allocates Memory — Physical and Virtual Addressing

All RSTS/E systems use the memory management feature available on PDP–11/23–PLUS, 24, 34, 35, 40, 44, 45, 50, 55, 60, and 70 computers. This feature extends the addressable memory range of the PDP–11 processor by using hardware registers called Active Page Registers (APRs).

The PDP–11 processor handles 16–bit operand addresses, allowing reference to 32K words. (Remember that the PDP–11 is byte-addressable, so the address range is from 0 through 2^{16}–1, which equals 64K bytes or 32K words.) With the memory management unit, a 16–bit address is treated as a relocatable (virtual) address that is combined with information in an APR to form an 18–bit (22–bit, for the PDP–11/23–PLUS, 24, 44, and 70) physical address.

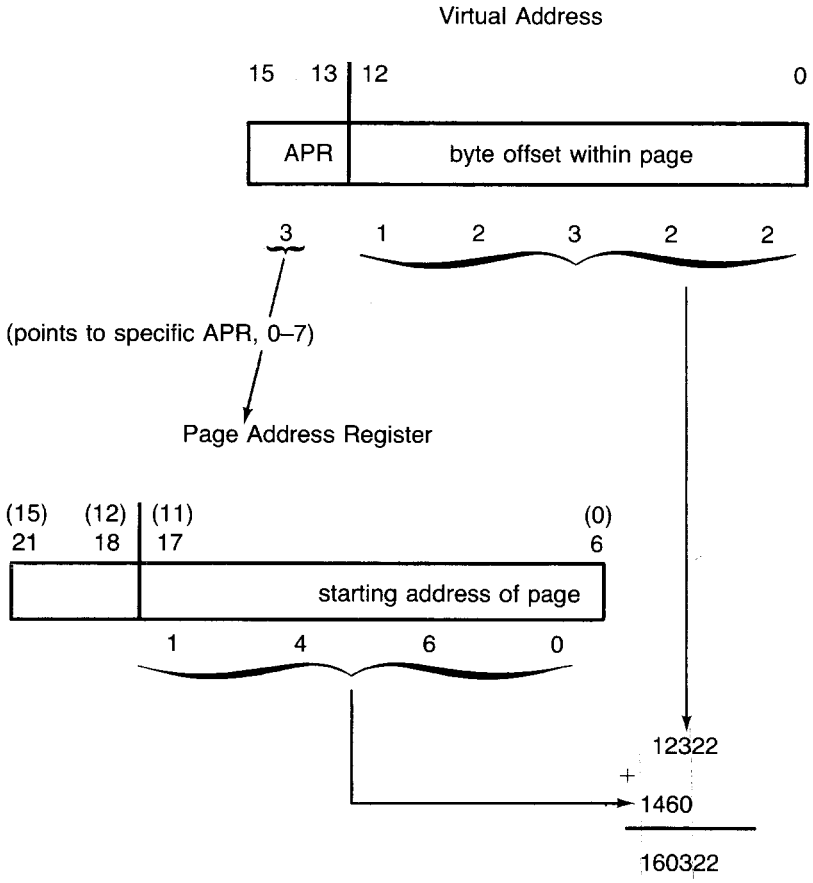
The Processor Handbook for the PDP–11 processors explains in detail how the APRs function. Briefly, an APR consists of two 16–bit registers. These registers define a "page" of contiguous memory. The Page Address Register (PAR) defines the physical memory location where the page begins. The Page Descriptor Register (PDR) defines, among other things, the maximum length of the page and how it can be accessed (for example, read/write, read-only).

Figure 2–1 shows how a virtual address and a Page Address Register are combined to form a physical address in physical memory. The 16–bit

virtual address defines which one of eight Active Page Registers is to be used and a byte offset within the page. The Page Address Register of the indicated APR is handled as though it contains bits 6–17 (6–21 for the PDP–11/23–PLUS, 24, 44, and 70) of an 18–bit (or 22–bit) physical address, defining the start of the page.

In Figure 2–1, the virtual address of 072322_8 identifies APR 3 and byte 12322 of the page defined by APR 3. The PAR of APR 3 indicates a starting address of 146000 for the page. The physical address obtained is 146000 + 012322, or 160322.

Figure 2–1: How a Physical Address Is Formed



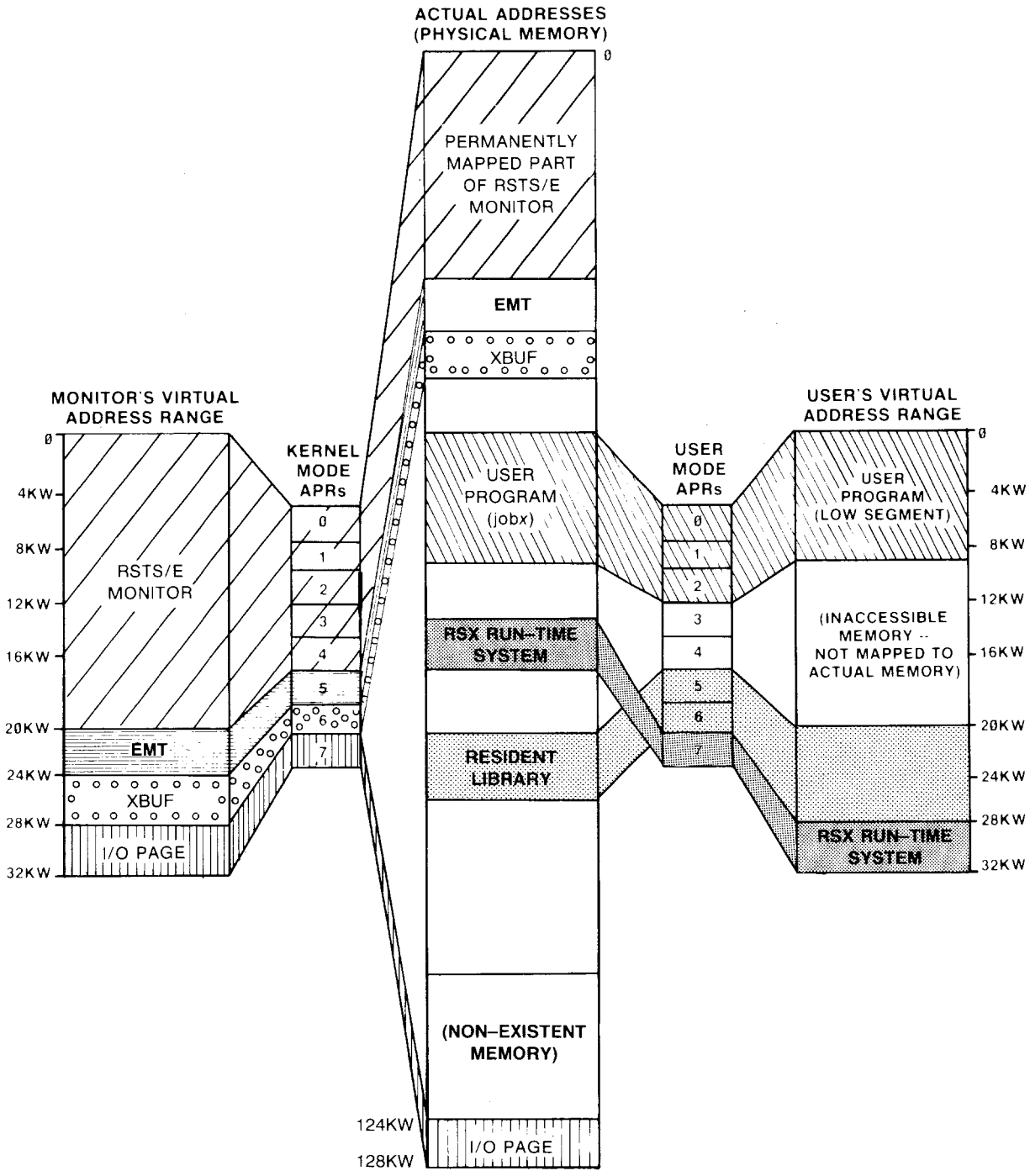
The byte offset field in the virtual address is 13 bits long. The maximum size of a page, then, is 2^{13} bytes, or 4096 words. In other words, one APR can "map" a virtual address range of up to 4K words into an equal extent of physical memory.

The memory management unit on the PDP–11 consists of two sets of APRs,* eight in each set. Since each APR can map a 4K segment of virtual memory to physical memory, each set of APRs can provide access to 32K words of physical memory.

* The PDP–11/44, 45, 50, 55, and 70 have three sets of APRs; the additional set is for "supervisor mode" mapping, which RSTS/E does not use.

The monitor uses one set, called the "kernel mode" APRs, to map itself in physical memory. It uses the other set, called the "user mode" APRs, to map the job that is active during the current time slice of time-shared processing. Figure 2–2 illustrates the concept of mapping through the APRs.

Figure 2–2: Memory Mapping with the APRs



On the PDP–11/44, 45, 50, 55, and 70, the RSTS/E monitor can take advantage of what is called "I and D space." On these processors, there are actually two sets of eight APRs for each mode. One set can be used to map instructions, and the other set maps data. The monitor may use this type of mapping, depending on the number of "small buffers" the system manager selects with INIT (see the *RSTS/E System Generation Manual*). For example, the monitor may, if the number of requested small buffers is large enough, use data-space APR 1 to map small buffers and instruction-space APR 1 to map common routines.

2.2 Job Space — High Segment and Low Segment

The RSTS/E monitor is designed to handle work requested by a user through an interface: the run-time system. For example, the BASIC–PLUS, DCL, RSX, and RT11 run-time systems (available as part of a RSTS/E system) each provide their own keyboard monitor to accept and process user commands. These run-time systems also contain code to handle their own sets of directives, accepting and expanding user program calls to the monitor. For example, the RSX run-time system provides I/O calls to the monitor (phrased in terms of logical units and records), which the run-time system itself translates and executes as the more device-oriented calls handled directly by the monitor.

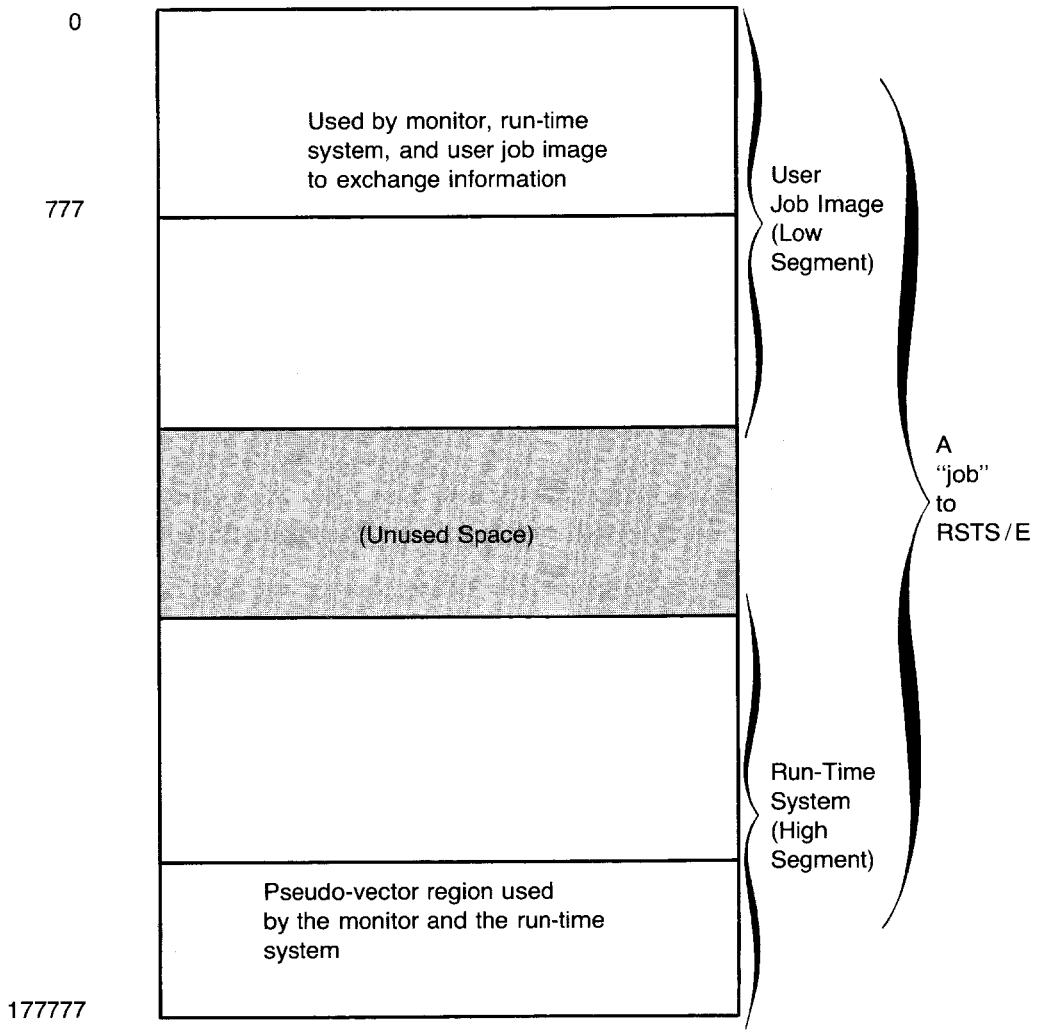
Thus, the run-time system communicates with both the user program and the monitor. Execution control passes back and forth between these three entities; data is passed between them using established ranges of virtual addresses. The monitor, then, needs to be able to access both the run-time system and the user job image during any given time slice. It does this by setting up the run-time system as part of the 32K words accessible through the eight user APRs.

The monitor assigns an area for the run-time system in the high portion of virtual address space, called the "high segment." The low portion of virtual address space, or "low segment," belongs to the "user job image"; that is, to the utility program, compiler, assembler, or executable user program that is currently being executed for the job. (As part of its housekeeping for each job, the monitor keeps track of where the currently appropriate run-time system is, where the user job image is, and what the values were in the program counter register (PC), program status word (PSW), and other job-context information at the end of the last time slice. Before the next time slice for the job, the monitor simply loads the APRs with the correct values for the job and loads the PC, PSW, and so forth, so that execution continues where it left off.)

In any case, the high segment, or run-time system, takes up some multiple of 4K words of virtual address space, due to the APR mapping as discussed in Section 2.1. The BASIC–PLUS run-time system, for example, may take from 13 to 16K words of physical memory, depending on options selected when the system is installed. Even though the physical memory required may be only 13K words, it still requires four APRs to map this range, leaving four APRs, or a maximum of 16K words, for a user program running under the BASIC–PLUS run-time system.

The monitor uses certain areas within the high segment and the low segment to get information from the job defining what work the monitor is to do for it and to pass information back to the job. Figure 2–3 illustrates the job area in virtual addresses. The first 1000_8 bytes are used to pass information between the monitor, the run-time system, and the user job image for certain types of monitor directives. The "pseudo-vector" region in high virtual memory is used by the monitor to determine, for example, where control is to be passed when a job is initially entered. The run-time system sets this area with entry points and values to define itself to the monitor.

Figure 2–3: Job Area in Virtual Memory



The following subsections give more detail on these areas. Read Section 2.4 if you are interested in using the general monitor directives described in Chapter 3. The RSX and RT11 run-time system directives set up the first 1000 bytes of memory for you if you are using only the directives described in Chapters 5 or 7. Similarly, Section 2.5 will be of interest mainly if you wish to code your own run-time system or modify one of the existing ones and need to know about the pseudo-vector region.

2.3 Important Installation Options

At system generation, the system manager can install options that affect the operation of the run-time system. The next two sections describe these options and their effect on the system.

2.3.1 The "Disappearing" RSX Run-Time System

When a RSTS/E system is generated, the system manager has the option of installing the "emulation" code of the RSX run-time system as a part of the monitor. When this is done, the emulation code—the part that handles traps and processes the RSX directives explained in Chapter 5—is permanently resident in memory. It is not permanently mapped; the monitor maps this section of code using kernel mode APR 5 when requested to do so by the "user command processing" code in the RSX run-time system.

The whole RSX run-time system still exists as a file that is loaded from disk when necessary, remains as long as anyone is using it, and is removed when it is no longer needed. When someone at a terminal types a command to run a program that was, for example, assembled with MAC and linked with TKB, the monitor passes control to the RSX run-time system to load the program. Once the program is loaded and ready for execution, however, the whole RSX run-time system is no longer needed. The emulation portion of the code, which must be present to process the execution of RSX directives and traps that might occur during execution of the program, is in the RSTS/E monitor. The RSX run-time system passes control to the monitor, with a request to "disappear" from the high segment of the user job space. When it gives control to the monitor, the RSX run-time system passes on any requests from the user or from the program itself to make use of the high-segment space that is freed when the run-time system disappears.

The monitor decreases the count of current users of the RSX run-time system and frees the physical memory taken by the full RSX run-time system if no one else is using it and it was not installed as permanently resident. The monitor then maps the RSX emulation code (using kernel mode APR 5), sets up the user APRs and other registers according to the information passed to it by the RSX run-time system, and passes control to the program.

Up to this point, the program itself is still limited to 28K words; user APR 7 was needed for the RSX run-time system to process the command and load the program. Now, however, the program can execute directives to expand itself—up to (32K–32)words. Or, it can access resident libraries of routines or data with user APR 7.

2.3.2 Resident Libraries

The system manager can also install the capability to handle resident libraries in the RSTS/E system.

A resident library, like a run-time system, can be shared by many user programs. In fact, resident libraries in RSTS/E have many parallels with run-time systems, both in concept and implementation.

The underlying concept of both run-time systems and resident libraries is that both are "shareable." The difference lies in their purpose. Run-time systems allow user programs to share code that extends the capabilities of the monitor. A user program runs under the control of a run-time system. A resident library, on the other hand, extends the capabilities of the user program. A user program can pass control to a routine within a resident library or access data in a resident library.

As with run-time systems, the system manager defines a file as a resident library with UTILTY (*RSTS/E System Manager's Guide*). A resident library can be defined as permanently resident; that is, it will always be in physical memory. Or, a resident library can be defined such that the monitor loads it from disk when necessary (when a job requests its use, or "attaches" to, the resident library). The resident library then remains in memory as long as at least one job is attached to it. It is removed when no jobs are using it and the space is needed for something else.

You can access a resident library of shareable routines or data most easily by using the Task Builder (TKB).* The Task Builder options allow you to link the resident library to your executable program in such a way that your program can reference mnemonic entry points and locations in the resident library.

However, system directives also exist (used by the Task Builder itself) by which a user program (user job image) can attach itself to a resident library, create a window of virtual addresses to refer to locations in the library, and map the virtual addresses to all or some portion of the memory occupied by the resident library.

As with run-time systems, APRs are used to create the windows of addresses that are mapped to actual memory locations. So, windows to access resident libraries take up space in the job area in 4K word units. Such windows cannot overlap the user job image (low segment) and cannot overlap the run-time system (high segment) unless it is the RSX run-time system installed so that it can "disappear."

The illustration of memory mapping in Figure 2–2 shows a resident library mapped as part of a job running with the RSX run-time system.

2.4 Low-Segment Details — First 1000 Bytes of the Low Segment

The monitor attaches special significance to the first 1000_8 bytes of virtual address space in the low segment. This space is automatically allocated by the RSX task builder and RT11 linker; relocatable addresses assigned by

---

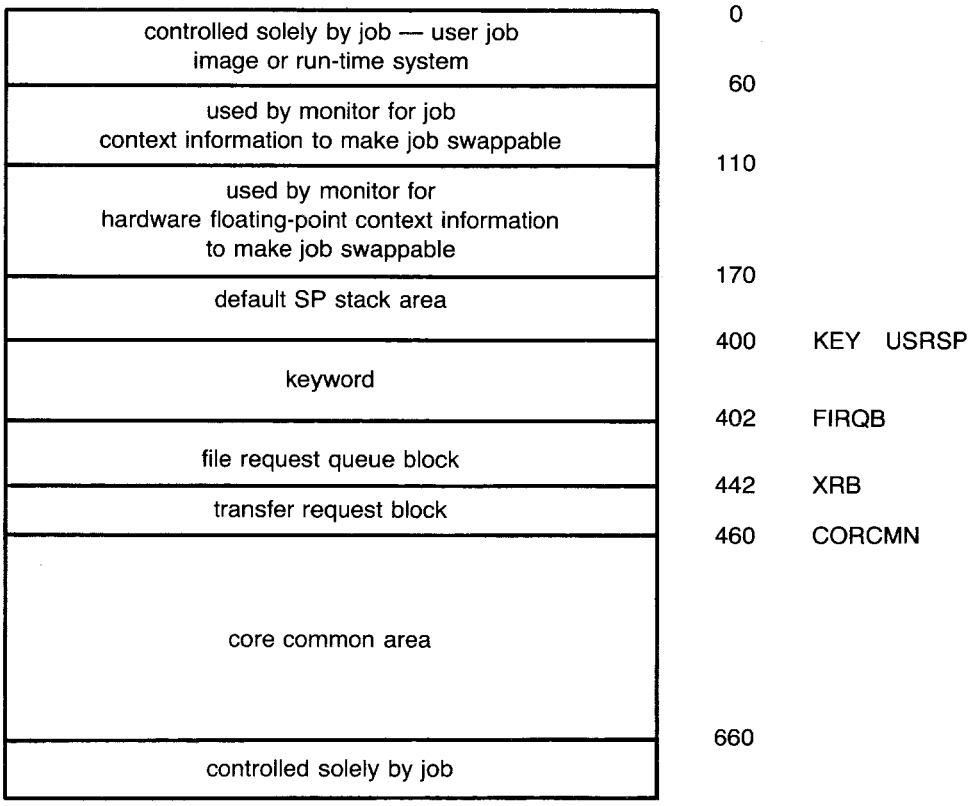
* See the description of the COMMON, LIBR, RESCOM, and RESLIB options in the *RSTS/E Task Builder Reference Manual.*

these programs always begin at location 1000_8 unless you request otherwise. If you wish to use the general monitor directives described in Chapter 3, your program must fill parts of this area with information for the monitor, and the monitor passes information back in this area. Rather than use octal addresses, you can use the COMMON.MAC prefix file, described in Section 3.1.2, to assign mnemonic names to commonly used addresses and offsets. COMMON.MAC does not allocate space, but rather assigns mnemonic names to areas within the first 1000_8 bytes of virtual address space. Using the mnemonics assigned with COMMON.MAC makes the code more readable and easier to maintain.

The general regions in this area are shown in Figure 2–4. Note that a run-time system may use some of the areas differently when it assumes control. The RSX run-time system, for example, uses the memory labeled "default SP stack area" as a table of logical units. The Task Builder automatically generates a "user stack" after the first 1000 bytes of virtual address space. (Section 4.5 briefly describes how RSX uses the first 1000 bytes.)

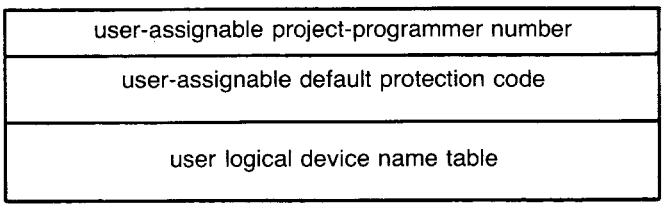
If you use the general directives in Chapter 3, you should reference only the areas that are shown with mnemonics (provided by COMMON.MAC). The mnemonics to the right in Figure 2–4 are assigned through COMMON.MAC.

Figure 2–4: First 1000 Bytes of Low Segment



(continued on next page)

Figure 2–4: First 1000 Bytes of Low Segment (Cont.)

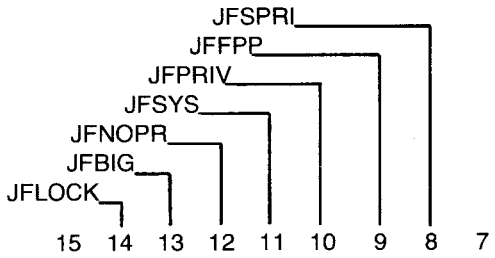


A general description of the named areas follows. The general monitor calls in Chapter 3 describe specific formats for the areas the calls use.

KEY

The keyword defines the job's status in the time-sharing environment, for example, the job's privilege. Bits in the keyword can be set and cleared by the monitor or by the job (either the run-time system or the user job image). The job can manipulate some bits in the keyword with the .SET and .CLEAR directives (Sections 3.21 and 3.5).

The keyword is "refreshed" by the monitor at certain points, for example, when a run-time system is entered at P.RUN, where the intent is to load and execute a program file in the user job image (.RUN directive, Section 3.20). For a keyword refresh, the monitor clears bit 15 and bits 7–0 and sets the remaining bits to indicate the job's current status. Only seven bits are significant to the monitor. The rest can be used by the job in whatever manner it wishes.



JFLOCK Set to 1 indicates that the job does not wish to be swapped. You can change this bit with .SET and .CLEAR. When this bit is set, the only normal condition that will cause the job to be swapped is when the job asks for a memory size expansion (see .CORE, Section 3.6) and there is not enough room to do the expansion where the job now is in memory.

JFBIG Set to 1 indicates that the job can exceed its private memory maximum (see .CORE directive, Section 3.6). This bit is implicitly set every time a privileged program is run (by either a privileged or a nonprivileged job). It can be changed with .SET and .CLEAR.

JFNOPR Set to 1 indicates that the job is not yet logged in. It is an informational bit and can be altered only by the monitor when the job is logged in.

JFSYS Set to 1 indicates that the job is currently running with temporary privileges. It is set by the monitor when a nonprivileged job runs a privileged program. Once the program is run, the job can regain temporary privilege by setting this bit and can drop privilege temporarily by clearing it. (This bit is never 1 when JFPRIV, below, is 1.)

JFPRIV Set to 1 indicates that the job can exceed its private memory maximum (see .CORE directive, Section 3.6). This bit is implicitly set every time a privileged program is run (by either a privileged or a nonprivileged job). It can be changed with .SET and .CLEAR.

JFFPP Set to 1 indicates that the contents of the hardware floating-point unit (if any) should be part of the context of this job. That is, information in the floating-point registers should be saved and restored along with the rest of the user job image during swapping. Any program that uses the hardware floating-point unit should set this bit. It can be changed with .SET and .CLEAR.

JFSPRI Set to 1 indicates that the job is running with a special run priority—at 1/2 level higher than normal. This bit can be changed with .SET and .CLEAR.

USRSP

This mnemonic is assigned the value 400 (by COMMON.MAC). The monitor automatically loads this value into the stack pointer register (SP) when a job is created. SP is also reset to this value under certain conditions, effectively establishing a default user stack area for the job beginning at word 376. The user stack area ends at location 170 (octal). Any attempt to push the stack past location 170 results in a "stack overflow" error that is handled by the run-time system (see the description of P.BAD in Section 2.5).

You can change SP if you want. However, any attempt to reset SP to any location between 0 and 167 (octal) causes a "stack overflow" error. In addition , the monitor resets SP to 400 when a run-time system is entered with a .RUN, .CCL, or .RTS directive (Sections 3.20, 3.3, and 3.19, respectively), and when certain catastrophic errors occur, such as a fatal disk error while the user job image was being swapped (see the discussion of P.BAD in Section 2.5). You need to be aware that the monitor resets SP at these times only if you are coding or modifying a run-time system. The system does not return control to a user program under these conditions, because the program cannot recover.

---

* All privileged utilities that can be executed by nonprivileged users — SYSTAT for example — clear this bit before exiting, so that the temporary privilege set up for the job cannot be used further.

FIRQB

The FIRQB (file request queue block) is the main communication area between the monitor and the job for monitor directives that involve file or device operations such as open, close, and so forth. Either the run-time system or the user job image may use this area. If, for example, you use the general monitor directives described in Chapter 3, your MACRO program must store values in the FIRQB before issuing some of the directives. If you choose to use the directives in either the RSX or RT11 run-time systems, code within the run-time system will intercept the request, set up the FIRQB and other relevant areas, and then call the monitor to handle the request.

The general format of the FIRQB, with all possible mnemonics assigned by COMMON.MAC, is shown in Figure 2–5. In addition, the size of the FIRQB (40_8) has the mnemonic FQBSIZ.

Figure 2–5: General FIRQB Format

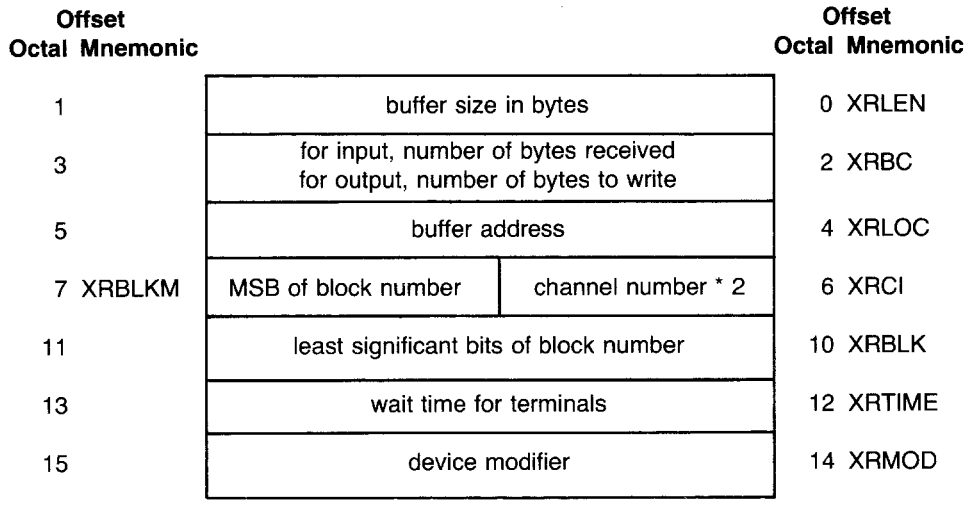
Offset Octal	Mnemonic			Offset Octal	Mnemonic
1			returned status	0	FIRQB
3	FQFUN	CALFIP/.UUO subfnc.	job number * 2	2	FQJOB
5	FQSIZM	MSB of file size	channel number *2	4	FQFIL FQERNO
7		project number	programmer number	6	FQPPN
11		file name (2 words in RAD50 format)		10	FQNAM1
13				12	
15		file type (1 word in RAD50 format)		14	FQEXT
17		least significant bits of file size		16	FQSIZ
21		buffer length		20	FQBUFL FQNAM2
23		mode		22	FQMODE
25		status flags		24	FQFLAG
27	FQPROT	protection code	≠0, prot. code real	26	FQPFLG
31		device name (2 ASCII characters)		30	FQDEV
33		≠0,unit number real	device unit number	32	FQDEVN
35		cluster size		34	FQCLUS
37		number of entries in directory lookup		36	FQNENT

XRB

The XRB (transfer request block) is the main communication area between the monitor and the user for monitor directives handling file or device I/O.

It is also the area in which the monitor stores information requested by straightforward information-request calls. As with the FIRQB, the general monitor directives described in Chapter 3 require that you store and retrieve information directly to and from the XRB. The RSX and RT11 runtime systems handle more general directives, which they translate to a call or calls using the XRB. The general format of the XRB, with all possible mnemonics assigned by COMMON.MAC, is shown in Figure 2–6. In addition, the size of the XRB (16_8) has the mnemonic XRBSIZ.

Figure 2–6: General XRB Format



CORCMN

The core common area (CORCMN) is used as a common data exchange area when it is necessary to exchange lengthy data (usually strings) between the monitor and the job or between programs running under the same job number.

For example, the monitor uses CORCMN to pass to the job a string that is the full name of a command that has been recognized as a valid CCL command. The Concise Command Language (CCL) of RSTS/E allows users to type one-line commands to call utilities that might otherwise require several input lines from the terminal. For example:

CCL Form

```
$ PIP FILE1.=FILE2.

$
```

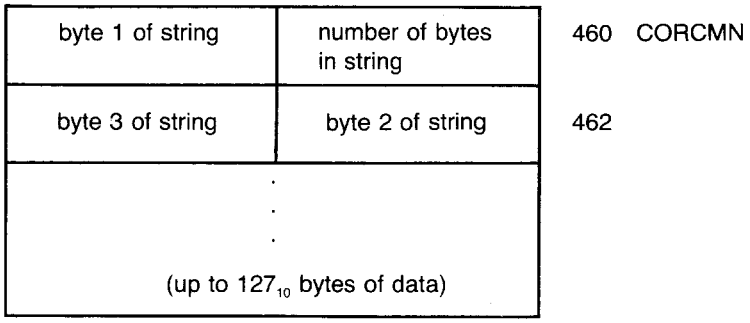
Regular Form

```
$ RUN $PIP
*FILE1.=FILE2.
*CTRL/Z

$
```

To centralize decoding, the monitor analyzes CCL commands by comparing them to those defined by the system manager when the system starts up timesharing. With the .CCL directive (Section 3.3), a job can ask the monitor to analyze a string to see if it is an acceptable command. If it is, the monitor passes control to the run-time system associated with that CCL command and passes the command and any arguments on to the job in the CORCMN area.

The general format of the CORCMN area is:



USRPPN, USRPRT, USRLOG

The job can set these areas (see .ULOG, Section 3.31) to the assigned project-programmer number (USRPPN), default protection code (USRPRT), and assigned logical device names (USRLOG), which the monitor then uses when an .FSS directive (Section 3.10) is executed. The .FSS directive causes the monitor to convert a file name string to the standard RSTS/E file specification format, that is, to the FIRQB format.

The .ULOG and .FSS directives also allow you to define and use some nonstandard area to contain these values, as described in Sections 3.10 and 3.31. However, the .ULOG directive will set up 18 words in the same basic format; the .FSS directive will expect these values in the same relative locations.

USRPPN

A nonzero value in this word is interpreted as a project-programmer number (high byte = project number, low byte = programmer number). The monitor uses this value to translate an at sign character (@) encountered in a file specification string for .FSS. If this word is zero, an .FSS will produce an error if an @ character appears in the string to be translated.

USRPRT

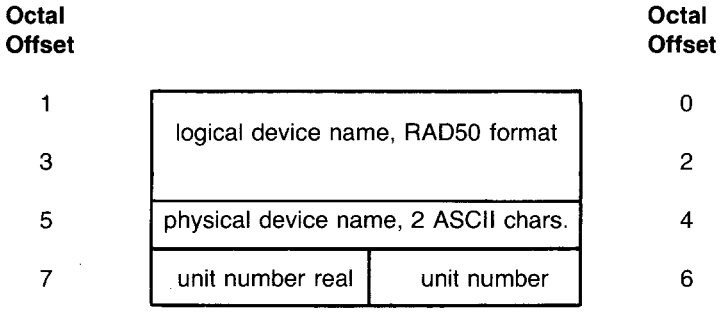
A nonzero value in this word is interpreted as a protection code to be used as a default if no protection code is specified in a file specification string translated by an .FSS directive. The value of the protection code should be in the high byte; the low byte should be nonzero, to indicate an explicit protection code. (Protection code values may range from 0 through 377_8; the meanings associated with various values are described in the *RSTS/E System User's Guide.*)

A zero in this word indicates that there is no default protection code for this user. Therefore, the system default protection code (normally 60_{10}) is used.

USRLOG

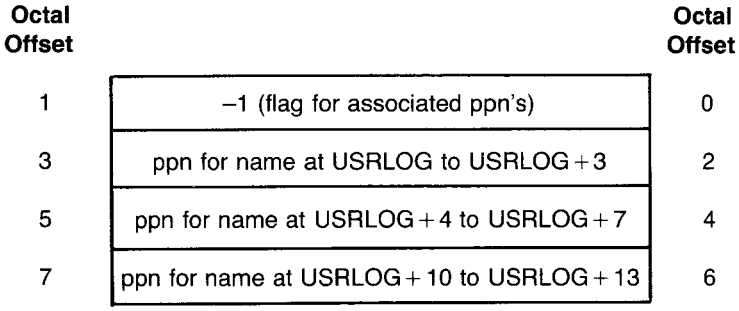
This area holds the user's private logical device name table. It consists of 16_{10} words, allowing either three or four logical names to be associated with devices. (If a project-programmer number is associated with a logical name, only three logical names can be assigned. If no project-programmer number is associated with any logical name, four logical names can be assigned.) The .FSS directive uses this table for logical-to-physical device translation; the user logicals here will supersede any system-wide logical names defined by the system manager.

The format of each entry in the first 12_{10} words is:



Offset	Meaning
0	This two-word area contains the logical device name in RAD50 format. If the first word is zero, then this entry is currently unused, and the remaining three words of the entry are random.
4	This word holds the physical device name as two ASCII characters. This physical device name is the one to be substituted for the given logical device name in an .FSS directive.
6	The low byte of this word contains the unit number of the physical device. The high byte is set to a nonzero value to indicate an explicit device number. The unit number defines the particular unit of the physical device substituted for the given logical device name in an .FSS directive. If the entire word is zero, then no unit number is associated with the device (for example, SY:).

If no project-programmer number is associated with a logical name, the format of the last four words of the USRLOG area is the same as described above. If a project-programmer number is associated with a logical name, the format of the last four words is:



(Note: The .ULOG directive automatically sets up these areas—see Section 3.31.)

2.5 High-Segment Details — Pseudo Vectors

The monitor and the run-time system use the pseudo-vector area to communicate with each other. The general layout of this area is shown in Figure 2–7. As with the low 1000 bytes of virtual address space, the file COMMON.MAC assigns mnemonic names to locations in this area. These names are shown to the right in Figure 2–7. Each of the areas is described in detail in the text following. If you wish to modify or code your own run-time system, the format and meaning of these areas is of considerable interest. Otherwise, you might wish to examine them simply to get an idea of the type of communication between the run-time system and the monitor.

In general, the pseudo-vector region contains:

- Values and flags that define the capabilities of the run-time system for the monitor. For example, one flag indicates whether the run-time system can handle user-typed commands—a keyboard monitor capability.
- Addresses pointing to locations within the run-time system where the monitor is to pass control when certain conditions occur. These addresses fall into three categories:

  1. Addresses for Synchronous System Traps (SSTs).* Control is passed to these locations when the job executes an instruction that causes a trap to the monitor. The monitor passes control to the run-time system along with the contents of the program counter (PC) and program status word (PSW). The term "synchronous" is used in the sense that the trap occurs at the same time as (is a direct result of) some instruction executed by the job. These traps may or may not indicate an error. For example, if the job executes an instruction with an odd address, control is passed to one of these trap addresses. If the job simply executes a BPT instruction, control is passed to another of these addresses.
  2. Addresses for Asynchronous System Traps (ASTs).* Control is passed to these locations (a) as a result of some event external to the execution of the job (for example, the user types a CTRL/C at the terminal) or (b) as a result of some internal but asynchronous process (such as an error in the hardware floating-point processor, whose execution overlaps that of the PDP–11 central processor). When such conditions occur, control is passed to the monitor, which passes control on to the run-time system, along with the contents of the PC and PSW. In the case of a floating-point trap, the monitor also passes along the floating exception code (FEC) and floating exception address (FEA). For the asynchronous traps, the PC and PSW do not refer to the instruction that caused the trap, but to the instruction that was executing in the central processor when the trap occurred.

* The term "pseudo vector" arises from the relationship of some of these (one-word) trap addresses in the pseudo-vector region to the (two-word) vector addresses in kernel-mode memory set up to handle error traps and interrupts in the PDP–11. When the RSTS/E monitor receives control as a result of a trap to certain of these vector addresses, it passes control on to the run-time system at addresses specified in the pseudo-vector region.

3. Entry Point Addresses. The monitor passes control to the run-time system at entry-point addresses when some major transition point is reached for the job. For example, when the user types a RUN or CCL command at the terminal, the monitor passes control to an entry point in the appropriate run-time system, to load and execute the requested program.

Figure 2–7: Format of Pseudo-Vector Region of High Segment

flags describing the run-time system	177732	P.FLAG	P.OFF
normal executable file type	177734	P.DEXT	
(reserved)	177736		
minimum size, in K words, of user job image	177740	P.MSIZ	
trap address for FIS hardware floating point option	177742	P.FIS	
crash entry point (primary run-time system only)	177744	P.CRAS	
start entry point (primary run-time system only)	177746	P.STRT	
entry point for new user	177750	P.NEW	
entry point for new user with program to run	177752	P.RUN	
trap address for various "bad" errors	177754	P.BAD	
trap address for BPT instruction and T-bit traps	177756	P.BPT	
trap address for IOT instruction	177760	P.IOT	
trap address for non-monitor EMT instructions	177762	P.EMT	
trap address for all TRAP instructions	177764	P.TRAP	
trap address for FPP or FPU floating point unit	177766	P.FPP	
trap address when user types one CTRL/C	177770	P.CC	
trap address when user types two CTRL/Cs	177772	P.2CC	
maximum size (in K words) of user job image	177774	P.SIZE	
(reserved for future use)	177776		

The pseudo-vector region is described in detail below.

Normally, you would code the contents of the pseudo-vector region as part of the run-time system file. Please note, however, that the UTILTY program's ADD command, used to define a file as an auxiliary run-time system, has switches that will cause the monitor to override certain portions of the pseudo-vector region and use values assigned in the ADD. For example, one bit in one word of the pseudo-vector region states whether the run-time system is read/write or read-only when it is loaded in memory. Normally, this would be read-only, but for debugging a run-time system with ODT (which allows you to change memory), the run-time system must be read/write. The /RW switch in the ADD command of UTILTY lets you tell the

monitor that until further notice, this run-time system is read/write, regardless of what is specified in the pseudo-vectors. The UTILTY program and its ADD command are described in the *RSTS/E System Manager's Guide.*

2.5.1 Run-Time System Capability and Default Definitions

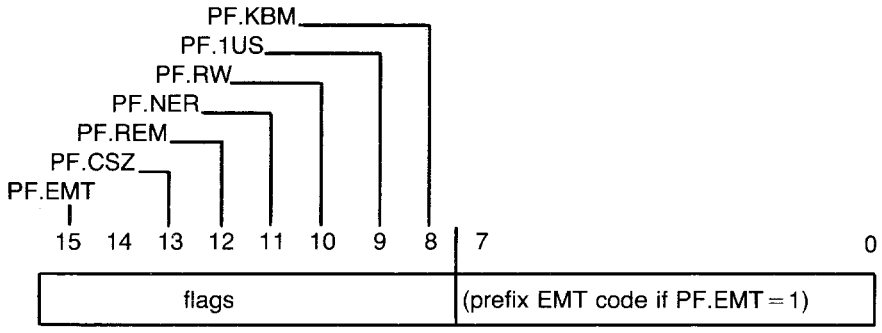
These mnemonics refer to values and flags that define run-time system capabilities for the monitor.

P.OFF

The P.OFF mnemonic is simply used to define the first word of the pseudo-vector region. It is currently set equivalent to 177732, the same as P.FLAG.

P.FLAG

The monitor expects the P.FLAG word to be set with flags that define the capabilities of the run-time system:



PF.EMT

This bit is set to 1 to indicate that the run-time system wishes to handle a call that would normally be handled by the monitor. To show how the bit works, we must first describe what normally happens when a monitor directive is translated and executed.

All the monitor directives described in this manual are translated to EMT (Emulator Trap) instructions. The direct monitor calls (Chapter 3) are one-for-one translations; that is, one call is translated to one EMT. The code to process the call is in the monitor itself. The RSX and RT11 emulator calls may be translated to more than one instruction, but the code always contains an EMT. The direct monitor calls, furthermore, are translated to an EMT with a low byte that is an even number within the range $0–100_8$. When such an instruction is executed, control transfers directly to the monitor, the call is processed, and control returns to the instruction following the EMT.

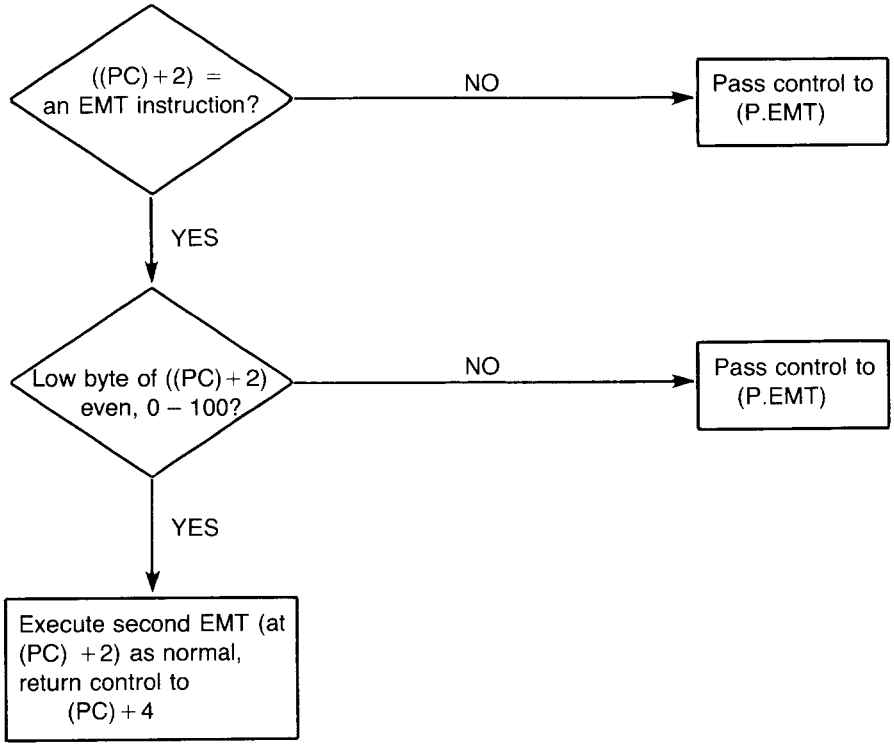
An EMT instruction with an odd value in the range 1–77 in the low byte, or any value in the range 101–377, also transfers control to the monitor. The monitor examines the low byte, discovers that the EMT is not one of its "own," and transfers control to the run-time system at the entry point defined by location P.EMT in the pseudo-vector region.

Now—the PF.EMT bit is set to 1 to indicate that the run-time system wishes to process EMTs that are normally processed by the monitor, that is, with an even low byte in the range 0–100. When PF.EMT is set to 1, all EMTs will cause control to pass to the run-time system at entry P.EMT except those immediately preceded by a "special prefix" EMT—an EMT whose low byte is equal to the low byte of P.FLAG.

Specifically, when PF.EMT equals 1, the monitor handles all EMT instructions as follows:

1. Any EMT whose low byte is not equal to the low byte of P.FLAG will cause control to pass through the monitor (unprocessed except for examination), back to the run-time system at the address contained in the P.EMT word.

2. An EMT whose low byte is equal to the low byte of P.FLAG will cause control to pass to the monitor, which looks at the word following the EMT with the special code (that is, at the word in location (PC) + 2). Action is taken according to the value of this word:



In other words, special processing is done by the run-time system for all EMTs except those preceded by a "special prefix" EMT. The RT11 run-time system uses this feature so that it can emulate the RT11 operating system's directives properly.

NOTE

All EMT instructions are reserved for use by DIGITAL.

PF.CSZ

For a user job image executed as a result of a .RUN directive, the monitor preallocates memory based on information provided by the run-time system the image is executing under. When this bit is set to 1, the monitor preallocates memory based on the size of the file referenced in the .RUN directive:

space (in K words) = (*filesize* + 3)/4

Filesize is the number of 512–byte blocks required for the file on disk. (The division by 4 is performed because there are four 512–byte blocks in 1K word. The addition of 3 "rounds up" any fraction of the integer divide to the next whole integer.)

When PF.CSZ is set to 0, the monitor preallocates memory for the image according to the value specified in the P.MSIZ word of the pseudo-vector region.

PF.REM

When the PF.REM bit is set to 1, the monitor immediately removes the run-time system from memory when no job is using it. When this bit is 0, the monitor leaves the run-time system in memory until the space is actually needed by something else.

PF.NER

When this bit is set to 1, the monitor does not log errors occurring within the run-time system to the system error log.

PF.RW

When this bit is set to 1, the monitor maps the run-time system as read/write (recall the read/write feature of the Page Descriptor Register of an APR, Section 2.1). This is a useful feature when debugging a run-time system. In normal operation, this bit should be set to 0, indicating that the run-time system is to be mapped read-only.

PF.1US

When the PF.1US bit is set to 1, the monitor allows only one job to use the run-time system. That is, it is not handled as shareable code.

PF.KBM

When this bit is set to 1, the monitor expects that the run-time system can function as a keyboard monitor. The run-time system can function as a job keyboard monitor only when this bit is set. (See the .RTS directive, Section 3.19, for a discussion of job keyboard monitors.)

P.FLAG COMBINATIONS

The PF.1US, PF.RW, PF.NER, and PF.REM bits are useful flags when you are debugging a run-time system. PF.1US limits access to the run-time system to one user; PF.RW is necessary if you wish to use the ODT routine to change memory. PF.NER keeps the run-time system from logging useless errors while debugging, and PF.REM ensures that the run-time system will be reloaded each time it is used. (Otherwise, an old copy might still remain in memory when you really wanted to debug a new copy.)

P.DEXT

This word can be set to three Radix–50 characters that the monitor will use as a default runnable file type. If a .RUN (Section 3.20) is executed with no file type given, the monitor scans its list of installed run-time systems (in the order they were installed*). For the first run-time system in the list (the primary run-time system), the monitor looks for a file with the file name given in the .RUN and a type that is the default type for the run-time system. If such a file is found, it is set up for the .RUN. If no such file is found, the monitor searches for a file with the given file name and the next run-time system's default runnable file type, and so forth. Note that the order in which the file types are chosen does not depend in any way on the run-time system executing the .RUN.

For example, the BASIC–PLUS run-time system fills this word with .BAC; RT11, with .SAV; and RSX, with .TSK. If the run-time system has no runnable file type, this word should be set to 0.

P.MSIZ

The P.MSIZ word gives the minimum allowable size for a user job image, in K words, for this run-time system. The monitor uses this value as a check when the job issues a .CORE directive (Section 3.6) to change the size of the user job image in memory. The value of P.MSIZ must be an integer between 1 and 28, inclusive.

P.SIZE

The P.SIZE word contains the maximum size (in K words) that a user job image can be for this run-time system. The monitor uses this value as a check when a job issues a .CORE directive (Section 3.6) to change the size of the user job image in memory. P.SIZE must be an integer between 1 and 28, inclusive. The effective upper limit is 32 minus the size of the run-time system rounded up to a multiple of 4. (Remember that the APR mapping requires that space for the run-time system be allocated in units of 4K words.) Thus, a run-time system that required 5K words could set an upper limit here of 24 (32–8). It could set P.SIZE to some smaller value, however.

The RSX run-time system, when the emulator is installed as part of the monitor, is an exception to this rule. As described in Section 2.3, an executing program can expand to 32K words. So P.SIZE in this particular case is set to 32.

---

* The system manager installs run-time systems with the ADD command of UTILTY, as described in the *RSTS/E System Manager's Guide*. The order of installation shows up in the display produced by the SYSTAT utility.

2.5.2 Synchronous System Trap Addresses

These mnemonics refer to locations in the run-time system where control is to pass for synchronous system traps.

P.FIS

The monitor interprets the P.FIS word as the trap address for the hardware floating-point instruction set available on the PDP–11/35 and 40. Whenever an instruction from this set is executed that causes a trap to the kernel mode vector at 244_8, the monitor passes control on to the run-time system at the location specified by the contents of the P.FIS word.

This trap pushes two words onto the user's SP stack: the contents of the PC and PS registers at the time of the exception.

SP → (PC) at the time of the trap
(PS) at the time of the trap
word to which SP pointed before the trap

Whatever action the run-time system wishes to take for this trap should be done at the location specified by the contents of P.FIS. (A return from interrupt (RTI) instruction will return control to the point where it was when the trap occurred.)

P.BAD — Synchronous Traps

The monitor passes control on to the run-time system at the location specified by the contents of P.BAD when any of the following synchronous traps occur:

1. Memory management unit exception (trapped to kernel mode vector at 250_8).
2. The job tries to execute a reserved instruction (trapped to kernel mode vector at 10_8).
3. The job issues an instruction with an odd address (trapped to kernel mode vector at 4_8).

This trap pushes two words onto the user's SP stack: the contents of the PC and PS registers at the time of the trap.

SP → (PC) at the time of the trap
(PS) at the time of the trap
word to which SP pointed before the trap

An error code is returned in the first byte of the FIRQB so that the run-time system can determine which error occurred. The codes are:

B.4	Odd address
B.10	Reserved instruction
B.250	Memory management unit exception

The run-time system is responsible for processing these errors in whatever manner it sees fit. In general, most run-time systems provided with RSTS/E systems report the error, using the UU.ERR subfunction of the .UUO call

(Section 3.32.19), and perhaps print the PC (program counter) value from the top of the stack. An RTI instruction can be used to return control to the point where it left off when the trap occurred. Note that some asynchronous traps also use this address (Section 2.5.3).

P.BPT

The P.BPT word contains the trap address for a BPT instruction and for T-bit traps. When the job issues a BPT instruction or a T-bit trap occurs (to the kernel mode vector at 14_8), the monitor passes control on to the run-time system for the job at the address specified by the contents of this word.

The trap pushes two words onto the user's SP stack: the contents of the PC and PS registers.

```
SP ──►  (PC) at the time of the trap
        (PS) at the time of the trap
        word to which SP pointed before the trap
```

The run-time system would process these traps in any fashion it sees fit at the location specified by the contents of P.BPT. The RTI or RTT instructions can be used to return control to the user's program at the point where it was when the trap occurred.

P.IOT

The P.IOT word contains the trap address for an IOT instruction. Whenever the job issues an IOT instruction (trapped to kernel mode vector at 20_8), the monitor passes control on to the run-time system at the address specified by the contents of this word. This trap pushes two words onto the user's SP stack: the contents of the PC and PS registers at the time of the trap.

```
SP ──►  (PC) at the time of the trap
        (PS) at the time of the trap
        word to which SP pointed before the trap
```

The run-time system can process the trap in any fashion it sees fit. An RTI instruction can be used to return control to the point where it was when the trap occurred.

P.EMT

This word contains the location to which control is transferred for non-monitor EMT instructions; that is, for EMT instructions whose low byte is odd within the range 0 – 100_8 or any value in the range 101 – 377. If the PF.EMT bit is set in the P.FLAG word in the pseudo-vector region, control is transferred here for all EMT instructions except those preceded by the "special prefix" EMT, as described previously.

The trap pushes two words onto the user's SP stack: the contents of the PC and PS registers at the time of the trap.

```
SP ──►  (PC) at the time of the trap
        (PS) at the time of the trap
        word to which SP pointed before the trap
```

The run-time system is responsible for processing the EMT as it sees fit. The RTI instruction can be used to return control to the point where it was when the trap occurred.

NOTE

All EMT instructions are reserved for use by DIGITAL.

P.TRAP

This is the location to which control is transferred for all TRAP instructions (operation codes 104400 through 104777, inclusive). Whenever the job executes such an instruction (trapped to kernel mode vector 34_8), the monitor passes control on to the run-time system at the location specified by the contents of this word.

The trap pushes two words onto the user's SP stack: the contents of the PC and PS registers at the time of the trap.

```
SP—►  (PC) at the time of the trap
      (PS) at the time of the trap
      word to which SP pointed before the trap
```

The run-time system is responsible for processing the trap as it sees fit. An RTI instruction will return control to the point where it was when the trap occurred.

2.5.3 Asynchronous System Trap Addresses

These mnemonics refer to locations within the run-time system where control is to pass for asynchronous traps.

P.FPP

This location is the trap address for the FPP (or FPU) hardware floating-point processor (the PDP–11/34A, 44, 45, 50, 55, 60, and 70 asynchronous unit or the KEF11–AA for the PDP–11/23–PLUS and 24). Whenever the unit takes an exception trap (to kernel mode vector at 244_8), the monitor passes control to the run-time system at the location specified by the contents of this word. Since the Floating Exception Code (FEC) and Floating Error Address (FEA) of this unit are not otherwise accessible, the monitor pushes these two values onto the user's SP stack, in addition to the contents of the PC and PS registers at the time of the interrupt.

```
SP —►  Floating-Point Exception Code (FEC)
       Floating-Error Address (FEA)
       (PC) at the time of the trap
       (PS) at the time of the trap
       word to which SP pointed before the trap
```

The run-time system can process the trap as appropriate, clean the stack (pop the FEC and FEA), and issue an RTI instruction to return control to the user's program at the point where it was when the trap occurred.

P.CC

This is the location to which control passes when the user types a CTRL/C. ("The user" in this case is any terminal input being accepted by this job, on any channel.)

The monitor inhibits further programmed output for the job (CTRL/O effect) and cancels any pending character output. The user's SP stack is modified at entry as follows:

```
SP ──►  (PC) at the time of the trap
        (PS) at the time of the trap
        word to which SP pointed before the trap
```

The run-time system can process the CTRL/C as desired. All run-time systems supplied with a RSTS/E system abort the job, unless the user job image has indicated that it wishes to handle CTRL/C traps itself (see the SCCA$ directive, Section 5.19, and the .SETCC directive, Section 7.39).

P.2CC

This is the trap address taken when the user types a second CTRL/C before the run-time system has been able to respond to the first CTRL/C typed. (That is, the monitor has received two CTRL/Cs before it has been able to pass control on to the run-time system at the location specified by the contents of P.CC in the time-sharing environment.)

As with one CTRL/C, when the P.2CC point is entered, further programmed output has been canceled (CTRL/O effect), and any pending character output has been canceled. Two words are pushed onto the user's SP stack:

```
SP ──►  (PC) at the time of the trap
        (PS) at the time of the trap
        word to which SP pointed before the trap
```

The run-time system can process the condition as desired (BASIC–PLUS exits immediately, returning control to the P.NEW entry point in the default keyboard monitor). An RTI instruction would return control to the point where the program left off, but this will annoy the user who typed the two CTRL/Cs expecting to get out.

P.BAD — Asynchronous Traps

The monitor passes control to the location specified by P.BAD whenever any of the following asynchronous errors occur:

1. The user's SP stack overflows.
2. A fatal disk error occurs when the job is swapped. The original contents of the user job image are lost.
3. A memory parity fault occurs in the user job image. The original contents of the user job image are lost.

4. A fatal disk error occurs when a run-time system or resident library is loaded. Control passes to P.BAD in the default keyboard monitor when the load error occurs for a run-time system.

None of these errors are really recoverable; an error is returned in the first byte of the FIRQB to indicate which error occurred, the keyword (KEY) is refreshed, and the contents of the general registers (R0 through R5) are random. The SP (stack pointer) is reset to the value USRSP.

In general, most run-time systems provided with RSTS/E systems report the error, using the UU.ERR subfunction of the .UUO call (Section 3.32.19), and also the "?Program lost-sorry" message (UU.ERR call with FUCORE value). Then, the run-time systems exit to the job keyboard monitor, using .RTS (Section 3.19). The reason for printing the "?Program lost-sorry" message is that it warns the user that user logical values have been destroyed.

The error codes returned in the first byte of the FIRQB are:

B.STAK	The user's SP stack overflowed.
B.SWAP	Fatal disk error on swap.
B.PRTY	Memory parity fault.
NRRTS	Fatal disk error on run-time system or resident library load.

Control is also transferred to P.BAD for some synchronous traps, as described in Section 2.5.2.

2.5.4 Entry Points

These mnemonics refer to locations within the run-time system where control is to pass at certain transition points for the job.

P.CRAS

This word is a special entry point used only by the primary run-time system. Control passes to this location in the primary run-time system only when the whole system is restarted after a crash.

P.STRT

This word is a special entry point used only by the primary run-time system. Control passes to this location only when the system starts up from a normal startup procedure.

P.NEW

The monitor passes control to this entry point under the assumption that "new user" or "next request" processing is to be done, as opposed to the P.RUN entry point, where a specific program is to be run under this run-time system. This entry point is commonly used as the entry point to switch back to a job's keyboard monitor. For example, the .EXIT directive (Section 3.9) passes control to this entry point in the default keyboard monitor. (The system manager defines a keyboard monitor as the default for all jobs.) The .RTS directive (Section 3.19) can be used to pass control to this entry point

in a job's keyboard monitor or a specifically named run-time system. (As described in Section 3.19, a job can establish its own job keyboard monitor, different from the default keyboard monitor.)

By examining the keyword (KEY) and the XRB, the run-time system can determine how and by whom it was entered at P.NEW, if this is significant. (Run-time systems that do not have keyboard monitors would probably wish to simply exit (using .EXIT) to the default keyboard monitor here.) The three conditions under which control is passed to the P.NEW entry are:

1. Brand New Job on the System. In this case, JFNOPR (bit 12_{10} in KEY) is set to 1 (the job is not yet logged in), and the words at location XRB + 2 and XRB + 4 are 0 (the monitor requested the entry, not a run-time system). This indicates that the monitor has passed control to this location having received terminal input over channel 0 in a "logged out" state (occurs only for the primary run-time system).

   The run-time system should run some predetermined program to read (.READ directive, Section 3.17) the fully assembled line that the monitor has buffered. The BASIC–PLUS run-time system loads and executes the file SY:[1,2]LOGIN.BAC (the LOGIN utility) in this case.

2. Switch to This Run-Time System when Job Logged Out. In this case, JFNOPR (bit 12_{10} in KEY) is set, and the name of the calling run-time system* is given as two RAD50 words in locations XRB + 2 and XRB + 4.

   For this case, the run-time system should issue a "logged out" prompt message. For example, the BASIC–PLUS run-time system prints "Bye" and returns control to the monitor. (Normally, control does not pass to the run-time system in this case. If LOGIN does not recognize the line that it read (in case 1, above), it "kills itself," destroying the job and returning control to the monitor.)

3. Switch to This Run-Time System when Job Logged In. In this case, JFNOPR (bit 12_{10} in KEY) will be clear (0). The name of the calling run-time system* is given as two words of RAD50 in locations XRB + 2 and XRB + 4 or is 0 if this job was just created by UU.JOB (see Section 3.32.23).

   For this situation, the run-time system should issue its "logged in" prompt and attempt to read the next command from the terminal open on channel 0. BASIC–PLUS prints "Ready", DCL prints "$", RSX prints ">", and RT11 prints ".". At this point, all then wait for further input.

Keyboard monitors should read channel 0 (the job's terminal) using the "keyboard monitor wait" feature of .READ. The monitor will kill jobs that execute this read in a logged-out state; otherwise, it is an "infinite-wait" read.

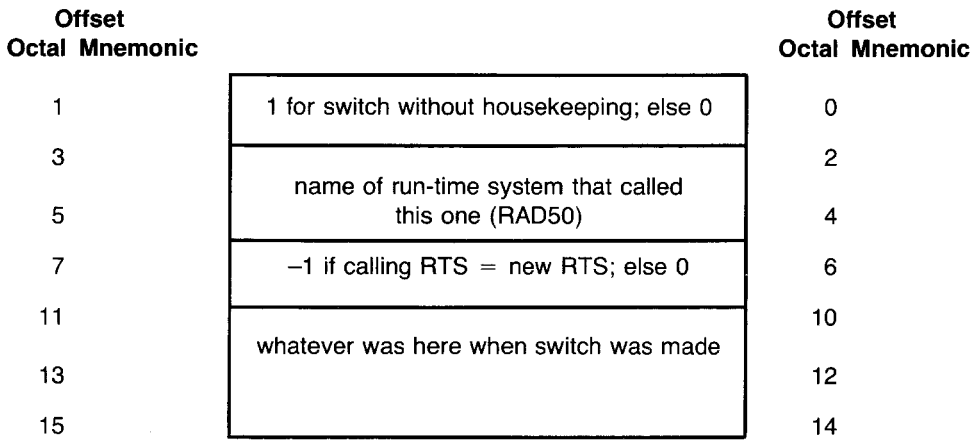
---

* The calling run-time system is the run-time system under whose control the directive was issued that caused the switch.

The monitor usually* does some housekeeping for the job at the time the P.NEW entry point is entered. Specifically, the word at location FIRQB + FQJOB is always set to two times the job number assigned by the monitor when the job was created, and the keyword (KEY) is refreshed with current information about the job, as described in Section 2.4. Furthermore, the stack pointer, hardware register SP, is reset to 400 (see USRSP description in Section 2.4), and all the general registers (R0 through R5) contain random values. Note, however, that I/O channels are left open. Therefore, the run-time system should reset all I/O channels.

The following information exists in the XRB at the time the P.NEW entry point is entered:

XRB on P.NEW Entry



XRB + 0 This word contains a value of 1 if control was transferred by an .RTS directive using the "switch without changing job context" option (Section 3.19).

XRB + 2 The two words beginning here contain the name of the calling run-time system, in RAD50 format. If control has been transferred here directly by the monitor, these two words contain 0.

XRB + 6 This word contains –1 if the calling run-time system is the same run-time system that now has control. This word is 0 otherwise; that is, if the calling run-time system is not the same as the called run-time system.

XRB + 10 The contents of the next three words will be the same as they were when the switch occurred. That is, data can be passed from run-time system to run-time system here. If control has been transferred to P.NEW directly by the monitor, these three words are 0.

---

* This housekeeping is not done if a specific request is made to pass control to a run-time system without changing the job-context information. (See .RTS, Section 3.19.)

P.RUN

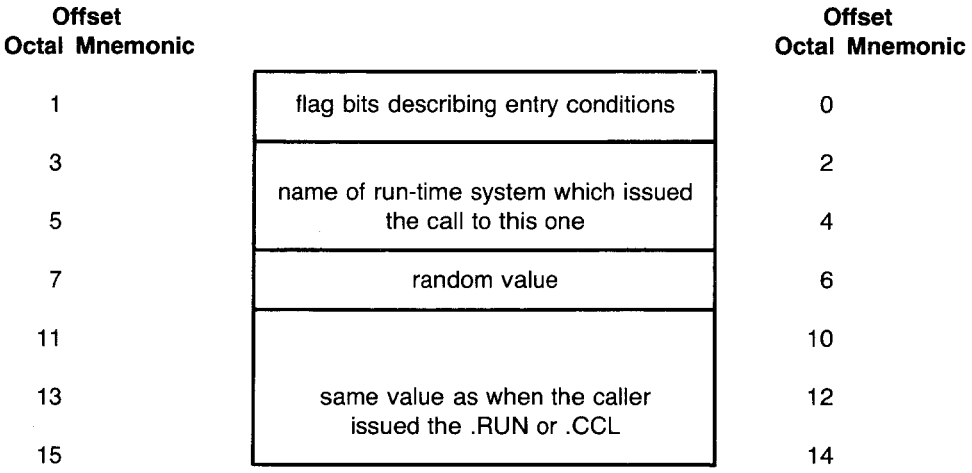
The monitor passes control to the P.RUN entry point when an executable program is to be run for a job under control of this run-time system. This can occur as the result of either a .RUN directive (in which a job has directly asked for a file to be run) or a .CCL directive (in which a job has asked the monitor to check a string to see if it is a valid Concise Command Language (CCL) command, and if so, execute the appropriate file).

The file to be run has been opened by the monitor on channel 15_{10} and is a disk file. The file has not been read; it is up to the run-time system to load and execute the file. The run-time system should also reset all I/O channels except 15_{10}, as they may be open.

As with P.NEW, the monitor has "refreshed" the keyword (KEY), set SP to 400, and left R0–R5 containing random values. In addition, the monitor sets the JFBIG bit in KEY if the program to be run is a privileged program. If the program to be run is privileged and the caller is normally nonprivileged, the monitor sets the JFSYS bit in KEY.

Data is passed to the run-time system in the XRB, FIRQB, and KEY areas of the user job image (low segment).

XRB on P.RUN Entry



XRB+0 This word contains flag bits that describe the entry conditions. (The STATUS variable in BASIC–PLUS returns these values.)

Bit	Meaning
15	Set to 1 indicates the entry was made as the result of a .CCL directive. A value of 0 indicates the entry was made as the result of a .RUN directive.
14	Set to 1 indicates the caller issued a .CCL directive with a /DETACH switch, with the intent that this run-time system run the file in detached mode. It is up to the run-time system to take action on this flag. (Detaching a job can be done with the UU.DET subfunction of the .UUO directive, as described in Section 3.32.)

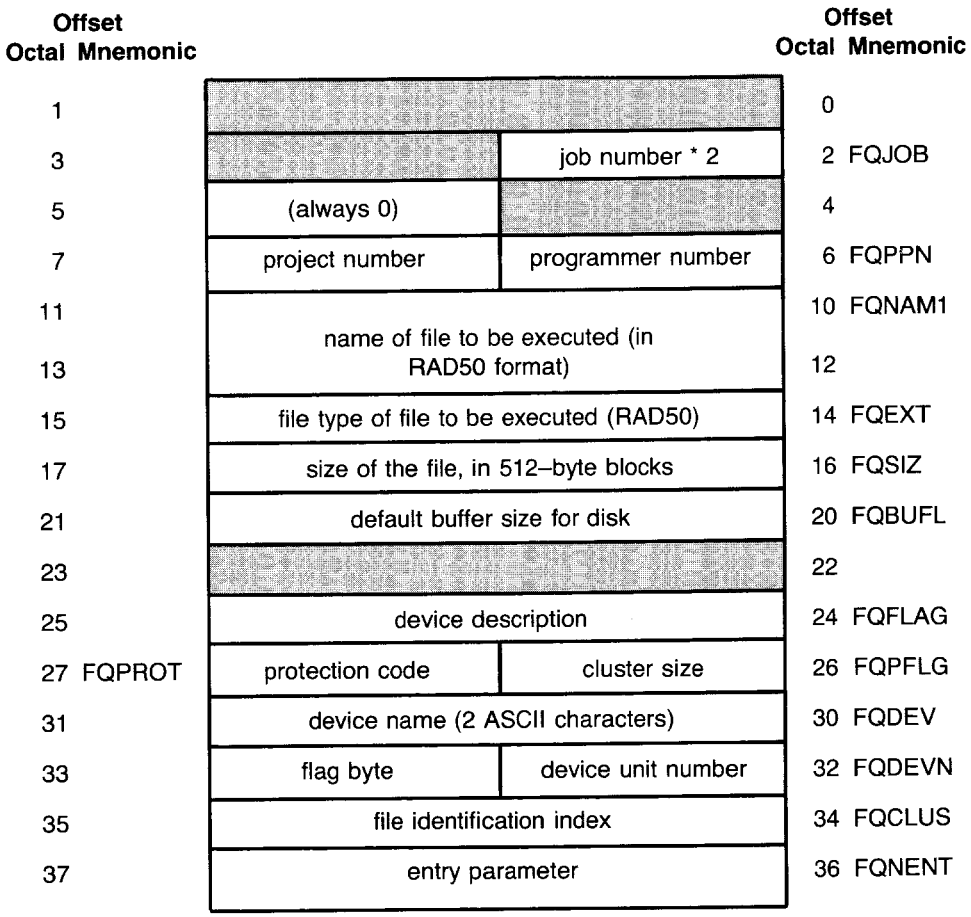
Bit	Meaning
13	Set to 1 indicates that the caller issued a directive with a /SIZE switch; that is, that the file is to be run at a specific size. When this bit is set, bits 0–7 of this word indicate the desired size (see below). It is up to the run-time system to set the size as indicated (see .CORE, Section 3.6.)
12–8	Reserved for future use.
7–0	If this byte is 0, then no special size for this program run is desired. If greater than zero, this byte indicates the size (in K words) that the program should be run at. If less than zero, the absolute value of this byte indicates an increment (in K words) to the size that the program would normally run at.

XRB+2 These two words contain the name of the run-time system under which the .RUN or .CCL directive to this run-time system was issued, in RAD50 format.

XRB+6 The contents of this word are random.

XRB+10 The three words beginning here contain the same information that they held when the job issued the .RUN or .CCL directive.

FIRQB on P.RUN Entry



FIRQB+FQJOB — The job number, times two, assigned by the monitor when this job was created.

FIRQB+FQPPN — The project-programmer number for the file that is to be run.

FIRQB+FQNAM1 — The file name of the file that is to be run, as two words in RAD50 format.

FIRQB+FQEXT — The file type of the file that is to be run, as one word in RAD50 format.

FIRQB+FQSIZ — The size of the file, in 512–byte blocks.

FIRQB+FQBUFL — The recommended size, in bytes, for the buffer size in a .READ for this (disk) file.

FIRQB+FQFLAG — Flag bits defining the device. They are set to indicate that this is a disk file. (See the FQFLAG description in the "open" function of the CALFIP directive, Section 3.2.12.)

FIRQB+FQPROT–1 — The file cluster size, modulo 256_{10}. (That is, a file cluster size of 256_{10} is indicated by a zero byte here.) This byte is the same as the FQCLUS value supplied in the "open" functions of the CALFIP directive, Section 3.2, except that it is returned in a byte instead of a word.

FIRQB+FQPROT — The protection code of the file.

FIRQB+FQDEV — The device name of the disk device, as two ASCII characters.

FIRQB+FQDEVN — The unit number of the disk device.

FIRQB+FQDEVN+1 — The low-order two bits of this byte are set to indicate whether or not the device is part of the public structure:

Bit 0 = 0 The device is in the public structure.
Bit 0 = 1 The device is a private disk.
Bit 1 = 0 A specific device was not specified in the open.
Bit 1 = 1 A specific device was specified in the open.

FIRQB+FQCLUS — The file identification index of this file. This word is significant mainly in that it can be used in place of the file name in subsequent opens of the file on disk. You can open the file with the OPNFQ subfunction of CALFIP (Section 3.2.12) using an explicit project-programmer number in FIRQB+FQPPN, a zero word in FIRQB+FQNAM1, an explicit device name in FQDEV and FQDEVN, and the file identification index in FIRQB+FQNAM1+2.

FIRQB + FQNENT

Parameter word from the caller. The .RUN directive, which causes entry at P.RUN in a run-time system, allows the caller to specify a parameter word to be passed to the run-time system. Bit 15 of this word may or may not be the same as the caller passed, however. If the caller was privileged, whatever the caller specified for bit 15 is passed. If the caller was nonprivileged, the monitor automatically sets this bit to 0. The intent of this bit is to "pass on" temporary privilege. If this bit is set, the run-time system should retain temporary privilege (the JFSYS bit in KEY is now set and should remain set). If this bit is off, the run-time system should consider whether or not to retain temporary privilege (if the JFSYS bit is on, perhaps it should be turned off).

For example, the BASIC–PLUS run-time system uses the FIRQB + FQNENT word as follows.

Temporary privilege is kept (JFSYS, if set, is left set) if either:

1. Bit 15 of FIRQB + FQNENT is set
2. Bit 15 of FIRQB + FQNENT is clear, but the remaining bits are also 0

If bit 15 of FIRQB + FQNENT is clear and the remaining bits are nonzero, temporary privilege is permanently dropped (by issuing .CLEAR for both JFSYS and JFPRIV).

In either case, the contents of bits 14–0 of FIRQB + FQNENT are used as the line number where execution is to be started for the program. A value of 0 in bits 14–0 means to start at the first executable line. Thus, clearing JFSYS (dropping temporary privilege) when execution is to start at other than the first line protects privileged programs from being started at odd locations to bypass their normal privilege checking.

Finally, if JFSYS was set upon entry (whether or not it gets immediately cleared), then the program will be destroyed upon completion (normal or abnormal). This protects a privileged program from being examined when it is not running.

PART II
Monitor Directives

Chapter 3
General Monitor Directives

3.1 Introduction

This chapter describes the general directives to the RSTS/E monitor. These directives are available to the MACRO programmer under both the RSX and RT11 run-time systems. They are EMT instructions that are processed directly by the monitor. They are not examined or processed by a run-time system.

3.1.1 Summary of General Monitor Directives

Table 3–1 summarizes the general monitor directives. Detailed descriptions are given in Sections 3.2 – 3.33. These sections are arranged alphabetically by mnemonic name.

Some directives, which cause a change in run-time system or change in job size, should not be executed by a program (user job image) running under the RT11 run-time system, or unpredictable results may occur. These directives are marked with an asterisk (*) in Table 3–1. (Note, however, that these are not restrictions for assembling using MACRO — the assembler for the RT11 run-time system. If you are coding a run-time system, you can use these directives and assemble under either MACRO or MAC.)

Table 3–1: Summary of General Monitor Calls

Name	EMT Code (Octal)	Description
CALFIP	0	Call the File Processor portion of the RSTS/E monitor. Includes "housekeeping" functions for file/device I/O such as open, close, and so forth.
.READ	2	Read from a previously opened file or device.
.WRITE	4	Write to a previously opened file or device.
.CORE*	6	Change memory size allocated for user job image.
.SLEEP	10	Sleep job for n seconds.
.PEEK	12	Peek at the monitor's memory (privileged).
.SPEC	14	Special function.
.TTAPE	16	Enter tape mode.
.TTECH	20	Enable echo on a channel.
.TTNCH	22	Disable echo on a channel.
.TTDDT	24	Enter ODT submode on a channel.
.TTRST	26	Cancel CTRL/O effect.
.TIME	30	Get timing information.
.POSTN	32	Get device's horizontal position.
.DATE	34	Get current date.
.SET	36	Set keyword bits.
.STAT	40	Get statistics for job.
.RUN*	42	Run new program (user job image).
.NAME	44	Install a new program name.
.EXIT*	46	Exit to default keyboard monitor.
.RTS*	50	Switch to new run-time system.
.ERLOG	52	Log an error from run-time system.
.LOGS	54	Check for logical devices.
.CLEAR	56	Clear keyword bits.
.MESAG	60	Message send/receive.
.CCL*	62	Check string to see if Concise Command Language.
.FSS*	64	Scan a string for valid RSTS/E file specification.
.UUO	66	Execute monitor FIP call (access to BASIC–PLUS SYS calls to FIP).
.CHAIN*	70	Execute user job image under same run-time system.
.PLAS	72	Access a shared library.
.RSX*	74	Used only by RSX run-time system to "disappear", if possible.
.ULOG	76	Assign/reassign/deassign device or user logical.

* These directives should not be used by a program running under control of the RT11 run-time system.

3.1.2 Prefix File COMMON.MAC

The monitor directives described in this chapter require that you pass parameters to the monitor in the FIRQB and XRB; values are also returned to your program in these areas. The file COMMON.MAC, provided with all RSTS/E kits, relates mnemonics to often-used addresses, offset values, and function codes, eliminating the need for octal coding and addressing. These mnemonics are used in the directive descriptions that follow, and their use is recommended for readable, maintainable code.

3.1.2.1 How to Assemble with COMMON.MAC — COMMON.MAC is a prefix file; it is assembled with your other MACRO source files under either the RSX or RT11 run-time systems. For example, under the RT11 run-time system, the following sequence would assemble the files COMMON.MAC, SRC1.MAC and SRC2.MAC into the object module file OBJ.OBJ with an assembly listing file OBJ.LST:

```
RUN $MACRO
*OBJ,OBJ=COMMON,SRC1,SRC2
```

Similarly, under the RSX run-time system, this sequence would assemble the files COMMON.MAC, SRC1.MAC, and SRC2.MAC into the object module file OBJ.OBJ with an assembly listing file OBJ.LST:

```
RUN $MAC
MAC>OBJ,OBJ=COMMON,SRC1,SRC2
```

3.1.2.2 Macros Provided in COMMON.MAC — In addition to providing mnemonics, the COMMON.MAC file contains macros that can be used in programs assembled under either the RSX or RT11 run-time systems, as long as COMMON.MAC is assembled with the source, as described previously.

TITLE name,desc,nn,date,editors

The TITLE macro sets a title (.TITLE) from the name and description (desc) parameters and builds an identification (.IDENT) from the specified number nn. The IDENT has the form xx.x.nn, where xx.x is the current RSTS/E version number (08.0 for V8.0), and nn is the edit level you specify. Descriptive information is placed in the table of contents as follows:

```
EDIT:       DATE:       BY:
 nn         date        editors
```

ORG section[,offset]

ORG defines the origin address of a program section. The first occurrence of an ORG with a given section name causes all instructions requiring memory space following the ORG to be assigned consecutive relocatable addresses starting with 0 or, if an offset is given, with the octal address given. Following occurrences of an ORG with the same section name causes resumption of addressing wherever it left off before, because of an intervening ORG.

The ORG macro also defines a symbol with the same name as the section at the first relative location within the section. Every invocation of ORG also defines (or redefines) the section to be returned to by the macro UNORG (see below).

DEFORG section

The DEFORG macro is the same as the ORG macro except that the symbol at relative 0 (the section name) is declared as a global symbol. By convention, the module that defines the section (rather than just uses it) issues the DEFORG macro.

TMPORG section[,offset]

TMPORG is the same as ORG except that it does not define (or redefine) the section to be reentered by the UNORG macro. In this way, the module can temporarily enter a new section and then return to the main section using UNORG without having to know the main section name.

UNORG The UNORG macro will reenter the section most recently declared in an ORG or DEFORG macro.

INCLUDE name1[,name2,...]

The INCLUDE macro indicates that the module issuing the INCLUDE requires the named modules (name1, and so forth). The name(s) should be declared with DEFORG(s) in the required modules.

INCLUDE declares the listed section names as global symbols and issues the macro directive .SBTTL with the heading "INCLUDE FROM LIBRARY 'name' " to be inserted in the assembly listing table of contents. INCLUDE documents the named sections as required by this section.

.DSECT [start][,cref]

The .DSECT macro starts a dummy program section (with the MACRO directive .ASECT) at relocatable address 0 or at the address given by the optional argument start. If the cross-reference (cref) parameter is given (nonblank), the program section is included in the cross-reference listing, if you request one for the assembly.

.DSECT (continued)

The .DSECT macro is used in the file COMMON.MAC to define the system parameters and offsets.

For example, coding of this form is used in COMMON.MAC to assign the proper values to the mnemonics in the pseudo-vector region:

```
.DSECT    177776,NOCREF
          .BLKW    -1
P.SIZE:   .BLKW    -1
P.2CC:    .BLKW    -1
P.CC:     .BLKW    -1
    .
    .
    .
```

.BSECT

[HIGH][,cref]

The .BSECT macro is like the .DSECT macro except that the default starting address is 1 instead of 0. If the argument HIGH is used, the starting address is 400_8. This starting address lets you use .BSECT to generate bit values. The .BSECT macro is used in COMMON.MAC to define mnemonics for bit locations. For example, the following coding assigns the mnemonics to the bit locations in the keyword (KEY; see Section 2.4). Note that the period (.) after .BLKB is required.

```
.BSECT     HIGH,NOCREF
JFSPRI:    .BLKB   .
JFPP:      .BLKB   .
    .
    .
    .
```

NOTE

A .BSECT is used at the end of the file COMMON.MAC. This means that you must explicitly start your MACRO program with an ORG macro or .PSECT directive to begin your program at relocatable address 0. Otherwise, your code will be regarded as a continuation of the .BSECT, and the program will not assemble properly.

.EQUATE

symbol,value

.EQUATE defines the given symbol to have the supplied value (which may be an expression) by using the equivalent of:

```
.DSECT    value
.symbol:
UNORG
```

.BLKW0 [amount][,value]

The .BLKW0 macro is similar to the MACRO directive .BLKW, which reserves a specified number of words of storage space. The default for amount is 1, but amount can be any expression. While .BLKW just reserves space, .BLKW0 fills the space with zeros (by default) or the value you specify.

.BLKB0 [amount][,value]

The .BLKB0 macro is similar to the MACRO directive .BLKB, which reserves a specified number of bytes of storage space. As with .BLKW0, the default for amount is 1, but amount can be any expression. .BLKB0 fills the space you reserve with zeros (by default) or the value you specify.

GLOBAL <name1[,name2,...]>

GLOBAL declares the name symbols as external global symbols.

RETURN [register]

The RETURN macro generates an RTS PC by default but can generate any other RTS instruction by specifying an explicit register.

JMPX label

JMPX is just like the JMP instruction but will also declare the label to be an external global (that is, jump external).

CALL subroutine[,register[,argument list]]

You can use CALL instead of JSR PC to call subroutines. If an explicit register is specified, then the call is JSR on that register. If an argument list is specified, it generates a list of .WORD arguments inline with the subroutine call.

CALLR subroutine

CALLR is equivalent to a CALL to a subroutine immediately followed by a RETURN. CALLR generates a JMP instruction.

CALLX subroutine

CALLX is just like CALL, but it also declares the subroutine name as an external global symbol.

CALLRX

CALLRX is just like CALLR except that the subroutine name is declared as an external global symbol.

3.1.3 Error Mnemonics: Link-File ERR.STB

When the monitor processes the directives described in this chapter, any errors that it detects are passed back to the job in the first byte of the FIRQB as a binary value. The ERR.STB file, provided with all RSTS/E kits, relates mnemonic values to these binary codes, so that you do not have to analyze and process errors in octal. The descriptions in this chapter all refer to the mnemonics provided by ERR.STB. (Appendix A lists all possible errors, the ERR.STB mnemonics, and the error text produced by the BASIC–PLUS run-time system when these errors occur.)

The symbols are automatically resolved at link time if you include ERR.STB with the files you link with either TKB (the Task Builder for the RSX run-time system) or LINK (the linker for the RT11 run-time system). For example, under the RSX run-time system, the following code would link ERR.STB and MAIN.OBJ to produce the executable file IMG1.TSK, a memory allocation file MP1.MAP, and a symbol definition file SF1.STB:

```
RUN $TKB
TKB>IMG1,MP1,SF1=ERR,STB,MAIN
```

Similarly, under the RT11 run-time system, the following sequence would link ERR.STB and MAIN.OBJ to produce the executable file IMG1.SAV, a memory allocation file MP1.MAP, and a symbol definition file SF1.STB:

```
RUN $LINK
*IMG1,MP1,SF1=ERR,STB,MAIN
```

3.1.4 Programming Hints

Preset the FIRQB and XRB to 0

The monitor directives in this chapter pass information to the monitor in the FIRQB and XRB areas of the low 1000 bytes of memory. It is usually a good idea to clear the entire FIRQB and XRB before issuing a call, to ensure that no extraneous information has been left in the areas (from data returned on a call, for example) that could affect how the call executes. In some cases, however, you may want to leave the FIRQB and XRB alone. The .FSS call, for example, scans a string and, if it is a valid file specification, returns to the FIRQB the information needed to open the file with the CALFIP call. You would not want to clear the FIRQB before opening the file with CALFIP.

Note that to ensure compatibility with future versions of RSTS/E, you should always set to zero any fields in the FIRQB and XRB diagrams that are shaded or are documented as reserved or not used.

The following example contains three routines that clear the FIRQB and XRB:

CLRFQX Clears both the FIRQB and XRB.

CLRFQB Clears the FIRQB.

CLRXRB Clears the XRB.

The values FQBSIZ and XRBSIZ used in these routines are defined in COMMON.MAC.

```
        .ENABL  LSB

CLRFQX::PUSH    <R0,R1>                 ;Save R0,R1
        MOV     #FIRQB,R0               ;Point to FIRQB.
        MOV     #<<FQBSIZ+XRBSIZ>/2>,R1 ;Compute how many words to clear.
        BR      10$

CLRFQB::PUSH    <R0,R1>                 ;Save R0,R1
        MOV     #FIRQB,R0               ;Point to FIRQB.
        MOV     #<FQBSIZ/2>,R1          ;Compute how many words to clear.
        BR      10$

CLRXRB::PUSH    <R0,R1>                 ;Save R0,R1
        MOV     #XRB,R0                 ;Point to XRB
        MOV     #<XRBSIZ/2>,R1          ;Compute how many words to clear.

10$:    CLR     (R0)+                   ;Zero it out ...
        SOB     R1,10$                  ;'til all done
        POP     <R1,R0>                 ;Restore R0,R1.
        RETURN

        .DSABL  LSB
```

Data Returned to FIRQB and XRB

If a call completes without error, the monitor sets byte 0 of the FIRQB to zero. If an error occurs on a call, the monitor sets byte 0 of the FIRQB to an error code. Likewise, the monitor always sets the byte at FIRQB+2 to the current job number times two when a call completes.

In some circumstances, it may be useful to know what happens to the "data passed" when a call completes. (Is the file name still there? and so forth.) Bytes not specified as containing "Data Returned" are undefined. You should not rely on these values when coding your programs because DIGITAL reserves the right to change the values returned in these bytes at any time. In addition, if an error occurs, the data returned may or may not have replaced the data passed. It depends on how far processing for the call got before the error occurred.

Channel Numbers for I/O

Directives that handle input/output use a "channel number" to refer to a device. In device or file opens, a channel number is related to a specific device defined in the call. Directives that transfer data (.READ, .WRITE, .MESAG) can then refer to a channel number rather than define a device or file.

Valid channel numbers range from 0 through 15_{10}. Channel 0 is the job's terminal; for example, a .WRITE to channel 0 will write to the terminal which is running the job. Channel 0 is always open. Similarly, the monitor opens a file to be run on channel 15_{10} when control transfers to the P.RUN entry point in a run-time system. Thus, user jobs may define and use channels 1 – 14_{10}.

Directives That Do I/O

The CALFIP subfunctions OPNFQ, CREFQ, CRBFQ, and CRTFQ open a file or device and relate the specified channel number to that file or device. OPNFQ opens a file or device for input; CREFQ "creates" a file, that is, opens a file or device for output. CRBFQ creates a binary (executable) output file on disk, and CRTFQ creates a temporary file on disk.

The directives .READ and .WRITE transfer data between memory and a device or file specified by channel number.

The CLSFQ (close) and RSTFQ (reset) subfunctions of CALFIP close a device or file and free the associated channel number so it can be used with another device or file.

Directives That Support I/O

The "file string scan" (.FSS) directive is useful for programs that process files specified by a terminal user. The .FSS directive examines a string of characters and, if it is a valid RSTS/E file specification, converts it to the FIRQB format used to open a file. Thus, your program can accept a typed string from the job's terminal and use .FSS to convert the string to the FIRQB format to do I/O on the file.

The .LOGS directive examines a string and, if it is a system logical device name defined by the system manager, returns the actual device name and unit number that corresponds to the logical name.

You can use the LOKFQ subfunction of CALFIP to search for disk files that meet "wildcard" file specifications. For example, you could search an account on disk for all files with names beginning with the characters DD.

3.2 CALFIP — Call the File Processor

Form

```
CALFIP
```

Function

The CALFIP directive to the RSTS/E monitor handles "housekeeping" necessary for input/output on RSTS/E. For example, CALFIP allows you to open a channel for file or device I/O.

You select the particular function desired by setting a function field in the FIRQB (at offset FQFUN). Other parameters are also passed to the monitor in the FIRQB, depending on the function requested. The functions are described in the following subsections; a summary is given below:

FQFUN Value (Octal)	Mnemonic	Action Performed (BASIC–PLUS Equivalent)
0	CLSFQ	Close an open channel (CLOSE)
2	OPNFQ	Open a channel (OPEN FOR INPUT)
4	CREFQ	Create/extend/open a channel (file-structured OPEN FOR OUTPUT)
6	DLNFQ	Delete a file by name (KILL)
10	RENFQ	Rename a file (NAME...AS)
12	DIRFQ	Get directory information
14	UUOFQ	Process UUO
16	ERRFQ	Get error message text
20	RSTFQ	Reset (close) all channels (except channel 0)
22	LOKFQ	Look up a file
24	ASSFQ	Assign a device
26	DEAFQ	Deassign a device
30	DALFQ	Deassign all devices
32	CRTFQ	Create/extend/open a temporary file on disk (file-structured OPEN FOR OUTPUT, space deallocated on close)
34	CRBFQ	Create/extend/open a compiled image file on disk (file-structured OPEN FOR OUTPUT, protection code bit 6 always set)

3.2.1 ASSFQ (Assign a Device) – Privileged and Not Privileged

Form

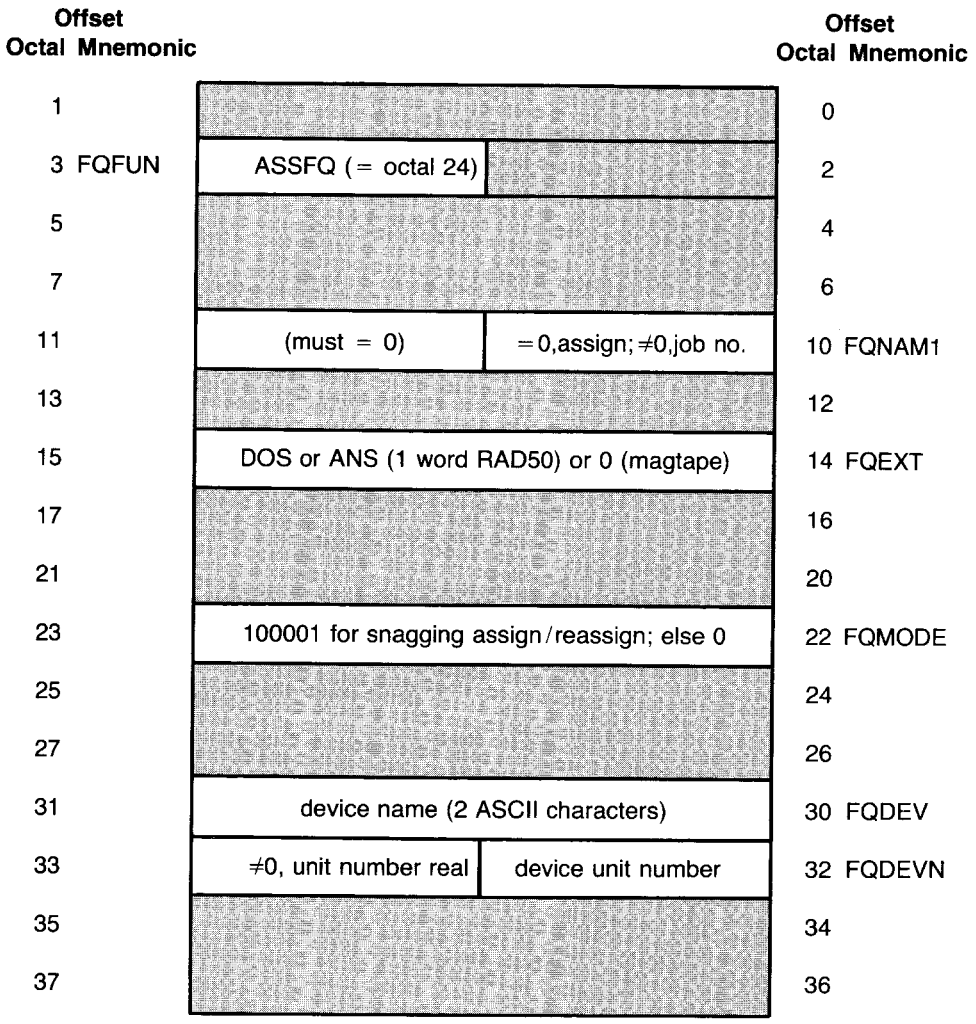
```
MOVB #ASSFQ,FIRQB+FQFUN
   .
   .
   .
(Set up FIRQB to define device)
   .
   .
   .
CALFIP
```

Function

The ASSFQ subfunction reserves a physical device for a job or transfers assignment of a currently owned device to another job.

Data Passed

FIRQB



FIRQB+FQFUN	The function code ASSFQ (octal value = 24).
FIRQB+FQNAM1	This byte is set to 0 to indicate an assign; if nonzero, it is used as the job number to which the device is to be reassigned. The high byte of this word (FIRQB+11) must be set to 0. If you are nonprivileged, you can reassign a device only to a job that is logged in to the same account as your current account.
FIRQB+FQEXT	When the device is magnetic tape, this word can contain either DOS or ANS in RAD50 format, to indicate DOS or ANSI label format for the tape drive. Or it can be set to 0 to indicate the system default for the drive.
FIRQB+FQMODE	This word is set to 100001 to assign a device that is currently assigned to another user. This is a "snagging" assign and is available only to privileged users. If you do not want a snagging assign, this word must be zero.
FIRQB+FQDEV	Device name, as two ASCII characters.
FIRQB+FQDEVN	The device unit number is passed here in binary. A nonzero value in FIRQB+FQDEVN+1 indicates an explicit device unit number. A zero value in FIRQB+FQDEVN+1 indicates no unit number.

Data Returned

Other than a possible error in byte 0 of the FIRQB, no data is returned by the ASSFQ subfunction.

Errors

For Assign (byte at FIRQB+FQNAM1 = 0):

NODEVC	The device name specified at FIRQB+FQDEV is not a valid device name.
NOTAVL	The device and unit specified exists on the system, but the attempt to reserve it is prohibited because: 1. The device is currently reserved by another job. 2. Ownership of the device requires privilege that the user does not have. For example, a nonprivileged user tried to assign a device that is currently assigned to another user.

NOTAVL (cont.)

3. The device or its controller has been disabled by the system manager.
4. The device is a keyboard line for a pseudo keyboard only.

PRVIOL — The device requires privilege for an assign.

For Reassign (byte at FIRQB+10 ≠ 0):

INUSE — The device specified is currently open or has an open file.

NODEVC — The device name is a logical device name for which a physical device is not currently assigned.

NOTAVL — (See description for Assign, above.)

BDNERR — The job number specified does not exist.

PRVIOL — You are not privileged and tried to reassign a device to a job that is logged in to an account other than your current account.

Example

The following code reassigns magnetic tape unit 0 (MT0:) to job 12_{10}. See Section 3.1.4 for information on the CLRFQB routine.

```
MAGT:   .ASCII  /MT/
        CALL    CLRFQB                  ;CALL ROUTINE TO CLEAR FIRQB
        MOVB    #ASSFQ,FIRQB+FQFUN      ;SET FUNCTION CODE
        MOVB    #12.,FIRQB+FQNAM1       ;ASSIGN TO JOB 12
        MOV     #^RANS,FIRQB+14         ;ANSI-LABEL TAPE
        MOV     MAGT,FIRQB+FQDEV        ;MAGTAPE DEVICE
        CLRB    FIRQB+FQDEVN            ;UNIT NO. 0
        MOVB    #377,FIRQB+FQDEVN+1     ;UNIT NO. REAL
        CALFIP
        TSTB    FIRQB                   ;ANY ERRORS?
        BNE     ERRTN                   ;BRANCH TO PROCESS ERROR
```

3.2.2 CLSFQ (Close a Channel) – Not Privileged

Form/Example

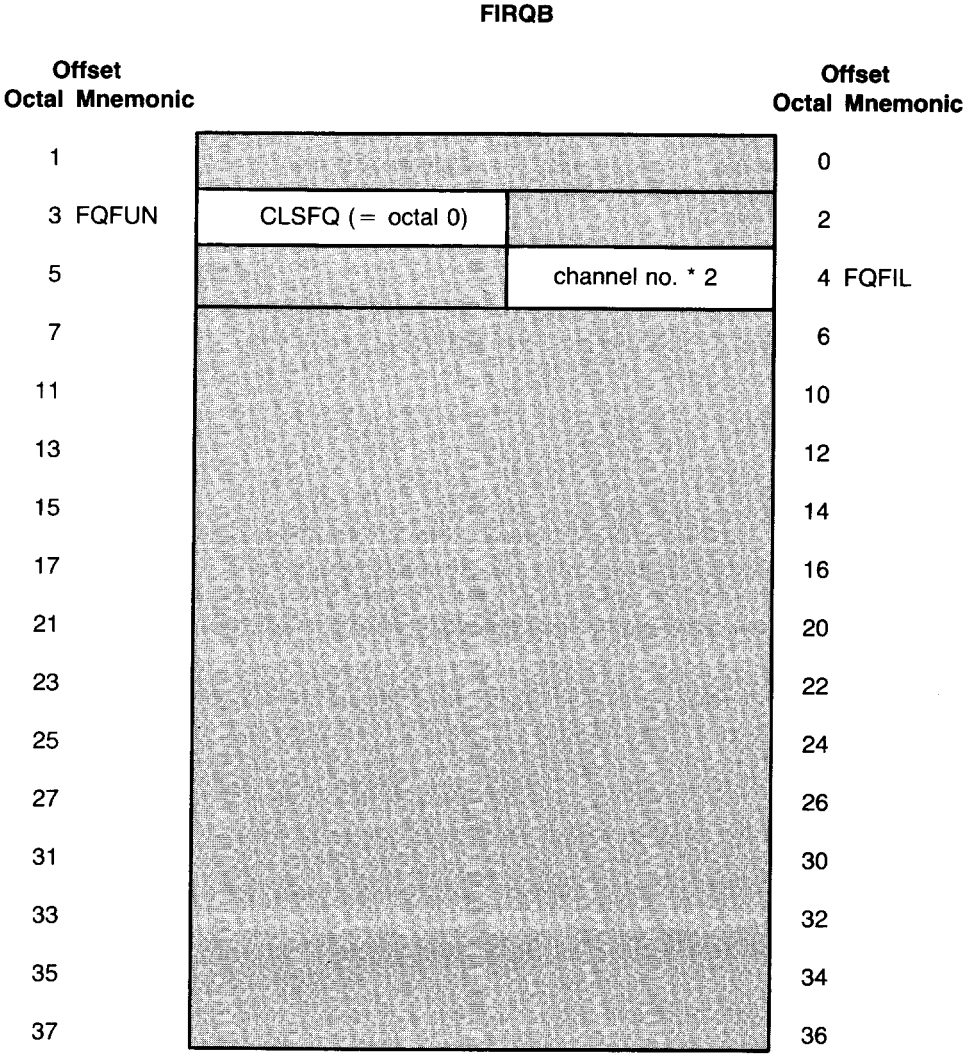
```
CHANO=8.                          ;SET VALUE FOR CHANNEL
MOVB    #CLSFQ,FIRQB+FQFUN        ;SET FUNCTION CODE IN FIRQB
MOVB    #CHANO*2,FIRQB+FQFIL      ;SET CHANNEL 8 FOR CLOSE
CALFIP                            ;EXECUTE MONITOR DIRECTIVE
```

Function

The CLSFQ function closes a channel. The specific action taken depends on the device or file that was previously opened on the channel, whether it was opened for input or output, and the mode with which it was opened. For example, closing a channel on which a magnetic tape was opened for input in file-structured mode will cause the monitor to position the tape at the end-of-file. (Actions taken on closing various devices/files are described in the *RSTS/E Programming Manual.)*

Requesting CLSFQ for a channel that is not currently open will simply return with no error indicated.

Data Passed



FIRQB + FQFUN The function code CLSFQ (octal value = 0).

FIRQB + FQFIL Channel number times 2; defines the channel to be closed.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the CLSFQ function of CALFIP.

Errors

All possible errors with the CLSFQ function of CALFIP are device-dependent; see Appendix A for a full list of errors.

Example

The following MACRO code closes the file or device on channel 12_{10}. See Section 3.1.4 for information on the CLRFQB routine.

```
CALL        CLRFQB                ;CLEAR FIRQB
MOVB        #CLSFQ,FIRQB+FQFUN    ;SET FUNCTION CODE IN FIRQB
MOVB        #12.*2,FIRQB+FQFIL    ;SET CHANNEL 12 FOR CLOSE
CALFIP                            ;EXECUTE MONITOR DIRECTIVE
TSTB        FIRQB                 ;TEST BYTE 0 FOR ERROR
BNE         ERRTN                 ;BRANCH TO PROCESS ERROR
```

3.2.3 CRBFQ (Create a Binary [Executable] File and Open It on a Channel) – Not Privileged

Form

```
MOVB #CRBFQ,FIRQB+FQFUN     ;SET FUNCTION CODE
   .
   .
   .
(Set parameters in FIRQB appropriate to device)
   .
   .
   .
CALFIP
```

Function

The CRBFQ function creates and opens a binary (executable) file. It is identical to the CREFQ function (Section 3.2.4), except that (1) the protection code is automatically set to indicate an executable file, and (2) the file must be opened on a disk device.

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	CRBFQ (= octal 34)		2	
5	FQSIZM	(must = 0)	channel number * 2	4	FQFIL
7		project number	programmer number	6	FQPPN
11		file name (2 words in RAD50 format)		10	FQNAM1
13				12	
15		file type (1 word in RAD50 format)		14	FQEXT
17		file size (in 512_{10}-byte blocks)		16	FQSIZ
21				20	
23		mode		22	FQMODE
25				24	
27	FQPROT	file protection	≠0,prot. code real	26	FQPFLG
31		device name (2 ASCII characters)		20	FQDEV
33		≠0, unit number real	device unit number	32	FQDEVN
35		file cluster size		34	FQCLUS
37		device cluster number for first block		36	FQNENT

FIRQB+FQFUN The function code CRBFQ (octal value = 34).

FIRQB+FQFIL Channel number times 2; defines the channel upon which the file is to be opened.

FIRQB+FQSIZM On other types of "create" opens, this byte contains the most significant bits of the file size. Executable (binary) files cannot be greater than $65{,}535_{10}$ blocks, however, so this byte must always be passed as zero.

FIRQB+FQPPN The project-programmer number (ppn) with which the file is to be created. The project number is in the high byte (FQPPN+1) and the programmer number in the low byte (FQPPN). A value of 0 in both bytes defaults to the ppn under which the calling program is running.

FIRQB+FQNAM1 The file name created, as two words of RAD50 data.

FIRQB+FQEXT The file type, as one word of RAD50 data.

FIRQB+FQSIZ The desired file size, in 512_{10}–byte blocks. The file is preextended to the specified size; that is, the space for the file is allocated when the file is opened, rather than as it is written.

FIRQB+FQMODE The mode with which the file is to be opened; values and actions taken are as described for the MODE modifier in file-structured OPEN FOR OUTPUT statements for disk, as described in the *RSTS/E Programming Manual.* If a mode value is used, bit 15_{10} of this word must be set to 1.

FIRQB+FQPROT File protection code; values for this field define read/write and execute access to the created file (see *RSTS/E System User's Guide*). If you want a default protection code, then set a full word of zeros at FIRQB+FQPFLG. In this case, either the system default for protection code will be used, or, if the CRBFQ will be deleting a previously existing file with the same file name, type, ppn, and device, the file protection code of the previously existing file will be used.

To assign a specific file protection code, a nonzero value is passed in byte FIRQB+FQPFLG (by convention, 377_8) and the specific file protection code in byte FIRQB+FQPROT. Bit 6 is automatically set, indicating that the file is executable. Meanings for protection codes for executable files are described in the *RSTS/E System User's Guide.*

FIRQB+FQDEV The device name is passed here as two ASCII characters; it must be a disk device. If this word is 0, the public disk structure is assumed.

FIRQB+FQDEVN The device unit number is passed here in binary. A nonzero value in FQDEVN+1 indicates an explicit device unit number. A zero value in FQDEVN+1 indicates no unit number.

FIRQB+FQCLUS This parameter has the same function as the CLUSTERSIZE option in BASIC–PLUS. A description of the CLUSTERSIZE option for disk is given in the *BASIC–PLUS Language Manual.*

FIRQB+FQNENT Device cluster number for placement of block 1 of the file. When you are creating a new file, you can place block 1 of the file on a particular block by specifying the disk device cluster number in this word. If this word is zero, no placement is done. If it is nonzero, the monitor will try to place the file at the specified device cluster or as near after it as possible. If the first block of the file can be placed at or after the specified device cluster number, the monitor sets a bit in the file's entry in the User File Directory (an internal monitor directory). If the first block of the file cannot be placed at or after the specified device cluster number, the file is placed at the lowest free block on the disk, the bit in the file's entry in the User File Directory is not set, and no error is returned.

Data Returned

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job number * 2	2	FQJOB
5	FQSIZM	(always 0)	channel number * 2	4	FQFIL
7		project number	programmer number	6	FQPPN
11		file name (2 words in RAD50 format)		10	FQNAM1
13				12	
15		file type (1 word in RAD50 format)		14	FQEXT
17		size of the file, in 512_{10}–byte blocks		16	FQSIZ
21		reasonable buffer size for device		20	FQBUFL
23		(as passed)		22	FQMODE
25		device description		24	FQFLAG
27	FQPROT	protection code	clustersize, mod256	26	FQPFLG
31		device name (2 ASCII characters)		30	FQDEV
33		flag byte	device unit number	32	FQDEVN
35		file identification index		34	FQCLUS
37		(as passed)		36	FQNENT

FIRQB + FQJOB	Current job number times two.
FIRQB + FQFIL	Channel number times two; defines the channel on which the file is open.
FIRQB + FQPPN	The project-programmer number under which the file is open. An actual project-programmer number is returned here even if this word was passed as 0.
FIRQB + FQNAM1	The file name created, as two words of RAD50 data.
FIRQB + FQEXT	The file type created, as one word of RAD50 data.
FIRQB + FQSIZ	The size to which the file was preextended, in 512_{10}–byte blocks.
FIRQB + FQBUFL	Reasonable buffer size for disk reads and writes, in bytes. (Always 512_{10} for disk.)
FIRQB + FQFLAG	Description of the device just opened (the same information as the BASIC–PLUS STATUS variable). The low byte contains the device's handler index, always 0 (DSKHND) for disk. The high byte contains a set of status flags, irrelevant here since the device is always disk. (See the OPNFQ subfunction if you are interested in these settings.)
FIRQB + FQPFLG	The file cluster size, modulo 256. That is, a file cluster size of 256_{10} is indicated by a zero byte here. This is the same as the value passed at FIRQB + FQCLUS, except that it is returned in a byte instead of a word.
FIRQB + FQPROT	The protection code of the file. Bit 7 is 0, bit 6 is 1, and bits 5 – 0 are as passed.
FIRQB + FQDEV	The device name of the disk device, as two ASCII characters. The actual device name is returned here, even if this word was passed as 0.
FIRQB + FQDEVN	The device unit number. The actual unit number is returned here, even if FIRQB + FQDEVN + 1 was passed as 0.
FIRQB + FQCLUS	The file identification index of this file. This word is significant mainly in that it can be used in place of the file name in subsequent opens of the file on disk. You can open the file with the OPNFQ subfunction of CALFIP using an explicit project-programmer number in FIRQB + FQPPN, a zero word in FIRQB + FQNAM1, an explicit device name in FQDEV and FQDEVN, and the file identification index in FIRQB + FQNAM1 + 2.

FIRQB+FQCLUS (cont.)

Note that there is no performance gain for using the file identification index rather than the file name. The file identification index is provided for compatibility with RSX. Furthermore, the file identification index is changed when the REORDR utility is run (see the *RSTS/E System Manager's Guide*).

Errors

NOTCLS	The specified channel is already open. It must be closed before it can be opened again.
PRVIOL	The specified device is not a disk device. The CRBFQ function can be executed only for a disk device.
xxxxx	Other errors are device-dependent. See Appendix A for a full list of possible error codes.

Example

The following MACRO code sets up the FIRQB for the CRBFQ function of CALFIP. The ppn is set to 2,210; the file name and type are set to FILNAM.TYP. The protection code is set such that the file is read/write-protected against everyone but the caller (user with ppn 2,210), and execute-protected against all but the caller and those in the caller's project (users with project number=2). The file is opened on disk unit 2 (DK2:). File size and cluster size are not specified. The cluster size defaults to the pack cluster size and the file size is not preallocated.

See Section 3.1.4 for information on the CLRFQB routine used in this example.

```
DK:   .ASCII   /DK/
      CALL     CLRFQB                        ;CLEAR FIRQB
      MOVB     #CRBFQ,FIRQB+FQFUN            ;SET FUNCTION CODE
      MOVB     #4*2,FIRQB+FQFIL              ;SET CHANNEL = 4
      MOVB     #2,FIRQB+FQPPN+1              ;SET PROJECT NUMBER =2
      MOVB     #210.,FIRQB+FQPPN             ;SET PROG. NO.=210.
      MOV      #^RFIL,FIRQB+FQNAM1           ;SET FILE NAME AND
      MOV      #^RNAM,FIRQB+FQNAM1+2         ;TYPE TO
      MOV      #^RTYP,FIRQB+FQEXT            ;"FILNAM.TYP"
      MOVB     #<8.+16.+32.>,FIRQB+FQPROT    ;SET PROTECTION CODE
      MOVB     #377,FIRQB+FQPFLG             ;SET PROTECTION CODE REAL
      MOV      DK,FIRQB+FQDEV                ;SET DEVICE TO DISK,
      MOVB     #2,FIRQB+FQDEVN               ;UNIT 2
      MOVB     #377,FIRQB+FQDEVN+1           ;(EXPLICIT DEVICE NO.)
      CALFIP
```

3.2.4 CREFQ (Create a File and Open It on a Channel) – Not Privileged

Form

```
MOVB    #CREFQ,FIRQB+FQFUN    ;SET FUNCTION CODE
  .
  .
  .
(set parameters appropriate to device)
  .
  .
  .
CALFIP
```

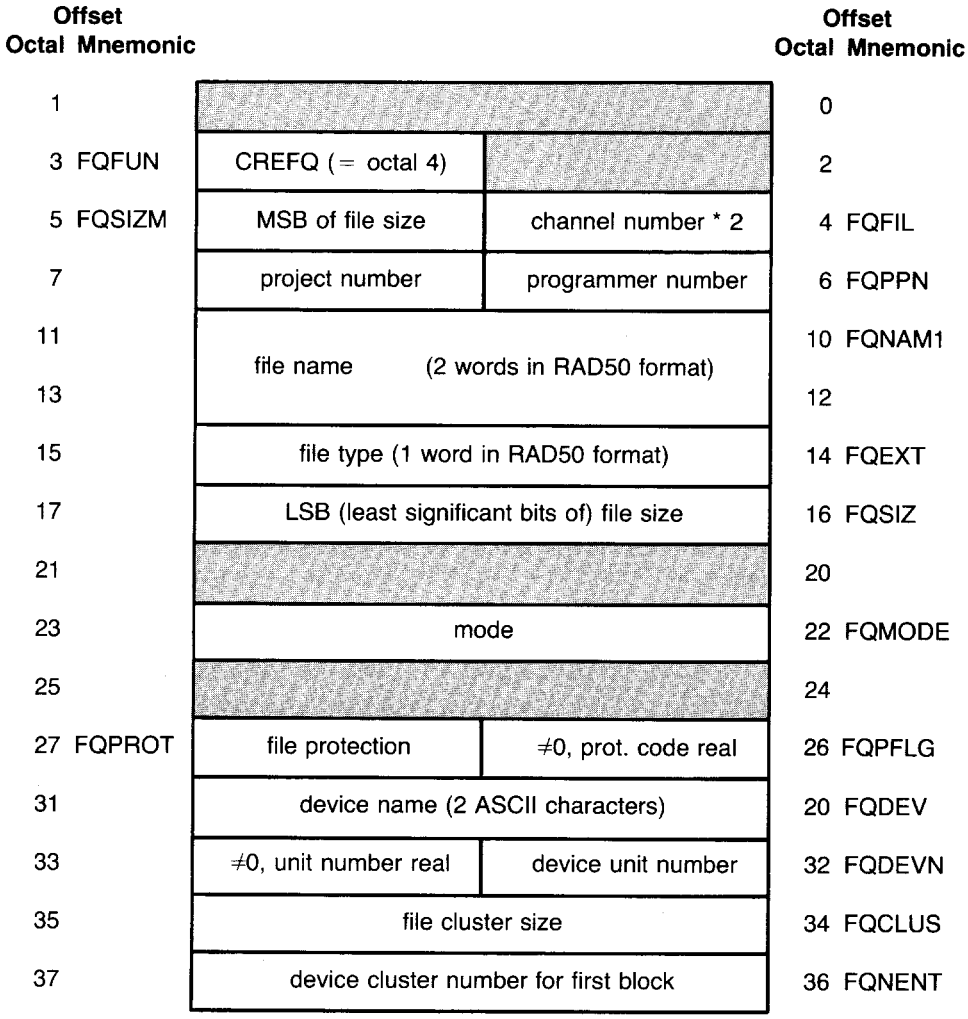
Function

The CREFQ function performs the same action as a file-structured OPEN FOR OUTPUT statement in BASIC–PLUS. Parameters defining the device, file name and type, protection code, mode, file size, and cluster size can be used by setting values in the FIRQB. The choice depends upon the device.

For example, CREFQ with a file name and type on a magtape device with mode = 128_{10} would cause an "open for append" operation. A search for an existing file with the specified name and on the specified device would be made; the file would have to be the last file on the tape. When found, the tape would be positioned after the last record in the file, ready for data to be written and appended. The *RSTS/E Programming Manual* describes the file-structured OPEN FOR OUTPUT operation for the various devices.

Data Passed

FIRQB



FIRQB+FQFUN The function code CREFQ (octal value = 4).

FIRQB+FQFIL Channel number times 2; defines the channel upon which the file is to be opened.

FIRQB+FQSIZM For large disk files (greater than 65,535 blocks), this byte contains the most significant bits of the file size. See FIRQB+FQSIZ, below, for a discussion of the entire 24–bit field used for large files on disk.

FIRQB+FQPPN The project-programmer number (ppn) with which the file is to be created. The project number is in the high byte (FIRQB+FQPPN+1), and the programmer number in the low byte (FIRQB+FQPPN). A value of 0 defaults to the ppn under which the calling program is running.

FIRQB + FQNAM1 The file name to create, as two words of RAD50 data.

FIRQB + FQEXT The file type, as one word of RAD50 data.

FIRQB + FQSIZ The desired file size, in 512_{10}–byte blocks. This parameter is relevant only for disk and ANSI magtape files. For disk files, this word forms the least significant bits of the file size. It is combined with the byte at FIRQB + FQSIZM to indicate the file size. The disk file is automatically preextended to the indicated size. (That is, the space for the file is allocated when the file is opened, not as it is written.)

For ANSI magtape, this word has the same function as the FILESIZE option in BASIC–PLUS, as described in the *RSTS/E Programming Manual.*

FIRQB + FQMODE The mode with which the file is to be opened; values and actions taken for specific devices are as described for the MODE modifier for file-structured OPEN FOR OUTPUT statements in the *RSTS/E Programming Manual.* If a mode value is used, bit 15 of this word must be set to 1.

FIRQB + FQPROT File protection code; values for this field define subsequent read and write access to the opened file (see the *RSTS/E System User's Guide*). If you want a default protection code, then set a full word of zeros at FIRQB + FQPFLG. In this case, either the system default for protection code will be used, or, if the CREFQ will be deleting a previously existing file with the same file name, type, ppn, and device, the file protection code of the previously existing file will be used.

To assign a specific file protection code, put a nonzero value in byte FIRQB + FQPFLG (by convention, 377_8) and the specific file protection code in byte FIRQB + FQPROT. This allows an explicit file protection code of 0. Bit 6 is always cleared by the CREFQ function, regardless of whether or not it is set in FIRQB + FQPROT. Bit 6 is the "compiled file" bit; compiled files should be opened with the CRBFQ function (Section 3.2.3).

FIRQB + FQDEV The device name is passed here as two ASCII characters. A zero word indicates the public disk structure.

FIRQB + FQDEVN The device unit number is passed here in binary. A nonzero value in FIRQB + FQDEVN + 1 indicates an explicit device unit number. A zero value in FIRQB + FQDEVN + 1 indicates no unit number.

FIRQB+FQCLUS This parameter has the same function as the CLUSTERSIZE option in BASIC–PLUS. It is relevant only for disk files and ANSI magtape files. A description of the CLUSTERSIZE option for disk is given in the *BASIC–PLUS Language Manual*; for ANSI magtape, in the *RSTS/E Programming Manual.*

FIRQB+FQNENT For disk files, the device cluster number for placement of block 1 of the file. When creating a new file, you can place block 1 of the file on a particular block by specifying the disk device cluster number in this word. If this word is zero, no placement is done. If it is nonzero, the monitor will try to place the file at the specified device cluster or as near after it as possible. If the first block of the file can be placed at or after the specified device cluster number, the monitor sets a bit in the file's entry in the User File Directory (an internal monitor directory). If the first block of the file cannot be placed at or after the specified device cluster number, the file is placed at the lowest free block on the disk, the bit in the file's entry in the User File Directory is not set, and no error is returned.

Data Returned

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job number * 2	2	FQJOB
5	FQSIZM	MSB of file size	channel number * 2	4	FQFIL
7		project number	programmer number	6	FQPPN
11		file name (2 words in RAD50 format)		10	FQNAM1
13				12	
15		file type (1 word in RAD50 format)		14	FQEXT
17		LSB (least significant bits) of file size		16	FQSIZ
21		reasonable buffer size for device		20	FQBUFL
23		(as passed)		22	FQMODE
25		device description		24	FQFLAG
27	FQPROT	protection code	clustersize, mod256	26	FQPFLG
31		device name (2 ASCII characters)		30	FQDEV
33		flag byte	device unit number	32	FQDEVN
35		file identification index		34	FQCLUS
37		(as passed)		36	FQNENT

NOTE

For non-disk devices, the relevant information returned with the CREFQ subfunction is in the two words at FIRQB+FQBUFL and FIRQB+FQFLAG. All other words are simply returned as passed.

FIRQB+FQJOB — Current job number times two.

FIRQB+FQFIL — Channel number times two; defines the channel on which the file is open.

FIRQB+FQSIZM — For large files, this byte contains the most signficant bits of the size to which the file was preextended, in 512_{10}–byte blocks. This byte is combined with the word at FIRQB+FQSIZ to form a 24–bit field giving the file size.

FIRQB+FQPPN — The project-programmer number under which the file is open. An actual project-programmer number is returned here even if this word was passed as 0.

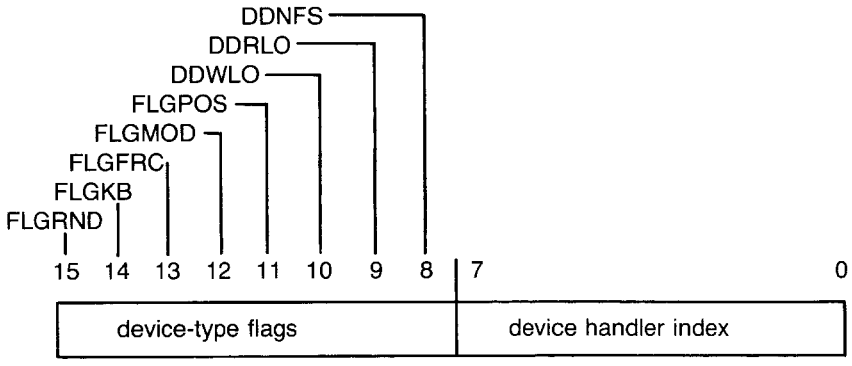
FIRQB+FQNAM1 — The file name created, as two words of RAD50 data.

FIRQB+FQEXT — The file type created, as one word of RAD50 data.

FIRQB+FQSIZ — The size to which the file was preextended, in 512_{10}–byte blocks.

FIRQB+FQBUFL — Reasonable buffer size for this device, in bytes. If you are doing device-independent I/O (that is, if you do not wish to keep track of which device is being opened and perform specific opens, reads, and writes, depending on the device), this value is the monitor's "best guess" for a buffer size to use in subsequent reads and writes on the opened channel. (See Sections 3.17 and 3.33 on .READ and .WRITE.)

FIRQB+FQFLAG — Description of the device just opened (same information as the BASIC–PLUS STATUS variable). The low byte contains the device's handler index. There is one unique handler index for all device types. The high byte contains a set of status flags to allow for device-independent I/O routines.



High Byte — Device-Type Flags

The bits in the high byte of the flags word are set to indicate the type of file or device just opened:

FLGRND = 1 The device or file is random-access.
= 0 The device or file is sequential.

FLGKB = 1 The file or device is a terminal-type file or device (or is generically a terminal).
= 0 The file or device is not a terminal-type file or device.

FLGFRC = 1 The file or device is byte-oriented. That is, the .READ and .WRITE directives handle data in byte units.
= 0 The file or device is block-oriented. The .READ and .WRITE directives handle data in block units.

FLGMOD = 1 The file or device accepts modifiers in .READ and .WRITE directives (Sections 3.17 and 3.33, XRB+XRMOD).
= 0 The file or device does not accept modifiers in .READ and .WRITE directives.

FLGPOS = 1 The file or device keeps track of its horizontal position and expands characters such as TAB into whatever is appropriate for the file or device. You can determine the current horizontal position with the .POSTN directive.
= 0 The file or device does not keep track of its horizontal position.

DDWLO = 1 The file or device has been write-locked (with the protection code or mode value in the open) or is generically a write-only device.
= 0 The file or device is not write-locked.

DDRLO = 1 The file or device has been read-locked (with the protection code in the open) or is generically a read-only device.
= 0 The file or device is not read-locked.

DDNFS = 1 The file or device is non-file-structured (or is generically not a file-structured device).
= 0 The file or device is file-structured.

Low Byte — Device Handler Index

Bits 0–7 of the flags word contain a handler index that indicates the generic kind of device. Currently defined values are:

Octal Value	Symbol	Meaning
0	DSKHND	All disks
2	TTYHND	All terminals
4	DTAHND	DECtape
6	LPTHND	All line printers
10	PTRHND	Paper tape reader
12	PTPHND	Paper tape punch
14	CDRHND	Card reader
16	MTAHND	Magtape
20	PKBHND	Pseudo keyboards
22	RXDHND	Flexible diskettes
24	RJEHND	2780 remote job entry
26	NULHND	The null device
30	DMCHND	The DMC11/DMR11 DDCMP interface
36	DT2HND	DECtape II
40	KMCHND	KMC11
42	IBMHND	IBM interconnect
46	DMPHND	DMP11/DMV11 device

FIRQB+FQPFLG The file cluster size, modulo 256. That is, a file cluster size of 256_{10} is indicated by a zero byte here. This is the same as the value passed at FIRQB+FQCLUS, except that it is returned in a byte instead of a word.

FIRQB+FQPROT The protection code of the file. Bits 6 and 7 are 0; bits 5 – 0 are as passed.

FIRQB+FQDEV The device name of the disk device, as two ASCII characters. The actual device name is returned here, even if this word was passed as 0.

FIRQB+FQDEVN The device unit number. The actual unit number is returned here, even if FIRQB+FQDEVN+1 was passed as 0.

FIRQB+FQCLUS The file identification index of this file. This word is significant mainly in that it can be used in place of the file name in subsequent opens of the file on disk. You can open the file with the OPNFQ subfunction of CALFIP using an explicit project-programmer number in FIRQB+FQPPN, a zero word in FIRQB+FQNAM1, and the file identification index in FIRQB+FQNAM1+2.

Note that there is no performance gain for using the file identification index rather than the file name. The file identification index is provided for compatibility with RSX. Furthermore, the file identification index is changed when the REORDR utility is run (see the *RSTS/E System Manager's Guide*).

Errors

NOTCLS The specified channel is already open. It must be closed before it can be opened again.

xxxxx Other errors are device-dependent. See Appendix A for a full list of errors.

Example

The following MACRO code sets up the FIRQB for a CREFQ that opens a file called FILE01.LST on the public disk structure. (We assume here that previous code has filled the FIRQB with zeros, so that the words at FIRQB+FQDEV and FIRQB+FQDEVN are zero, indicating the public disk. Similarly, the mode (FIRQB+FQMODE) and cluster size (FIRQB+FQCLUS) are zero, indicating normal read/write and the default cluster size.) A protection code of 56 is assigned (write-protected against all but the owner, read-protected against all but those in the owner's project).

```
MOVB      #CREFQ,FIRQB+FQFUN      ;SET FUNCTION CODE IN FIRQB
MOVB      #5*2,FIRQB+FQFIL        ;SET CHANNEL = 5
MOV       #^RFIL,FIRQB+FQNAM1     ;SET FILE NAME
MOV       #^RE01,FIRQB+FQNAM1+2   ;AND TYPE TO
MOV       #^RLST,FIRQB+FQEXT      ;FILE01.LST
MOVB      #56.,FIRQB+FQPROT       ;SET PROTECTION CODE
MOVB      #377,FIRQB+FQPFLG       ;EXPLICIT PROT. CODE
CALFIP
```

3.2.5 CRTFQ (Create and Open a Temporary File) – Not Privileged

Form

```
MOVB      #CRTFQ,FIRQB+FQFUN
  .
  .
  .
(set appropriate parameters)
  .
  .
  .
CALFIP
```

Function

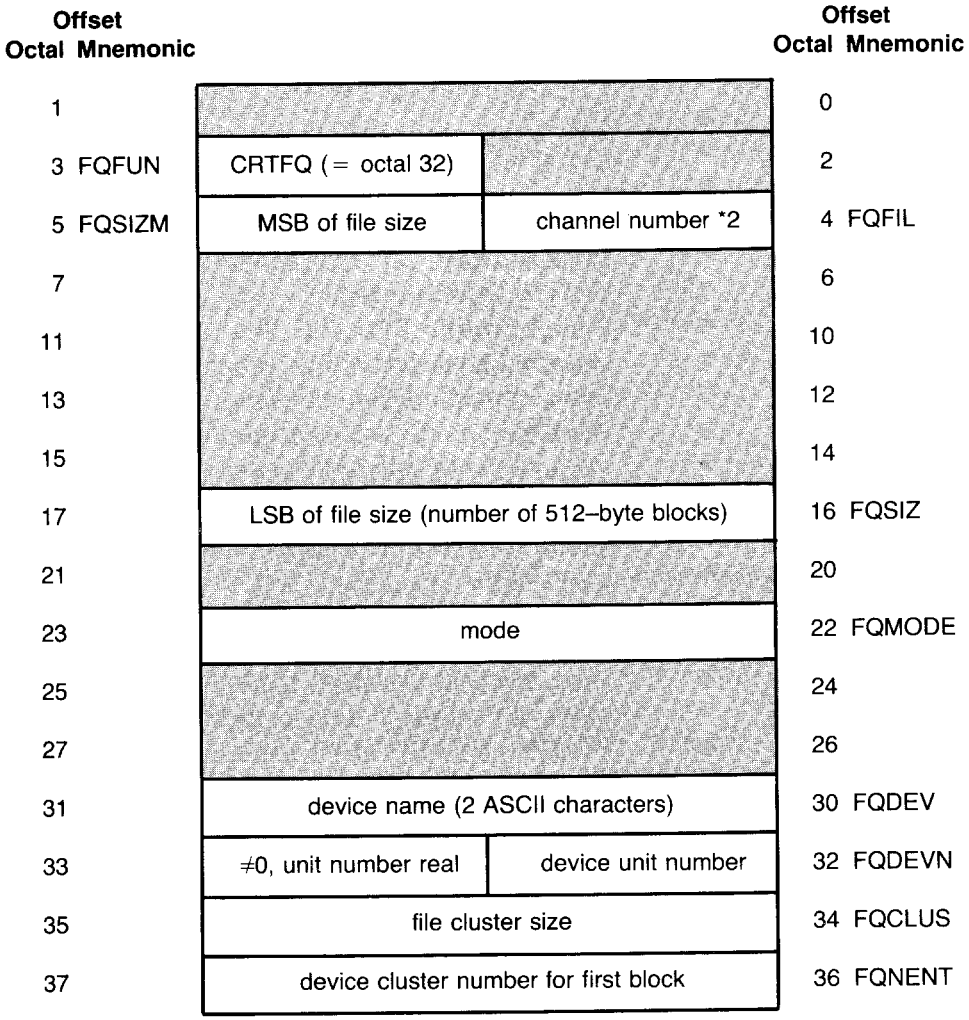
The CRTFQ function can be used to create and open a temporary file on disk. The file is "temporary" only in that the monitor generates a file name and type for the file, which is recognized by the LOGOUT utility. LOGOUT destroys such files when the user logs out; the monitor does not inherently destroy temporary files created with CRTFQ.

Most parameters relevant on an open are defaulted. You do not define or refer to the file by name; subsequent read and write operations refer to the channel on which the temporary file is open.

In addition to the function code, you specify a channel and, if desired, a file size and/or file cluster size. If an explicit cluster size is specified, it will be used. A specified file size may or may not be used. The monitor will attempt to reuse an existing temporary file for the same job by simply reopening that file. In this case, the file's size will be the size of the previous file. If no previous temporary file for the job exists, or if one does exist but is already in use by somebody else, then a new file will be created, with the specified file size. A new file will also be created if an explicit device name or cluster size is given.

Data Passed

FIRQB



FIRQB + FQFUN — The function code CRTFQ (octal value = 32).

FIRQB + FQFIL — Channel number times 2; defines the channel on which the file is to be opened.

FIRQB + FQSIZM — For large disk files (greater than $65{,}535_{10}$ blocks), this byte contains the most significant bits of the file size. See FIRQB + FQSIZ, below, for a description of how the entire 24_{10}–bit field is used.

FIRQB+FQSIZ — This word contains the least significant bits of the file's size in 512_{10}–byte blocks. (It is combined with the byte at FIRQB+FQSIZM.) The file size may or may not be used. If a temporary file already exists that is not in use, the monitor will use the space allocated to the previous temporary file. However, if cluster size is specified, a new file will be created, and the file size indicated by the 24–bit file size will be used.

FIRQB+FQMODE — The mode with which the file is to be opened. Values are as described for the MODE modifier in OPEN FOR OUTPUT statements for disk, as described in the *RSTS/E Programming Manual*. The only relevant modes are for creating a tentative file, creating a contiguous file, creating a conditionally contiguous file, and for data caching. All other mode bits are ignored, except bit 15. If a mode value is used, bit 15 of this word must be set to 1.

FIRQB+FQDEV — The device name is passed here as two ASCII characters; it must be a disk device. A value of 0 in this word indicates "SY", the public disk.

FIRQB+FQDEVN — The device unit number is passed here in binary. A nonzero value in the high byte (FIRQB+FQDEVN+1) indicates an explicit device unit number. A zero value in FIRQB+FQDEVN+1 indicates no unit number.

FIRQB+FQCLUS — File cluster size. Performs the same function as the CLUSTERSIZE option in BASIC–PLUS.

FIRQB+FQNENT — The device cluster number for placement of block 1 of the file on disk. When creating a new file, you can place block 1 of the file on a particular block by specifying the disk device cluster number in this word. If this word is zero, no placement is done. If it is nonzero, the monitor will try to place the file at the specified device cluster or as near after it as possible. If the first block of the file can be placed at or after the specified device cluster number, the monitor sets a bit in the file's entry in the User File Directory (an internal monitor directory). If the first block of the file cannot be placed at or after the specified device cluster number, the file is placed at the lowest free block on the disk, the bit in the file's entry in the User File Directory is not set, and no error is returned.

Data Returned

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job number * 2	2	FQJOB
5	FQSIZM	MSB of file size	channel number * 2	4	FQFIL
7		project number	programmer number	6	FQPPN
11		file name (2 words in RAD50 format)		10	FQNAM1
13				12	
15		file type (1 word in RAD50 format)		14	FQEXT
17		LSB (least significant bits) of file size		16	FQSIZ
21		reasonable buffer size for device		20	FQBUFL
23		(as passed)		22	FQMODE
25		device description		24	FQFLAG
27	FQPROT	protection code	clustersize, mod256	26	FQPFLG
31		device name (2 ASCII characters)		30	FQDEV
33		flag byte	device unit number	32	FQDEVN
35		file identification index		34	FQCLUS
37		(as passed)		36	FQNENT

FIRQB+FQJOB — Current job number times two.

FIRQB+FQFIL — Channel number times two; defines the channel on which the file is open.

FIRQB+FQSIZM — For large disk files (greater than $65{,}535_{10}$ blocks), this byte contains the most significant bits of the file size in 512_{10}–byte blocks. It is combined with the word at FIRQB+FQSIZ to form a 24–bit field giving the file size.

FIRQB+FQPPN — The project-programmer number under which the file is open. An actual project-programmer number is returned here even if this word was passed as 0.

FIRQB + FQNAM1	The file name created, as two words of RAD50 data.
FIRQB + FQEXT	The file type created, as one word of RAD50 data.
FIRQB + FQSIZ	The size to which the file was preextended, in 512_{10}–byte blocks.
FIRQB + FQBUFL	Reasonable buffer size for disk reads and writes, in bytes. (Always 512_{10} for disk.)
FIRQB + FQFLAG	Description of the device just opened (the same information as the BASIC–PLUS STATUS variable). The low byte contains the device's handler index, always 0 (DSKHND) for disk. The high byte contains a set of status flags, irrelevant here since the device is always disk. (See the OPNFQ subfunction if you are interested in these settings.)
FIRQB + FQPFLG	The file cluster size, modulo 256. That is, a file cluster size of 256_{10} is indicated by a zero byte here. This is the same as the value passed at FIRQB + FQCLUS, except that it is returned in a byte instead of a word.
FIRQB + FQPROT	The protection code of the file.
FIRQB + FQDEV	The device name of the disk device, as two ASCII characters. The actual device name is returned here, even if this word was passed as 0.
FIRQB + FQDEVN	The device unit number. The actual unit number is returned here, even if FIRQB + FQDEVN + 1 was passed as 0.
FIRQB + FQCLUS	The file identification index of this file. This word is significant mainly in that it can be used in place of the file name in subsequent opens of the file on disk. You can open the file with the OPNFQ subfunction of CALFIP using an explicit project-programmer number in FIRQB + FQPPN, a zero word in FIRQB + FQNAM1, and the file identification index in FIRQB + FQNAM1 + 2. Note that there is no performance gain for using the file identification index rather than the file name. The file identification index is provided for compatibility with RSX. Furthermore, the file identification index is changed when the REORDR utility is run on the directory (see the *RSTS/E System Manager's Guide*).

Errors

xxxxx	All errors for this directive are device-dependent. See Appendix A for a full list of errors.

Example

The following MACRO code opens a temporary file on channel 13. (We assume that the FIRQB has been initialized to all zeros. Hence, the temporary file is created on the public disk structure.)

```
MOVB    #CRTFQ,FIRQB+FQFUN      ;SET FUNCTION CODE
MOVB    #13.*2,FIRQB+FQFIL      ;SET CHANNEL TO 13
CALFIP
```

3.2.6 DALFQ (Deassign All Devices) – Not Privileged

Form

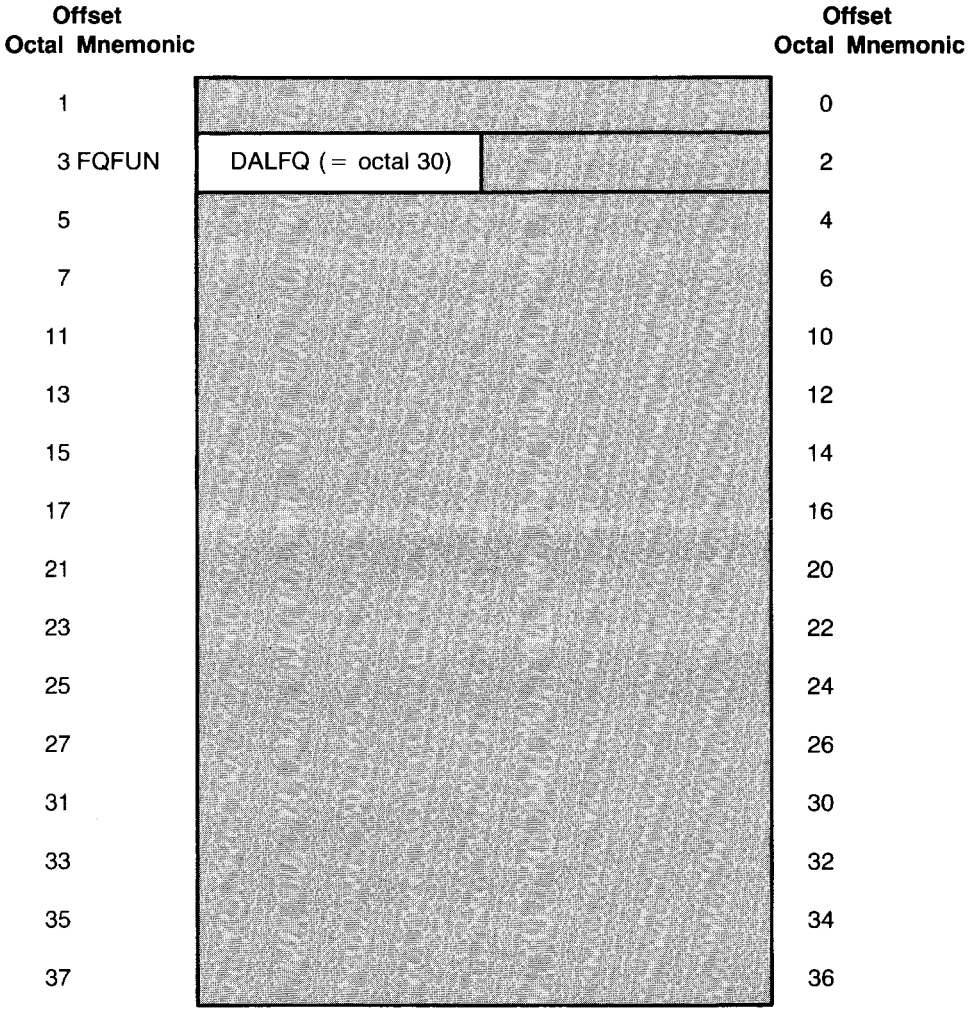
```
MOVB    #DALFQ,FIRQB+FQFUN
CALFIP
```

Function

The DALFQ subfunction deassigns all devices currently assigned to the job.

Data Passed

FIRQB



FIRQB+FQFUN The function code DALFQ (octal value = 30).

Data Returned

No data is returned by the DALFQ subfunction of CALFIP.

Errors

No errors are possible with DALFQ.

3.2.7 DEAFQ (Deassign a Device) – Not Privileged

Form

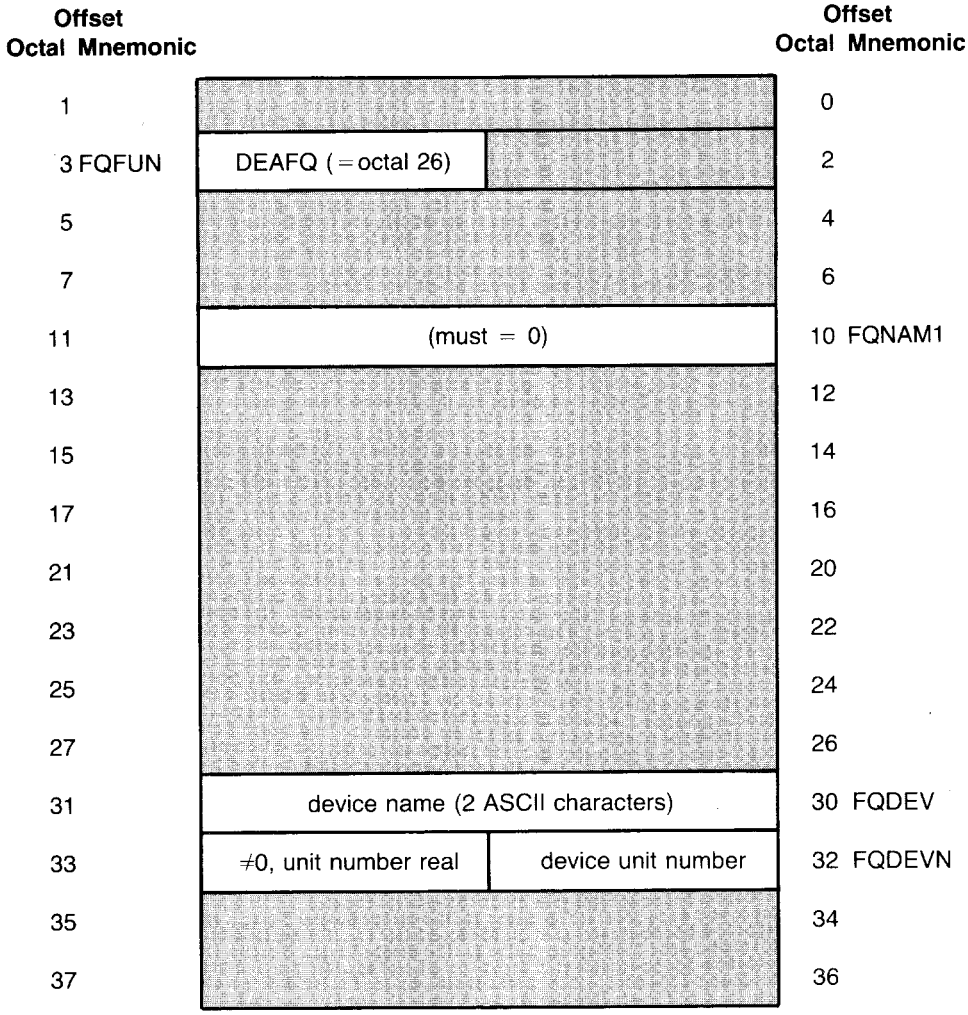
```
MOVB     #DEAFQ,FIRQB+FQFUN
   .
   .
   .
(Define device to be deassigned in FIRQB)
   .
   .
   .
CALFIP
```

Function

The DEAFQ subfunction deassigns a device from the current job (releases it for use by other jobs).

Data Passed

FIRQB



FIRQB+FQFUN	The DEAFQ function code (octal value = 26).
FIRQB+FQNAM1	The word at this location must be set to 0.
FIRQB+FQDEV	The name of the device to be deassigned, as two ASCII characters.
FIRQB+FQDEVN	The device unit number is passed here in binary. A non-zero value in FIRQB+FQDEVN+1 indicates an explicit device unit number, while a zero value indicates no device unit number.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the DEAFQ subfunction of CALFIP.

Errors

NODEVC	The device or its type specified at FIRQB+FQDEV and FIRQB+FQDEVN is not part of your system configuration.

Example

The following code deassigns LP: from the current job. See Section 3.1.4 for information on the CLRFQB routine.

```
CALL    CLRFQB                  ;CLEAR FIRQB
MOVB    #DEAFQ,FIRQB+FQFUN      ;SET FUNCTION CODE
MOV     #"LP,FIRQB+FQDEV        ;DEVICE=LINE PRINTER
CALFIP
```

3.2.8 DIRFQ (Get Directory Information) – Privileged and Not Privileged

Form

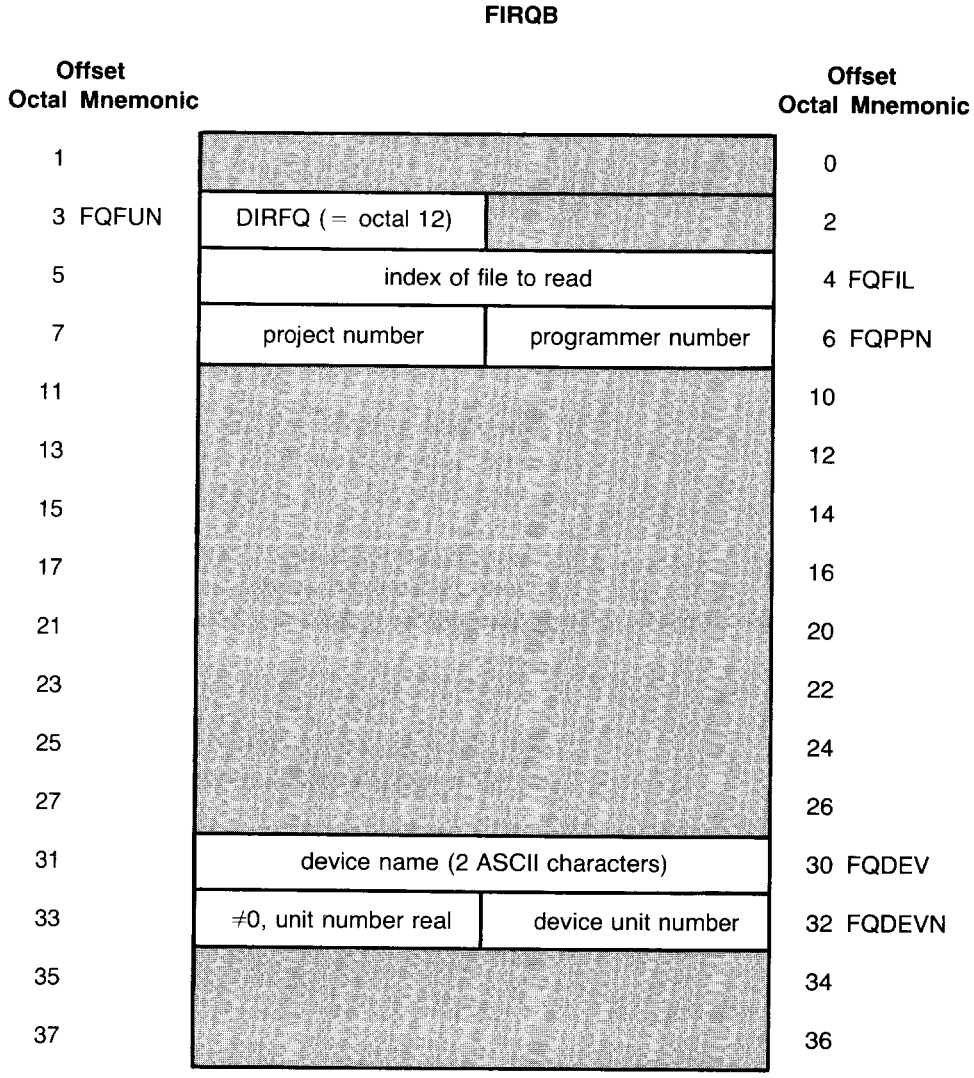
```
MOVB     #DIRFQ,FIRQB+FQFUN
   .
   .
   .
(Define device for the directory wanted)
   .
   .
   .
CALFIP
```

Function

The DIRFQ subfunction of CALFIP returns directory information about a disk, DECtape, or magtape file. Two forms of the call are available. One leaves a magtape file positioned at the end-of-file and returns the size of the file as part of the directory information. The second form, for magtape only, leaves a magtape file positioned at the beginning of the file, and does not return the size of the file.

Note that for disk, DIRFQ works differently for privileged and nonprivileged users. Privileged users always receive directory information. Nonprivileged users receive directory information only if they have read access to the file. The system manager can install an optional patch to give nonprivileged users unrestricted access to directory information.

Data Passed — Directory Lookup on Index



FIRQB + FQFUN — The function code DIRFQ (octal value = 12).

FIRQB + FQFIL — The index of the file to read. If this word is zero, the monitor returns data for the first file in the directory. For some positive value n, the monitor returns data for the n + 1 file in the directory. For magtape, a value of 0 causes the monitor to rewind the tape before it gets the information for the first file (by reading the label record of the file). The monitor then spaces the tape forward to the next end-of-file record and calculates the number of records in the file. The tape is left in that position. A nonzero value performs the same action, except that the tape is not rewound.

FIRQB + FQFIL (continued)

For DECtape, the first call issued must have a value of zero in this word to read the directory blocks from the tape before reading the first file. Subsequent calls with this word nonzero read the directory from the BUFF.SYS file. (Directory information for DECtape is kept in this system file on disk to speed up DECtape file processing, as described in the *RSTS/E Programming Manual*.)

FIRQB + FQPPN

The project-programmer number of the directory to look up, for disk or magtape. (The monitor does not use these bytes if the device is DECtape but simply returns information for each file read on the device.)

If this word is zero and the device is disk, this directive returns information for the project-programmer number under which this job is being executed.

If this word is zero and the device is magtape, this directive returns information for each file read, regardless of the project-programmer number under which it was written.

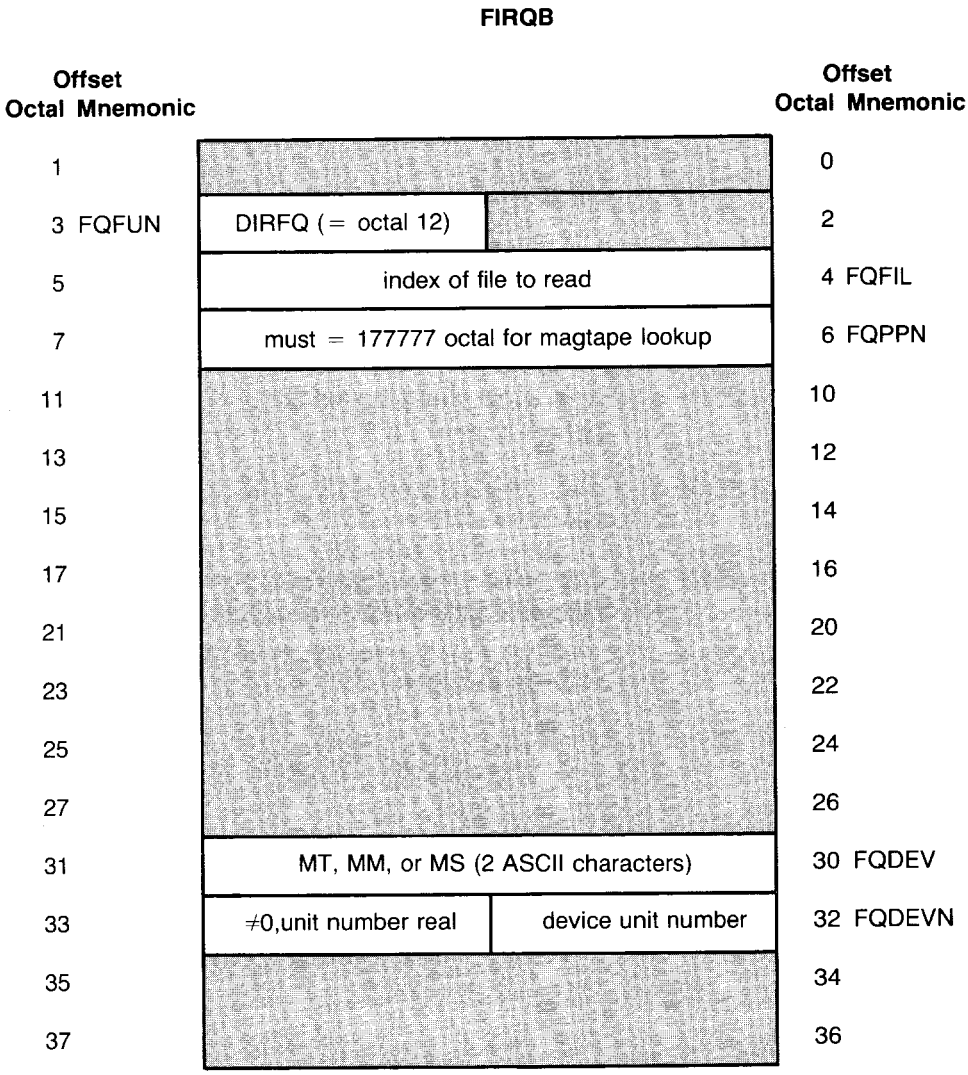
FIRQB + FQDEV

The device name, as two ASCII characters. Must be disk, magtape, or DECtape. If this word is zero, the public disk structure (SY:) is used.

FIRQB + FQDEVN

The device unit number is passed here in binary. A nonzero value in FQDEVN + 1 indicates an explicit device unit number. A zero value in FQDEVN + 1 indicates no unit number.

Data Passed — Special Magtape Lookup



FIRQB + FQFUN — The function code DIRFQ (octal value = 12).

FIRQB + FQFIL — Index number of the file to be read. If this word is zero, information is returned for the first file in the directory. If this word is some positive value n, information is returned for the n + 1 file in the directory. A value of 0 causes the monitor to rewind the tape before getting information from the first file (by reading the label record). It then backspaces the tape one record, leaving it positioned at the beginning of file. (This action leaves the tape positioned such that an open on this file will succeed on a single read from tape.) A nonzero value does not

FIRQB+FQFIL (continued)
: cause the tape to be rewound; the next record is read (it must be a label), and the tape is backspaced one record. When you are searching a tape for specific files to read, the normal action is to execute this directive with a value of 0 first. If the file is one to be read, open the file requesting no rewind, process the file, and close it to position the tape at the end-of-file. If the file is not one to be read, space the tape forward to the next end-of-file. (You can do this in MACRO with the .SPEC directive, Section 3.23.) Then issue the DIRFQ call with a nonzero value in the word beginning at FIRQB+FQFIL, and continue the process.

FIRQB+FQPPN
: This word must be set to 177777_8 for the special magtape lookup operation.

FIRQB+FQDEV
: The device name (MT, MM, or MS) as two ASCII characters.

FIRQB+FQDEVN
: The device unit number, in binary. A nonzero value in FIRQB+FQDEVN+1 indicates an explicit device unit number. A zero value in FIRQB+FQDEVN+1 indicates no unit number.

Data Returned (Both)

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3		current job number * 2	2	FQJOB
5	(same as data passed)		4	FQFIL
7	project number	programmer number	6	FQPPN
11	file name (2 words in RAD50 format)		10	FQNAM1
13			12	
15	file type (1 word in RAD50 format)		14	FQEXT
17	LSB of file length in blocks (not magtape)		16	FQSIZ
21	MSB of file length	protection code	20	FQNAM2
23	date of last access		22	FQMODE
25	date of creation		24	FQFLAG
27	time of creation		26	FQPFLG
31	(same as data passed)		30	FQDEV
33			32	FQDEVN
35	file cluster size (disk only)		34	FQCLUS
37	USTAT byte	number entries returned	36	FQNENT

FIRQB + FQJOB — The current job number times two.

FIRQB + FQPPN — The project-programmer number of the file. The project number is in the high byte (FIRQB + FQPPN + 1) and the programmer number is in the low byte (FIRQB + FQPPN).

FIRQB + FQNAM — The file name, as two words in RAD50 format.

FIRQB + FQEXT — The file type, as one word in RAD50 format.

FIRQB + FQSIZ — The least significant bits of the file size, in 512_{10}–byte blocks. This word is combined with the byte at FIRQB + 21 to form a 24–bit value giving the file's size. (This word is not returned with the special magtape directory lookup.)

FIRQB + FQNAM2 This byte contains the protection code of the file.

FIRQB + 21 This byte contains the most significant bits of the file's size in 512_{10}–byte blocks (see FIRQB + FQSIZ). (This byte is not returned with the special magtape directory lookup.)

FIRQB + FQMODE The date the file was last accessed, in system internal format:

[(year –1970) * 1000.] + day-within-year

(See the .DATE directive, Section 3.7, for a discussion of the system internal format for dates, if necessary.)

FIRQB + FQFLAG The date the file was created, in system internal format.

FIRQB + FQPFLG The time that the file was created, in system internal format: minutes before midnight, where midnight = 1440. (See the .DATE directive for a discussion of the system internal format for time, if necessary.)

FIRQB + FQCLUS For disk devices, this word contains the file cluster size. For tape, not used.

FIRQB + FQNENT Number of entries returned: for disk, 8; for tape, 6.

FIRQB + 37 Internal flag information (disk only):

Bit	Meaning
002	Placed file.
004	Some job has write access now.
010	File is open in update mode.
020	File is contiguous; no extend available.
040	No delete or rename allowed.
200	File is marked for deletion.

Errors

NOSUCH	Either account (project-programmer number) does not exist on the device specified, or no more files exist on the account (the index number is greater than the number of files on the account). Or, for the special magtape lookup, no more files exist on the tape.
DEVNFS	The device specified is not a file-structured device.
xxxxx	Other errors are device-dependent. See Appendix A for a full list of errors.

Example

The following code searches the disk directory to examine files for user [2,101]:

```
LOOP:   MOVB    #DIRFQ,FIRQB+FQFUN      ;SET FUNCTION CODE
        CLR     FIRQB+FQFIL
        MOV     #<2,*400+101,>,FIRQB+FQPPN ;SET PROJ,,PROG.NO.
        CLR     FIRQB+FQDEV             ;SEARCH SYSTEM DISK
        CLR     FIRQB+FQDEVN
        CALFIP
            .
            .
            .
  (Error processing, examine file name, process file)
            .
            .
            .
        INCB    FIRQB+FQFIL             ;INCREMENT INDEX
        JMP     LOOP                    ;GO BACK FOR NEXT ROUND
```

3.2.9 DLNFQ (Delete a File) – Not Privileged

Form

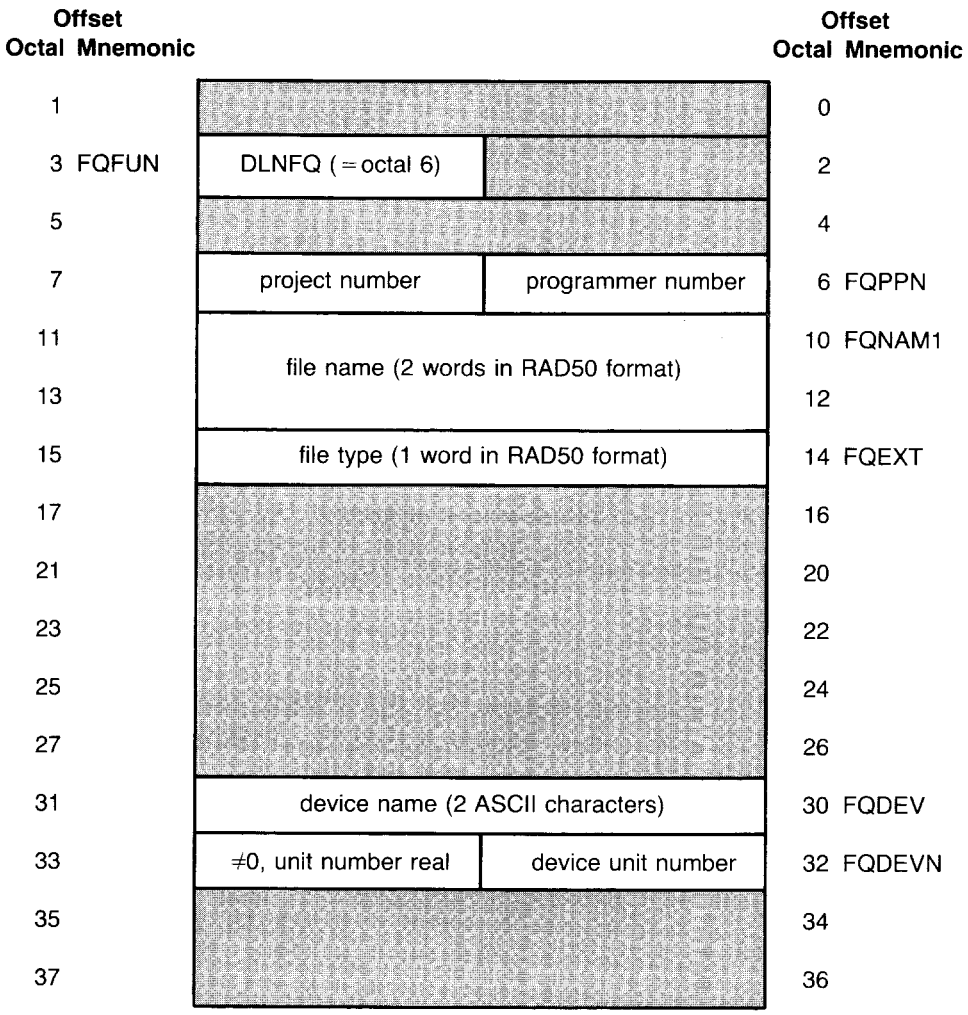
```
MOVB      #DLNFQ,FIRQB+FQFUN
  .
  .
  .
(Define file to be deleted in FIRQB)
  .
  .
  .
CALFIP
```

Function

The DLNFQ function of CALFIP deletes a file from disk or DECtape. The monitor's internal data on the location of the file are destroyed, and the file's space on the device is made available for general use.

Data Passed

FIRQB



FIRQB+FQFUN — The function code DLNFQ (octal value = 6).

FIRQB+FQPPN — The project-programmer number (ppn) of the file to be deleted. The project number is in the high byte (FIRQB+FQPPN+1), and the programmer number is in the low byte (FIRQB+FQPPN). A value of 0 defaults to the ppn under which the calling program is running.

FIRQB+FQNAM1 — The name of the file to be deleted, as two words of RAD50 data.

FIRQB+FQEXT — The file type, as one word of RAD50 data.

FIRQB+FQDEV — The name of the device containing the file to be deleted, as two ASCII characters; it must be a disk or DECtape device. A value of 0 in this word indicates the public disk structure.

FIRQB+FQDEVN — The device unit number is passed here in binary. A non-zero value in FQDEVN+1 indicates an explicit device unit number. A zero value in FQDEVN+1 indicates no unit number.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the DLNFQ function of CALFIP.

Errors

NOSUCH — The file specified in the data passed cannot be found.

PRVIOL — Protection violation. An attempt was made to delete a file that is either write-protected against the caller or marked for no delete.

Example

The following code deletes the file MYFIL.LST from the user's account on the public structure. (Assume that the FIRQB has been filled with zeros previously.)

```
MOVB    #DLNFQ,FIRQB+FQFUN
MOV     #^RMYF,FIRQB+FQNAM1       ;SET FILE NAME
MOV     #^RILE,FIRQB+FQNAM1+2     ;AND TYPE
MOV     #^RLST,FIRQB+FQEXT        ;TO "MYFILE.LST"
CALFIP
```

3.2.10 ERRFQ (Return Error Message Text) – Not Privileged

Form / Example

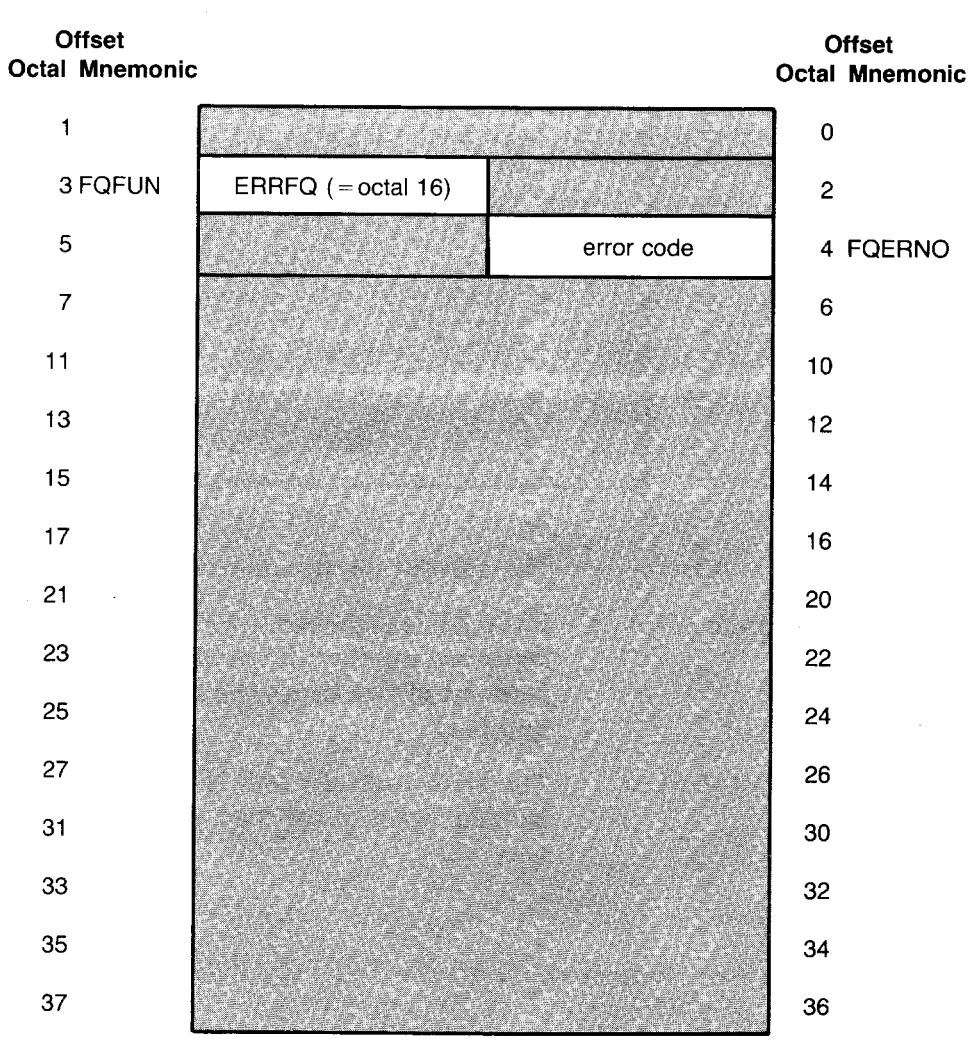
```
MOVB    #ERRFQ,FIRQB+FQFUN
MOVB    #ERR,FIRQB+FQERNO
```

Function

The ERRFQ subfunction of CALFIP returns error message text from the system error message file or from the default error message file if an error message file is not currently installed. The text is associated with the value of the error code passed as byte 4 of the FIRQB. This call returns the full RSTS/E error message text associated with errors returned in byte 0 of the FIRQB on all the monitor directives.

Data Passed

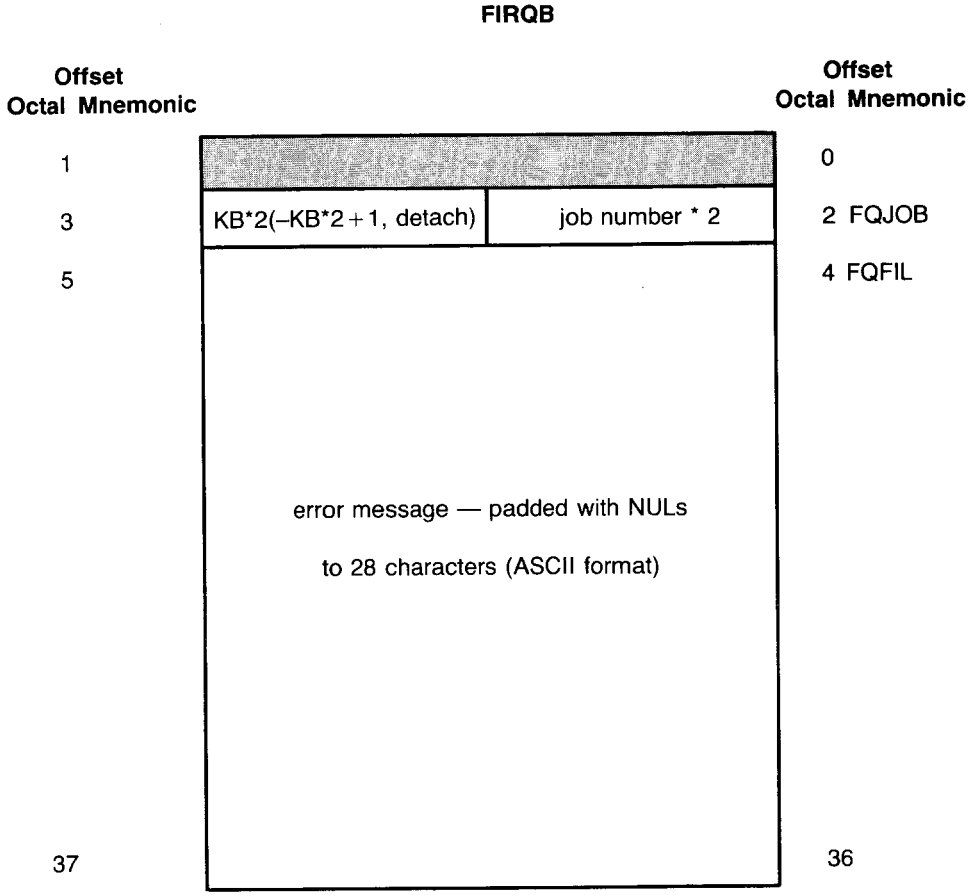
FIRQB



FIRQB+FQFUN The function code ERRFQ (octal value = 16).

FIRQB+FQERNO The error code value (in binary) for which the corresponding error message text is to be returned.

Data Returned



FIRQB+FQJOB The current job number times two.

FIRQB+3 If the job is attached, two times the currently attached keyboard number. If the job is detached, the one's complement of two times the currently detached keyboard number.

FIRQB+FQFIL The error message text begins in this byte. The text is padded with octal 000 bytes to 28 characters, if necessary.

Errors

No errors are returned with the ERRFQ subfunction of CALFIP.

3.2.11 LOKFQ (Disk File/Wildcard Lookup) – Privileged and Not Privileged

Form

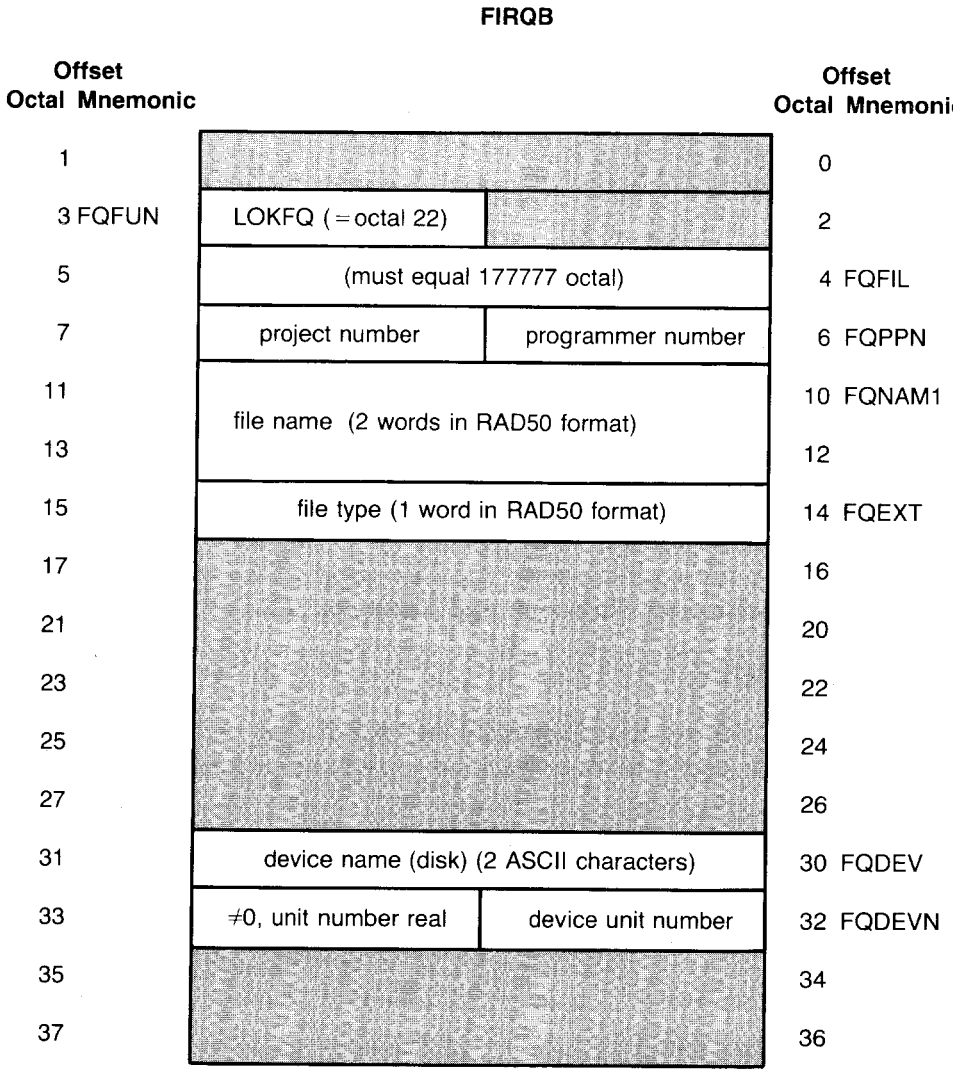
```
MOVB     #LOKFQ,FIRQB+FQFUN
   .
   .
   .
(Set up FIRQB to define file/wildcard)
   .
   .
   .
CALFIP
```

Function

The LOKFQ subfunction of CALFIP will do one of two things. It will look for a file on disk by name, returning directory information (date of creation, and so forth). Or it will perform a "wildcard" file search. For example, with a file name and type in the FIRQB of *.TXT it will search an account for a file with any file name and a type of .TXT. By incrementing an index and reexecuting LOKFQ, you can search through an entire directory for all such files.

LOKFQ works differently for privileged and nonprivileged users. Privileged users always receive directory information. Nonprivileged users receive directory information only if they have read access to the file. Note that the system manager can install an optional patch to give nonprivileged users unrestricted access to directory information.

Data Passed — Disk Directory Lookup



FIRQB + FQFUN	The function code LOKFQ (octal value = 22).
FIRQB + FQFIL	The word beginning at this location must be set to 177777 (octal) to indicate the "disk directory lookup by file name" option of LOKFQ.
FIRQB + FQPPN	The project-programmer number (ppn) for the file to be looked up. A value of 0 for this word defaults to the ppn under which the calling program is running.
FIRQB + FQNAM1	The file name to be looked up; two words in RAD50 format.
FIRQB + FQEXT	The file type of the file to be looked up; one word in RAD50 format.

FIRQB + FQDEV — The device name (must be disk), as two ASCII characters. If both bytes are 0, the public disk structure (SY:) is used.

FIRQB + FQDEVN — The disk device unit number is passed here in binary. A nonzero value in FQDEVN + 1 indicates an explicit device unit number. A zero value in FQDEVN + 1 indicates the system default.

Data Returned — Disk Directory Lookup by File Name

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3		job number * 2	2	FQJOB
5	same as data passed (177777 octal)		4	FQFIL
7	project number	programmer number	6	FQPPN
11	file name (2 words in RAD50 format)		10	FQNAM1
13			12	
15	file type (1 word in RAD50 format)		14	FQEXT
17	LSB of file length in 512–byte blocks		16	FQSIZ
21	MSB of file length	protection code	20	FQNAM2
23	date of last access		22	FQMODE
25	date of creation		24	FQFLAG
27	time of creation		26	FQPFLG
31	device name (2 ASCII characters)		30	FQDEV
33	≠0, unit number real	device unit number	32	FQDEVN
35	file cluster size		34	FQCLUS
37	file identification index		36	FQNENT

FIRQB + FQJOB	The current job number times two.
FIRQB + FQFIL	The word at this location is the same as the data passed, in this case, 177777 (octal).
FIRQB + FQPPN	The project-programmer number of the file (same as data passed).
FIRQB + FQNAM1	The file name, two words in RAD50 format (same as data passed).
FIRQB + FQEXT	The file type, one word in RAD50 format (same as data passed).
FIRQB + FQSIZ	This word contains the least significant bits of the file's size in 512_{10}–byte blocks. This word is combined with the byte at FIRQB + 21 to form a 24–bit field giving the file size.
FIRQB + FQNAM2	The file's protection code, in binary, is returned in this byte.
FIRQB + 21	This byte contains the most significant bits of the file size in 512_{10}–byte blocks. It is combined with the word at FIRQB + FQSIZ to form a 24–bit field giving the file size.
FIRQB + FQMODE	This word contains the date the file was last accessed, in system internal format: [(year – 1970) * 1000_{10}] + day-within-year
FIRQB + FQFLAG	This word contains the date the file was created, in system internal format (see FIRQB + FQMODE).
FIRQB + FQPFLG	This word contains the time the file was created, in system internal format: minutes until midnight, with 1440 equal to midnight.
FIRQB + FQDEV	The device name, as two ASCII characters. Always a specific name, even if 0 was passed here.
FIRQB + FQDEVN	The device unit number, in binary. A specific number is always returned here; FIRQB + FQDEVN + 1 is always nonzero.
FIRQB + FQCLUS	The file cluster size is returned in this word.

FIRQB+FQNENT

The file identification index is returned in this word. You can use the file identification index instead of the file name to open a file on disk with the OPNFQ subfunction of CALFIP. To do so, specify an explicit device name at FIRQB+FQDEV, a device unit number at FIRQB+FQDEVN, an explicit project-programmer number at FIRQB+FQPPN, a zero word at FIRQB+FQNAM1, and the file identification index at FIRQB+FQNAM1+2. (The file identification index is used by utilities that access software subroutines in the RMS libraries, for example.)

Note that there is no performance gain in using the file identification index rather than the file name. The file identification index is provided for compatibility with RSX. Furthermore, the file identification index is changed when the REORDR utility is run on the directory (see the *RSTS/E System Manager's Guide*).

Errors

BADNAM

The file name in bytes FIRQB+FQNAM1 through FIRQB+13 is missing.

NOSUCH

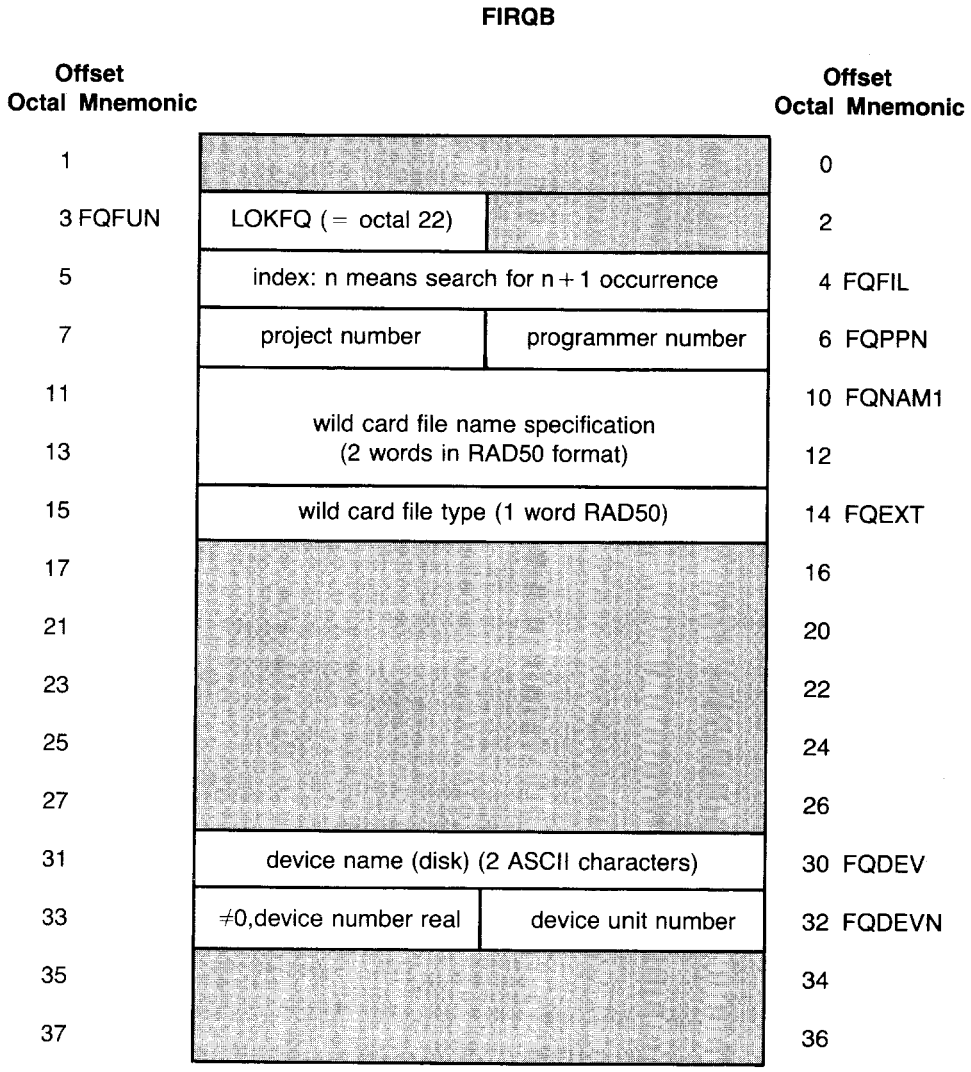
The device specified at FIRQB+FQDEV is not disk, or the file specified does not exist on the specified disk. This error also occurs when a nonprivileged user does not have read access to the specified file.

Example

The following code looks for the file MATRIX.DAT on the system disk. See Section 3.1.4 for information on the CLRFQB routine.

```
CALL    CLRFQB                  ;CLEAR FIRQB
MOV     #LOKFQ,FIRQB+FQFUN      ;SET FUNCTION CODE
MOV     #177777,FIRQB+FQFIL     ;FILENAME LOOKUP
CLR     FIRQB+FQPPN             ;CALLER'S ACCOUNT
MOV     #^RMAT,FIRQB+FQNAM1     ;SET FILENAME
MOV     #^RRIX,FIRQB+FQNAM1+2   ;AND TYPE
MOV     #^RDAT,FIRQB+FQEXT      ;TO "MATRIX.DAT"
CLR     FIRQB+FQDEV             ;SYSTEM DISK
CLR     FIRQB+FQDEVN            ;DEVICE
CALFIP
```

Data Passed — Disk Wild Card Directory Lookup



FIRQB + FQFUN — The function code LOKFQ (octal value = 22).

FIRQB + FQFIL — An index number specifying the occurrence of the file name meeting the wildcard specifications. A value of 0 in this word causes the monitor to search for the first file name in the directory that meets the specification. A value of 1 causes the monitor to search for the second file name, and so forth.

FIRQB + FQPPN — The project-programmer number of the account whose directory of disk files is to be searched.

FIRQB + FQNAM1

The wildcard file name, as two words in RAD50 format. An * character can replace the entire file name, or a ? character can replace any character in the file name. For example, a file name of FILE?? would cause the monitor to search the directory for any file name beginning with the characters FILE. An * character indicates that the file name does not matter in the search.

FIRQB + FQEXT

The wildcard file type, as one word in RAD50 format. An * character can replace the entire file type, or a ? character can replace any character in the type. For example, a file type of BA? causes the monitor to search the directory for any file type beginning with the characters BA. An * character indicates that the file type does not matter in the search.

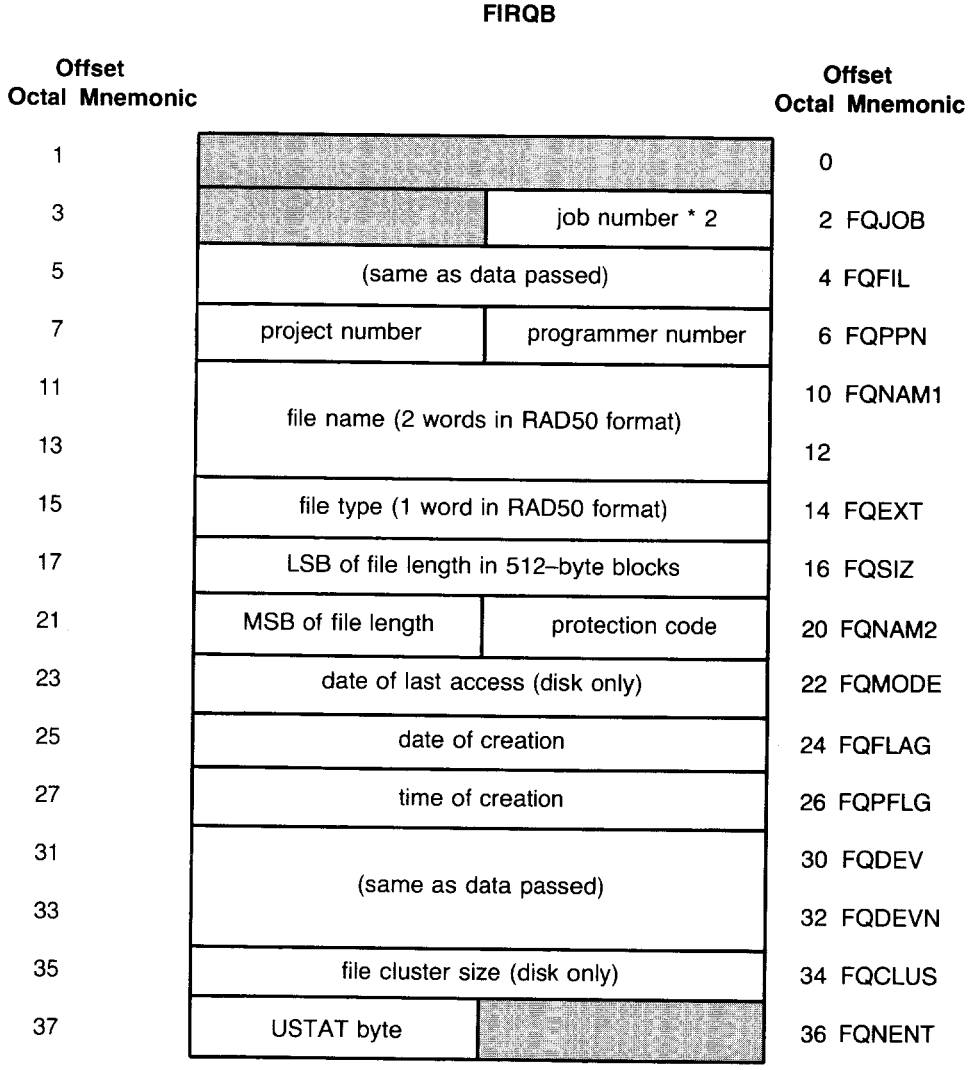
FIRQB + FQDEV

The name of the device to be searched (must be disk). A value of 0 in this word indicates the public disk structure (SY:).

FIRQB + FQDEVN

This byte contains the device unit number in binary. A nonzero value in FIRQB + FQDEVN + 1 indicates an explicit device unit number. A zero value in FIRQB + FQDEVN + 1 indicates the system default.

Data Returned — Disk Wildcard Directory Lookup



FIRQB + FQJOB — The current job number times two.

FIRQB + FQPPN — The project-programmer number of the file (same as data passed).

FIRQB + FQNAM1 — The actual file name of a file meeting the wildcard specification in the data passed. Two words in RAD50 format.

FIRQB + FQEXT — The actual file type of a file meeting the wildcard specification in the data passed. One word in RAD50 format.

FIRQB+FQSIZ — This word contains the least significant bits of the file's size in 512_{10}–byte blocks. This word is combined with the byte at FIRQB+21 to form a 24–bit field giving the file's size.

FIRQB+FQNAM2 — The file's protection code, in binary, is returned in this byte.

FIRQB+21 — This byte contains the most significant bits of the file's size in 512_{10}–byte blocks. The byte is combined with the word at FIRQB+FQSIZ to give the file's size.

FIRQB+FQMODE — This word contains the date the file was last accessed, in system internal format:

[(year – 1970) * 1000_{10}] + day-within-year

FIRQB+FQFLAG — This word contains the date the file was created, in system internal format (see FIRQB+FQMODE).

FIRQB+FQPFLG — This word contains the time the file was created, in system internal format: minutes until midnight, with 1440 equal to midnight.

FIRQB+FQDEV — The device name, as two ASCII characters. Always a specific name, even if 0 was passed here.

FIRQB+FQDEVN — The device unit number, in binary. A specific number is always returned here; FIRQB+FQDEVN+1 is always nonzero.

FIRQB+FQCLUS — The file cluster size is returned in this word.

FIRQB+37 — Internal flag information:

Bit	Meaning
002	Placed file.
004	Some job has write access now.
010	File is open in update mode.
020	File is contiguous; no extend available.
040	No delete or rename allowed.
200	File is marked for deletion.

Errors

BADNAM — No file specification appears at FIRQB+FQNAM1.

NOSUCH — Either the device specified at FIRQB+FQDEV and FIRQB+FQDEVN is not a disk, or no match exists for the occurrence specified in the word at FIRQB+FQFIL.

PAKLCK — The disk is locked and the caller is nonprivileged.

Example

The following code asks the monitor to search the directory for account [2,130] for the first occurrence of a file specification beginning with the letter X. See Section 3.1.4 for information about the CLRFQB and CLRXRB routines.

```
CALL      CLRFQB                        ;MAKE SURE FIRQB
CALL      CLRXRB                        ;AND XRB ARE CLEAR
MOV       #FILNAM,XRB+XRLOC             ;POINT TO FILE NAME
MOV       #NAMSIZ,XRB+XRLEN             ;SET ITS LENGTH
MOV       #NAMSIZ,XRB+XRBC              ;AND AGAIN
.FSS                                    ;CONVERT TO FIRQB FORMAT
MOVB      #LOKFQ,FIRQB+FQFUN            ;SET FUNCTION CODE
CALFIP

FILNAM:    .ASCII "[2,130]X?????.*"     ;STRING FOR.FSS TO CONVERT
NAMSIZ = .-FILNAM                       ;AND ITS LENGTH
```

3.2.12 OPNFQ (Open a File/Device on a Channel) – Privileged and Not Privileged

Form

```
MOVB     #OPNFQ,FIRQB+FQFUN
   .
   .
   .
(Set parameters appropriate to file or device)
   .
   .
   .
CALFIP
```

Function

The OPNFQ function has the same effect as an OPEN FOR INPUT statement in BASIC–PLUS; it opens a device or already-existing file on a channel. Parameters defining the device, file name and type, protection code, and mode are passed to the monitor in the FIRQB. If a file name is given in the FIRQB, a file-structured open for input is performed. If no file name is given, a non-file-structured open for input is performed. The *RSTS/E Programming Manual* describes file- and non-file-structured open for input and the actions taken for the mode parameter (MODE modifier in BASIC–PLUS) for each device.

NOTE

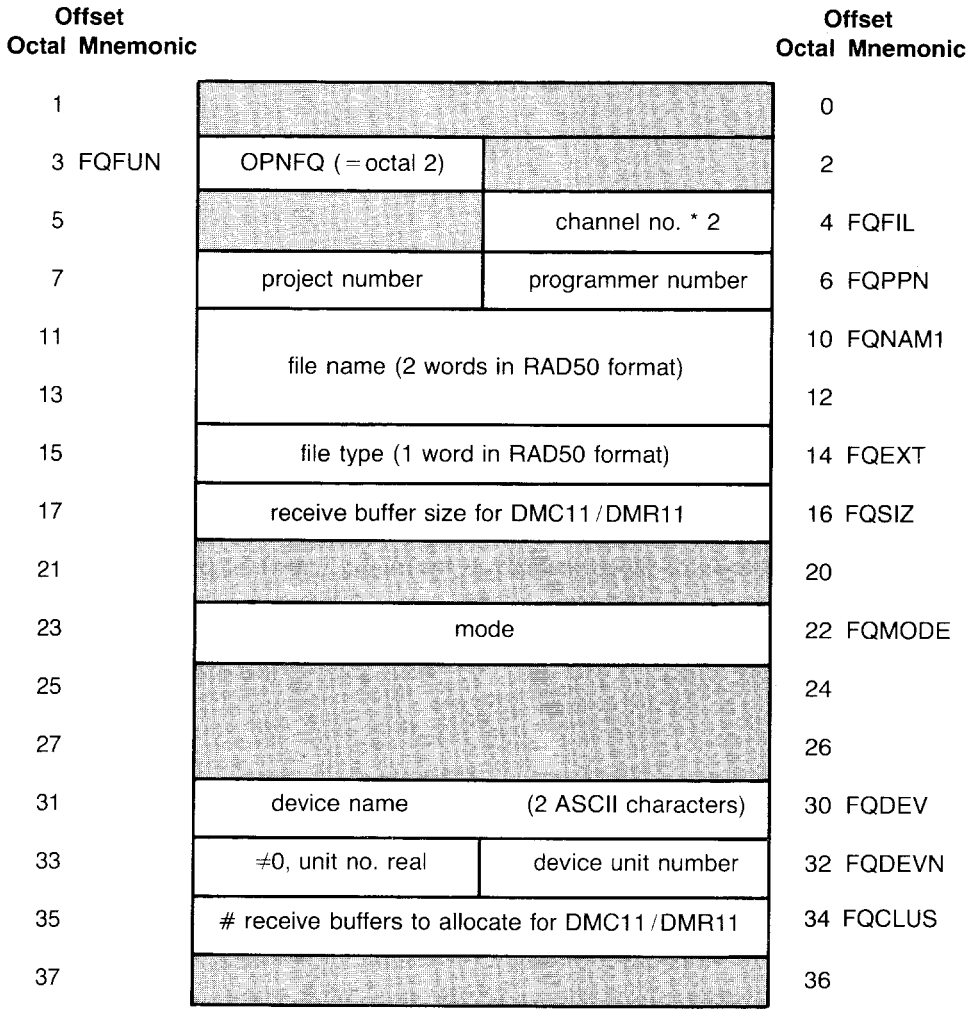
Only privileged users can open a disk for non-file-structured processing. An optional patch is available to extend this capability to nonprivileged users; refer to the *RSTS/E Maintenance Notebook* for more information.

Whenever you use a disk as a non-file structured device, be aware that all RSTS/E data structures you access are subject to change at any time.

CALFIP
OPNFQ

Data Passed

FIRQB



FIRQB+FQFUN — The function code OPNFQ (octal value = 2).

FIRQB+FQFIL — Channel number times 2; defines the channel on which the file is to be opened.

FIRQB+FQPPN — The project-programmer number (ppn) of the file to be opened. The project number is in the high byte (FQPPN+1), and the programmer number in the low byte (FQPPN). A value of 0 defaults to the ppn under which the calling program is running. (Not used for non-file-structured open.)

FIRQB + FQNAM1 — The file name to be opened, as two words of RAD50 data. May also be specified as a word of 0 followed by the file identification index (see LOKFQ, Section 3.2.11).

(Must be two words of zero for a non-file-structured open.)

FIRQB + FQEXT — The file type, as one word of RAD50 data. (Must be zero for a non-file-structured open.)

FIRQB + FQSIZ — This parameter is useful for the DMC11/DMR11 only, where it specifies the receive buffer size. You can specify a value between 1 and 632_{10} (1170_8). (See the *RSTS/E Programming Manual* for more information.)

FIRQB + FQMODE — The mode with which the file is to be opened; values and actions taken are as described for the MODE modifier in OPEN FOR INPUT and non-file-structured OPEN statements for various devices, as described in the *RSTS/E Programming Manual.* If you use a mode value at all, you must set bit 15 of this word to 1.

FIRQB + FQDEV — The device name is passed here as two ASCII characters. A value of zero indicates "SY", public disk.

FIRQB + FQDEVN — The device unit number is passed here in binary. A nonzero value in the high byte of this word (FIRQB + FQDEVN + 1) indicates an explicit device unit number. A zero value in FIRQB + FQDEVN + 1 indicates no unit number.

FIRQB + FQCLUS — This parameter has the same function as the CLUSTERSIZE option in BASIC–PLUS. For OPNFQ, it is useful only for the DMC11/DMR11, where it specifies the number of receive buffers to allocate. You can specify a value between 1 and 177_8, but values above 4 are not recommended. (See the *RSTS/E Programming Manual* for more information.)

Data Returned

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job number * 2	2	FQJOB
5	FQSIZM	MSB of file size	channel number * 2	4	FQFIL
7		project number	programmer number	6	FQPPN
11		file name (2 words of RAD50)		10	FQNAM1
13				12	
15		file type (1 word of RAD50)		14	FQEXT
17		LSB (least significant bits of) file size		16	FQSIZ
21		reasonable buffer size for device		20	FQBUFL
23		(as passed)		22	FQMODE
25		device description		24	FQFLAG
27	FQPROT	protection code	clustersize, mod 256	26	FQPFLG
31		device name (2 ASCII characters)		30	FQDEV
33		flag byte	device unit number	32	FQDEVN
35		file identification index		34	FQCLUS
37				36	

NOTE

For nondisk devices, the relevant information returned with the OPNFQ subfunction is in the two words at FIRQB+FQBUFL and FIRQB+FQFLAG. All other words are simply returned as passed.

FIRQB+FQJOB — The current job number times two.

FIRQB+FQFIL — Channel number times two; defines the channel on which the file is open.

FIRQB + FQSIZM — For large disk files (greater than $65{,}535_{10}$ blocks), this byte contains the most significant bits of the file's size in 512_{10}–byte blocks. This byte is combined with the word at FIRQB + FQSIZ to form a 24_{10}–bit field giving the file size.

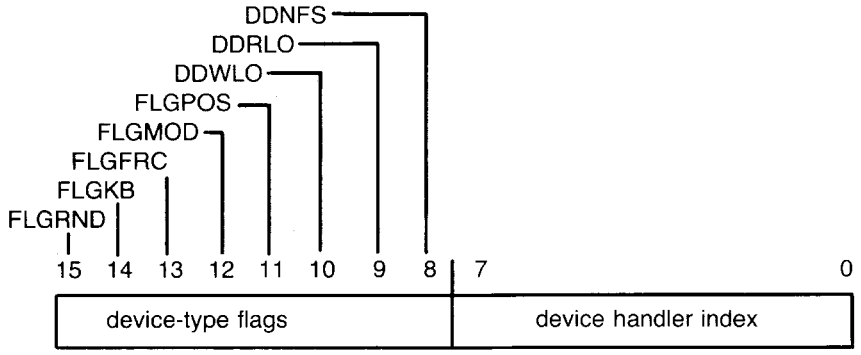
FIRQB + FQPPN — The project-programmer number under which the file is open. An actual project-programmer number is returned here even if this word was passed as zero.

FIRQB + FQNAM1 — The file name, as two words of RAD50 data.

FIRQB + FQEXT — The file type, as one word of RAD50 data.

FIRQB + FQBUFL — Reasonable buffer size for this device, in bytes. If you are doing device-independent I/O (that is, if you do not wish to keep track of which device is being opened and perform specific opens, reads, and writes, depending on the device), this value is the monitor's "best guess" for a buffer size to use in subsequent reads and writes on the opened channel. (See .READ and .WRITE, Sections 3.17 and 3.33.)

FIRQB + FQFLAG — Description of the device just opened. The low byte contains the device's handler index. There is one unique handler index for all device types. The high byte contains a set of status flags to allow for device-independent I/O routines.



High Byte — Device-Type Flags

The bits in the high byte of the flags word are set to indicate the type of file or device just opened.

FLGRND = 1 The device or file is random-access.
= 0 The device or file is sequential.

FLGKB = 1 The file or device is a terminal-type file or device (or is generically a terminal).
= 0 The file or device is not a terminal-type file or device.

FLGFRC = 1 The file or device is byte-oriented. That is, reads and writes handle data in byte units.
= 0 The file or device is block-oriented. Reads and writes handle data in block units.

FLGMOD = 1 The file or device accepts modifiers in reads and writes (Sections 3.17 and 3.33, XRB + XRMOD).
= 0 The file or device does not accept modifiers in reads and writes.

FLGPOS = 1 The file or device keeps track of its horizontal position and expands such characters as TAB into whatever is appropriate for the file or device. You can determine the current horizontal position with the .POSTN directive.
= 0 The file or device does not keep track of its horizontal position.

DDWLO = 1 The file or device has been write-locked (with the protection code in the open) or is generically a write-only device.
= 0 The file or device is not write-locked.

DDRLO = 1 The file or device has been read-locked (with the protection code in the open) or is generically a read-only device.
= 0 The file or device is not read-locked.

DDNFS = 1 The file or device is non-file-structured (or is generically not a file-structured device).
= 0 The file or device is file-structured.

Low Byte — Device Handler Index

Bits 0–7 of the flags word contain a handler index that indicates the generic kind of device.

Octal Value	Symbol	Meaning
0	DSKHND	All disks
2	TTYHND	All terminals
4	DTAHND	DECtape
6	LPTHND	All line printers
10	PTRHND	Paper tape reader
12	PTPHND	Paper tape punch
14	CDRHND	Card reader
16	MTAHND	Magtape
20	PKBHND	Pseudo keyboards
22	RXDHND	Flexible diskette
24	RJEHND	2780 remote job entry
26	NULHND	The null device
30	DMCHND	The DMC11/DMR11 DDCMP interface
36	DT2HND	DECtape II
40	KMCHND	KMC11
42	IBMHND	IBM interconnect
46	DMPHND	DMP11/DMV11 device

FIRQB + FQPFLG — The file cluster size, modulo 256. That is, a file cluster size of 256_{10} is indicated by a zero byte here.

FIRQB + FQPROT — The protection code of the file.

FIRQB + FQDEV — The device name, as two ASCII characters. (For disk, the actual device name is returned, even if a zero word was passed in this word.)

FIRQB + FQDEVN — The device unit number. (For disk devices, the actual unit number is returned here, even if FIRQB + FQDEVN + 1 was passed as 0.)

FIRQB+33

For a file-structured open, this byte contains two relevant bits:

Bit 0 = 0 The device is in the public structure.
Bit 0 = 1 The device is a private disk.
Bit 1 = 0 A specific device was not specified.
Bit 1 = 1 A specific device was specified.

These bits are meaningless for a non-file-structured open.

FIRQB+FQCLUS

The file identification index of this file. This word is significant mainly in that it can be used in place of the file name in subsequent opens of the file on disk. You can open the file with the OPNFQ subfunction by using an explicit project-programmer number in FIRQB+FQPPN, a zero word in FIRQB+FQNAM1, and the file identification index in FIRQB+FQNAM1+2.

Note that there is no performance gain in using the file identification index rather than the file name. The file identification index is provided for compatibility with RSX. Furthermore, the file identification index is changed when the REORDR utility is run on the directory (see the *RSTS/E System Manager's Guide*).

Errors

NOTCLS

The specified channel is already open. It must be closed before it can be opened again.

PRVIOL

You are not privileged and tried to:

1. Open a disk for non-file-structured access.
2. Open a device that the system manager has restricted to privileged users.

xxxxx

All other possible errors are device-dependent. See Appendix A for a full list of errors.

Example

The following MACRO code sets up the FIRQB for the OPNFQ function. A non-file-structured open of magtape unit 2 is done on channel 3. See Section 3.1.4 for information on the CLRFQB routine.

```
MAG:    .ASCII   /MT/
         .
         .
         .
        CALL     CLRFQB                    ;CLEAR FIRQB
        MOVB     #OPNFQ,FIRQB+FQFUN        ;SET FUNCTION CODE
        MOVB     #3*2,FIRQB+FQFIL          ;SET CHANNEL = 3
        MOV      MAG,FIRQB+FQDEV           ;SET DEVICE = MT
        MOVB     #377,FIRQB+FQDEVN+1       ;SET FLAG DEVICE NO. EXPLICIT
        MOVB     #2,FIRQB+FQDEVN           ;SET DEVICE NO. = 2
        CALFIP
```

3.2.13 RENFQ (Rename a File) – Not Privileged

Form

```
MOVB      #RENFQ,FIRQB+FQFUN
   .
   .
   .
(Define file to be renamed and new name in FIRQB)
   .
   .
   .
CALFIP
```

Function

The RENFQ function renames an existing file on disk or DECtape and, if requested, deletes any existing file with the new name.

Data Passed

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3	FQFUN	RENFQ (=octal 10)	2	
5			4	
7		project number / programmer number	6	FQPPN
11		file name (2 words in RAD50 format)	10	FQNAM1
13			12	
15		file type (1 word in RAD50 format)	14	FQEXT
17		−1 to delete existing file (disk only)	16	FQSIZ
21		new file name (2 words in RAD50 format)	20	FQNAM2
23			22	
25		new file type (1 word in RAD50 fmt.)	24	FQFLAG
27	FQPROT	file protection / word = 0 if no change	26	FQPFLG
31		device name (2 ASCII characters)	30	FQDEV
33		≠0, unit number real / device unit number	32	FQDEVN
35			34	
37			36	

FIRQB+FQFUN	The function code RENFQ (octal value = 10).
FIRQB+FQPPN	The project-programmer number (ppn) of the existing file to be renamed. The project number is in the high byte (FIRQB+FQPPN+1), and the programmer number is in the low byte (FIRQB+FQPPN). A value of 0 defaults to the ppn under which the calling program is running.
FIRQB+FQNAM1	The old (existing) name for the file, as two words of RAD50 data.
FIRQB+FQEXT	The old type, as one word of RAD50 data.
FIRQB+FQSIZ	This word is set to –1 to indicate that any existing file on the specified device with the new name is to be deleted. If any other value is given here and a file already exists with the new name, the RENFQ function will return an error.
FIRQB+FQNAM2	The new file name and type are given here, as three words of RAD50 data. You can use RENFQ to change the protection code on a file by setting FQPROT and making these three words the same as the three words beginning at FIRQB+FQNAM1.
FIRQB+FQPROT	The new protection code for the file, if any, is specified in this byte. To retain the old protection code, this entire word (FIRQB+FQPFLG and FIRQB+FQPROT) must be zero. If the word is nonzero, the high byte will be used as the new protection code.
FIRQB+FQDEV	The device name is passed here as two ASCII characters; it must be a disk or DECtape device. A value of 0 in this word indicates the public disk structure.
FIRQB+FQDEVN	The device unit number is passed here in binary. A nonzero value in FIRQB+FQDEVN+1 indicates an explicit device unit number. A zero value in FIRQB+FQDEVN+1 indicates the system default.

Data Returned

Other than a possible error in byte 0 of the FIRQB, no data is returned by the RENFQ function.

Errors

FIEXST	The new file name specified already exists.
NOSUCH	The old file specified cannot be found.

Example

The following code renames the file OLDNAM.TXT to NEWNAM.TXT on the public disk structure under the caller's account. FQSIZ is set to –1, so any existing file named NEWNAM.TXT will be deleted. See Section 3.1.4 for information about the CLRFQB routine.

```
CALL     CLRFQB                  ;CLEAR FIRQB
MOVB     #RENFQ,FIRQB+FQFUN      ;SET FUNCTION CODE
MOV      #^ROLD,FIRQB+FQNAM1     ;SET OLD FILE NAME
MOV      #^RNAM,FIRQB+FQNAM1+2   ;AND TYPE
MOV      #^RTXT,FIRQB+FQEXT      ;TO "OLDNAM.TXT"
MOV      #-1,FIRQB+FQSIZ         ;DELETE EXISTING FILE
MOV      #^RNEW,FIRQB+FQNAM2     ;SET NEW FILE NAME
MOV      #^RNAM,FIRQB+FQNAM2+2   ;AND TYPE
MOV      #^RTXT,FIRQB+FQNAM2+4   ;TO "NEWNAM.TXT"
CALFIP
```

3.2.14 RSTFQ (Reset a Channel) – Not Privileged

Form

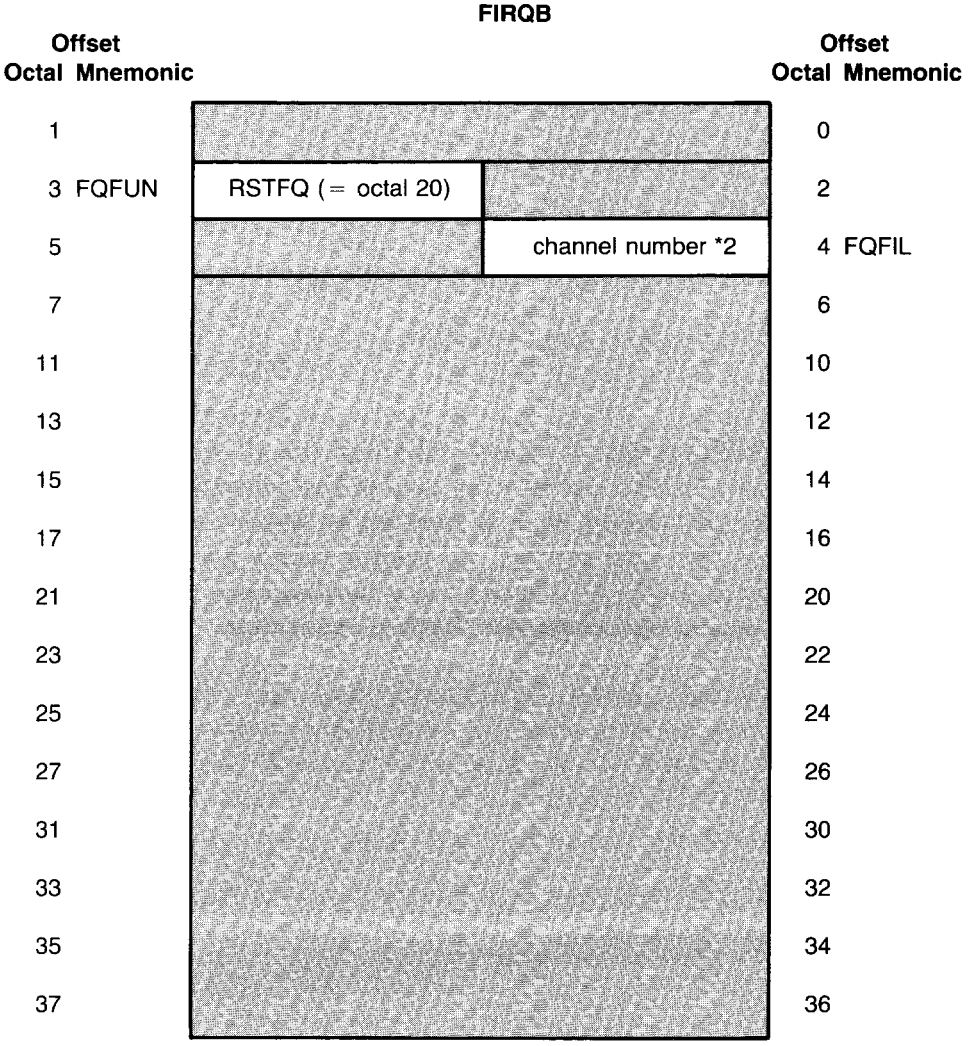
```
MOVB      #RSTFQ,FIRQB+FQFUN
   .
   .
   .
(Define channel to be reset)
   .
   .
   .
CALFIP
```

Function

The RSTFQ function closes a channel (or all channels, or all channels except one) without performing any of the normal "cleanup" operations. For example, no trailer tape is written to paper tape punch, no form feed is given on the line printer, no trailer labels are written to magtape. This function is useful as a backup to a normal close operation. If a normal close fails, RSTFQ will close the channel regardless. You can also use RSTFQ to close a channel on which a tentative file is open if you do not want to make the file permanent. (Tentative files are described in the *RSTS/E Programming Manual.*) The RSTFQ directive functions the same as a CLOSE statement with a negative channel number in BASIC–PLUS.

CALFIP
RSTFQ

Data Passed



FIRQB+FQFUN — The function code RSTFQ (octal value = 20).

FIRQB+FQFIL — Three possibilities:

1. Reset one channel. Set FIRQB+FQFIL to the channel number to reset times two.
2. Reset all channels. Set FIRQB+FQFIL to zero.
3. Reset all but one channel. Set FIRQB+FQFUN to –(channel number times two); that is, the negative of two times the channel number to remain open.

Data Returned

No data is returned with the RSTFQ function.

Errors

No errors are possible with the RSTFQ function. If the channel or channels specified are not open, the call simply returns without error.

Example

The following code resets all channels currently open for the job, except channel 2. See Section 3.1.4 for information about the CLRFQB routine.

```
CALL    CLRFQB                  ;CLEAR FIRQB
MOVB    #RSTFQ,FIRQB+FQFUN      ;SET FUNCTION CODE
MOVB    #-4,FIRQB+FQFIL         ;SET ALL CHANNELS BUT 2
CALFIP
```

3.2.15 UUOFQ (Hook to File Processor) – Privileged and Not Privileged

Form

```
MOVB      #UUOFQ,FIRQB+FQFUN
   .
   .
   .
(Set up FIRQB for UUOFQ subfunction)
   .
   .
   .
CALFIP
```

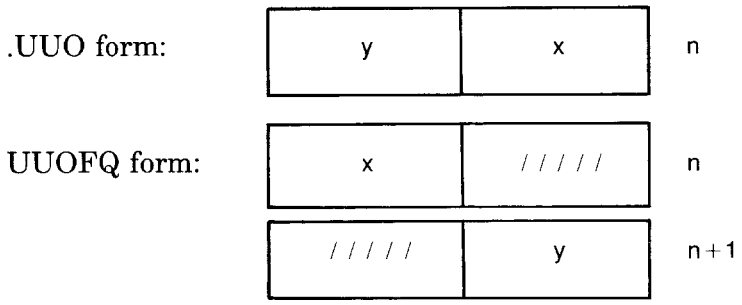
Function

The UUOFQ subfunction of CALFIP performs the same operations as the .UUO directive (Section 3.32). The same subfunctions are available, and a similar format is used for the data passed. The data returned is in the same format as for the .UUO directive.

In general, the data passed for the UUOFQ subfunction of CALFIP is moved down one byte from the .UUO subfunction format. The value UUOFQ is stored in byte 3 in the FIRQB, the UU.xxx code is stored in byte 4 in the FIRQB, whatever is shown in byte 4 for .UUO is stored in byte 5 for UUOFQ, and so forth.

The UUOFQ subfunction of CALFIP is not recommended (although it does work), because values that are normally set up as whole words must be stored as a high-byte in location n and a low-byte in location n+1.

For example:



3.3 .CCL — Check String for CCL Command – Not Privileged

Form

```
.CCL
```

Function

The .CCL directive asks the monitor to check a string (defined in the XRB) to see if it is a valid Concise Command Language (CCL) command. If the string is a valid CCL command, the monitor passes control to the appropriate run-time system for that CCL (as defined by the system manager), using the equivalent of a .RUN directive (Section 3.20). If the directive is successful, control does not return in-line.

Control will pass to the run-time system at the location specified in the P.RUN word in the pseudo-vector region (Section 2.5). Data will be passed to the run-time system in the job's CORCMN, XRB, FIRQB, and KEY areas in the low segment. The file containing the program to be run as a result of the .CCL command will be open on channel 15. (The contents of these areas are described under the P.RUN description (see Section 2.5.4), because they are of interest to the run-time system being entered as a result of a .CCL, not to the caller.)

If the string is not a valid CCL command, the monitor returns control to the caller (the run-time system or user job image that issued the .CCL directive) with no error. Control resumes with the instruction following the .CCL. Since control would not be returned here otherwise, the program does whatever processing it deems necessary here for an unsuccessful CCL.

As mentioned, the system manager defines the CCL command, an abbreviation point, the name of the file that is to be executed when the command is given, and an entry point for the program (see the *RSTS/E System Manager's Guide*).

The command can be a string from one to nine characters long. The allowed single-character commands are A through Z, @, $, and #. Commands that are longer than one character must begin with a letter; the remaining characters can be letters or digits.

The abbreviation point defines how many characters must be specified before the command is accepted as valid. For example, if "DIRECTORY" were defined as a CCL command, the system manager could indicate three characters as the abbreviation point for the command. Then DIR, DIRE, DIREC and so forth, up to the full DIRECTORY, would all be interpreted by the monitor as correct CCL commands. (The monitor always fills in the full CCL command in CORCMN when it passes control to the appropriate run-time system.)

When a .CCL directive is issued, then, the monitor compares the indicated string with the commands defined by the system manager:

1. All characters are trimmed to 7–bit ASCII; that is, each byte will lose its high-order bit. (Note that this may change in future releases.)
2. All null (ASCII code 000 octal) and delete (ASCII code 177 octal) characters are ignored and are never passed on to the run-time system in CORCMN.
3. Leading spaces (ASCII code 040 octal) and tabs (ASCII code 011 octal) are ignored and are never passed on to the run-time system in CORCMN.
4. When not enclosed by the quote characters " (ASCII code 042 octal) and ' (ASCII code 047 octal):
   - All tabs (ASCII code 011 octal) are changed to spaces (ASCII code 040 octal).
   - All control characters (ASCII codes 001 through 037, octal, inclusive) are ignored and never passed in CORCMN.
   - Adjacent spaces (ASCII code 040 octal) are merged into a single space.
   - All lowercase alphabetics (ASCII codes 141 through 172, octal, inclusive) are changed into their uppercase equivalents (ASCII codes 101 through 132, octal, inclusive).
5. When enclosed by the quote characters " (ASCII code 042 octal) and ' (ASCII code 047 octal), all characters are kept as is and passed on to the run-time system in CORCMN.

The monitor will also analyze two switches itself, immediately following the CCL command text (for example, DIR/SI:n/DET). These switches will be passed on to the run-time system as status flags set in the XRB, as described in the P.RUN description in Section 2.5. The two switches may appear in either order but must immediately follow the command. Both are optional switches. The /SIZE switch has the format:

[space]/SI[Z[E]]:[+][#]n[.]

where n indicates the size, in K words, of the user job image that the program, when executed, will require. If the + sign is given, n indicates the additional amount of space, in K words, that the file will require over that indicated by the computed size or minimum size (see description of PF.CSZ bit in P.FLAGS word, Section 2.5). If the + sign is omitted, then n simply indicates the size, in K words, at which the invoked file should run. If the # sign is given, n is assumed to be octal. If the period is given, n is assumed

to be decimal. If both are given, an error is returned, and if neither is given, n is assumed to be decimal. The value of n must be between 1 and the system-wide maximum for a user job image (see SWAP MAX, *RSTS/E System Generation Manual*).

The /DETACH switch has the format:

[space]/DET[A[C[H]]]

This switch indicates that the invoked program should be run "detached." In this state, channel 0 (the terminal associated with the job) is marked as detached while the program is running. This can be useful for noninteractive programs; it frees the terminal for other use and prevents the user from interrupting the job by typing a CTRL/C.

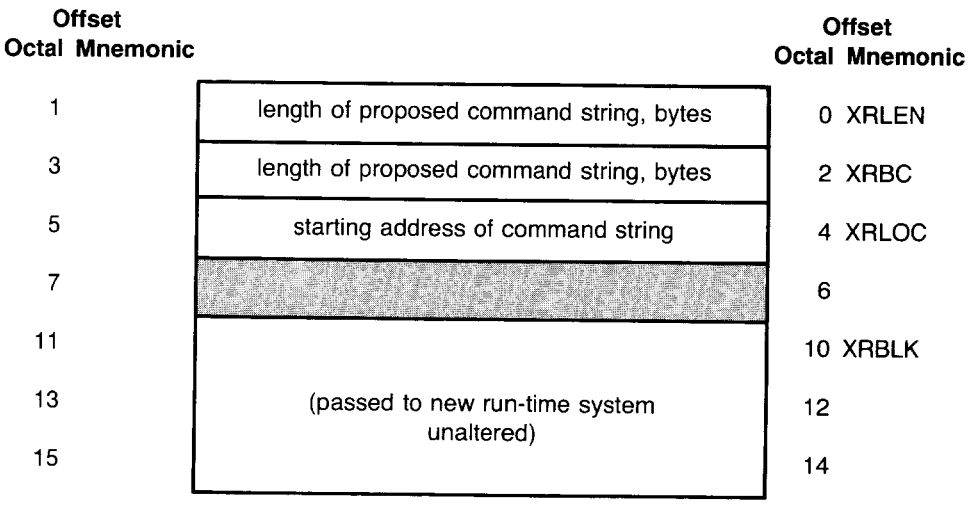
Remember that the monitor simply examines these switches and passes the information on to the run-time system. The run-time system is responsible for doing the appropriate processing. For more information on the CCL facility, see the *RSTS/E Programming Manual.*

NOTE

This directive should not be used from a user job image running with the RT11 run-time system, since the lowest 1000 (octal) bytes are used by RT11 differently from other run-time systems.

Data Passed

XRB



XRB+XRLEN The length of the proposed CCL command string, in bytes.

XRB+XRBC The length of the proposed CCL command string, in bytes, is also passed in this word.

XRB+XRLOC The starting address of the proposed CCL command string.

XRB+XRBLK The remaining three words will be unaltered here if the string is indeed a CCL command and the monitor takes over and does the equivalent of a .RUN directive. That is, the run-time system that is given control as a result of this command will find the same three words here that the caller left. These three words will also be unchanged if control returns back to the caller for any reason. See the .RUN monitor call and the P.RUN entry point for details.

Data Returned

No real arguments are returned to the calling program. (Data passed on to the run-time system if the call is successful is described in the P.RUN description in Section 2.5.) If the call is unsuccessful, the last three words of the XRB are unaltered, but the first four words will be random. In addition, an error code will be returned in the first byte of the FIRQB.

Errors

(none) No error is returned if the command part of the string passed was not a valid CCL command. The contents of CORCMN have not been altered; the XRB (except for the last three words) has been altered.

BADCNT The first three words of the XRB, which describe the CCL command string, are illegal.

LINERR The indicated string is too long to be passed in CORCMN.

BADSWT An illegal switch was given in the CCL command string.

BDNERR An illegal number was the argument to one of the switches found in the CCL command string. For example, the "n" value in the /SIZE switch was greater than the system-wide maximum for a user job image (see SWAP MAX, *RSTS/E System Generation Manual*).

xxxxxx Any other error returned results from the monitor's execution of a .RUN directive for the program. (See the .RUN directive, Section 3.20.)

Example

The following example asks the monitor to check a 72–byte string beginning at location BUFFER to see if it is a CCL command. See Section 3.1.4 for information on the CLRXRB routine.

```
BUFFER:   .BLKB   72.
            .
            .
            .
          CALL    CLRXRB              ;CLEAR XRB
          MOV     #72.,XRB+XRLEN      ;SET LENGTH
          MOV     #72.,XRB+XRBC       ;SET LENGTH AGAIN
          MOV     #BUFFER,XRB+XRLOC   ;SET STARTING ADDRESS
          .CCL
```

3.4 .CHAIN — Execute Under Same RTS – Not Privileged

Form

```
.CHAIN
```

Function

The .CHAIN directive is the same as the .RUN directive, except that it returns an error* if the program to be run would cause a new run-time system to be entered. That is, if the call succeeds, the current run-time system is entered at the P.RUN entry point. In addition, there is no change in the user job image size.

This call can be used to bypass the special protection afforded by the "compiled file" bit in the protection code. That is, this protection can be supplied by the run-time system without this bit being set in the protection code of the file. The .CHAIN reenters the run-time system so that a user cannot take control of the file once it is open on channel 15.

Please see the .RUN directive, Section 3.20, for Data Passed, Data Returned, and Errors. For the example, substitute .CHAIN for .RUN.

NOTE

This directive should not be used from a user job image running under the RT11 run-time system. Use the RT11 .CHAIN directive (Section 7.2) to transfer control to another program running under RT11.

---

* The error returned is NORTS (in byte 0 of the FIRQB).

3.5 .CLEAR — Clear Keyword Bits – Privileged and Not Privileged

Form

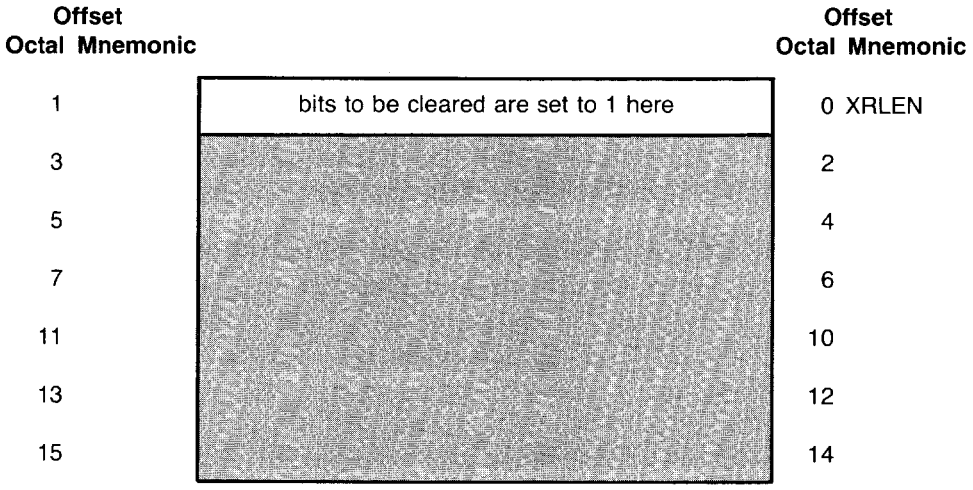
```
.CLEAR
```

Function

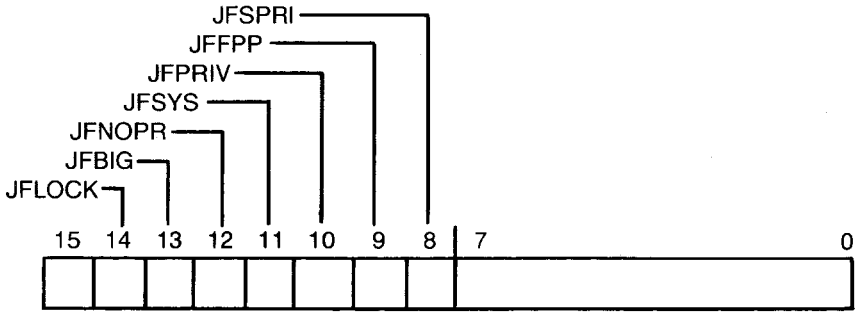
The .CLEAR directive can be used to clear certain bits in the keyword (KEY) location in the user job image (Section 2.4). The bits to be cleared are passed to the monitor in the XRB.

Data Passed

XRB



XRB+XRLEN The bits to be cleared are set to 1 here.



JFLOCK Clearable by any caller. Clearing JFLOCK indicates that the job wishes to be swapped. When JFLOCK is clear, the monitor will swap the job (that is, the user job image) to and from disk as necessary.

JFBIG Clearable by any caller. Clearing JFBIG drops the job's right to exceed its private memory maximum.

JFNOPR Not clearable by any caller; masked off.

JFSYS Clearable by any caller. If the job has temporary privileges and this bit is cleared in a .CLEAR call, the temporary privileges are temporarily lost.

JFPRIV This bit, when set, indicates permanent privilege. It cannot actually be cleared with the .CLEAR call. However, if a .CLEAR call is issued with this bit indicated for clearing, any temporary privilege that the job has or had is permanently lost.

JFFPP Clearable by any caller. Clearing JFFPP indicates that this job no longer wishes the hardware floating-point unit (if any) to be swapped along with the job's normal context information.

JFSPRI Clearable by any caller. Clearing JFSPRI lowers the job's run priority by one-half step. (That is, it clears bit 2 of the system-controlled low-order three bits of the run priority. See UTILTY, *RSTS/E System Manager's Guide.*)

All other bits in the XRB are masked off; that is, the corresponding bits in KEY cannot be cleared by the job with the .CLEAR directive.

Data Returned

No data is returned with the .CLEAR directive.

Errors

No errors are possible with the .CLEAR directive.

Example

The following code clears the JFLOCK bit. See Section 3.1.4 for information on the CLRXRB routine.

```
CALL    CLRXRB              ;CLEAR XRB
MOV     #JFLOCK,XRB+XRLEN   ;SET JFLOCK FOR CLEAR
.CLEAR
```

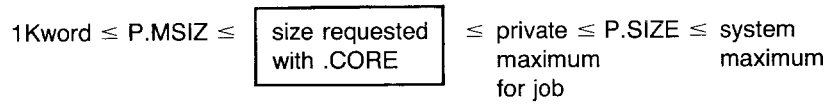
3.6 .CORE — Change Memory Size – Not Privileged

Form

```
.CORE
```

Function

The .CORE directive asks the monitor to change the amount of memory currently allocated for the user job image (low segment) for this job. The monitor preallocates space for a user job image at the time a .RUN directive is issued. The space is based on the file's size (PF.CSZ = 1 in the P.FLAG word of the pseudo-vector region) or is equal to the P.MSIZ word in the pseudo-vector region. This initial size can be changed with the .CORE directive as many times as desired, as long as the requested size (1) falls within a maximum and minimum value and (2) does not overlap any address windows created by the job for use with resident libraries (see .PLAS, Section 3.15). The monitor first checks the size requested against maximum and minimum values:



The monitor determines the maximum allowable amount of space for a user job image as follows:

1. Set <max> (the maximum size that a job image can be) to the system-wide maximum. This maximum is set by the system manager at startup time (see SWAP MAX, *RSTS/E System Generation Manual*).
2. If the maximum user job image size imposed by the run-time system (P.SIZE in the pseudo vectors) is less than the current <max>, set <max> to P.SIZE.
3. If the job's private memory maximum is less than <max> and if JFBIG in the job's keyword (KEY) is 0, then set <max> to the job's private memory maximum. The system manager can set a particular job's private memory maximum with the UTILTY system program. You can also set a private memory maximum with the UU.PRI subfunction of the .UUO directive, Section 3.32. A job's private memory maximum is initially defaulted to the system maximum.

Thus, the size requested with .CORE is checked against <max>, as determined by the three steps above. The size requested with .CORE is also checked against a minimum (the P.MSIZ word in the pseudo vectors, which must be greater than or equal to 1K words). Any size between P.MSIZ and <max>, inclusive, is legal.

There are two special cases:

1. If the size requested with .CORE is exactly equal to the run-time system's minimum size (P.MSIZ), that request is considered legal even if the requested size is greater than the job's private maximum.
2. If the size requested with .CORE is less than the job image's current size and within the allowable bounds for the run-time system, but it is still larger than the private maximum, that request is considered legal. This could happen if JFBIG was equal to 1, allowing the current size to be greater than the private maximum, and then JFBIG was cleared to 0. The monitor would still allow the size greater than the private maximum.

If .CORE requests a decrease in job image size, no further checks are made. If .CORE requests an expansion and the size is legal according to the tests described above, the monitor then checks the base APRs of any address windows created by the job (see the CRAFQ subfunction of .PLAS, Section 3.15.2). If the size requested in the .CORE overlaps a created address window, the .CORE fails and returns an error.

If .CORE requests a legal expansion that cannot be made "in place," that is, if there is not enough free memory available for the expansion, the job will be swapped out and swapped back in at the larger size. (This swap will occur even when JFLOCK = 1 in the keyword (KEY).)

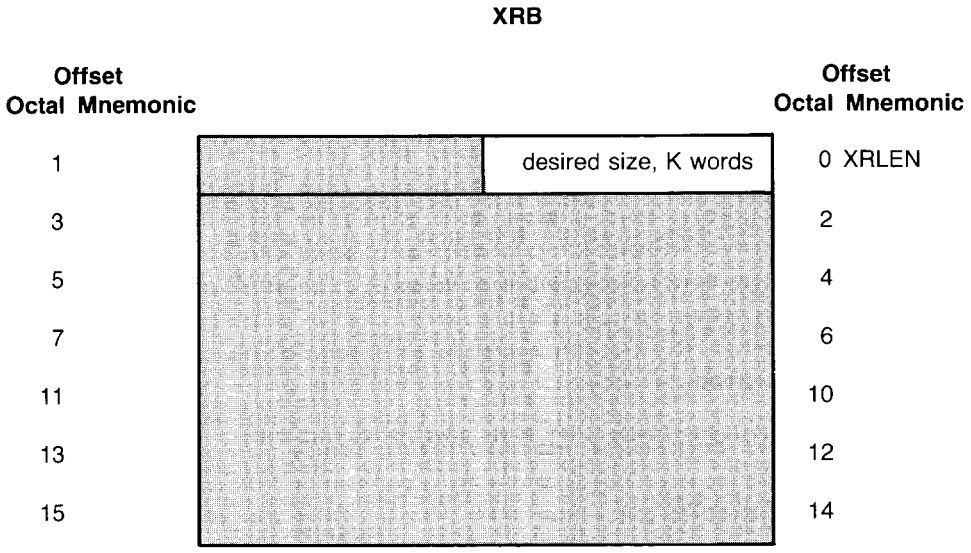
When a user job image expands, the content of the newly added memory is zeroed as protection against a malicious user reading memory to look for passwords.

NOTE

This directive should not be used from a user job image running under the RT11 run-time system. Expanding memory size should be done through the RT11 emulator, using the appropriate RT11 directive.

When the image runs under the RSX run-time system or its derivatives, use the EXTK$ directive to extend the task size, so that subsequent GTSK$ directives can return the task size correctly.

Data Passed



XRB + XRLEN — This byte contains the desired size for the user job image, in K words.

Data Returned

Other than a possible error in the first byte of the FIRQB, no data is returned with the .CORE directive.

Errors

EDBMCE — The requested user job image size is illegal. It is either too large or too small according to the rules described above, or it overlaps a mapped window.

Example

The following code requests a user job image of 24K words. See Section 3.1.4 for information on the CLRXRB routine.

```
CALL      CLRXRB          ;CLEAR THE XRB
MOVB      #24.,XRB+XRLEN  ;SET XRB TO INDICATE 24K WORDS
.CORE
```

3.7 .DATE — Return Current Date and Time – Not Privileged

Form

```
.DATE
```

Function

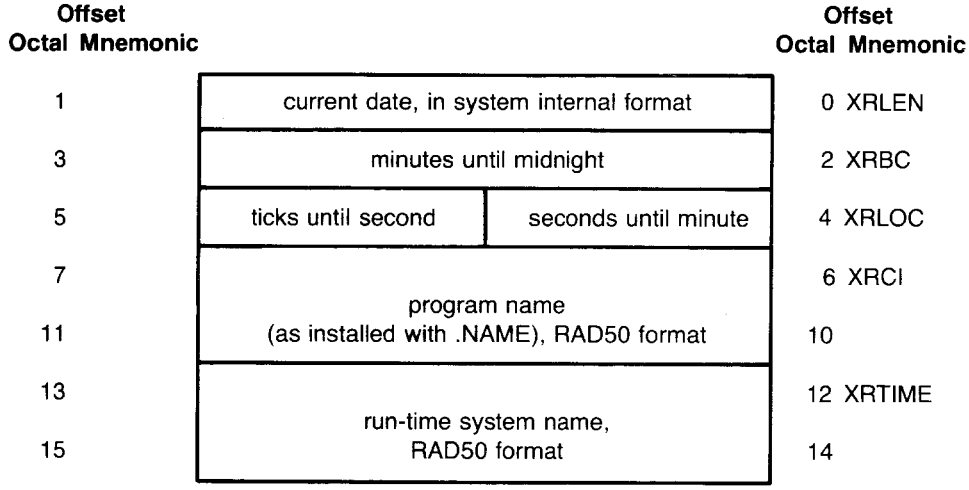
The .DATE directive returns the current date and time, the current program name (as installed by .NAME, Section 3.13), and the current run-time system name in the XRB.

Data Passed

No data is passed with the .DATE directive.

Data Returned

XRB



XRB+XRLEN — The current date, in system internal format. The monitor calculates the date as:

$[(\text{year} - 1970) * 1000_{10}] + (\text{day-within-year})$

where day-within-year is 1 for January 1, 2 for January 2, and so forth. (Every leap year, the day-within-year value for March 1 and following is one higher than in other years.)

XRB+XRBC — The number of minutes until midnight. A value of 1440 (decimal) is midnight; 720 is noon; 0 is never returned.

XRB+XRLOC — This byte contains the number of seconds until the next minute. A value of 60 is xx:xx:00; 1 is xx:xx:59; 0 is never returned.

XRB + 5	This byte contains the number of "ticks" until the next second. A tick is either 1/60th or 1/50th of a second, depending on the clock in use and/or the line frequency. (Systems running with the KW11P clock at crystal speeds, rather than at line frequency, have a "tick" of 1/50th of a second. If the system is operating off a 60 Hz power line, a "tick" is 1/60th of a second.)
XRB + XRCI	The current program name (as installed by the most recent .NAME monitor call) is returned here as two words in RAD50 format.
XRB + XRTIME	The current run-time system name is returned here as two words in RAD50 format.

Errors

No errors are possible with the .DATE monitor call.

Example

Since no data is passed to the monitor, the call is simply:

```
.DATE
```

3.8 .ERLOG — Log an Error from RTS – Not Privileged

Form

```
.ERLOG
```

Function

The .ERLOG directive can be issued from the high segment (run-time system) only. It allows the run-time system to log an error into the RSTS/E error log file, which can then be printed by the system manager (see the *RSTS/E System Manager's Guide*). For example, you might wish to place an entry into the RSTS/E error logging scheme on a hardware floating-point unit exception that has an illegal error code — the monitor makes no such checks.

This directive can be issued only from the job's current run-time system (high segment). Since this call is not privileged, it is deemed wise not to allow users to fill up the system error log with unimportant errors. If .ERLOG is issued from the user job image (low segment), it is ignored.

Data Passed

The .ERLOG directive records in the system error log file the contents of the program counter (PC) and program status word (PS) at the time of the call, as well as the contents of the general registers (R0 through R5). These registers will then be displayed at the system manager's request. Hence, the registers should contain whatever information you wish to record when the .ERLOG is executed.

Data Returned

No data is returned by .ERLOG.

Errors

No error is possible with .ERLOG.

Example

Assuming the general registers contain relevant information, the call is simply:

```
.ERLOG
```

3.9 .EXIT — Exit to Default Keyboard Monitor – Not Privileged

Form

```
.EXIT
```

Function

The .EXIT directive returns control to the default keyboard monitor at the P.NEW entry point (Section 2.5)*. When a program exits, it would normally pass control to the job's keyboard monitor with the .RTS directive (Section 3.19). You can use the .EXIT call as a backup to return control to the default keyboard monitor should the .RTS fail, or for any other reason when it is desirable to enter the default keyboard monitor at the entry point specified by P.NEW. The .EXIT directive needs no arguments and never returns in-line to the caller.

Data Passed

The .EXIT directive needs no arguments; however, the three words beginning at XRB+10 are passed unaltered to the default keyboard monitor. The monitor also passes information to the default keyboard monitor when .EXIT is executed; for details, see the discussion of the P.NEW entry point in Section 2.5.

Data Returned

No data is returned with .EXIT; control never returns in-line.

Errors

No errors are possible with .EXIT.

Example

Since no data is passed or returned with .EXIT, the call is simply:

```
.EXIT
```

NOTE

This (RSTS/E) directive should not be used from a user job image running under the RT11 run-time system. The proper way to terminate such a program is to exit to the RT11 emulator, which will return control to the job's keyboard monitor.

---

* The default keyboard monitor is defined by the system manager. It displays the prompt you see immediately after login.

3.10 .FSS — Check File Specification String – Not Privileged

Form

```
.FSS
```

Function

The .FSS directive examines a string of characters presumed to be a file specification and, if possible, converts it to the internal RSTS/E file specification format; that is, the FIRQB format. The monitor returns information to the XRB describing what it found in the string and returns the converted file specification to the FIRQB. Thus, programs that manipulate files can use .FSS to translate a user-typed string to the FIRQB format.

The monitor examines the string from left to right and stops without error when it encounters:

1. The end of the string.
2. An equal sign (= ASCII code 075 octal).
3. A semicolon (; ASCII code 073 octal).
4. A slash (/ ASCII code 057 octal) that is followed by anything other than the switches described below, which the monitor translates to the FIRQB format:

   - /CL[USTERSIZE]:[–][#]n[.]

     where n is the cluster size used in opening files and devices. The variable n may specify a value ranging from -32768_{10} through 32767_{10}, inclusive.

   - /MO[DE]:[#]n[.]

     where n is the mode used in opening files and devices. The variable n may specify a value between 0 and 32767_{10}, inclusive.

   - /FI[LESIZE]:[#]n[.] or /SI[ZE]:[#]n[.]

     where n is the filesize used in opening files and devices. The value of n defines the file's size in 512–byte blocks and, with the large file capability for disk, can range from 0 through $2^{23}-1$, or greater than 8 million blocks.

   - /PO[SITION]:n

     where n is the position used in creating files (to position block 1 of the file at a device cluster). The variable n may specify a value between 0 and $65{,}535_{10}$, inclusive.

Function (Cont.)

- /PR[OTECTION]:[#]n[.]

  where n is the protection code used in opening or creating files. The variable n may specify a value between 1 and 255_{10}, inclusive. The value of n determines the file's protection from users, as described in the *RSTS/E System User's Guide.*

The brackets [ ] in the switches above enclose optional characters. Where more than one character is enclosed in brackets, any or all of the enclosed characters can be omitted. For example, MO, MOD, and MODE would all be accepted and the following quantity translated to the mode location in the FIRQB. The value n is assumed to be decimal, unless the optional pound sign [#] appears, indicating that n is octal. The optional decimal point also indicates a decimal value.

5. A comma (, ASCII code 054 octal). An exception is the comma separating the project-programmer numbers in a ppn.

The monitor will translate the following components in a file specification string:

device name — A device name can be either a logical device name or a physical device name:

logical — A logical device name is a string of alphanumeric characters terminated with a colon (:). Only the first six characters are examined; the remainder are ignored. The monitor first checks a logical device name against the user's own logical device name assignments in USRLOG (or its equivalent, as defined in the XRB). If the monitor finds a definition, it returns the physical device name associated with that logical device name to the FIRQB. If the logical name is not found in the user-logical area, the monitor then makes a similar search against its own internal table of system-wide user logicals. If the logical name is not there either, the monitor simply returns the logical device name and sets a flag in the XRB to indicate that it could make no association. For a logical device name beginning with an underscore, the monitor does not attempt any translation to a physical device name.

physical — A physical device name consists of two alphabetic characters optionally followed by digits and ended with a colon (:). The digits are translated as decimal and must have a value between 0 and 127 (decimal). Leading zeros are allowed.

account or ppn

A project-programmer number can be expressed either as a single special character or as two separate numbers enclosed in square brackets [ ] or parentheses ( ) and separated by a comma.

The following special characters are translated as described:

$ — The $ is translated to the account assigned by the system manager to the system library. It is usually assigned as [1,2].

! — The ! is translated to an account assigned by the system manager. It is usually assigned as [1,3].

% — The % is translated to an account assigned by the system manager. It is usually assigned as [1,4].

& — The & is translated to an account assigned by the system manager. It is usually assigned as [1,5].

— The # is translated to the caller's "group library." It is always equivalent to [*proj*,0] where *proj* is the project number of the user issuing the .FSS directive.

@ — The @ is translated to the caller's "assignable ppn," the USRPPN value described in Section 2.4. If USRPPN is set, its value is placed in the FIRQB at offset FQPPN. If USRPPN is 0, a string with an @ causes an error.

[n,m] — This is the explicit construct for a project-programmer number. The value n specifies the project number, and m specifies the programmer number. The variables n and m may specify any value from 0 through 254 (decimal) inclusive, except for [0,0]. If a pound sign (#) precedes either n or m, the string is assumed to specify an octal value. Either n or m, or both, can also be the asterisk (*). The * is converted to 255 (decimal) and placed in its corresponding FIRQB location. The asterisk character indicates a "wildcard" specification.

(n,m) — This is an alternate way to specify an explicit project-programmer number. The same rules for n and m apply as when they are enclosed by brackets.

file name — A file name may consist of alphanumeric characters and the question mark (?). It is the only field in the file specification with no explicit delimiter. Only the first six characters are examined; the rest are ignored. The asterisk character (*) is also an acceptable file name. It is parsed to two words of RAD50, where each RAD50 character is the "unused" code (35_8). Each question mark is also converted to this "unused" code. This indicates that the file name field is "wild." (The LOKFQ subfunction of CALFIP (Section 3.2.11) and UU.LOK subfunction of .UUO (Section 3.32.26) can be used to look up wildcard files.)

type — A file type may consist of alphanumeric characters and the question mark (?), preceded by a period (.). The asterisk (*) is also an acceptable file type. It is translated to one word of RAD50, where each RAD50 code is the "unused" code (035 octal). Each question mark in the file type is also converted to this "unused" code. This indicates that the file type field, or character in the file type field, is "wild."

If given, the file type must always follow the file name.

protection code — A file protection code can consist of numeric digits enclosed by angle brackets. The general form of a protection code is <nnn>, where n may be numeric characters indicating a value from 0 through 255_{10}. If the numeric characters are preceded by a pound sign (#), they are converted as specifying an octal value. If no file protection code is specified in the string and a default value has been assigned in USRPRT (see Section 2.4), the default value will be placed in the FIRQB.

The components described above can appear in the string in any order, with the exception of the file type and ppn. The file type must follow the file name, if specified. In addition, if the device name is a system or user logical device name that has an account (ppn) associated with it, the position of an explicit ppn in the file specification string is significant. If the order is device:ppn, then the explicit ppn overrides the ppn associated with the logical device name.

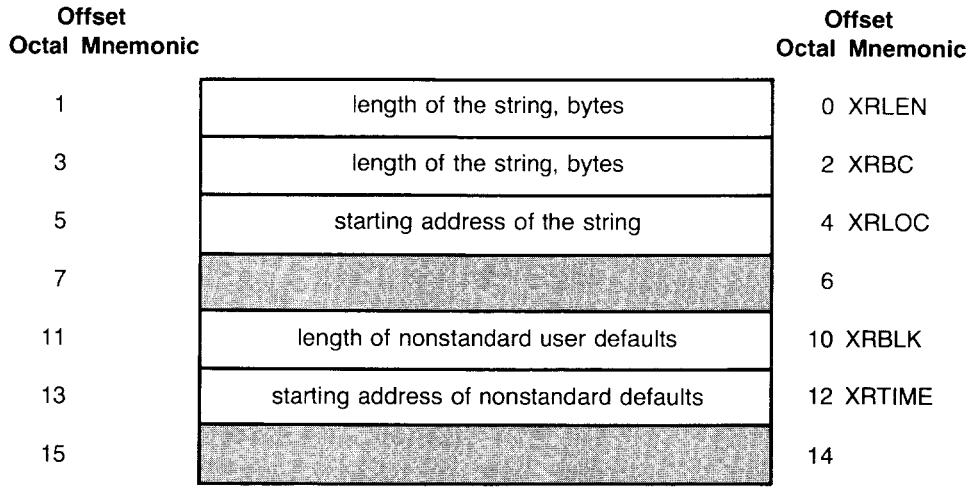
If the order is [ppn]device:, then an illegal device name error is reported. (This error occurs only when a ppn is associated with the logical name.)

NOTE

Do not use this directive from a user job image running under the RT11 run-time system, since the user logical area is not in the standard location.

Data Passed

XRB



XRB + XRLEN — This word contains the length of the character string to be translated, in bytes.

XRB + XRBC — This word also contains the length of the character string to be translated, in bytes.

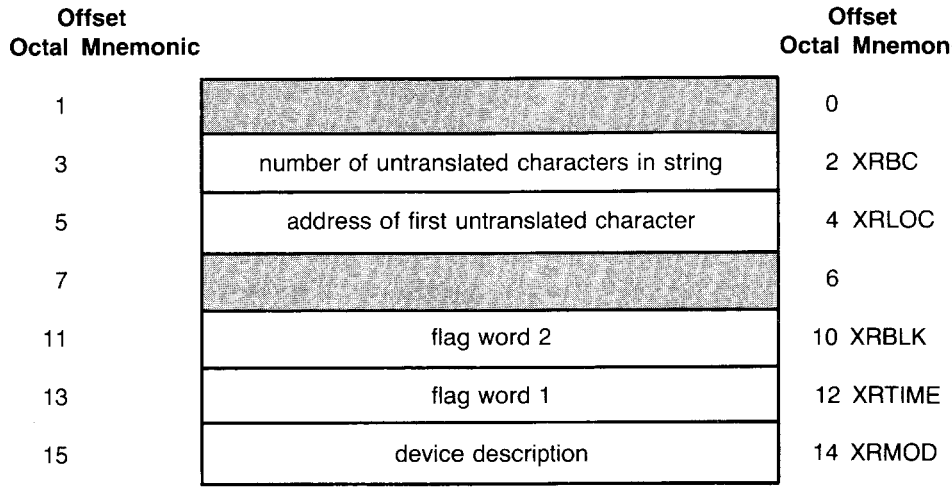
XRB + XRLOC — This word contains the starting address of the string to be translated.

XRB + XRBLK — If the user logical information (USRPPN, USRPRT, and USRLOG) is in its standard location, this word is passed as 0. If some nonstandard set of locations is being used, then the length of that information, in bytes, is specified here.

XRB + XRTIME — If the word at XRB + XRBLK is nonzero, then this word defines the starting location for the user logical information. (The order of the information is assumed to be the same as in its standard location; that is, the user logical ppn, user logical protection code, user logical device name table. The format is also expected to be the same — see USRPPN, USRPRT, and USRLOG descriptions in Section 2.4 for details.)

Data Returned

XRB



XRB+XRBC This word contains a count of the untranslated characters in the string. If all characters were translated, the value of this word is 0.

XRB+XRLOC This word contains the address of the first untranslated byte of the string. (If XRB+XRBC is 0, this word identifies the end of the string.)

XRB+XRBLK Bit flags describing the translated string. (Note: This word is the same as "flag word 2" for the BASIC–PLUS file name string scan SYS call, as described in the *RSTS/E Programming Manual.*)

Bit	Octal Value	Meaning
0	1	A file name was found in the source string and is returned as two words in RAD50 format at FIRQB+FQNAM1.
	0	No file name was found (and bits 1 and 2 of this word are also 0).
1	2	The translated file name consisted of a single * character and has been translated to two words at FIRQB+FQNAM1 consisting of the RAD50 representation of the string "??????".
	0	The translated file name was not an * character.
2	4	The file name contained at least one ? character.
	0	The file name did not contain any ? characters.

(continued on next page)

Bit	Octal Value	Meaning
3	10	A period (.) was found in the source string.
	0	No period was found, implying that no file type was specified (and bits 4, 5, and 6 of this word are also 0).
4	20	A file type was found (that is, the field after the period was not null).
	0	No file type was found (the field after the period was null), and bits 5 and 6 of this word are also 0.
5	40	The file type was an * character and is returned in the word at FIRQB+FQEXT as the RAD50 representation of the string "???".
	0	The file type was not an * character.
6	100	The file type contained at least one ? character.
	0	The file type did not contain any ? characters.
7	200	A project-programmer number was found in the source string.
	0	No project-programmer number was found (and bits 8 and 9 of this word are also 0).
8	400	The project number was an * character. (That is, the project-programmer number was of the form [*,n].) The byte at FIRQB+FQPPN+1 is returned as 255_{10} (377_8).
	0	The project number was not an * character.
9	1000	The programmer number was an * character. (That is, the project-programmer number was of the form [n,*].) The byte at FIRQB+FQPPN is returned as 255_{10} (377_8).
10	2000	A valid protection code was found.
	0	No protection code was found.
11	4000	No file protection code was found in the string, but there was a default output file protection code in location USRPRT. The default has been returned in the FIRQB.
	0	The user-assignable default protection code (at location USRPRT) was not used. Either zero or the protection code given in the string is returned to the FIRQB.
12	10000	A colon (:), but not necessarily a device name, was found in the source string.
	0	No colon was found (no device was specified); bits 13, 14, and 15 of this word are also 0.
13	20000	A device name was found in the source string.
	0	No device name was found; bits 14 and 15 of this word are also 0.
14	40000	The device name in the string was a logical device name.
	0	The device name in the string was an actual device name; bit 15 of this word is also 0.

(continued on next page)

Bit	Octal Value	Meaning
15	100000	This bit set indicates an invalid device name. (The characters that were specified are simply returned at FIRQB + FQDEV as two words in RAD50 format.) This bit can be set in one of two ways: 1. If the device name contained an underscore but was not a recognizable device name for any device on the system, this bit is set. 2. If the device name did not contain an underscore but the name could not be translated to a physical device name, this bit is set.
	0	The device name specified, if any, was either an actual device name or a logical device name to which a physical device has been assigned. The physical device name has been returned to the word at FIRQB + FQDEV as two ASCII characters, and the unit information has been returned appropriately at FIRQB + FQDEVN.

XRB + XRTIME Remaining bit flags describing what was translated. Some of these bits duplicate information returned at XRB + XRBLK. DIGITAL recommends that you use the bits at XRB + XRBLK to allow for enhancements in future releases. (Note: This word is the same as "flag word 1" for the BASIC–PLUS file name string scan SYS call, described in the *RSTS/E Programming Manual.*)

Bit	Octal Value	Meaning
0	1	The /CLUSTERSIZE:n switch was specified.
	0	The /CLUSTERSIZE:n switch was not specified.
1	2	Either the /MODE:n or /RONLY switch was specified.
	0	Neither the /MODE:n nor the /RONLY switch was specified.
2	4	The /FILESIZE:n or /SIZE:n switch was specified.
	0	Neither the /FILESIZE:n nor the /SIZE:n switch was specified.
3	10	The /POSITION:n switch was specified.
	0	No /POSITION:n switch was specified.
4–7		(Not currently used.)

(continued on next page)

Bit	Octal Value	Meaning
8	400	A file name was found in the source string (and is returned as two words in RAD50 format at FIRQB + FQNAM1). Note that this is the same meaning as for bit 0 at XRB + XRBLK.
	0	No file name was found in the source string.
9	1000	A period (.) was found in the source string. Note that this is the same meaning as for bit 3 at XRB + XRBLK.
	0	No period was found in the source string, implying that no file type was specified either.
10	2000	A project-programmer number was found in the source string. Note that this is the same meaning as for bit 7 at XRB + XRBLK.
	0	No project-programmer number was found.
11	4000	A valid protection code was found. Note that this is the same meaning as for bit 10 at XRB + XRBLK.
	0	No protection code was found.
12	10000	A colon (but not necessarily a device name) was found in the source string. Note that this is the same meaning as for bit 12 at XRB + XRBLK.
	0	No colon was found, implying that no device could have been specified.
13	20000	Device name was specified and was a logical device name. Note that this is the same meaning as for bit 14 at XRB + XRBLK.
	0	Device name (if specified) was an actual device name. (If device name was not specified, this bit will also be 0.)
14		(Not currently used.)
15	100000	Source string contained wildcard characters (either ? or * or both) in file name, type, or project-programmer number fields. In addition, the device name specified, although a valid logical device name, does not correspond to any of the logical device assignments currently in effect or contains an underscore as the first character. Flag word 2 contains more specific information.
	0	None of the above.

XRB + XRMOD This word contains the device description (the same information returned by the BASIC–PLUS STATUS variable and returned at FIRQB + FQFLAG when a file or device is opened with the OPNFQ or CREFQ subfunctions of CALFIP). The device handler index is in the low byte and descriptive flags are in the high byte.

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job number * 2	2	FQJOB
5	FQSIZM	MSB of file size		4	
7		project number	programmer number	6	FQPPN
11		file name (two words in RAD50 format)		10	FQNAM1
13				12	
15		file type (in RAD50 format)		14	FQEXT
17		LSB (least significant bits) of file size		16	FQSIZ
21				20	
23		mode parameter		22	FQMODE
25				24	
27	FQPROT	protection code	=377, explicit prot. code	26	FQPFLG
31		device name		30	FQDEV
33		≠0,unit number real	device unit number	32	FQDEVN
35		cluster size parameter		34	FQCLUS
37		position parameter (dcn for first block)		36	FQNENT

NOTE

For each of the following field definitions that begin with the word "If," a corresponding statement applies: "If not, the field is left alone." That is, you can insert values in the FIRQB before executing the .FSS to serve as default values for fields when the .FSS returns no result.

FIRQB+FQJOB — The current job number times two.

FIRQB+FQSIZM — If a /FILESIZE:n or /SIZE:n switch was specified with n greater than 65,535, the most significant bits of the file size are contained in this byte.

FIRQB + FQPPN	If a project-programmer number was part of the translated string or if a logical name was found to be the same as a system or user logical name with an associated project-programmer number, this word contains the binary value of that ppn. The project number is in the high byte; the programmer number in the low byte. Any value returned here by .FSS has been verified by the monitor as syntactically correct; that is, within the range for ppns on a RSTS/E system.
FIRQB + FQNAM1	If a file name was encountered, it is translated to two words of RAD50, beginning at this location. If less than 6 characters, the file name is left-justified and padded with blanks (0 RAD50 characters).
FIRQB + FQEXT	If a file type was encountered, it is translated to 1 word of RAD50, beginning at this location. If less than 3 characters, the file type is left-justified and padded with blanks (0 RAD50 characters).
FIRQB + FQSIZ	If a /FILESIZE:n or /SIZE:n switch was encountered, the value of n is translated to binary and the least significant bits of the value are placed in this word. (If the file size indicated was greater than 65,535, the most significant bits are placed in the byte at FIRQB + FQSIZM.)
FIRQB + FQMODE	If a /MODE:n switch was encountered, the value specified is translated to binary and returned in this word. Bit 15 is set to indicate that a mode switch was translated and to differentiate between a mode of 0 and no mode at all. (Note that bit 15 must be set for a mode value to work on opens.)
FIRQB + FQPFLG	If a file protection code was encountered, a word is returned here. The high byte (FIRQB + FQPROT) is the binary value of the protection code, and the low byte (FIRQB + FQPFLG) is 377. Setting the low byte indicates that a protection code was specified and differentiates a protection code of 0 from no protection code at all.
FIRQB + FQDEV	If a device name was specified, it is returned here as two ASCII characters. If less than two characters were specified, the device name is left-justified and padded with blanks.

FIRQB+FQDEVN — If a device name but no explicit unit number was specified, this word is 0. If an explicit unit number was specified, then that unit number is in the low byte and 377_8 is in the high byte. Setting the high byte indicates an explicit device number and differentiates a device number of 0 from no device number at all.

NOTE

If a syntactically correct logical device name was encountered that could not be translated to a physical device name, then the logical device name is returned as two words of RAD50 starting at offset FQDEV. A status bit in the XRB is set to indicate that this was done.

FIRQB+FQCLUS — If a /CLUSTERSIZE:n switch was encountered, the value of n is returned here, in binary.

FIRQB+FQNENT — If a /POSITION:n switch was encountered, the value of n is returned here, in binary. (The value n is the device cluster number for the first block of the file.)

Errors

BADCNT — The first three words of the XRB are illegal or odd address for nonstandard user logical table.

BADNAM — Some illegal specification occurred in the string.

BADSWT — Some .FSS switch was encountered, but it was in an illegal format.

BDNERR — The numeric argument to one of the .FSS switches was illegal.

Example

The following code causes the monitor to scan a string beginning at location BUFFER as a possible file name. BUFFER is defined as an 80–byte area and is filled with zeros to terminate the string scan if what the user typed did not fill the buffer.

```
BUFFER:     .BLKW0    #40.
            .
            .
            .
 (read string into BUFFER from terminal)
            .
            .
            .
            MOV       #80.,XRB+XRLEN          ;DEFINE LENGTH
            MOV       #80.,XRB+XRBC           ;DEFINE LENGTH AGAIN
            MOV       #BUFFER,XRB+XRLOC       ;START OF BUFFER
            .FSS
 (test for error; if none, try open)
```

3.11 .LOGS — Check for Logical Devices – Not Privileged

Form

```
.LOGS
```

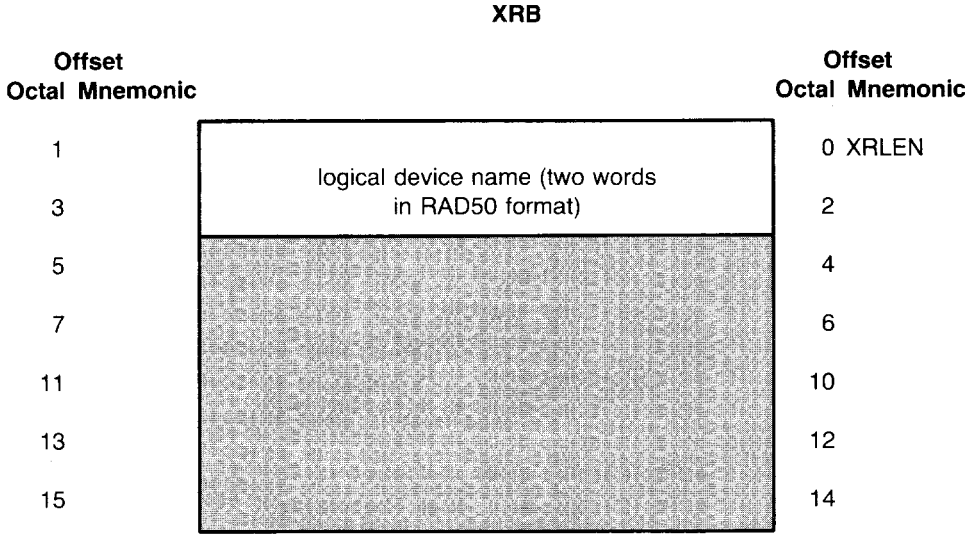
Function

The .LOGS directive (1) translates a system logical device name to a physical device name, (2) verifies that a physical device name is valid, or (3) obtains generic information about a particular device.

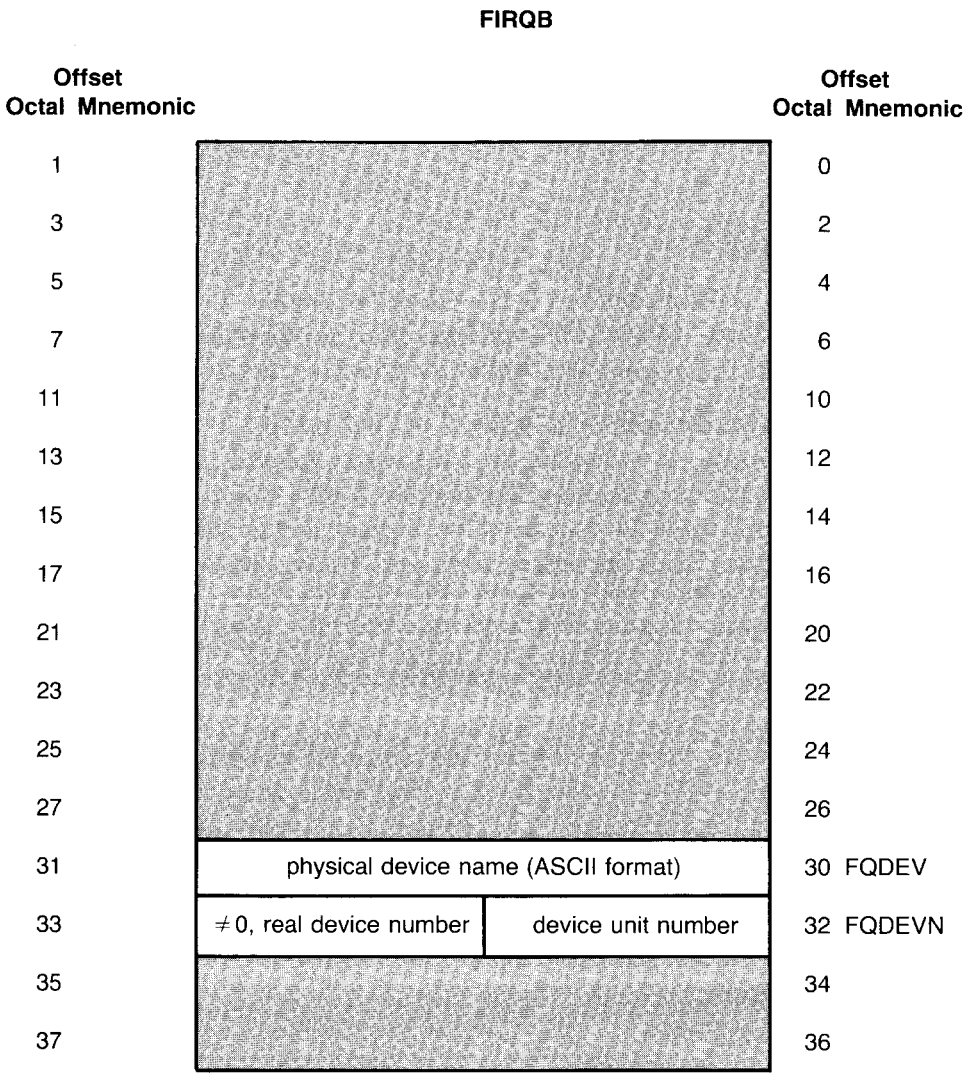
You specify either a logical device name, a physical device designation (name and, if relevant, unit number), or both, in the XRB and the FIRQB. The monitor compares the logical device name specified, if any, against its internal table of system-wide logical device names defined by the system manager. (This directive does not check user-specified logical names.) If it finds a match, the monitor returns the device designation associated with the logical device name to the FIRQB. This physical device designation consists of a name, unit number, and in some cases, a project-programmer number (ppn).

Next, the monitor checks the physical device designation (either the one passed or the one returned by the monitor in the logical-name translation) to be sure it is valid. If so, information describing the device is returned in the FIRQB.

Data Passed



XRB+XRLEN The logical device name to be checked is passed as two words in RAD50 format beginning at this location. If only a physical device name check or description is needed, then the first word of the XRB should be passed as 0. The physical device name is passed in the FIRQB.



FIRQB+FQDEV — The physical device name, as two ASCII characters. A value of 0 (and at offset FIRQB+FQDEVN) indicates the public disk structure (SY:). If only a translation from a logical device name to a physical device name is desired, a value of –1 can be passed here. (–1 is guaranteed not to be a valid physical device name.)

FIRQB+FQDEVN — The unit number of the physical device name is passed in this byte, in binary. To indicate an explicit device number, set the high byte (at FIRQB+FQDEVN+1) to some nonzero value. If the physical device name is of the form "XY:" (that is, no unit number is specified), then set the entire word at this offset to 0 to indicate no explicit unit number.

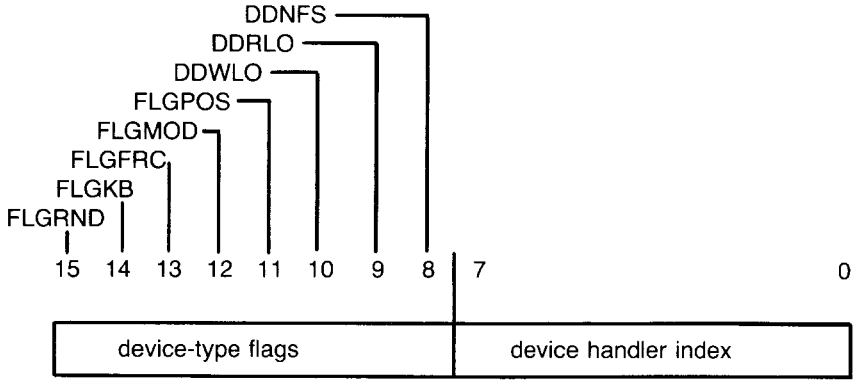
Data Returned

XRB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3			2	
5		logical device flag:–1 = no,–2 = yes,0 = N.A.	4	XRLOC
7		device description	6	XRCI
11		reasonable buffer size for device	10	XRBLK
13			12	
15			14	

XRB+XRLOC If the passed logical device name was translated successfully to a physical device name, this word is returned as –2; if not, as –1. If there was no logical device name specified, this word is returned as 0.

XRB+XRCI Description of the device. The low byte contains the device's handler index. The high byte contains a set of status flags.



High Byte — Device-Type Flags

The bits in the high byte of the flag word are set to indicate the type of file or device just opened:

FLGRND = 1 The device or file is random-access.
= 0 The device or file is sequential.

(continued on next page)

High Byte — Device-Type Flags (Cont.)

FLGKB = 1 The file or device is a terminal-type file or device (or is generically a terminal).
= 0 The file or device is not a terminal-type file or device.

FLGFRC = 1 The file or device is byte-oriented. That is, reads and writes handle data in byte units.
= 0 The file or device is block-oriented. Reads and writes handle data in block units.

FLGMOD = 1 The file or device accepts modifiers in reads and writes (Sections 3.17 and 3.33, XRB + XRMOD).
= 0 The file or device does not accept modifiers in reads and writes.

FLGPOS = 1 The file or device keeps track of its horizontal position and translates characters such as TAB to whatever is appropriate for the file or device. (You can determine the current horizontal position of such a device with the .POSTN directive.)
= 0 The file or device does not keep track of its horizontal position.

DDWLO = 1 The file or device has been write-locked (with the protection code in the open) or is generically a write-only device.
= 0 The file or device is not write-locked.

DDRLO = 1 The file or device has been read-locked (with the protection code in the open) or is generically a read-only device.
= 0 The file or device is not read-locked.

DDNFS = 1 The file or device is non-file-structured (or is generically not a file-structured device).
= 0 The file or device is file-structured.

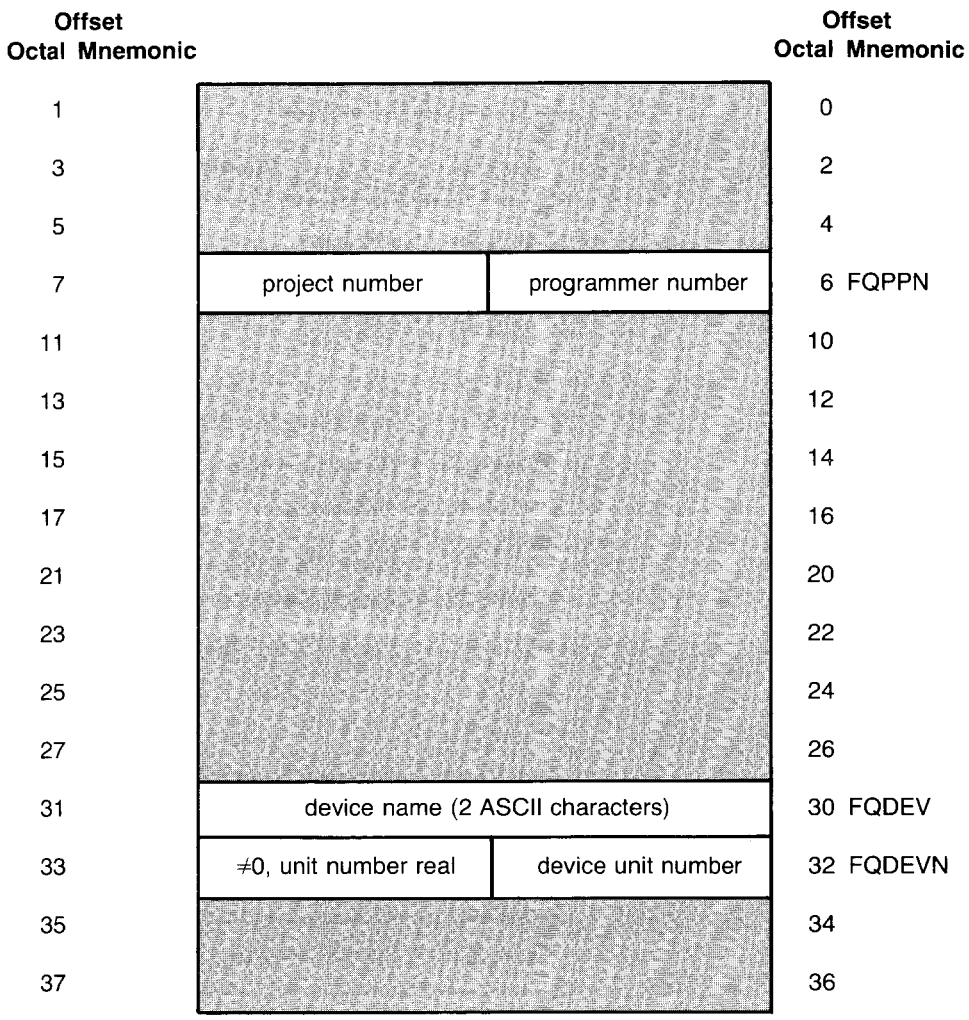
Low Byte — Device Handler Index

Bits 0–7 of the flag word contain a handler index that indicates the generic kind of device. Current values for this byte are:

Octal Value	Symbol	Meaning
0	DSKHND	All disks
2	TTYHND	All terminals
4	DTAHND	DECtape
6	LPTHND	All line printers
10	PTRHND	Paper tape reader
12	PTPHND	Paper tape punch
14	CDRHND	Card reader
16	MTAHND	Magnetic tape
20	PKBHND	Pseudo keyboards
22	RXDHND	Flexible diskettes
24	RJEHND	2780 remote job entry
26	NULHND	The null device
30	DMCHND	The DMC11/DMR11 DDCMP interface
36	DT2HND	DECtape II
40	KMCHND	KMC11
42	IBMHND	IBM interconnect
46	DMPHND	DMP11/DMV11 device

XRB+XRBLK If the physical device name is valid (either the one returned by the monitor's translation of logical device name or the one passed), this word contains the monitor's "best guess" as a reasonable buffer size for this device. (See .READ and .WRITE, Sections 3.17 and 3.33.)

FIRQB



FIRQB+FQPPN — If a logical device name was passed and it was translated to a device designation with an associated project-programmer number, the project-programmer number is returned in this word. Otherwise, this word is the same as before the .LOGS call was executed.

FIRQB+FQDEV — The physical device name, either the one returned when a successful translation of logical device name is made or the one passed, if no logical device name was passed. The physical device name is returned as two ASCII characters.

FIRQB + FQDEVN — The physical device unit number, either the one returned when a successful translation of logical device name is made or the one passed, if no logical device name was passed. The low byte contains the unit number, in binary. The high byte (at FIRQB + FQDEVN + 1) is either 0, to indicate no explicit device number, or nonzero, to indicate an explicit device number.

Errors

NODEVC — The physical device name (either the one passed or the one corresponding to the logical device name) is invalid.

Example

The following code asks the monitor to check the name "SYSDEV" to see if it is a defined system logical device name and, if so, to return the physical device name and characteristics to the XRB and FIRQB:

```
MOV     #^RSYS,XRB+XRLEN        ;SET XRB TO TRANSLATE LOGICAL
MOV     #^RDEV,XRB+XRBC         ;DEVICE NAME "SYSDEV"
.LOGS
```

3.12 .MESAG — Message Send/Receive

Form

```
   .
   .
   .
(Load FIRQB and/or XRB for appropriate subfunction)
   .
   .
   .
.MESAG
```

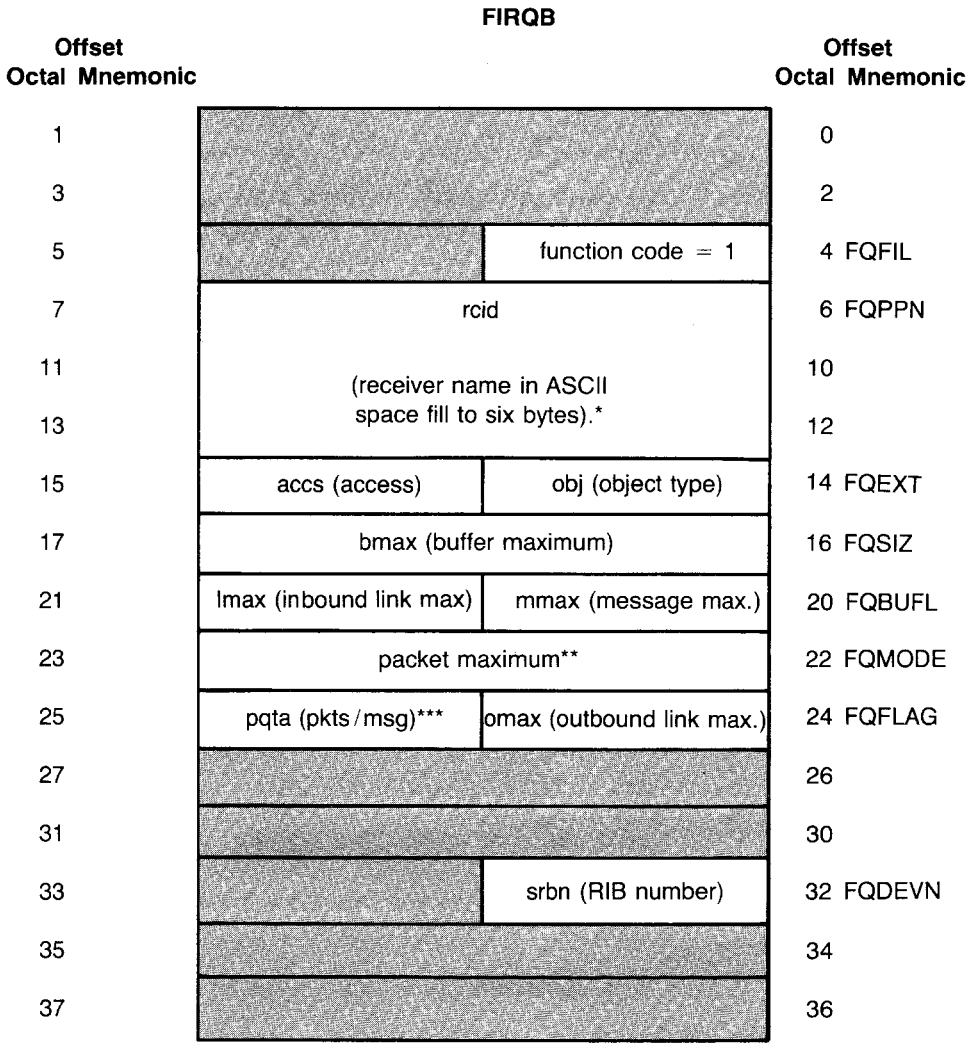
Function

The .MESAG directive provides access from a MACRO program to the RSTS/E local message send/receive services and, if your system is a DECnet/E system, to DECnet/E network message send/receive services.

This section contains FIRQB and XRB formats and error descriptions for local message send/receive. (Unless data passed and returned show specific values for the XRB, it should be all zeros.) For detailed information about each call, see the *RSTS/E Programming Manual.* For information about network message send/receive, see the manual *RSTS/E DECnet/E Network Programming in MACRO–11.*

3.12.1 Declare Receiver Subfunction – Privileged and Not Privileged

Data Passed



Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Declare Receiver subfunction of .MESAG.

---

* A nonprivileged caller must pass the job number in bytes 5 and 6 of the receiver name.

** Used only in an EMT logging program; specifies the maximum number of packets that can be queued at any one time. See the *RSTS/E Programming Manual* for more information.

*** Used only in an EMT logging program; specifies the number of packets that make up a complete message. See the *RSTS/E Programming Manual* for more information.

Errors

INUSE The calling job already exists in the system's list of declared receivers. This error may indicate a logic error in the program or that a previous program running under the same job number failed to remove itself from the receiver list before terminating. In the latter case, issue a remove receiver call, and then reissue the declare receiver. (It is common practice to code a remove receiver immediately before the declare receiver call.)

NOBUFS There were no small buffers available to hold the arguments passed in the declaration. Since the system's use of small buffers is dynamic, a retry may succeed.

PRVIOL
1. The specified RIB number is out of range.
2. A nonprivileged program tried to perform a function available to privileged programs only. (See the *RSTS/E Programming Manual* for details on privileged and nonprivileged use of declare receiver.)

FIEXST The receiver name passed is being used by another receiver, or the local object type you specified is "single instance" and is already in use.

BADFUO The receiver name, object type, and access parameters passed are inconsistent.

BADCNT The specified packet quota is out of range.

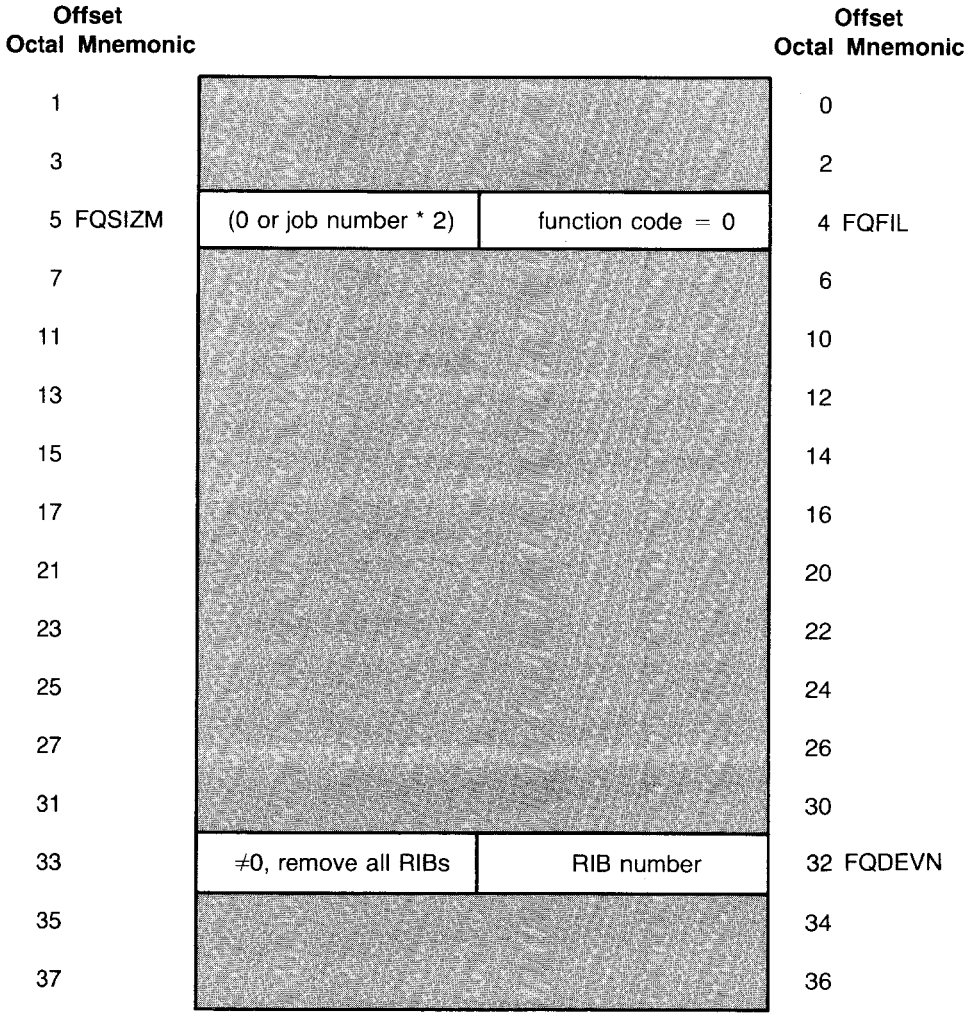
BADNAM
1. The receiver name passed contains nonprintable characters or leading or embedded spaces.
2. A nonprivileged job passed a nonblank receiver name whose fifth and sixth characters are not its job number.
3. The specified local object type is invalid.

ERRERR The call you attempted requires an optional feature (such as EMT logging) that is not available on your system.

3.12.2 Remove Receiver Subfunction – Privileged and Not Privileged

Data Passed

FIRQB



Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Remove Receiver subfunction of .MESAG.

Errors

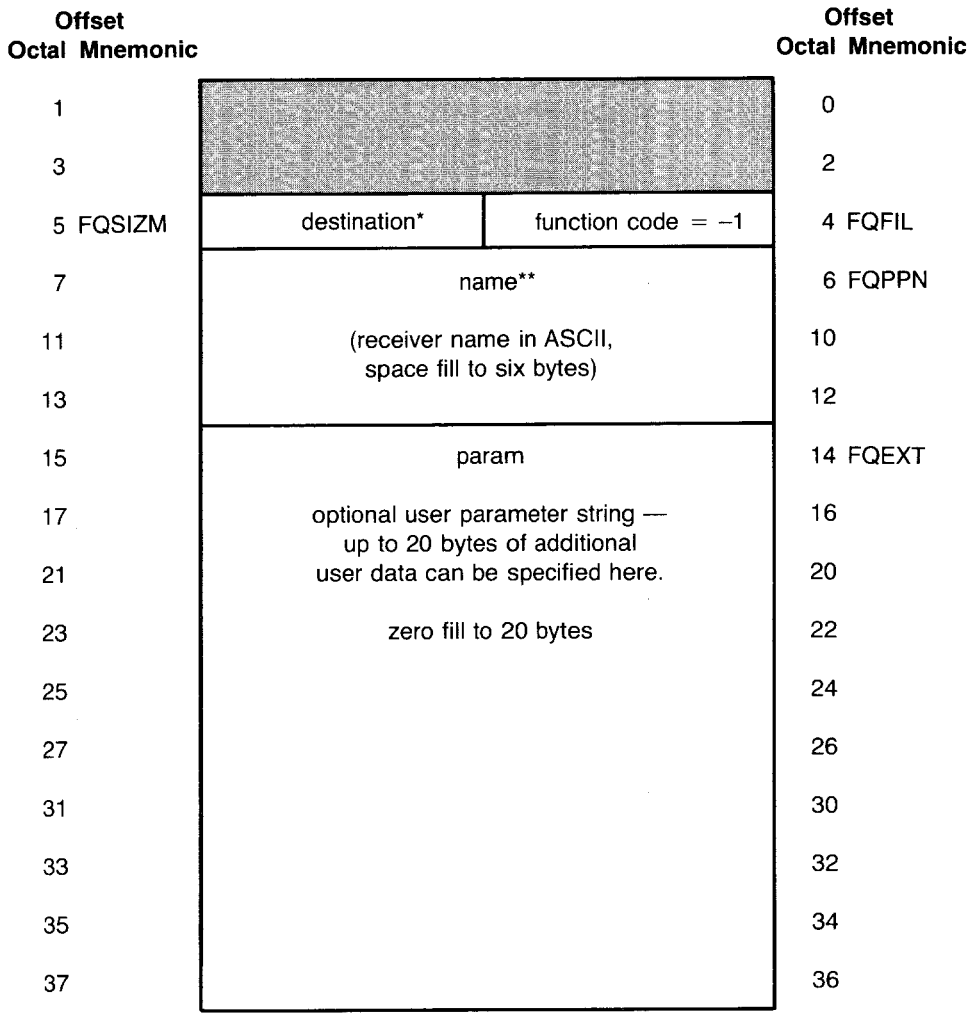
PRVIOL The caller is nonprivileged and has attempted to remove another job (that is, FIRQB + FQSIZM is nonzero).

BADFUO The argument at FIRQB + FQSIZM was odd. It must be zero to remove the calling program or job number times two to remove another job.

3.12.3 Send Local Data Message Subfunction – Privileged and Not Privileged

Data Passed

FIRQB

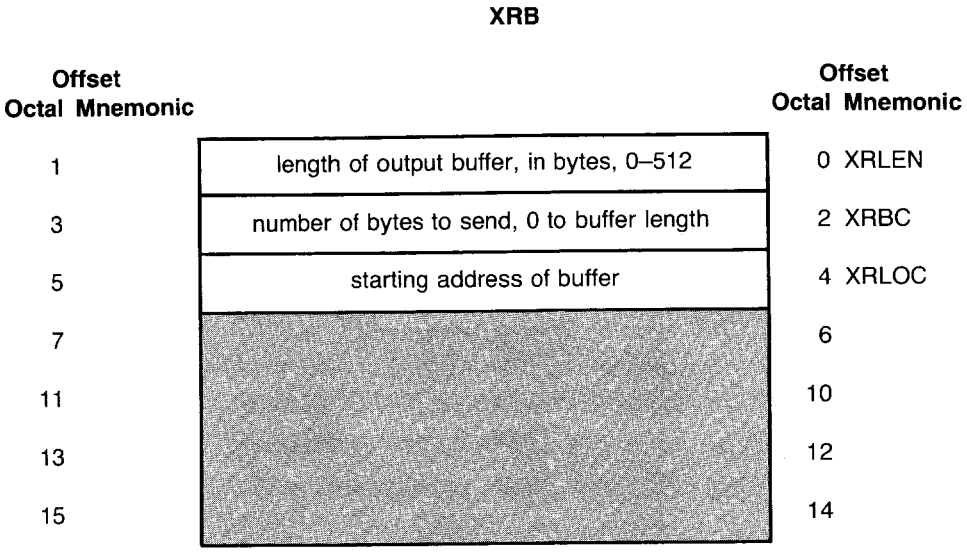


---

* You can specify the destination in one of two ways:

0	Indicates that the destination is the receiver name that starts at FIRQB+FQPPN.
job number *2	Indicates that the destination is this job number. A send by job number works only when the receiving job is receiving messages on RIB 0.

** The system uses the receiver name only if the byte at FIRQB+FQSIZM is 0.

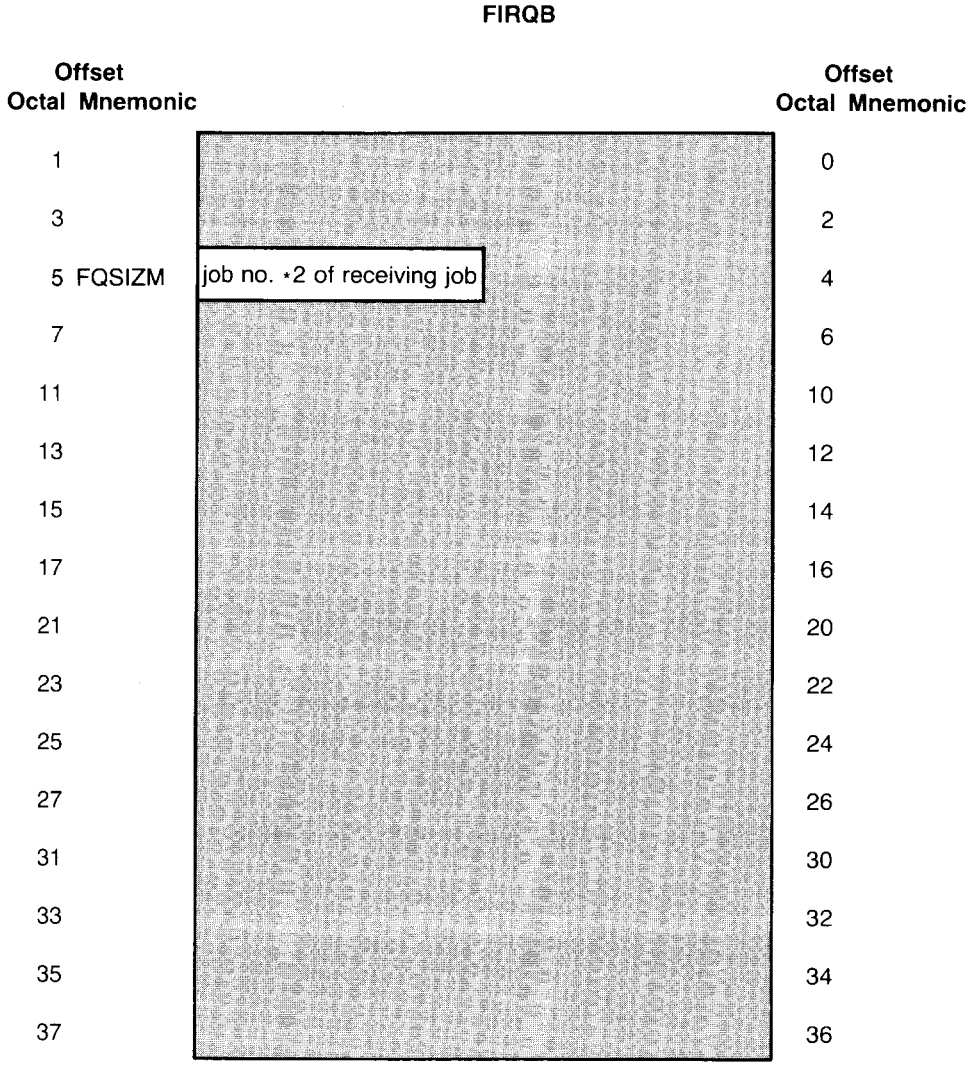


XRB + XRLEN — Length of the output buffer, in bytes. This value may range from zero through 512_{10}.

XRB + XRBC — The number of bytes to be sent. This value may range from zero through the size of the buffer, as specified at XRB + XRLEN.

XRB + XRLOC — Starting address of the buffer.

Data Returned



Errors

NOROOM	The number of pending messages for the intended local receiver is at its declared maximum. This program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
NOSUCH	The intended local receiver could not be located in the system's list of declared receivers. The receiver must be declared (with a declare receiver) before any data can be transmitted to it.
PRVIOL	Some access violation has occurred. Either the receiver does not allow any local senders, or the sender is nonprivileged and the receiver allows only privileged senders.

BADFUO — The value at FIRQB+FQSIZM is odd. It must be 0 or the receiver's job number times two.

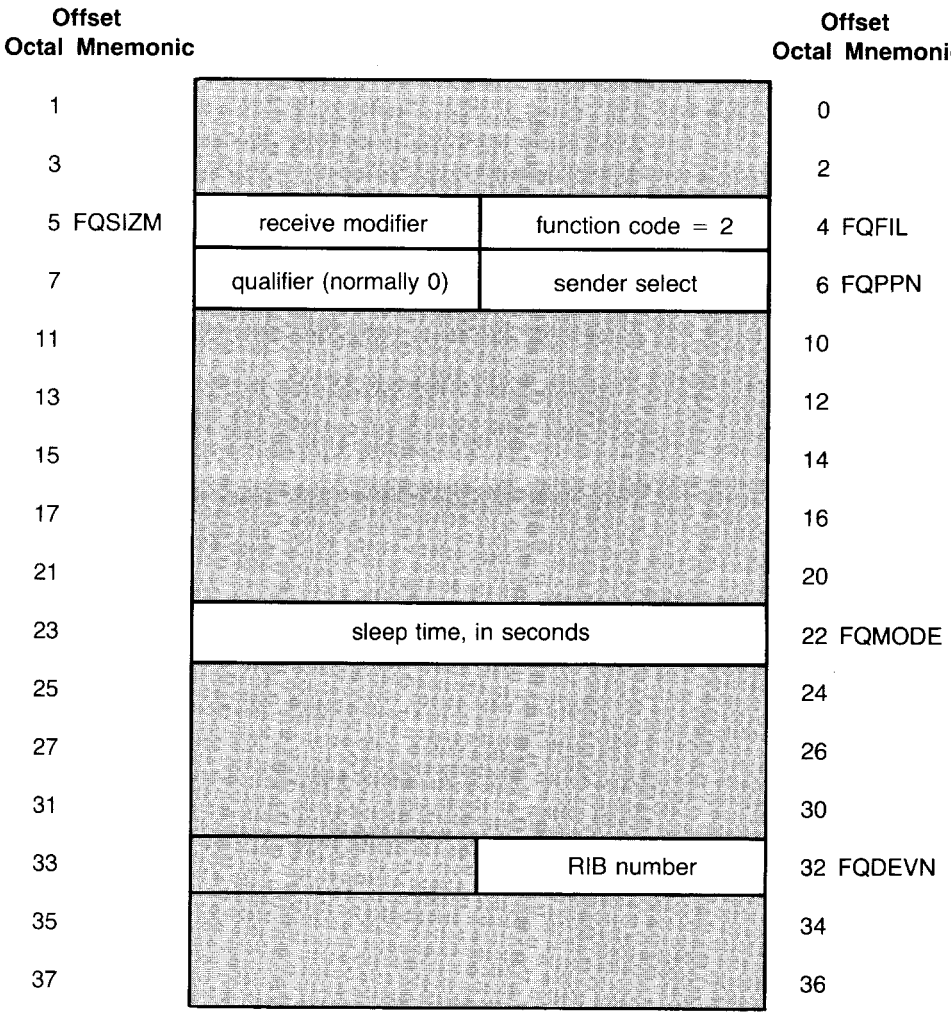
BADCNT — The XRB+XRLEN value is illegal. It may range from 0 through 512_{10}.

NOBUFS — System buffers are currently not available to store this message for the intended local receiver. A later retry may proceed without error.

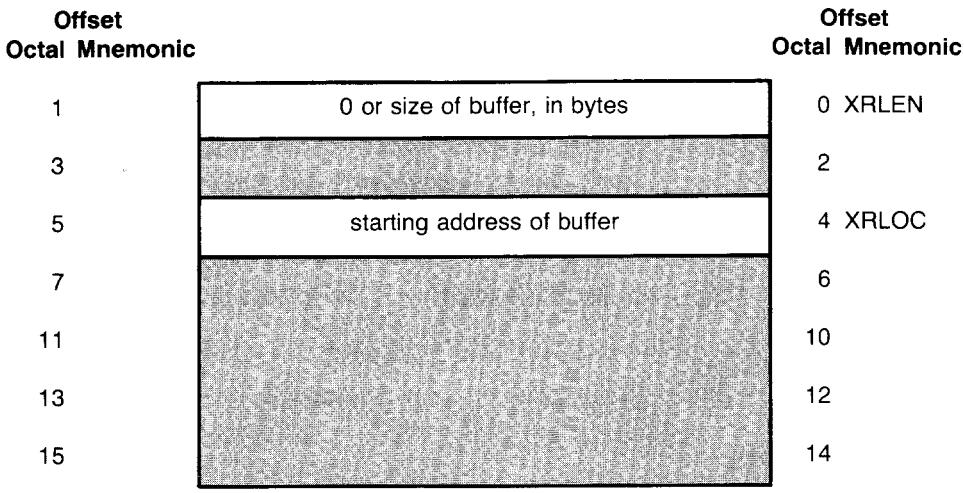
3.12.4 Receive Subfunction – Privileged and Not Privileged

Data Passed

FIRQB



XRB



XRB+XRLEN The size of the receive buffer, in bytes. This word may be zero if no user data is desired on the receive. The amount of data transferred from a pending message will never be greater than the buffer size.

XRB+XRBC This word must be passed as zero. The monitor returns the actual number of bytes of user data transferred in this word location, as shown in the Data Returned sections.

XRB+XRLOC The starting address of the buffer. The buffer, as defined by XRLOC for its start and XRLOC+XRLEN–1 for its last byte, must lie wholly within either the job image (low segment) or the run-time system (high segment).

If the buffer is in the low segment, the address defined by the contents of XRB+XRLOC must be greater than 170_8 to avoid destroying the job-context information used in swapping the job (Section 2.4).

If the buffer is in the high segment, it must not fall within the pseudo-vector region. That is, it must not fall above the location P.OFF (Section 2.5). In addition, the run-time system must currently be mapped read/write (see PF.RW bit description in P.FLAG word, Section 2.5). The run-time system must be read/write in this case, as the monitor will be writing data to the buffer for the receive.

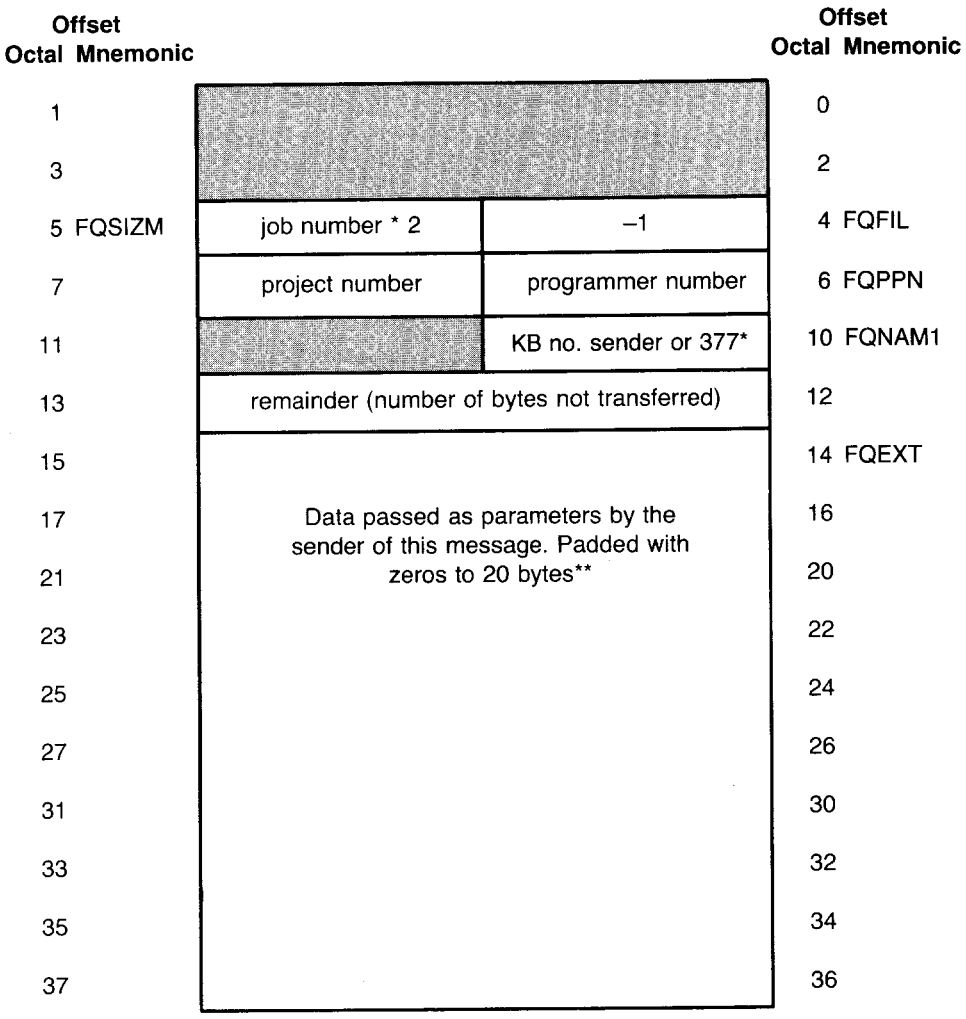
Data Returned

The Receive call returns data to the FIRQB and XRB, identifying the type of message received and user data, if any, to the buffer defined in the data passed.

The FIRQB and XRB formats for the local data message follow.

Data Returned (Local Data Message)

FIRQB



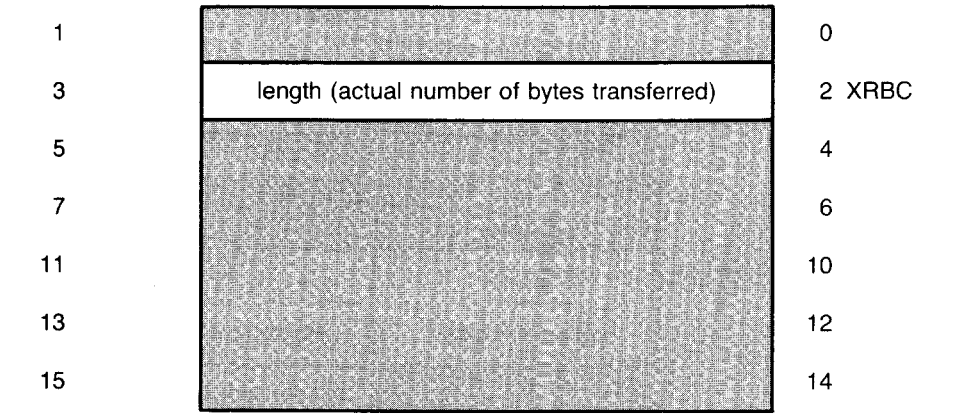
* 377 means sender is detached.

** For an EMT logger message, the monitor returns three values:

Bytes 14–15	Contain the number of data packets not transferred.
Bytes 16–17	Contain the number of EMTs your program "missed," either because it is not processing data packets quickly enough, or because not enough XBUF is available to store all the data packets that the monitor is creating.
Bytes 20–21	Contain the number of data packets transferred.

See the *RSTS/E Programming Manual* for more information.

XRB



Errors

NOSUCH — For a receive without sleep (bit 0 in receive modifier = 0), this error indicates that no appropriate messages are pending. For a receive with sleep (bit 0 in receive modifier = 1), this error is returned when the program is awakened from the sleep. The program must execute another receive call to retrieve any pending messages.

BADFUO — No receiver ID block. Before any receive can succeed, you must execute a declare receiver call to define the RIB number you want to use.

3.13 .NAME — Install Program Name with Monitor – Not Privileged

Form

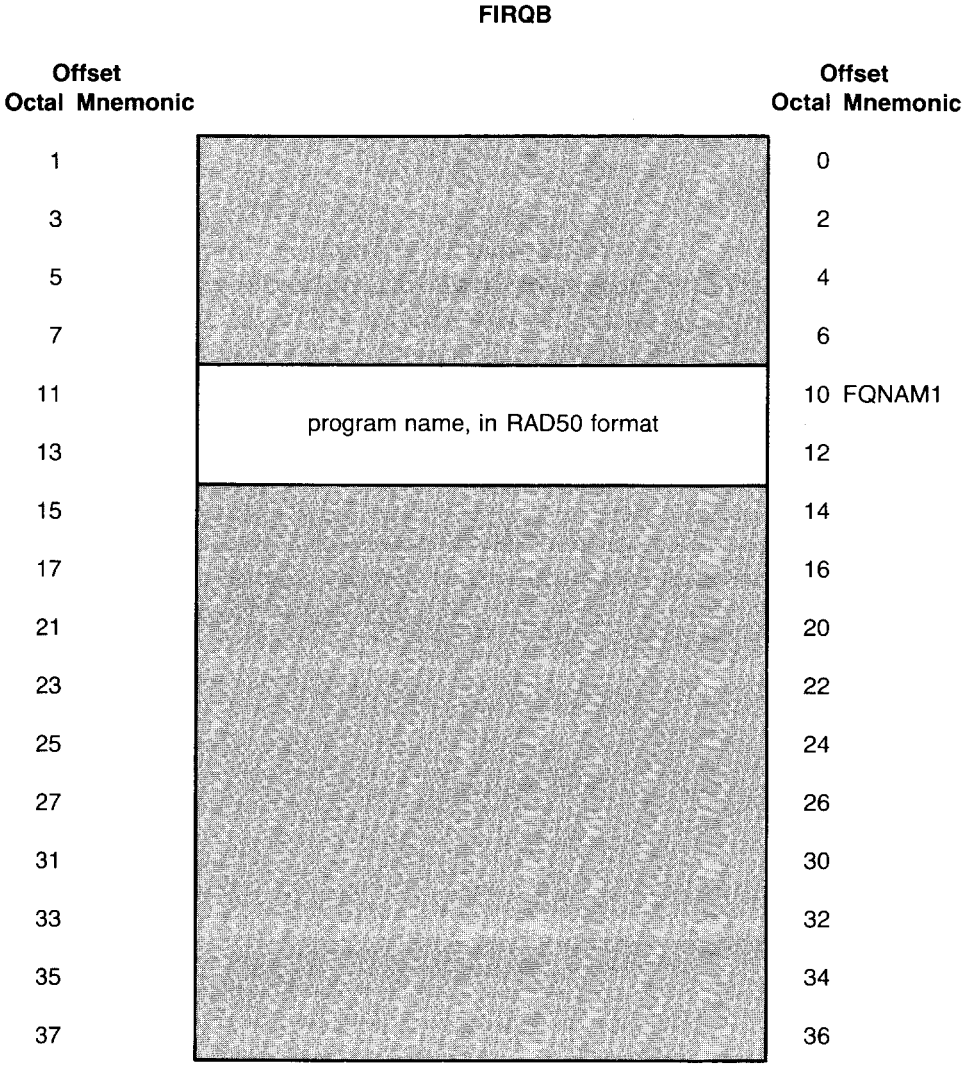
```
.NAME
```

Function

The .NAME directive installs a program "name" with the monitor. The monitor enters the name in an internal table; otherwise, it makes no use of the program name. However, the RSTS/E SYSTAT program uses the names in listing current information for jobs (under the "What" column) on the system. The BASIC–PLUS run-time system uses this directive when the user issues an OLD, NEW, or RENAME command, for example.

The program name is passed as two words of RAD50 data in the FIRQB. Note that the data is passed in the same location in the FIRQB where the file name exists at the P.RUN entry point into a run-time system. If you are coding or modifying a run-time system, one of the first things to do on entry at P.RUN is to install the program's name. Thus, the file name's position in the FIRQB at this point is handy.

Data Passed



FIRQB+FQNAM1 The program name to be installed, as two words in RAD50 format.

Data Returned

No data is returned with the .NAME directive.

Errors

No errors are possible with the .NAME directive.

Example

The following code installs the name "PROGRM" with the monitor.

```
MOV     #^RPRO,FIRQB+FQNAM1     ;SET FIRQB TO DECLARE
MOV     #^RGRM,FIRQB+FQNAM1+2   ;NAME OF "PROGRM"
.NAME
```

3.14 .PEEK — Look at Monitor Memory – Privileged

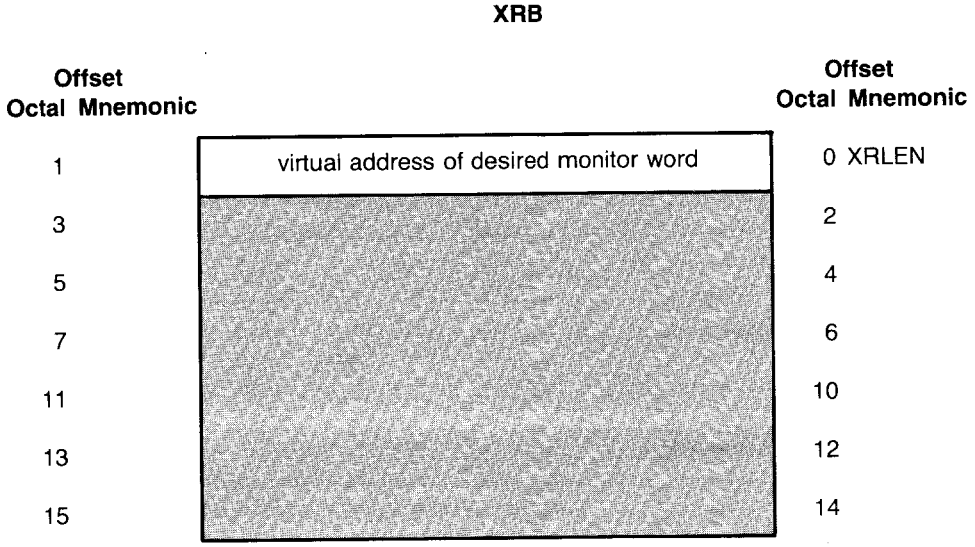
Form

```
.PEEK
```

Function

The .PEEK directive returns the contents of one word of the monitor's memory (that is, the memory mapped by the kernel mode APRs, as discussed in Section 2.1). .PEEK can be executed only by a privileged job. (Be very careful, however, at basing any of your coding on the contents of monitor memory. DIGITAL reserves the right to change the monitor structure and internal addresses at any time.)

Data Passed



XRB + XRLEN — This word contains the (virtual) address of the data word in monitor memory whose contents are to be returned. The value must be even, since word addresses on the PDP–11 are always even. Peeking at data in the I/O page (kernel APR 7, or 111 (binary) in bits 15, 14, and 13) can cause unpredictable system results and is not recommended. Furthermore, using .PEEK to obtain data in APRs 5 or 6 returns random data.

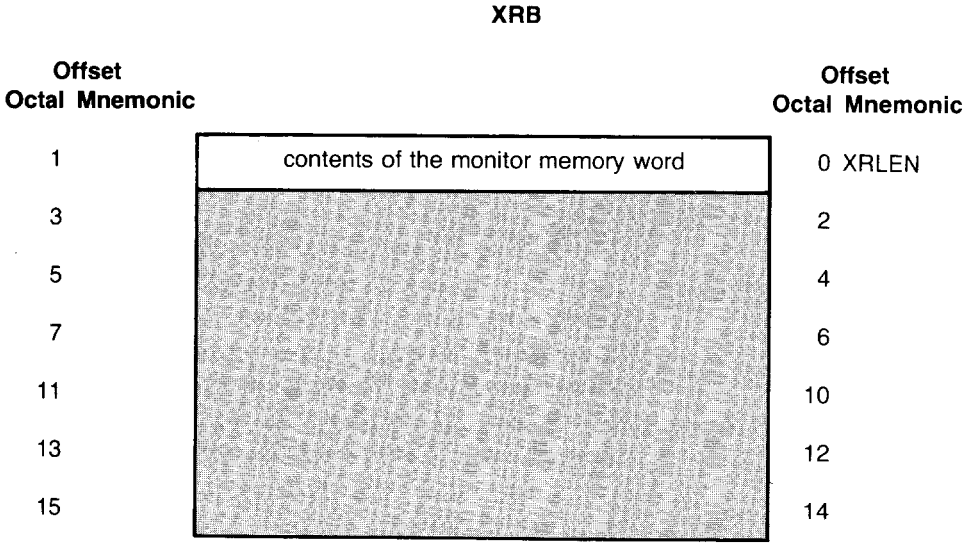
You generally use .PEEK to examine addresses returned by get monitor tables calls or addresses of fixed monitor locations.

Table 3–2 shows fixed monitor locations and their addresses.

Table 3–2: Fixed Monitor Locations

Address (decimal)	Name	Meaning
36(word)	IDATE	The date when the system was last started by START.
38(word)	ITIME	The time of day when the system was last started by START.
512(word)	DATE	Current system date.
514(word)	TIME	Current time of day.
518(byte)	JOB	Job number times 2 of the job currently running (always the user's own job number.)
520(word)	JOBDA	Address of the job data block (JDB) of the currently running job (always the user's own job data block).
522(word)	JOBF	Address of the JDFLG word in the job data block of the currently running job (always the user's own job data block).
524(word)	IOSTS	Address of the JDIOST (low) byte and JDPOST (high) byte in the job data block of the currently running job (always the user's own job data block).

Data Returned



XRB+XRLEN This word contains the contents of the requested monitor memory location.

Errors

B.4 The address specified caused a trap to the kernel mode vector at 4 (UNIBUS timeout, odd address, and so forth).

B.250 The address specified caused a memory management unit violation (trap to the kernel mode vector at 250).

PRVIOL The job is not privileged; .PEEK can be issued only from a privileged job.

Example

The following code obtains the contents of monitor memory location 518_{10} (the low byte of which is, incidentally, the current job number times 2).

```
MOV     #518.,,XRB+XRLEN        ;SET ADDRESS TO 518
.PEEK
```

3.15 .PLAS — Access Resident Library

The .PLAS (Program Logical Address Space) directive has six subfunctions that allow a MACRO program to access a resident library. Resident libraries must be so defined by the system manager with the ADD LIBRARY command (see *RSTS/E System Manager's Guide*). As noted in Section 2.3, the easiest way to do this is to link the resident library to your program using TKB — the Task Builder that links modules assembled or compiled under the RSX run-time system or its derivatives. However, you can use .PLAS subfunctions to access resident libraries using octal addresses.

A summary of the .PLAS subfunctions is given below; they are described in alphabetical order in following subsections.

FQFUN Value (Octal)	Mnemonic	Action Performed
0	ATRFQ	Attach resident library. Attaches the job to a resident library; necessary before the job can map a window to the library.
2	DTRFQ	Detach resident library. Detach the job from a resident library.
4	CRAFQ	Create address window. Defines a range of virtual addresses to be a "window" for looking at all or some portion of a resident library. Optionally, CRAFQ maps the window to all or some portion of a resident library. (The mapping can be done separately with MAPFQ.) The CRAFQ subfunction reserves one or more APRs, so CRAFQ "takes space" in the job area even though the window may not actually be mapped.
6	ELAFQ	Eliminate address window. Releases the APRs used by a particular window.
10	MAPFQ	Map window. Map an already created address window of virtual addresses to actual memory locations in an attached resident library. The monitor will load the library from disk if necessary.
12	UMPFQ	Unmap address window. Releases a window of virtual addresses from a mapping to actual memory locations.

When a job exits or a user logs out, the monitor automatically detaches all libraries and unmaps and eliminates all windows for the job.

3.15.1 ATRFQ (Attach Resident Library) – Not Privileged

Form

```
MOVB   #ATRFQ,FIRQB+FQFIL
       .
       .
       .
(set appropriate parameters)
       .
       .
       .
.PLAS
```

Function

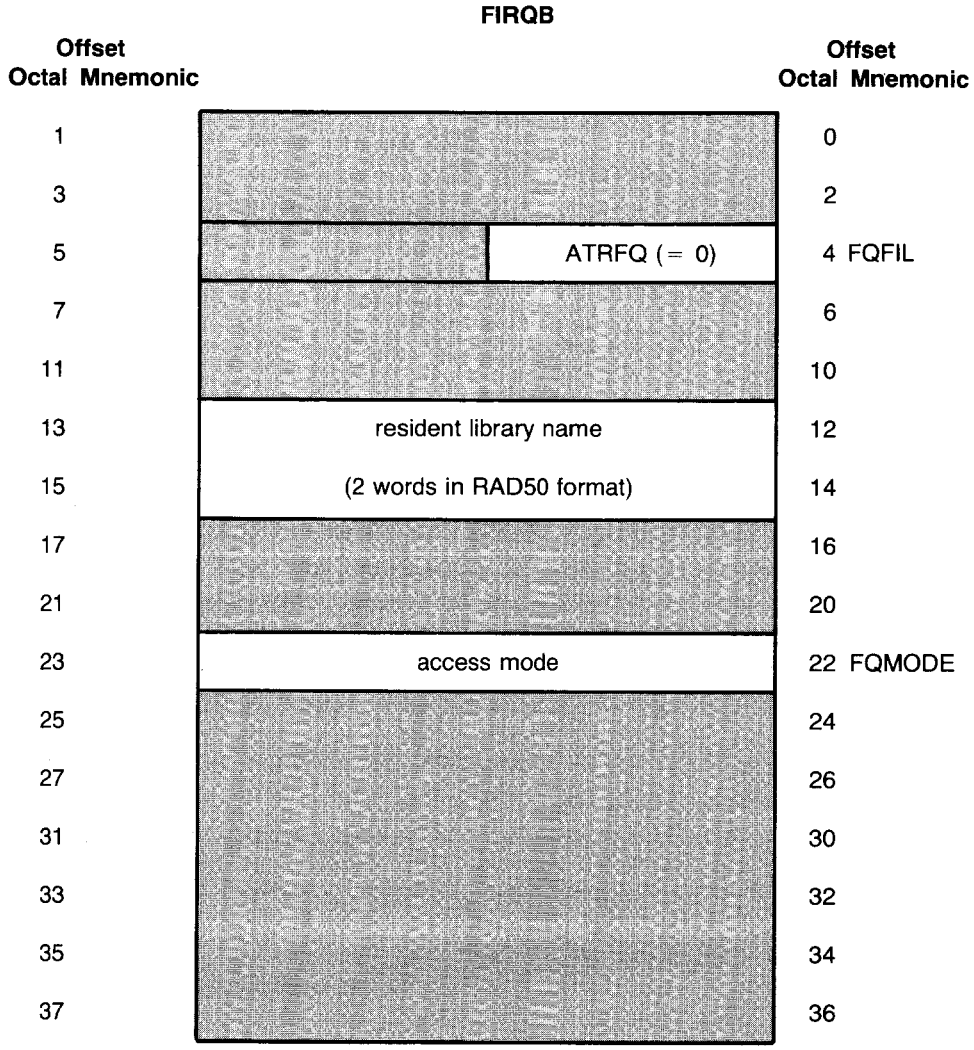
The ATRFQ (attach resident library) subfunction of .PLAS declares your intent to access a resident library. The type of access is specified in the call. If the calling job can access the library in that fashion,* the monitor loads the library from disk (if necessary) and sets up its own internal tables, which lay the groundwork for the job to map windows to the library. Note, however, that the resident library does not take up space in the job area (virtual memory) with an attach. APRs are assigned (virtual memory in the job area is taken) when a window is created (CRAFQ).

Up to five resident libraries can be attached to a job at any given time.

---

* The job's ability to access the resident library depends upon the protection assigned to the library by the system manager when the library was installed. The default protection grants read access to all users and denies write access to all users.

Data Passed



FIRQB + FQFIL — The function code ATRFQ (octal value = 0).

FIRQB + 12 — The name of the resident library to which the job is to be attached, as two words of RAD50 data. (Resident libraries are made known to the monitor by the system manager with the ADD LIBRARY command of UTILTY (*RSTS/E System Manager's Guide*). With this command, the system manager defines a file (filename.LIB) as a resident library. The monitor regards "filename" as the resident library's name.)

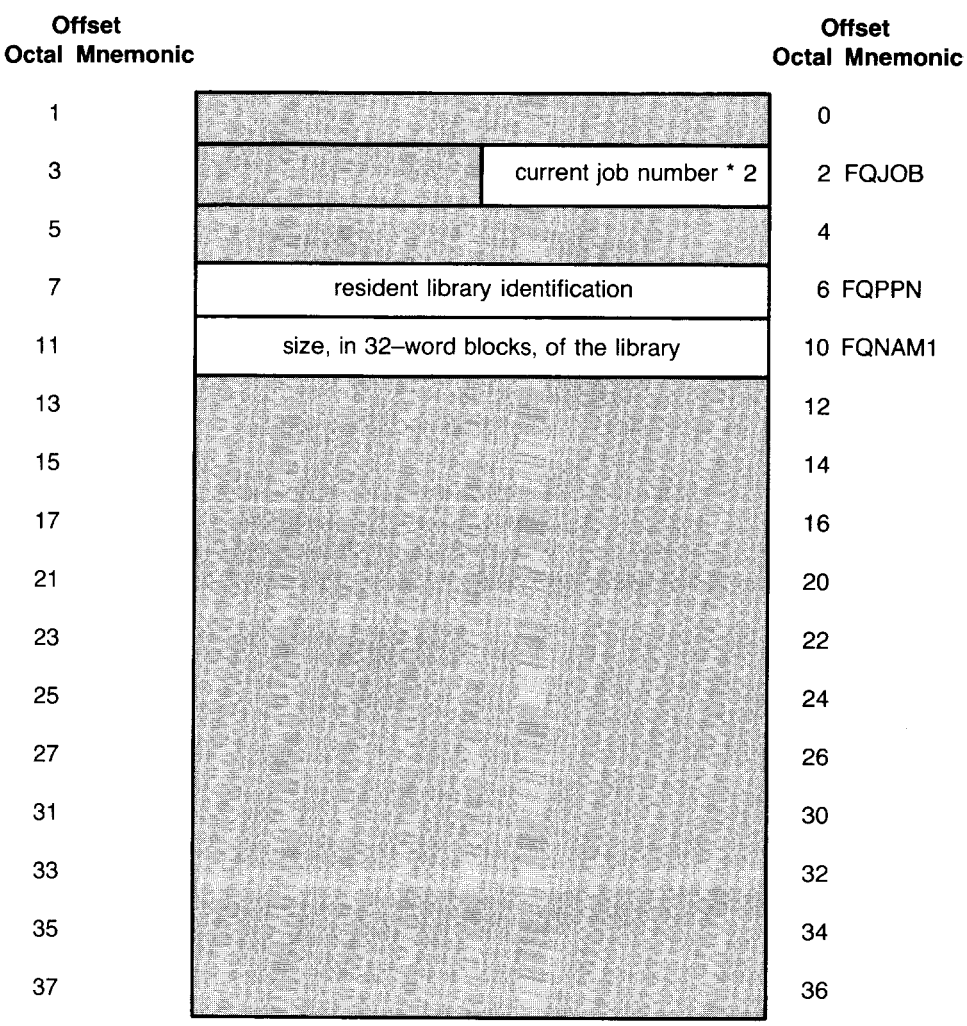
FIRQB + FQMODE — The low-order two bits of this word define the way the job wishes to access the library:

Bit 0 = 1 Read-only access is desired.

Bit 1 = 1 Read/write access is desired.

Data Returned

FIRQB



FIRQB + FQJOB — The current job number times two.

FIRQB + FQPPN — This word is an identifier that must be used, rather than the resident library name, in subsequent calls to identify a resident library. Thus, you will use this identifier to detach the job from the library (DTRFQ) and map and unmap windows to the library (MAPFQ and UMPFQ). (Keep it around!)

FIRQB + FQNAM1 — The size of the resident library, in 32–word blocks.

Errors

NOROOM — The job has tried to attach to more than five resident libraries. At least one must be detached from the job (DTRFQ) before another can be attached.

NOSUCH — The resident library specified in the data passed is not known to the monitor. The system manager must install a resident library before it can be used.

PRVIOL — The attach did not succeed because the caller's privilege did not allow the access specified in the data passed. This could happen either (1) because the access code specified in the data passed is not compatible with the possible access defined when the library was installed by the system manager or (2) because the protection code associated with the resident library file excludes access by the user.

Example

The following code attaches the job to a resident library called DATBAS. The access desired is defined as read/write.

```
MOVB    #ATRFQ,FIRQB+FQFIL      ;DEFINE FUNCTION CODE
MOV     #^RDAT,FIRQB+12         ;LIBRARY NAME IS
MOV     #^RBAS,FIRQB+FQEXT      ;DEFINED AS "DATBAS"
MOV     #2,FIRQB+FQMODE         ;ACCESS=READ/WRITE
.PLAS
```

3.15.2 CRAFQ (Create Address Window) – Not Privileged

Form

```
MOVB    #CRAFQ,FIRQB+FQFIL
        .
        .
        .
(set up parameters)
        .
        .
        .
.PLAS
```

Function

The CRAFQ subfunction of .PLAS can be used either to create a window (a range of virtual addresses) or to create a window of virtual addresses and map it to a range of actual addresses in an attached library. You define the range of addresses by (1) naming a base APR (which defines the starting address of the window) and (2) specifying the size of the window in 32–word blocks. Thus, a window always begins on a 4K–word boundary in virtual memory and always takes at least 4K words. It may take more than 4K words, depending on the size of the window.

If the address range overlaps the user job image, the call fails with an error. The address range cannot overlap the run-time system (high segment). However, if the RSX run-time system has been installed as "disappearing" (see .RSX, Section 3.18), this is not a consideration. APR 7, normally used to map the RSX run-time system, can be used instead to map a window to a resident library. If the address range overlaps an existing window, the previously created window is eliminated.

The difference between (1) creating a window and (2) creating and mapping a window is best illustrated by example. By using create without map, you can define one window, which can be mapped to a library or portion of a library and then remapped to another portion of the same library or another library, as many times as desired, using the MAPFQ subfunction of .PLAS. For example, suppose your program takes up 24K words and you want to access a 24K–word resident library of data values. You can use create without map to set up a 4K–word window in APR 6. You can then map the window (using MAPFQ) to the first 4K words of the library, process the data, map to the next 4K words of the library, and so forth.

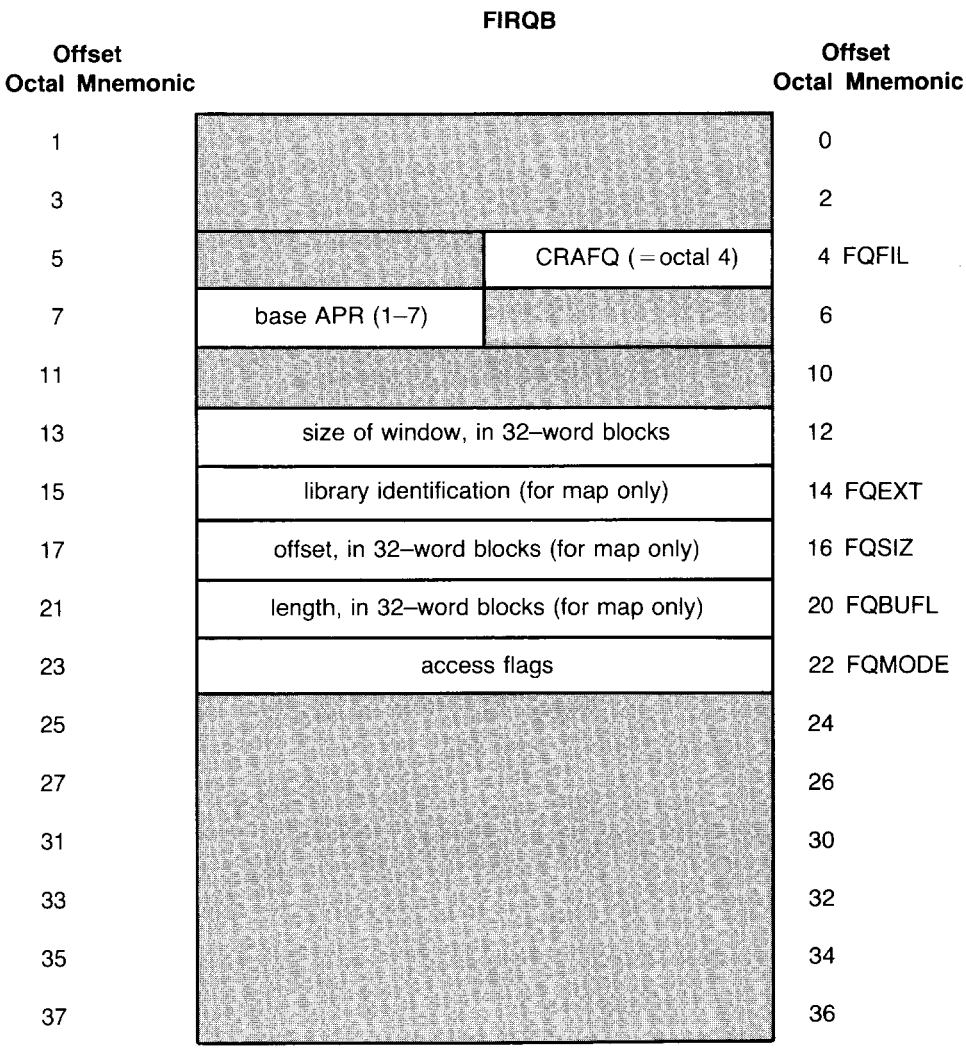
If, on the other hand, you had a 4K program and still wished to access a 24K–word library, you could use CRAFQ to create a 24K–word window and map it to the entire library in APRs 1 – 6.

A job can create a maximum of seven windows. A window takes at least one APR (it may take more, depending on the size you specify for the window). Thus, the maximum of seven assumes seven windows in APRs 1 through 7. APR 0 can never be used to create a window, since the user program takes at least this much space. As mentioned above, a window cannot overlap the user job image; thus, the size of the user job image determines the lowest base APR that can be used. If the program (user job image) is less than 4K words, APRs 1 and up (to the limit imposed by the run-time system boundary) can be used to create windows. If the user job image is between 4K words and 8K words, APRs 2 and up can be used to create windows, and so forth.

If a window is created that overlaps an already-existing window, the old window is eliminated. For example, if you create a 6K–word window using a base APR of 5, the window uses APRs 5 and 6. If you then create a 4K–word window using a base APR of 6, the entire old window is eliminated. APR 5 is then free for other use; APR 6 is used for the new window.

Data Passed

FIRQB



FIRQB+FQFIL — The function code CRAFQ (octal value = 4).

FIRQB+7 — The base APR of the window, 1 – 7. Implicitly defines the starting virtual address of the window. This byte cannot be zero, nor can it name an APR already being used to map the user job image.

FIRQB+12 — The desired size of the window, in 32–word blocks. For example, a value of 128_{10} = 4K words.

FIRQB+FQEXT — The identifier of the resident library to which the window is to be mapped. (This is the value returned by the ATRFQ function of .PLAS at FIRQB+FQPPN.) This word is ignored for calls requesting a create without mapping (bit 7 at FIRQB+FQMODE equals 0).

FIRQB+FQSIZ — The offset, in 32–word blocks, from the start of the library where the mapping is to begin. This word is ignored if no mapping is requested (bit 7 at FIRQB+FQMODE equals 0). A value of zero for this word indicates the window is to be mapped beginning at the first byte of the library. A value of 1 indicates the window is to be mapped beginning at the 33rd word of the library (starting address + 64), and so forth.

FIRQB+FQBUFL — The length, in 32–word blocks, of the area to be mapped (ignored if bit 7 at FIRQB+FQMODE equals 0). This value cannot be greater than the size of the window specified at FIRQB+12. Furthermore, this value, combined with the offset value at FIRQB+FQSIZ, cannot indicate an address beyond the end of the library or into the high segment (run-time system).

A value of 0 for this word defaults to either the size of the window (specified at FIRQB+FQEXT) or the space remaining in the library, whichever is smaller.

FIRQB+FQMODE — Two bits in this word define whether the window is to be mapped and whether write access to the window is desired.

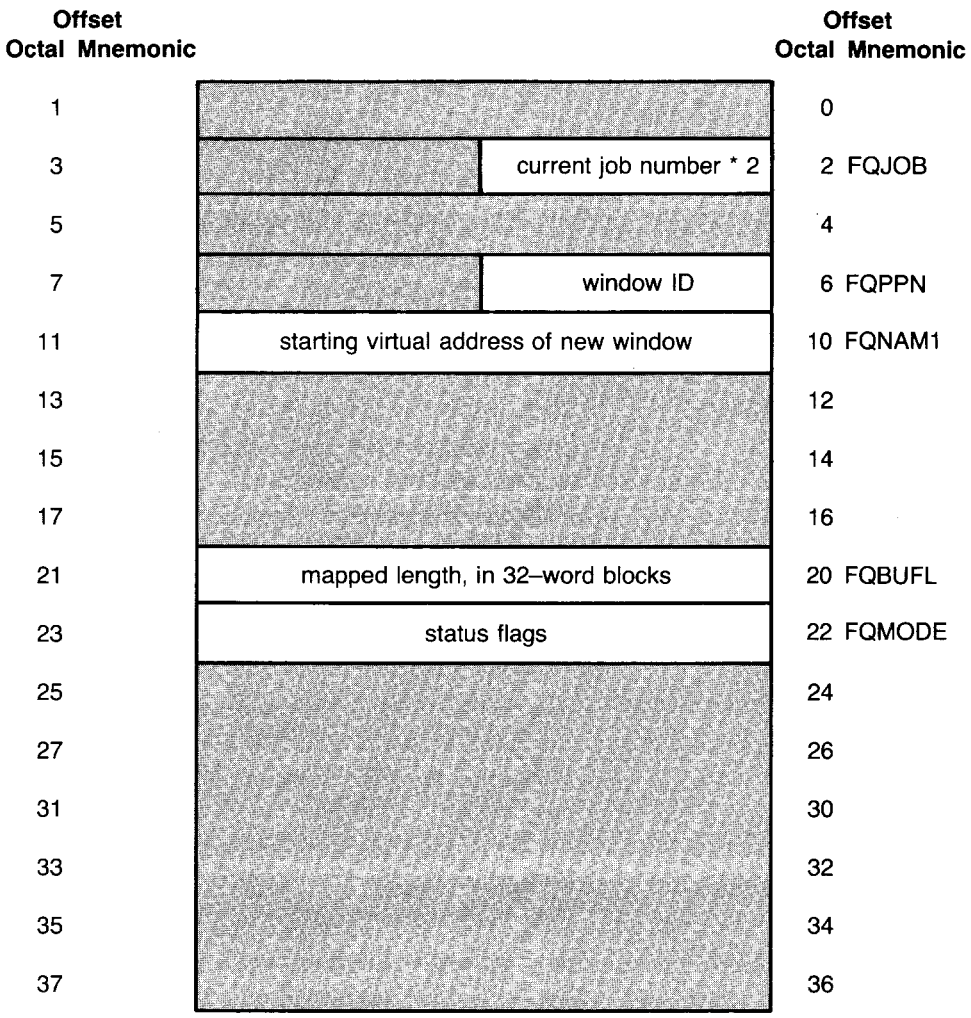
bit 1 = 1 Write access to the window is desired.
= 0 No write access to the window is desired.

bit 7 = 1 The window is to be mapped.
= 0 The window is not to be mapped.

The octal value to set bit 7 is 200_8; the value to set bit 1 is 2. Thus, a value of 202_8 for this word requests mapping and write access. A separate setting for write access in CRAFQ and in ATRFQ allows you to attach to a library read/write and map a portion of the library read-only.

Data Returned

FIRQB



FIRQB + FQJOB — The current job number times two.

FIRQB + FQPPN — This byte contains the window ID; it can be used in later MAPFQ calls to map the newly created window and must be used in any ELAFQ calls to eliminate the newly created windows. (Save it if you will need to use it.) The value returned may range from 1 through 7.

FIRQB + FQNAM1 — The starting virtual address of the new window.

FIRQB + FQBUFL — Length, in 32–word blocks, actually mapped by the window.

FIRQB+FQMODE Status flags.

bit 15 = 1 (Octal value equals 100000.) Window was created successfully.
= 0 Window was not created successfully.

bit 14 = 1 (Octal value equals 40000.) An existing window was unmapped because it overlapped the newly created mapping.
= 0 No existing windows were unmapped by this mapping.

bit 13 = 1 (Octal value equals 20000.) An existing window was eliminated because it overlapped the newly created window.
= 0 No existing windows were eliminated by this create.

Errors

BADFUO Either the base APR and window length specified were invalid, or the offset and mapping length values specified were invalid. (For example, an offset indicating a starting address for the mapping that is beyond the end of the library or into the run-time system is invalid.)

NOBUFS Creating a window requires a small buffer; a small buffer is not currently available.

NOROOM You attempted to create more than seven address windows.

NOSUCH The library ID specified for mapping is not a library currently attached to the job.

PRVIOL The create was unsuccessful because the user privileges do not allow the access desired. At this point, since the library has been attached successfully with some access defined, this error means that the access requested in the CRAFQ is not allowed by the access requested in the ATRFQ.

Example

The following code creates a 4K–word address window and maps it to the beginning of a library whose ID (returned from a previous ATRFQ) has been stored at location LIBID:

```
MOVB      #CRAFQ,FIRQB+FQFIL          ;DEFINE FUNCTION CODE
MOV       #6,FIRQB+7                  ;BASE APR = 6
MOV       #128.,,FIRQB+12             ;WINDOW = 4K WORDS
MOV       LIBID,FIRQB+FQEXT           ;SET LIBRARY ID
CLR       FIRQB+FQSIZ                 ;OFFSET = 0
CLR       FIRQB+FQBUFL                ;MAP 4K WORDS OR TO
                                      ;END OF LIBRARY
MOV       #128.,,FIRQB+FQMODE         ;MAP WINDOW, READ-ONLY
,PLAS
```

3.15.3 DTRFQ (Detach Resident Library) – Not Privileged

Form

```
MOVB     #DTRFQ,FIRQB+FQFIL
   .
   .
   .
(define library to be detached)
   .
   .
   .
.PLAS
```

Function

The DTRFQ function of .PLAS detaches a previously attached resident library. Any windows mapped to the library by the calling job are unmapped. If no other jobs are currently attached to the library and it was installed with the /REMOVE option, the monitor will remove the library from memory.

Data Passed

FIRQB

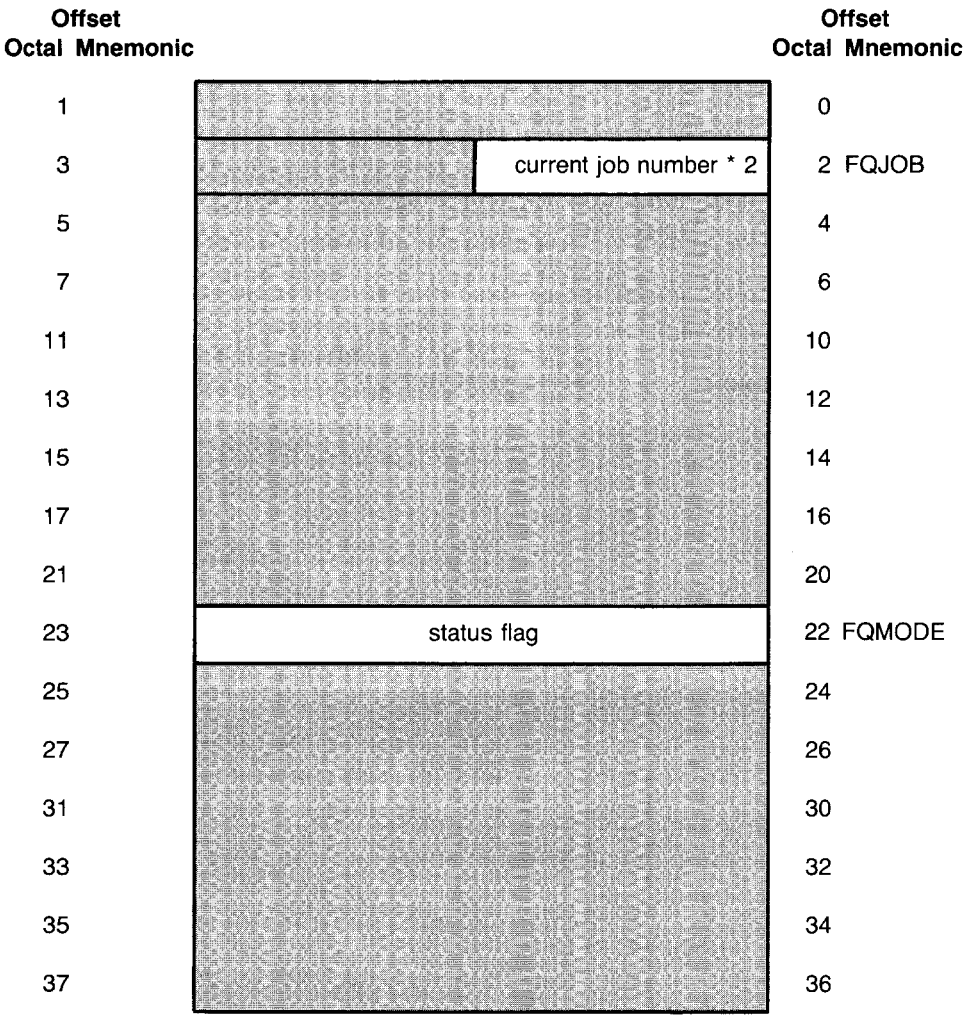
Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3			2	
5		DTRFQ (= octal 2)	4	FQFIL
7		library identification (returned by ATRFQ)	6	FQPPN
11			10	
13			12	
15			14	
17			16	
21			20	
23			22	
25			24	
27			26	
31			30	
33			32	
35			34	
37			36	

FIRQB+FQFIL The function code DTRFQ (octal value = 2).

FIRQB+FQPPN This word is the library identification returned at FIRQB+FQPPN by the ATRFQ that attached the job to the library.

Data Returned

FIRQB



FIRQB+FQJOB The current job number times two.

FIRQB+FQMODE Bit 14 of this word is set to 1 (octal value equals 40000) if any windows were unmapped as a result of this detach.

Errors

NOSUCH	The library ID specified at FIRQB + FQPPN in the data passed does not identify any library currently attached to the job.

Example

The following code detaches the library whose ID is stored at LIBID:

```
MOVB    #DTRFQ,FIRQB+FQFIL          ;SET FUNCTION CODE
MOV     LIBID,FIRQB+FQPPN           ;SET LIBRARY ID
.PLAS
```

3.15.4 ELAFQ (Eliminate Address Window) – Not Privileged

Form

```
MOVB     #ELAFQ,FIRQB+FQFIL
   .
   .
   .
(set up parameters in FIRQB)
   .
   .
   .
.PLAS
```

Function

The ELAFQ subfunction of .PLAS eliminates an address window that was created by the job, unmapping the window if necessary. ELAFQ frees the APRs used by the window and makes them available for creating another window or for expanding the user job image size.

Data Passed

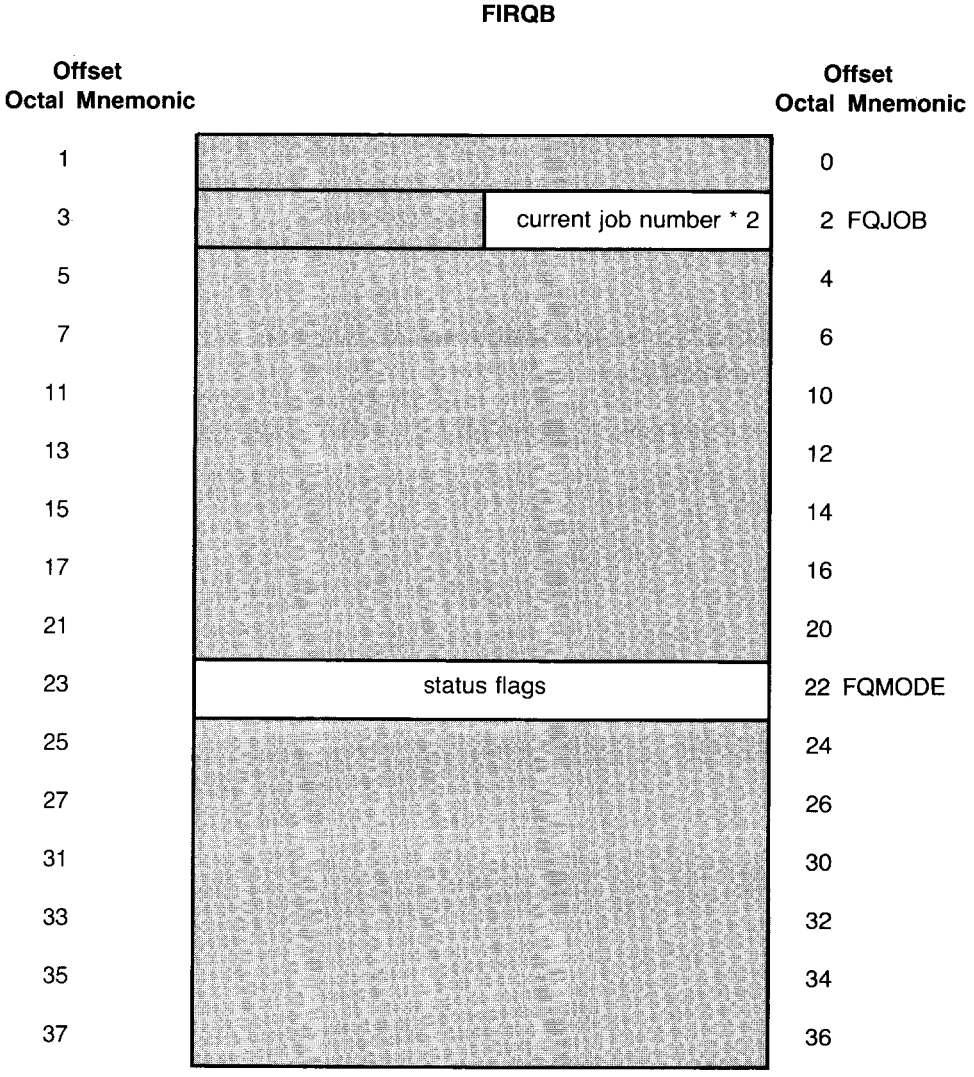
FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3			2	
5		ELAFQ (= octal 6)	4	FQFIL
7		window ID	6	FQPPN
11			10	
13			12	
15			14	
17			16	
21			20	
23			22	
25			24	
27			26	
31			30	
33			32	
35			34	
37			36	

FIRQB+FQFIL The function code ELAFQ (octal value equals 6).

FIRQB+FQPPN The ID of the window to be eliminated (returned at FIRQB+FQPPN by the CRAFQ that created the window).

Data Returned



FIRQB+FQJOB The current job number times two.

FIRQB+FQMODE Two bits in this word indicate the status of the window.

Bit 13 = 1 (Octal value equals 20000.) The window was successfully eliminated.
= 0 The window was not eliminated.

FIRQB+FQMODE (cont.)	Bit 14 = 1	(Octal value equals 40000.) The address window was mapped to a resident library and has been unmapped.
	= 0	The address window was not mapped; no unmapping was done.

Errors

BADFUO	An invalid window ID was given in the data passed at FIRQB+FQPPN (outside the range 1 through 7).
NOSUCH	The window ID specified at FIRQB+FQPPN is in the range 1 through 7 but matches no currently created window for this job.

Example

The following code eliminates an address window whose ID is stored in location WINID:

```
MOVB    #ELAFQ,FIRQB+FQFIL          ;SET FUNCTION CODE
MOVB    WINID,FIRQB+FQPPN           ;SET WINDOW ID
.PLAS
```

3.15.5 MAPFQ (Map Address Window) – Not Privileged

Form

```
MOVB     #MAPFQ,FIRQB+FQFIL
   .
   .
   .
(set up parameters)
   .
   .
   .
.PLAS
```

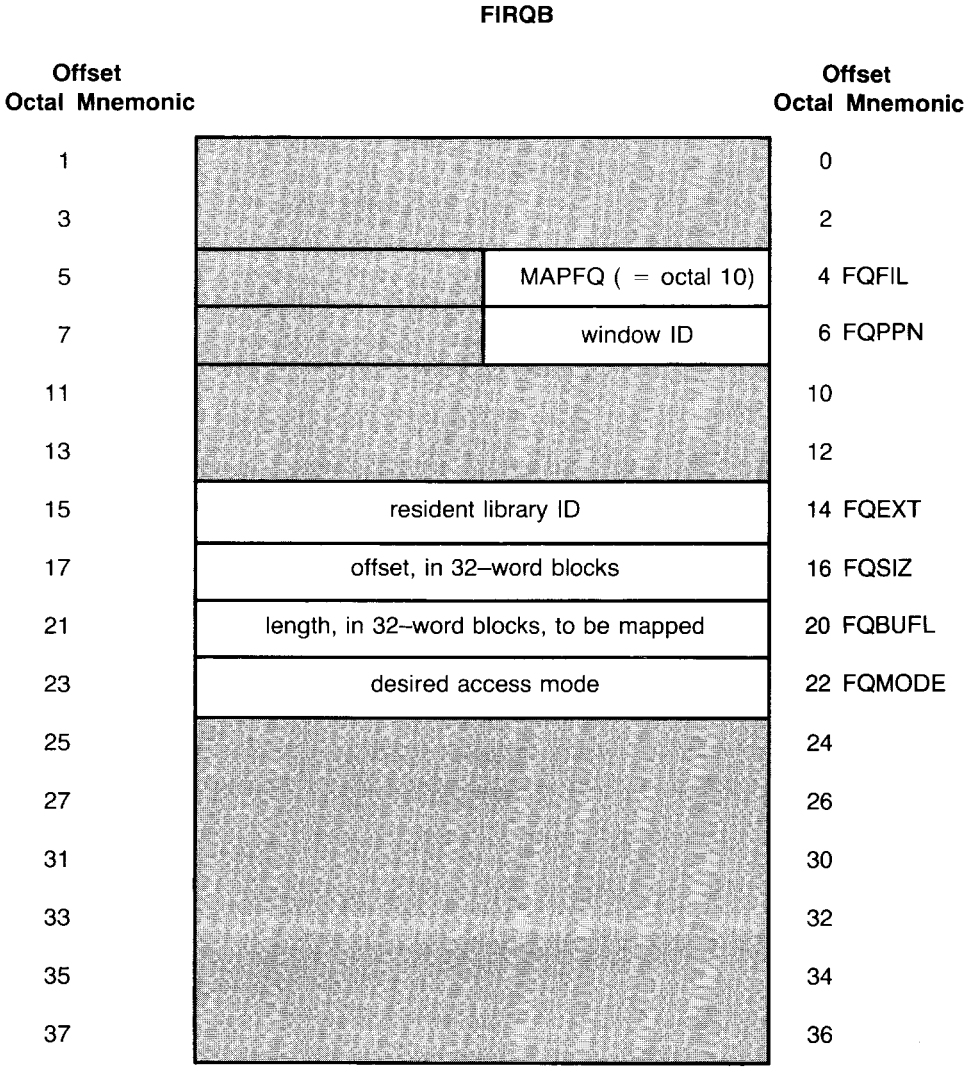
Function

The MAPFQ subfunction of .PLAS maps a previously created address window to an attached resident library. In other words, the MAPFQ relates the virtual address range defined by a CRAFQ to actual locations in memory within a resident library that has been attached to the job by an ATRFQ.

If the window specified is already mapped, it is unmapped from its previous actual memory locations and remapped to the new area. A job may map a maximum of seven address windows at any given time.

If the resident library being mapped is not in memory when the MAPFQ is executed, the system makes the library memory resident at that time.

Data Passed



FIRQB+FQFIL — The function code MAPFQ (octal value equals 10).

FIRQB+FQPPN — The ID of the window to be mapped (returned at FIRQB+FQPPN by the CRAFQ subfunction call that created the window).

FIRQB+FQEXT — The ID of the resident library to which the window is to be mapped (returned as a full word at FIRQB+FQPPN by the ATRFQ subfunction call that attached the job to the resident library).

FIRQB+FQSIZ — The offset, in 32–word blocks, from the start of the library where the mapping is to begin. A value of zero for this word indicates the window is to be mapped beginning at the first byte of the library. A value of 1 indicates that the window is to be mapped beginning at the 33rd word of the library (starting address + 64), and so forth.

FIRQB+FQBUFL — The length, in 32–word blocks, of the area to be mapped. This value cannot be greater than the size of the window (specified at FIRQB+12 in the CRAFQ which created the window). Furthermore, this value, combined with the offset value at FIRQB+FQSIZ, cannot indicate an address beyond the end of the library.

A value of 0 for this word defaults to either the size of the window or the space remaining in the library, whichever is smaller.

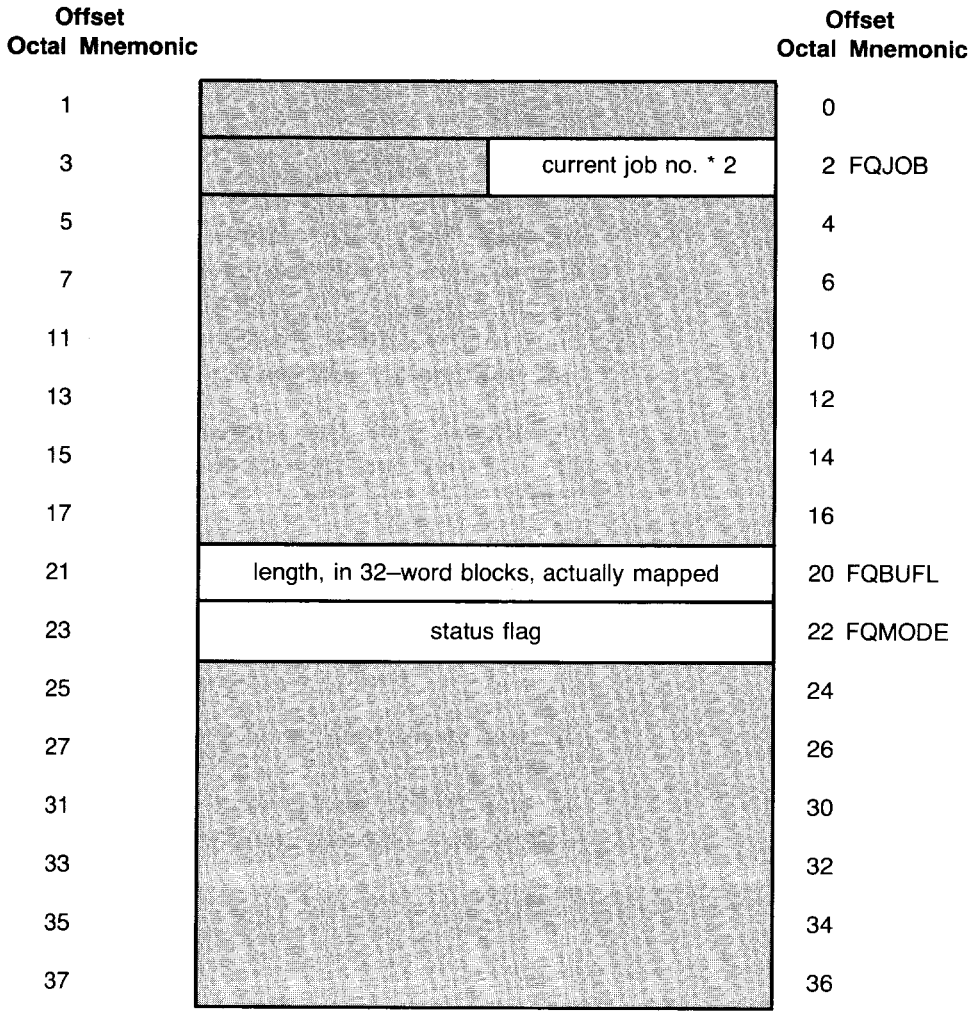
FIRQB+FQMODE — Bit 1 of this word specifies whether the window is to be mapped read/write or read-only.

bit 1 = 1 (Value equals 2.) Read/write access.
= 0 Read-only access.

A separate setting for access in MAPFQ and in ATRFQ allows you to attach to a library read/write and map a portion of the library read-only. You cannot, however, attach to a library read-only and then map to the library read/write.

Data Returned

FIRQB



FIRQB + FQJOB — The current job number times two.

FIRQB + FQBUFL — Length, in 32–word blocks, actually mapped by the call.

FIRQB + FQMODE — One bit of this word is set as a status flag.

Bit 14 = 1 (Octal value equals 40000.) The window specified was already mapped; the window was unmapped before this mapping was done.

= 0 The window specified had no previous mapping; no unmapping was done for this call.

Errors

BADFUO	The offset and length specified are inconsistent; either the mapping attempted to go beyond the end of the library or the length is greater than the created window.
NOSUCH	Either the resident library ID or the address window ID is incorrect. (The job is not currently attached to the specified resident library or no address window has been created with the specified window ID.)
PRVIOL	The mapping was unsuccessful because user privileges did not allow the access desired.

Example

The following code maps a window whose ID is stored at WINID to the attached library whose ID is stored at LIBID. The offset of 256_{10} indicates that the mapping is to begin 8K words from the start of the library. Access to the library is read-only.

```
MOVB    #MAPFQ,FIRQB+FQFIL      ;SET FUNCTION CODE
MOVB    WINID,FIRQB+FQPPN       ;SET WINDOW ID
MOV     LIBID,FIRQB+FQEXT       ;SET LIBRARY ID
MOV     #256.,FIRQB+FQSIZ       ;OFFSET 8K WORDS
CLR     FIRQB+FQBUFL            ;MAP WINDOW SIZE OR
                                ;TO END OF LIBRARY
CLR     FIRQB+FQMODE            ;READ-ONLY ACCESS
.PLAS
```

3.15.6 UMPFQ (Unmap Address Window) – Not Privileged

Form

```
MOVB     #UMPFQ,FIRQB+FQFIL
   .
   .
   .
(set up parameters)
   .
   .
   .
.PLAS
```

Function

The UMPFQ subfunction of .PLAS unmaps a specified address window from a resident library. (Note that a MAPFQ on an already-mapped window will unmap the existing windows.)

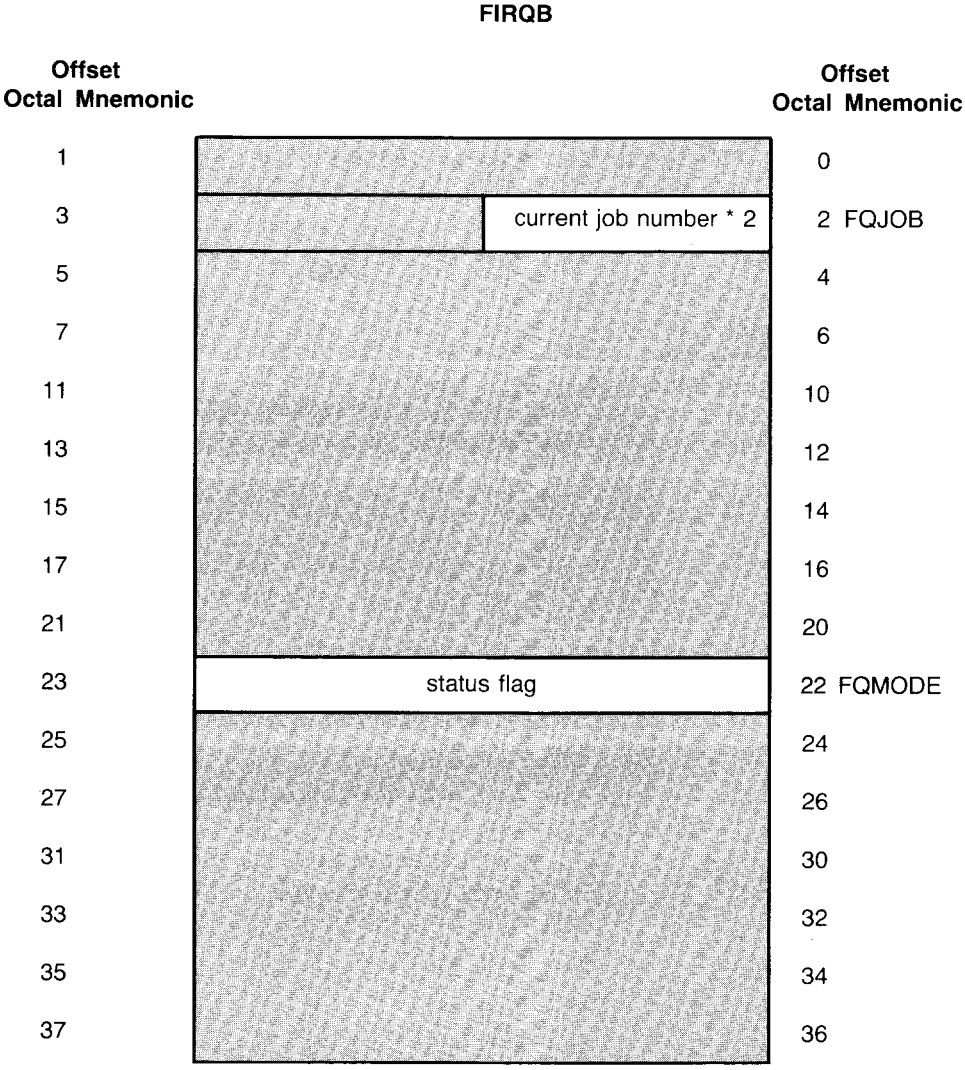
Data Passed

FIRQB

Offset Octal Mnemonic		Offset Octal Mnemonic
1		0
3		2
5	UMPFQ (= octal 12)	4 FQFIL
7	window ID	6 FQPPN
11		10
13		12
15		14
17		16
21		20
23		22
25		24
27		26
31		30
33		32
35		34
37		36

FIRQB + FQFIL	The function code UMPFQ (octal value equals 12).
FIRQB + FQPPN	The ID of the window to be unmapped (returned at FIRQB + FQPPN by the CRAFQ that created the window).

Data Returned



FIRQB + FQJOB	The current job number times two.
FIRQB + FQMODE	Bit 13 of this word is set (octal value equals 20000) if the unmapping was successful.

Errors

BADFUO	The window ID specified is invalid (not in the range 1 through 7).
NOSUCH	The window ID specified is in the range 1 through 7, but no such window is currently created for the job.

3.16 .POSTN — Return Current Horizontal Position – Not Privileged

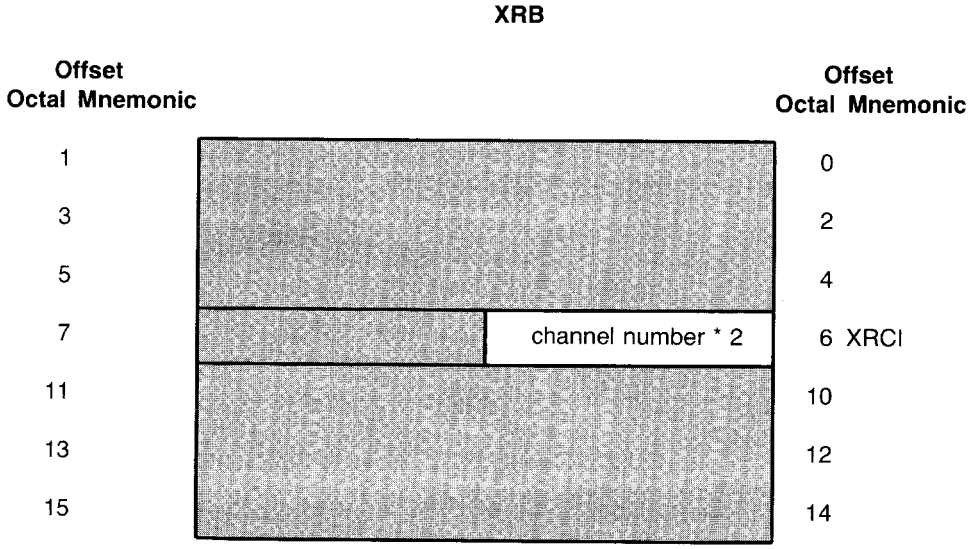
Form

```
.POSTN
```

Function

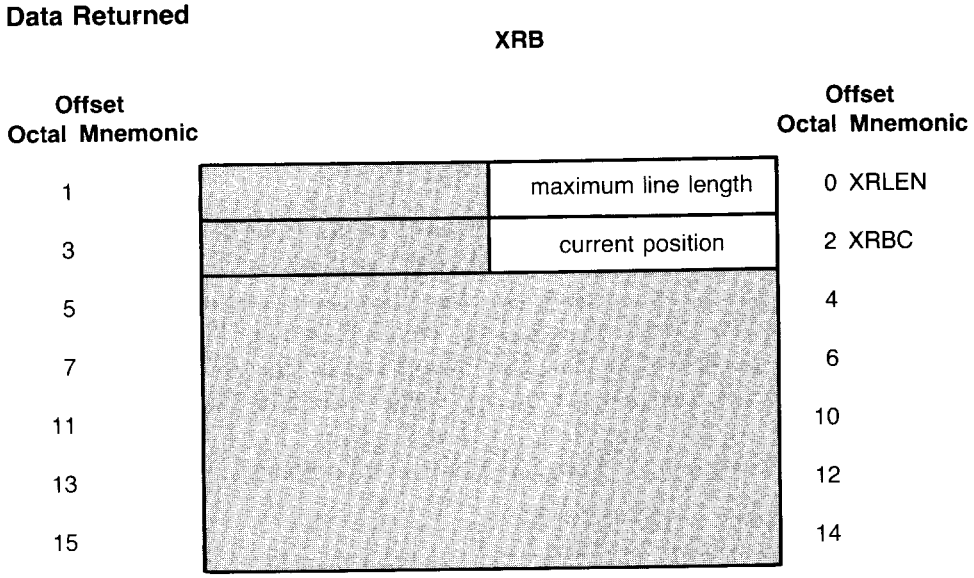
The .POSTN directive returns the maximum line width and the current horizontal position of devices for which this information is relevant (line printers and terminals). Data is passed in the XRB defining the channel where the device is currently opened. The information is returned in the XRB.

Data Passed



XRB+XRCI — Channel number times two; defines the channel on which the file or device is open.

Data Returned



XRB+XRLEN — The file/device's maximum line length plus one. A value of 81_{10} would indicate a maximum line length of 80 bytes (characters), for example.

XRB+XRBC — The file/device's current horizontal position is returned here. The value may range from 0 (leftmost character) to the value returned at XRB+XRLEN minus one (rightmost character). If the device does not keep track of its own horizontal position, then this value will be 0. The FLGPOS status bit returned in the FIRQB when the file/device was opened indicates whether the value returned here is meaningful.

Errors

NOTOPN — No file or device is open on the specified channel.

DETKEY — The device is a terminal that is detached.

Example

The following code requests the current horizontal position of the device open on channel 4.

```
MOVB    #4*2,XRB+XRCI           ;SET CHANNEL TO 4
.POSTN
```

3.17 .READ — Read Data from File or Device – Not Privileged

Form

```
.READ
```

Function

The .READ directive reads data from a file or device previously opened on a channel. The amount of data read depends on the device and the size of the buffer area, as defined in the XRB. The number of bytes transferred is always less than or equal to the buffer size. The actual number of bytes read is returned in the XRB when the directive is complete. Specific details for each device are given in Table 3–3. The "best guess" buffer sizes (returned by the monitor at FIRQB+FQBUFL when the device was opened) are also shown in Table 3–3, for comparison.

Table 3–3: Data Input with .READ

Device	Block Size, Bytes (Decimal)	"Best Guess,"* Bytes (Decimal)	.READ's Intent to Deliver	What Happens When: Buffer Size ≥Intent	What Happens When: Buffer Size <Intent
Byte-Oriented Devices (FLGFRC=1, FLGRND=0)					
Keyboard (Terminal)	N/A	128	1 line**	1 line**	Fill buffer; next .READ reads next part of line.
Pseudo Keyboard	N/A	128	Full buffer	N/A	N/A
Paper Tape Reader	N/A	128	Full buffer	N/A	N/A
Card Reader	N/A	160	1 card	1 card	Fill buffer; next .READ reads next part of card.
Block-Sequential Devices (FLGFRC=0, FLGRND=1)					
Magnetic Tape	18 to 30,000	512	1 block	1 block	Fill buffer; next .READ starts with new block. (Error returned.)
DECtape (file-structured)	510	510	1 block	1 block	Fill buffer; next .READ starts with new block. (No error returned.)
DMC/DMR	1 to 632	512	1 message	1 message	Delivers partial message. Next .READ delivers next message.

*Returned at FIRQB+FQBUFL when file or device was opened.

**A line is any number of characters terminated by RETURN, LINE FEED, ESCAPE, FORM FEED, CTRL/Z, CTRL/D, CTRL/C, or a user-set private delimiter.

(continued on next page)

Table 3–3: Data Input with .READ (Cont.)

Device	Block Size, Bytes (Decimal)	"Best Guess,"* Bytes (Decimal)	.READ's Intent to Deliver	What Happens When: Buffer Size ≥Intent	What Happens When: Buffer Size <Intent
Block-Random Devices (FLGFRC=0, FLGRND=0)					
Disk	512	512	Full buffer	Fills buffer; next .READ starts with new block.***	
Flexible Diskette	512 (block mode) or 128 (RX01 or RX02 single-density sector mode) or 256 (RX02 double-density sector mode)	512	Full buffer	Fills buffer; next .READ starts with new block or sector.***	
DECtape (non-file-structured)	512	512	1 block	1 block	Fill buffer; next .READ starts with new block. (No error returned.)

*Returned at FIRQB+FQBUFL when file or device was opened.

***If the buffer size is not a multiple of 512_{10} bytes, the remainder of the last block in will not be input. The next .READ will start with a new block: either the next sequential block (XRB+XRBLK=0) or the nth block (XRB+XRBLK=n). No error is returned.

Data Passed

XRB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1		size of the buffer in bytes (must not =0)		0	XRLEN
3		(must = 0)		2	XRBC
5		starting address of buffer		4	XRLOC
7	XRBLKM	MSB of block number	channel number * 2	6	XRCI
11		LSB of block number to begin (0 = next)		10	XRBLK
13		wait time for terminal input		12	XRTIME
15		device-dependent modifier		14	XRMOD

XRB+XRLEN — The length of the input buffer, in bytes. This word must be nonzero. The amount of data read depends on both the device and the buffer size. The amount of data will never be more than the buffer size, however. Details on reads for specific devices are given in Table 3–3.

XRB+XRBC — This word must be passed as zero. The monitor returns the actual number of bytes transferred in this word location, as described in the "Data Returned" section.

XRB+XRLOC — The starting address of the buffer. For disk, flexible diskette, and magtape devices, this address must be on a word boundary. For all other devices, the buffer can begin on an odd address.

The buffer, as defined by XRLOC for its start and XRLOC+XRLEN–1 for its last byte, must lie wholly within either the job image (low segment) or the run-time system's address space (high segment).

If the buffer is in the low segment, the address defined by the contents of XRB+XRLOC must be greater than 170 (octal) to avoid destroying job-context information used in swapping the job (Section 2.4).

If the buffer is in the high segment, it must not fall within the pseudo-vector region. That is, it must not fall above the location P.OFF (Section 2.5). In addition, the run-time system must currently be mapped read/write (see PF.RW bit description in P.FLAG word, Section 2.5). The run-time system must be read/write in this case, as the monitor will be writing data to the buffer for the .READ.

XRB+XRCI — Channel number times two; defines the channel for the read, as previously defined in an open (OPNFQ, CRTFQ, CREFQ, or CRBFQ functions of CALFIP, Section 3.2).

XRB+XRBLKM — For large files on disk (greater than 65,535 blocks), this byte contains the most significant bits of the block number to begin the read. This byte is combined with the word at XRBLK to form a 24–bit field defining the block number. This byte is ignored for nondisk devices.

XRB + XRBLK

For channels opened as file-structured, this word defines the starting block number for this read. (For large files, this word forms the least significant bits of the block number to start the read.) The value performs the same action as the BLOCK option for disk and the RECORD option for flexible diskette and non-file-structured DECtape, as described in the *RSTS/E Programming Manual.* This word is ignored if the device is not a random-access device. If the device is random-access and this field is nonzero, it is interpreted as the block number where the read is to start (1 to n, where n is the length of the file, in 512_{10}–byte blocks). If the field is zero, the next sequential block is read. For example, if a disk file is being read with a 1024_{10}–byte buffer size, a .READ with this parameter equal to 4 would cause the fourth and fifth blocks of the file (512_{10} bytes each) to be read into the buffer.

For channels opened on disk as non-file-structured MODE 0, this word defines the device cluster number where the read is to begin. In this case, each .READ begins with a new device cluster, so the buffer size at XRB + XRLEN should be a multiple of the device cluster size.

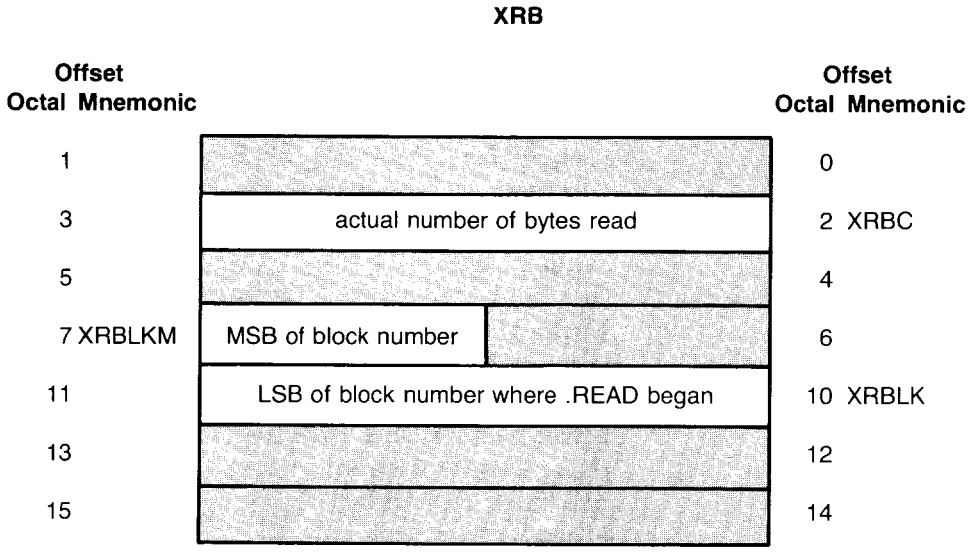
XRB + XRTIME

If positive, the maximum time to wait for terminal input data, in seconds. Zero indicates an infinite wait. A negative value (bit 15 equals 1) indicates an infinite "keyboard monitor wait." This wait is used by run-time systems that act as keyboard monitors for their command input. This type of wait time acts as a flag to the monitor and to the batch subsystem that the job is in a command input wait state.

XRB + XRMOD

Input operation modifier; significant only for card reader, terminal, DMC/DMR, or paper tape devices. (The monitor informs you with the FLGMOD bit of the flag word returned at FIRQB + FQFLAG on the open whether or not the device accepts modifiers.) This parameter performs the same action as the RECORD modifier in BASIC–PLUS for these devices, as described in the *RSTS/E Programming Manual.*

Data Returned



XRB + XRBC — Actual number of bytes just read. The value is between 0 and the value of XRB + XRLEN passed in the XRB. If an error is returned on the read (as indicated by byte 0 of the FIRQB), this word may or may not be zero. That is, data may be input even if an error occurs. For example, if the card reader detects an illegal card column punch combination, it places the decoded card data in the input buffer, substituting a special character for the bad column. The XRB + XRBC field is correctly set for the number of characters input, and an error is returned.

XRB + XRBLKM — For large disk files (greater than $65{,}535_{10}$ blocks), this byte contains the most significant bits of the block number just read. (See XRB + XRBLK.)

XRB + XRBLK — For random-access devices (see Table 3–3), this word contains the block number of the block just input with this .READ. Block numbers range from 1 to n (where n is the length of the file, in 512_{10}–byte blocks); they define the order in which the file was written. (For disks opened non-file-structured MODE 0, this is the device cluster number.)

Errors

BADCNT — The first three words describing the input buffer are illegal. (Illegal byte count for I/O.)

BSERR — The specified channel number is illegal.

NOTOPN — No file or device is open on the specified channel number.

PRVIOL — The file/device open on the specified channel is write-only, or the caller did not obtain read access to the file/device when it was opened.

All other errors are device-dependent. Some common errors are:

DATERR — Some data error occurred. There may or may not be any valid data in the input buffer. This error is issued for parity errors, bad card columns, and so on.

HNGDEV — Some hard device I/O error occurred. There is usually no data in the input buffer when this error occurs. (A .READ for an off-line device would cause this error.)

Example

In this example, we assume that a disk file has been opened on channel 2 and that the XRB has already been cleared to zero. Space for the buffer is allocated with the .BLKB directive, and the buffer size and location are defined. The next sequential block in the file is to be read. (Remember that the XRB has been filled with zeros, so XRB+XRBLK is already 0.)

```
BUFFER:   .BLKB    512.                ;ALLOCATE SPACE FOR BUFFER
             .
             .
             .
          MOV      #512.,XRB+XRLEN     ;SET BUFFER SIZE TO 512. BYTES
          MOV      #BUFFER,XRB+XRLOC   ;STARTING ADDRESS OF BUFFER
          MOVB     #2*2,XRB+XRCI       ;CHANNEL 2 FOR INPUT
          .READ
```

3.18 .RSX — Execute Job and Disappear (RSX only) – Not Privileged

Form

```
.RSX
```

Function

The .RSX directive is used only by the RSX run-time system to transfer control to the monitor to start a user job image that has been loaded by the RSX run-time system.

If RSX directive emulation has been installed as part of the monitor, the monitor passes control to the user job image, and the RSX run-time system will "disappear" from the high segment of the job space. If RSX directive emulation has not been installed as part of the monitor, processing continues as "normal," that is, with the run-time system as the high segment.

The .RSX call can be issued only from the high segment. The monitor checks the PC register, to ensure that the call originated in the high segment, and the SP register, to make sure that the stack is less than or equal to 1000. Further, this call is relevant only to the RSX run-time system. It is documented here for completeness and to provide an understanding of how the RSX run-time system "disappears."

The RSX run-time system passes relevant parameters to the monitor on the stack, including any requests from the program or from the user to make use of the extra space in APR 7. (Such information is built into the task header of the executable file by the Task Builder or can be requested by the user with the /SIZE switch in the CCL execute line.) There are three possibilities for requests: (1) no request to use the extra space, (2) a request to map a window with the extra space, or (3) a request to extend the user job image into the extra space.

3.18.1 No Requests for Extra Space

If there are no requests for extra space, the RSX run-time system places on the stack only the information needed to start the user job image:

```
SP ──►  New SP for program
        New PC for program
        New PSW for program
```

The RSX run-time system then issues the .RSX directive to pass control to the monitor. The monitor checks the value of the first word on the stack to see if it is even or odd. In this case, it is an address (of the program's stack), and so an even value. If RSX directive emulation has been installed as part of the monitor, the monitor releases APR 7 from mapping to the RSX run-time system and passes control to the user job image, loading the appropriate registers from the information provided in the stack. If RSX directive emulation has not been installed as part of the monitor, the monitor does not release APR 7 from mapping to the RSX run-time system. It simply passes control to the user program according to the information provided on the stack.

3.18.2 Request for Mapping a Window in APR 7

When there is a request for mapping a window in APR 7, the RSX run-time system places on the stack the information needed to create and map the window and to run the user job image. The first two words in the stack are in the same format as a "Create Address Window" request (see CRAW$, Section 5.6).

SP → High byte = 2, low byte = 117_{10}
Address of 8–word Window Definition Block
New SP for program
New PC for program
New PSW for program

The RSX run-time system then passes control to the monitor with the .RSX directive. The monitor checks the first word in the stack to see if it is even or odd. In this case, it is odd. If RSX directive emulation has been installed as a part of the monitor, the monitor releases APR 7 from its mapping to the RSX run-time system and loads it to map the requested window. Then it passes control to the user job image according to the information on the stack. If RSX directive emulation has not been installed as a part of the monitor, the monitor returns control to the run-time system with an error. (The requested mapping cannot be done.)

3.18.3 Request to Extend the User Job Image

When there is a request to extend the user job image, the RSX run-time system places on the stack the information needed for the extension as well as to start the user job image. The first three words in the stack are in the same format as an "Extend Task" directive to the RSX emulator (see EXTK$, Section 5.11).

SP → High byte = 3, low byte = 89_{10}
Number of 32–word blocks to extend
Reserved word
New SP for program
New PC for program
New PSW for program

The RSX run-time system then passes control to the monitor with the .RSX directive. The monitor checks the first word in the stack to see if it is even or odd. In this case, it is odd. If the RSX emulator has been installed as a part of the monitor, the monitor releases APR 7 from its mapping to the RSX run-time system, remaps part of the APR to the user, and passes control to the user job image according to the information passed in the stack. If RSX directive emulation has not been installed as a part of the monitor, the monitor returns control to the RSX run-time system with an error.

3.19 .RTS — Pass Control to Run-Time System – Not Privileged

Form

```
.RTS
```

Function

The .RTS directive passes control to a run-time system at the P.NEW entry point, where the intent is not to run an executable file (see P.NEW, Section 2.5.4). .RTS performs one of four functions:

1. Passes control to the job's keyboard monitor
2. Passes control to a named run-time system
3. Passes control to a named run-time system and establishes it as the job's keyboard monitor
4. Passes control to a named run-time system without changing job context information

The first function is generally part of a run-time system's exit processing; you do not normally use .RTS for this purpose in a user program. The following code shows how .RTS is used in exit processing (for example, in the RSX run-time system's EXIT$ directive):

```
EXIT:     CALL    CLRXRB          ;Clear XRB
          .RTS                    ;Exit to default KBM
          JMP     PROMPT          ;If that's this KBM, then
                                  ;prompt for another command
```

The other three functions of .RTS apply to both user programs and run-time systems.

Note that once established, a job's keyboard monitor replaces the default keyboard monitor as the one to which control passes in default situations. This concept is best explained by example.

The SWITCH program (see the *RSTS/E System User's Guide*) uses the .RTS directive to establish the run-time system to which it passes control as the job's keyboard monitor. Consider the following sequence:

```
Ready

SWITCH RT11
.MAC
MAC>CTRL/C
.CTRL/C
.SWITCH

Ready
```

The first line shows the BASIC–PLUS "Ready" prompt, indicating that the user is in the BASIC–PLUS keyboard monitor. The user then runs SWITCH to pass control to the RT11 run-time system. SWITCH establishes RT11 as the job's keyboard monitor. The RT11 run-time system displays the period prompt, and the user runs "MAC," the RSX run-time system's assembler. Control passes to the RSX run-time system, which loads the assembler and runs it. MAC displays the MAC> prompt. The user, realizing the mistake in typing MAC instead of MACRO, types CTRL/C to exit. Since RT11 has been established as the job's keyboard monitor, control passes back to RT11, and it displays the period prompt. The user, wanting to get back to BASIC–PLUS, types another CTRL/C. Since RT11 is the job's keyboard monitor, however, another period prompt appears. The user finally runs SWITCH from the RT11 run-time system, leaving out the run-time system name. SWITCH transfers control back to the default keyboard monitor, establishing it once again as the job's keyboard monitor.

In any case, the four ways that the .RTS directive can be used are:

1. The .RTS directive is used to switch control back to the run-time system already established as the job's keyboard monitor. In this case, the name of the run-time system is not known. The word in the FIRQB that would contain the first part of the run-time system name is 0. If the run-time system that issues the .RTS is not itself the job's keyboard monitor, control is passed to the job keyboard monitor at entry point P.NEW. If the run-time system that issues the .RTS is the job's keyboard monitor, control returns inline (to the instruction following the .RTS) with no error indicated.

2. The .RTS directive is used to switch control to the run-time system named in the FIRQB. (Run-time systems are made known to the monitor by the system manager with the ADD command of the UTILTY program. With this command, the system manager defines a file (filename.RTS) as an auxiliary run-time system. The monitor regards "filename" as the run-time system's name.) If the run-time system that issues the .RTS directive is not itself the named run-time system, control passes to the named run-time system at the P.NEW entry point. If the run-time system that issues the .RTS directive is the named run-time system, control returns to the instruction following the .RTS with no error indication. If the named run-time system does not exist or is not available for some reason, control returns to the instruction following the .RTS with an error in byte 0 of the FIRQB.

3. The .RTS directive is used to switch control to a named run-time system and establish the named run-time system as the job's keyboard monitor. If the run-time system that issues the .RTS is not itself the named run-time system, control passes to the named run-time system at the P.NEW entry point, and the named run-time system is established as the job's keyboard monitor. If the run-time system that issues the .RTS names itself as the run-time system, control returns to the instruction following the .RTS with no error, and the run-time system is established as the job's keyboard monitor. If the named run-time system does not exist or is unavailable, control returns to the instruction following the .RTS with an error in byte 0 of the FIRQB.

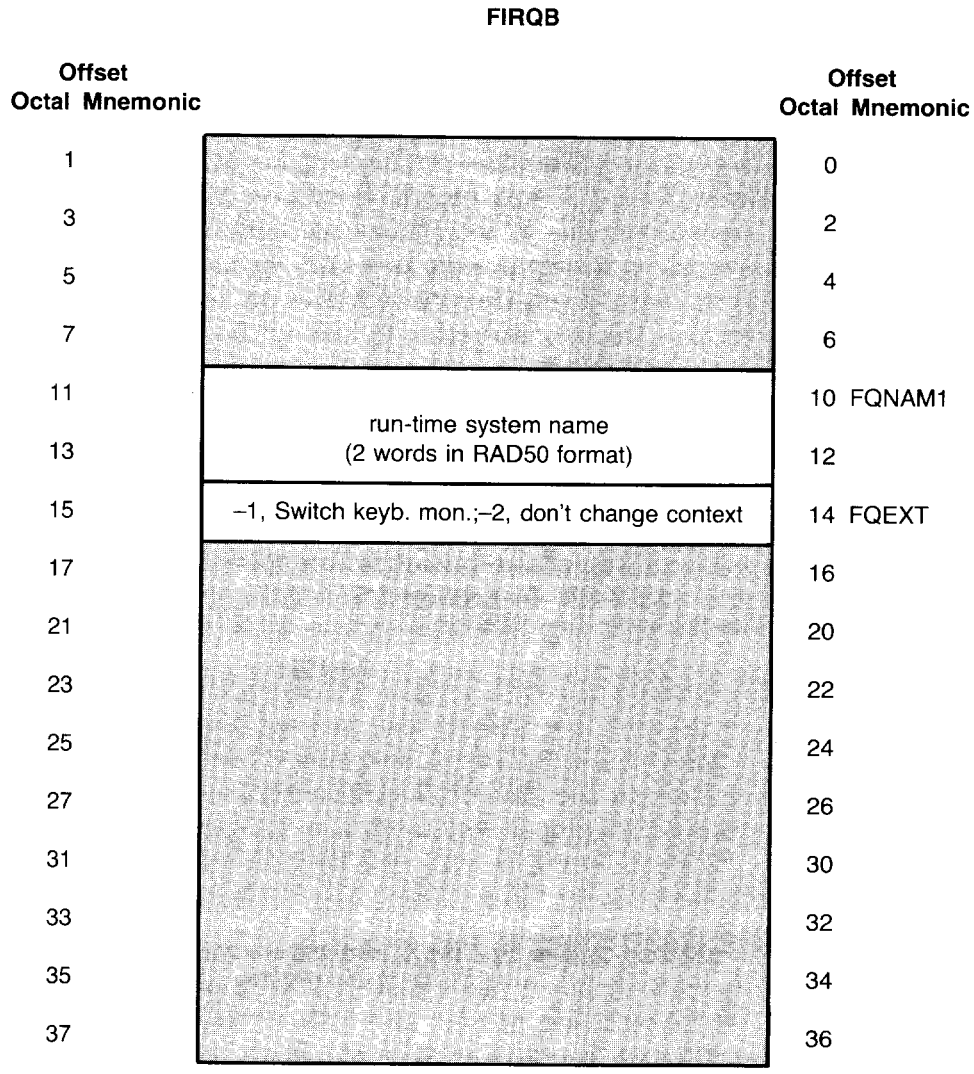
4. The .RTS directive is used to switch control to a named run-time system preserving the job's current job-context information. Control passes to the named run-time system at the P.NEW entry point, but the monitor does not refresh the keyword, reset the stack pointer, or perform any of the initialization operations described in the discussion of P.NEW in Section 2.5.

NOTE

This directive should not be used from programs running under the RT11 run-time system, because the lowest 1000_8 bytes are used by RT11 differently from other run-time systems. The proper way to terminate a program running under RT11 is to exit through the RT11 emulator.

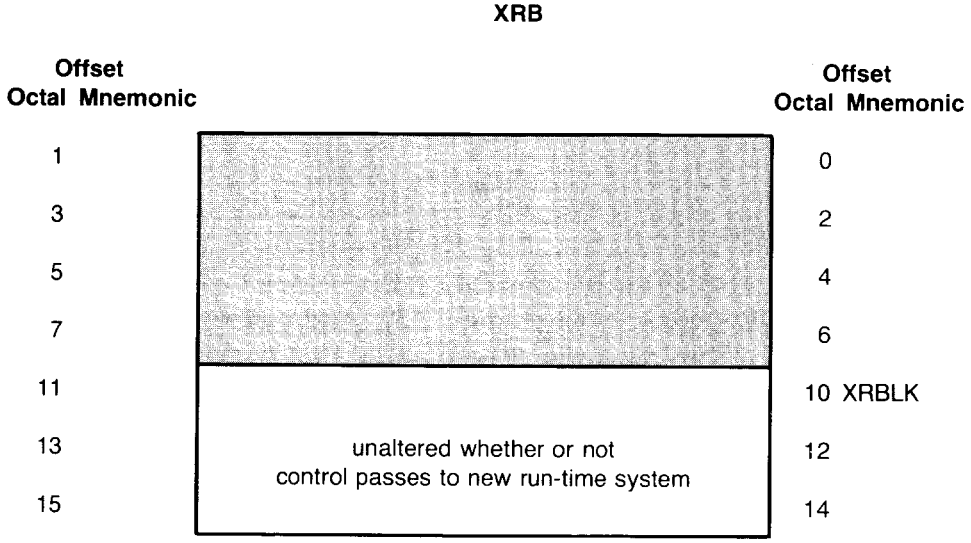
Although you can use this directive to terminate a program running under the RSX run-time system, DIGITAL recommends that you use the RSX EXIT$ or EXST$ directive instead (see Sections 5.9 and 5.10). Using .RTS may cause unexpected results. For example, if you use .RTS to exit from a program running under the RSX run-time system, and RSX is also your job keyboard monitor, control returns inline.

Data Passed



FIRQB + FQNAM1 If control is being passed to a named run-time system (cases 2, 3, and 4 described previously), the name of the run-time system is stored as two words in RAD50 format beginning here. If control is being passed to the job's keyboard monitor (case 1 described previously), the word beginning here must contain a value of 0.

FIRQB + FQEXT To establish a named run-time system as the job's keyboard monitor (case 3), set this word to –1. To switch control to a named run-time system without altering job-context information (case 4), set this word to –2. When the word at FIRQB + FQNAM1 is 0, this word is ignored.



XRB+XRBLK The three words starting at this location pass unaltered to the new run-time system. They are also unaltered if control returns inline.

Data Returned

Since control usually passes to some other run-time system at its P.NEW entry point, no arguments as such are returned by .RTS. The three words starting at XRB+10 remain unaltered if control returns inline.

Errors

NORTS No run-time system exists with the specified name.

PRVIOL The named run-time system does exist but cannot be switched to for some reason. (For example, if the switch is made to a named run-time system with the intent of establishing it as the job's keyboard monitor and its PF.KBM bit in the P.FLAG word is not set to 1.)

Example

The following example establishes "NEWRTS" (which the system manager must have added with UTILTY) as the job's keyboard monitor:

```
MOV     #^RNEW,FIRQB+FQNAM1          ;SET RUN-TIME SYSTEM
MOV     #^RRTS,FIRQB+FQNAM1+1        ;NAME TO "NEWRTS"
MOV     #-1,FIRQB+FQEXT              ;ESTABLISH AS JOB KBM
.RTS
```

3.20 .RUN — New Program to Run – Not Privileged

Form

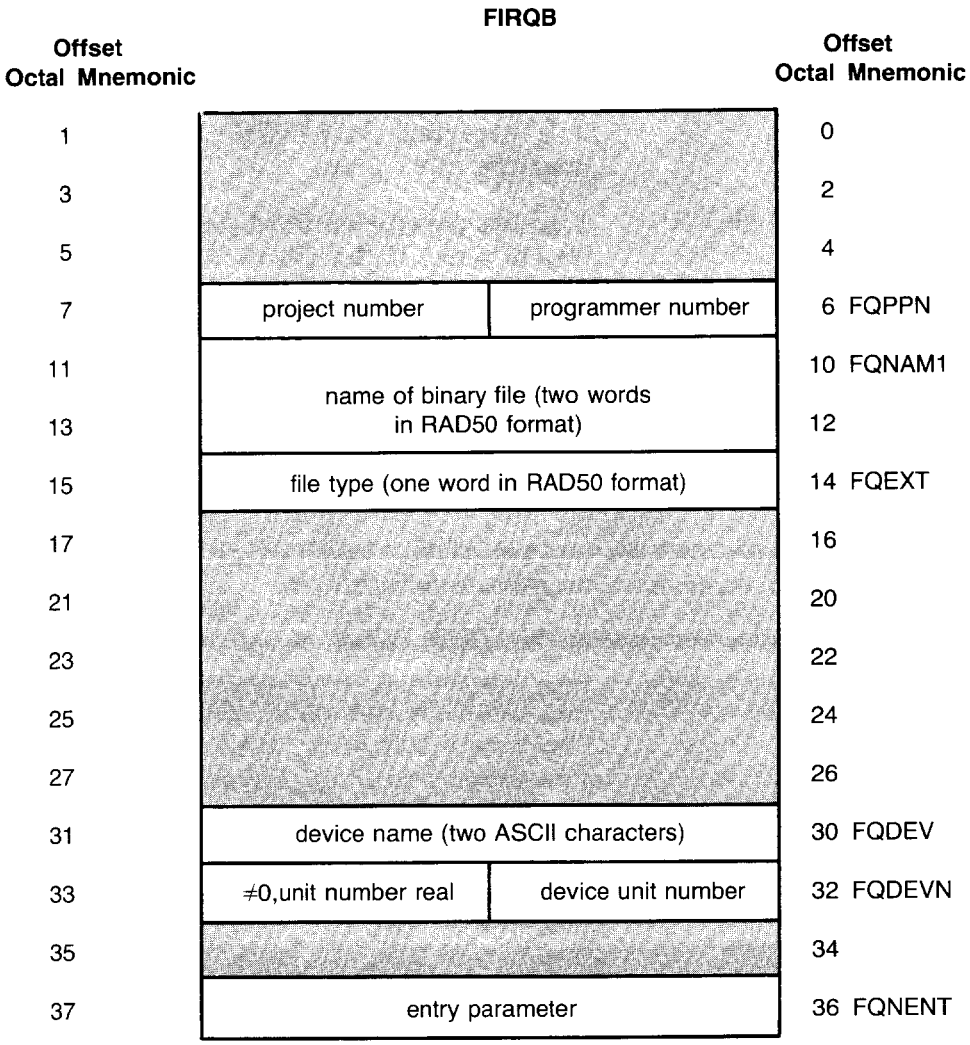
```
.RUN
```

Function

The .RUN directive searches for a binary (executable) file (defined in the FIRQB), opens it on channel 15_{10}, and passes control to the P.RUN entry point of the run-time system associated with the file. The associated run-time system is identified in the file's directory information. This directory information is initially set to indicate the run-time system under which the file was created. The system manager can change the run-time system associated with the file with the NAME command of UTILTY (see the *RSTS/E System Manager's Guide*).

The run-time system is responsible for loading and executing the file (see the P.RUN description in Section 2.5). Control is not returned inline unless an error occurs.

Data Passed



FIRQB + FQPPN — The project-programmer number of the file to be opened. The project number is specified in the high byte, the programmer number in the low byte. If this word is passed as 0, the ppn defaults to the project-programmer number of the job that issues the .RUN directive.

FIRQB + FQNAM1 — The file name of the file to be opened, as two words in RAD50 format, begins at this location.

FIRQB + FQEXT — The file type for the file to be opened, as one word in RAD50 format. If you set this word to –1, the monitor will search for the file name supplied, substituting the default file type for the currently installed set of run-time systems until a file with the given name and one of the default file types is found.

FIRQB+FQDEV — The device name of the file to be opened, as two ASCII characters. If the device name is not a disk, the .RUN directive returns inline with one of the "soft" errors described in the "Errors" section. If you pass a full word of 0 here and in FIRQB+FQDEVN, the public disk structure (SY:) is searched for the named file.

FIRQB+FQNENT — A 16–bit parameter word can be passed to the run-time system here. If the job issuing the .RUN is privileged, the word will be passed unaltered. If the job is nonprivileged, bit 15_{10} (the sign bit) will be cleared unconditionally. The monitor takes no other action as a result of the contents of this word; any processing is up to the run-time system. (See the P.RUN entry point, Section 2.5.)

Errors

When an error occurs in a .RUN, FIRQB+FQFLAG is set to 0 to indicate a "hard" error or –1 to indicate a "soft" error.

A hard error means the .RUN failed. A soft error also means the .RUN failed, but the run-time system may be able to recover. For example, when you use the BASIC–PLUS RUN command, BASIC–PLUS first executes a .RUN. If the .RUN returns a soft error, BASIC–PLUS performs an OLD (which compiles the program) and then executes .RUN again.

Hard Errors

BADNAM — The specified file name was 0. This is an illegal file name for disk files.

DEVNFS — The specified disk device is currently being used non-file-structured.

NOTCLS — Channel 15_{10} is currently open. It must be closed before any .RUN call.

NODEVC — The specified device does not exist or is in an illegal format.

NORTS — The run-time system named in the file's directory information has not been installed.

NOTMNT — The specified disk device is not now mounted.

PAKLCK — The disk pack on which the file exists is locked against further file opens.

PRVIOL — Some protection violation occurred, such as attempting to run a file that is protected against the caller.

Soft Errors

PRVIOL	The device is not disk but is still a legal device on the system. Or the file was found and is on disk but does not have the "compiled program" bit set in its file protection code (bit 6).
NOSUCH	The file was not found. Note that if FIRQB + FQEXT was –1 in the data passed, the monitor has looked for the given file name with all default runnable file types for the currently installed run-time systems. In this case, a source version of the file may yet exist.

Example

The following example uses the .FSS directive to translate a user-typed string to the FIRQB format. If no errors in the .FSS occur, a test is made to see if a file type was specified. If not, a –1 is supplied in FIRQB + FQEXT so that the monitor searches for the given file name with all possible default runnable file types.

```
(read user-typed line, set up FIRQB for .FSS)
                .
                .
                .
          .FSS
          TSTB    FIRQB                ;ERROR ON .FSS?
          BNE     ERRTN                ;BRANCH TO PROCESS ERROR
          BIT     #100000,XRB+XRBLK    ;INVALID DEVICE NAME?
          BNE     BADDEV               ;BRANCH TO PROCESS ERROR
          BIT     #10,XRB+XRBLK        ;FILE TYPE GIVEN?
          BNE     SKIP1                ;YES, SKIP NEXT
          MOV     #-1,FIRQB+FQEXT      ;NO, SEARCH ALL
SKIP1:    .RUN
```

3.21 .SET — Set Keyword Bits – Privileged and Not Privileged

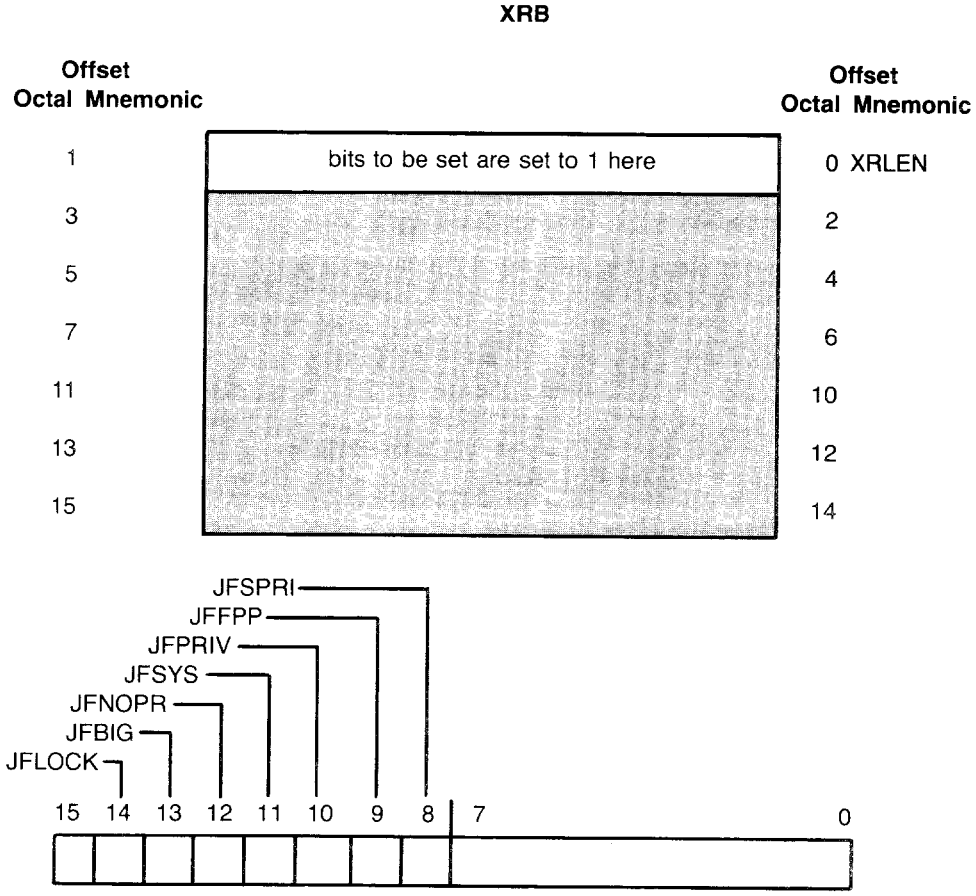
Form

```
.SET
```

Function

The .SET directive sets certain bits in the keyword (KEY) location in the user job image (Section 2.4). The bits to be set are passed to the monitor in the XRB.

Data Passed



JFLOCK Can be set only by privileged jobs. When this bit is set, the monitor will swap the user job image out of memory only when:

1. The job issues a .CORE directive to expand the memory allocated for the user job image, and there is not sufficient room in physical memory to do the expansion. In this case, the job is swapped out to disk and back in at the indicated size.

JFLOCK (cont.) 2. A fatal error, such as a memory parity failure, occurs.

JFBIG Can be set only by privileged jobs. When this bit is set, the job is allowed to exceed its private memory maximum (see .CORE, Section 3.6).

JFNOPR Cannot be set by any job; masked off.

JFSYS Can be set by a nonprivileged job only if JFSYS was set at one time and the temporary privileges gained were only temporarily dropped. (See description of KEY, Section 2.4.)

JFPRIV Cannot be set by any job; masked off.

JFFPP Can be set by any caller if the PDP–11/45 compatible hardware floating-point unit exists; masked off if it does not exist. When this bit is set, the monitor will save information in the floating-point unit as part of the job-context information kept when jobs are swapped in and out of memory.

JFSPRI Can be set only by a privileged caller. Setting JFSPRI raises the job's run priority by one-half step. (That is, it sets bit 2 of the system-controlled low-order three bits of the run priority. Priorities are normally set by the system manager with UTILTY.)

All other bits in the XRB word are masked off; that is, the corresponding bits in KEY cannot be set by the job with the .SET directive.

Data Returned

No data is returned with the .SET directive.

Errors

No errors are possible with the .SET directive.

Example

The following code sets JFBIG, allowing the job to exceed its private memory maximum:

```
MOV     #JFBIG,XRB+XRLEN        ;SET JFBIG
.SET
```

3.22 .SLEEP — Suspend Job – Not Privileged

Form

```
.SLEEP
```

Function

The .SLEEP directive causes the monitor to suspend the job for some specified time interval or until an event occurs that the job should be aware of. Optionally, the monitor checks, before the job is suspended, to see if some event has already occurred which would cause it to awaken. If so, the job is not suspended. Control returns inline in either case.

When a .SLEEP is executed, execution of the job is suspended until one of the following happens:

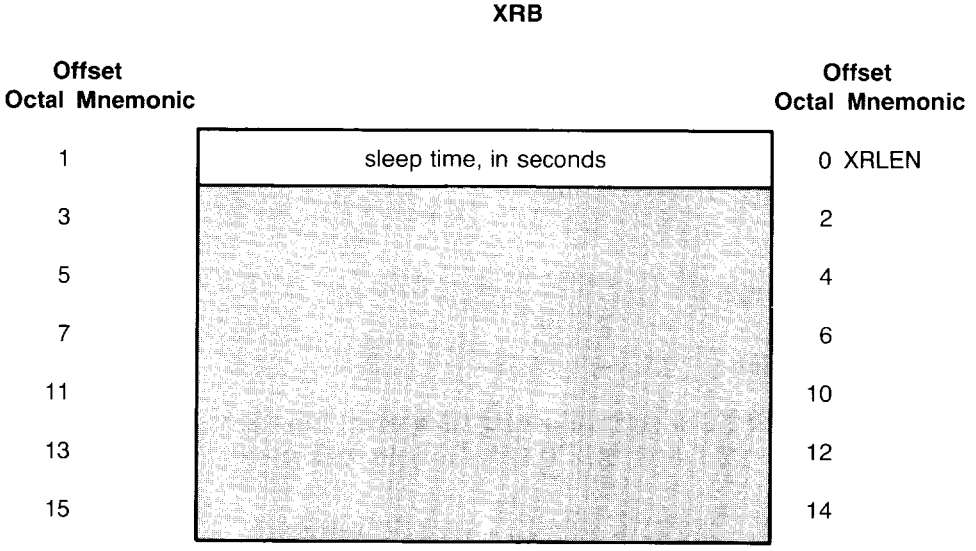
1. The sleep time (specified in the XRB) expires.
2. A local or network message is queued for the job (assuming that the job is using local or network send/receive services. See .MESAG, Section 3.12).
3. A delimiter is typed on a terminal that this job has opened or assigned.
4. The system manager disables logins. (This could occur if the system is being shut down.)
5. A state change occurs on a pseudo keyboard assigned to the job. (The job has printed output for the controlling job to read or has entered an input wait state.)
6. The DMC11/DMR11 driver (XM:) is open and a message is pending for the job.

If you request it, the monitor will check before suspending the job for the following conditions:

1. A delimiter has been typed on any terminal opened by the job or any terminal assigned to the job if the job also has a keyboard open on a nonzero channel.
2. A message has been queued for the job.
3. A state change has occurred on a pseudo keyboard opened by the job.
4. The job has a DMC11/DMR11 device driver open and a message has been received by that device driver.

If the monitor determines that any of these conditions are true, it does not execute the .SLEEP.

Data Passed



XRB+XRLEN This word defines the sleep interval, in seconds. If the value is 0, then .SLEEP returns immediately. If the sign bit (bit 15_{10}) is set, the monitor checks to see if any condition that would cause the sleep to terminate has already occurred. If so, the .SLEEP is not executed.

Data Returned

No data is returned with the .SLEEP directive.

Errors

No errors are possible with the .SLEEP directive.

Example

```
MOV     #5,XRB+XRLEN            ;SET TIMER TO 5 SECONDS
.SLEEP
```

3.23 .SPEC — Special Functions for I/O

Form

```
        ;
        ;
        ;
(set XRB for special function)
        ;
        ;
        ;
    ;SPEC
```

Function

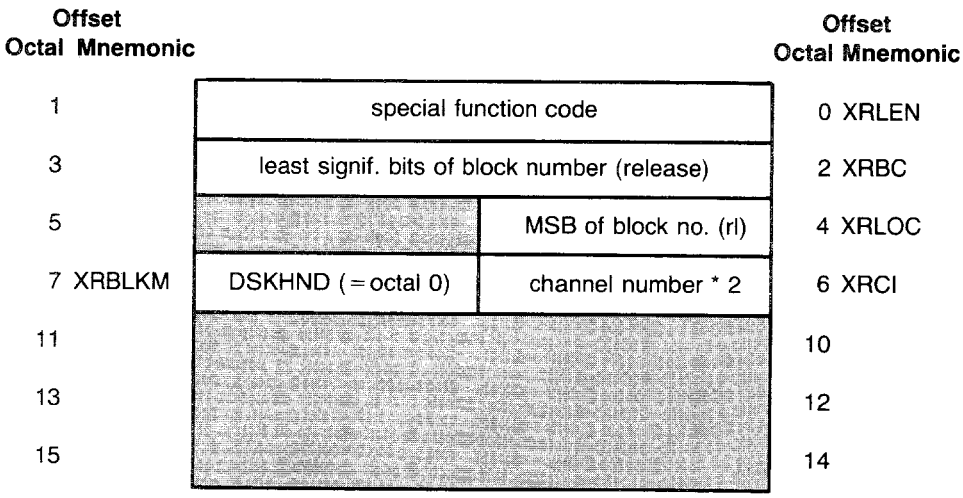
The .SPEC directive performs special functions for disk, terminal, magnetic tape, and RX01 and RX02 flexible diskettes.

3.23.1 .SPEC for Disk – Not Privileged

For disk, the .SPEC directive lets you explicitly lock up to seven disk blocks on a file open for update (mode parameter). A locked block cannot be accessed by another user (or from another channel). This extends the "implicit lock" feature, by which the last block or blocks read on a file open for update cannot be accessed by anyone else. The disk special functions also let you release explicit and implicit locks. (All locks, both explicit and implicit, are released when the file is closed.)

Data Passed (Disk)

XRB



XRB+XRLEN Defines function to be performed.

0 Release any implicit lock and all explicit locks. The monitor deallocates the extended internal table space it needed to do the explicit locks. (See code 3.)

1 Release implicit lock.

2 Make implicit lock into explicit lock.

3 Release the explicit lock on the block specified by the word at XRB+XRBC (least significant bits) and the byte at XRB+XRLOC (most significant bits). If all three bytes are zero, all explicit locks are released, but the monitor does not deallocate the extra space needed to do explicit locks. (This may be useful if you intend to use the explicit lock feature again during this run. An error occurs if no space is available for this purpose.)

4 Make implicit lock into explicit lock and release the implicit lock.

XRB+XRBC, XRB+XRLOC Both these bytes specify the starting block number for releasing an explicit lock. If these bytes are zero, all explicit locks are released, but the monitor retains the extended table area it needs to maintain these locks. (This may be useful if you wish to use this capability again during a run.)

XRB+XRCI Channel number times two; defines the channel for the lock/unlock operation.

XRB+XRBLKM Handler index for disk: DSKHND (octal value = 0).

Data Returned

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the disk subfunctions of .SPEC.

Errors

For code 2:

INTLCK Occurs if the implicit lock overlaps any current explicit lock. For example, if you read blocks 1 and 2 into a 1024–byte buffer in update mode, an implicit lock exists on blocks 1 and 2. If you explicitly locked these blocks, and then read blocks 2 and 3 and tried to explicitly lock blocks 2 and 3, you would get this error. An exact match is legal (for example, if the second read also read blocks 1 and 2) and results in a no-op.

NOBUFS	Occurs if the monitor needs to expand its internal table space but memory is not available.
NOROOM	There are already seven explicit locks on this channel.
PRVIOL	There is no current implicit lock; that is, no blocks have been read.

For code 3:

NOSUCH	The block number specified does not correspond to the first block number of an explicit lock.

3.23.2 .SPEC for Terminal – Privileged and Not Privileged

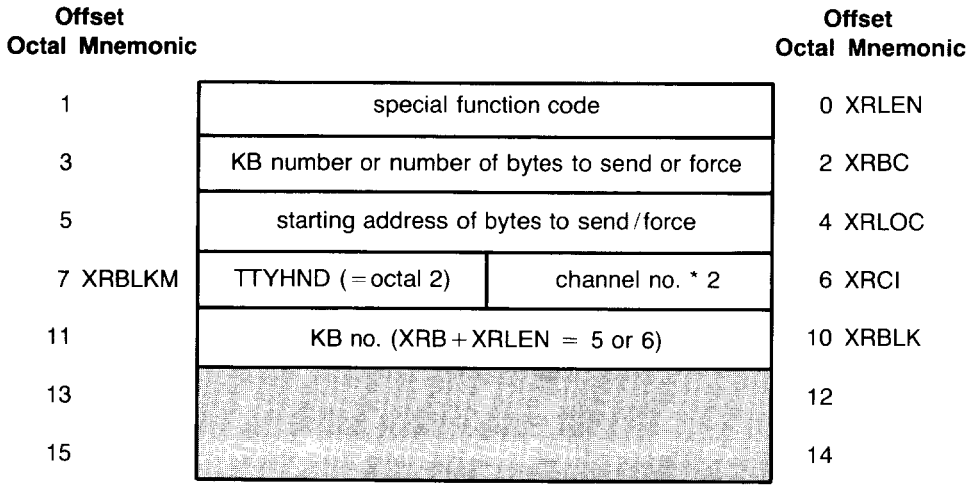
The .SPEC directive for terminals has two forms. The first form lets you perform several different functions, such as cancel CTRL/O, set modes for tape, echo, and ODT, and cancel type ahead. The second form lets you set, read, and clear private delimiters.

The next two sections show the data passed and returned for each form of .SPEC for terminals.

3.23.2.1 All but Private Delimiters —

Data Passed

XRB



XRB+XRLEN Defines a special function:

0 = Cancel CTRL/O (see .TTRST)
1 = Set tape mode (see .TTAPE)
2 = Enable echo and clear tape mode (see .TTECH)
3 = Disable echo (see .TTNCH)
4 = Set ODT mode (see .TTDDT)
5 = Force to keyboard (privileged)
6 = Broadcast to keyboard (privileged)
7 = Cancel all type-ahead

XRB+XRBC — For XRB+XRLEN = 0, 1, 2, 3, 4, or 7:

When XRB+XRBC equals 0, these functions take place on the terminal currently open for this job. When XRB+XRBC does not equal 0, these functions take place on the keyboard number specified in XRB+XRBC. This keyboard must be assigned to but not opened by the calling job.

For XRB+XRLEN = 5 or 6:

XRB+XRBC is the number of bytes to send or force.

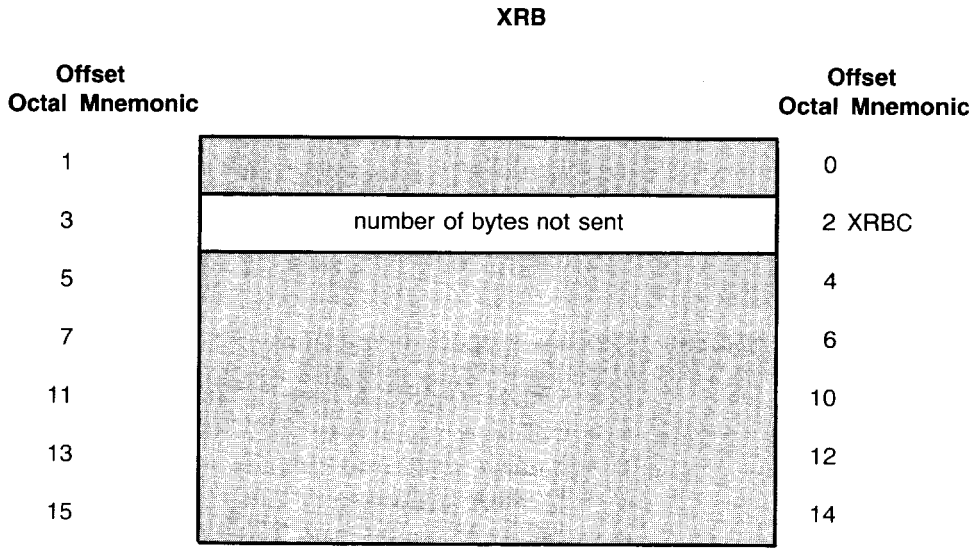
XRB+XRLOC — For XRB+XRLEN equal to 5 or 6, this word contains the starting address of the bytes to be sent or forced.

XRB+XRCI — Channel number times two; defines the channel for the terminal specified at XRB+XRBC.

XRB+XRBLKM — Handler index for terminals; TTYHND (octal value = 2).

XRB+XRBLK — For XRB+XRLEN = 5 or 6, this word contains the keyboard number to which the data is to be sent or forced.

Data Returned (Terminal)



XRB+XRBC — Number of bytes that could not be sent (returned only when XRB+XRLEN in the data passed was 6).

Errors (Terminal — Except Private Delimiter)

PRVIOL One of the following:

1. The calling job does not own the specified keyboard and is not privileged.
2. The function code is not 0 – 7 or 11 (returned for all .SPEC calls for terminals).
3. An illegal terminal number at XRB+XRBLK.
4. The device open on the channel at XRB+XRCI is not a terminal.

BSERR An illegal channel number is specified at XRB+XRCI.

NOTOPN The channel specified at XRB+XRCI is not open.

3.23.2.2 Private Delimiters —

A "private delimiter" is a character used as a delimiter within a program. You can define any printing or nonprinting character to be a private delimiter:

- A letter
- A function key such as DELETE
- A control character such as CTRL/Z
- A standard delimiter such as LINE FEED

A private delimiter is useful on a data entry terminal with a specialized keyboard. You can use a large or conveniently located key as the delimiter key. Private delimiters are also useful in keypad applications.

You can declare one character as a private delimiter on any RSTS/E system. In addition, a system generation option allows the use of multiple private delimiters. If your system has this feature, you can declare up to 256 private delimiters.

Multiple private delimiters let you do special character processing without using single character I/O. For example, by combining escape sequences with private delimiters, you can define your own function keys in keypad applications.

The rest of this section:

- Provides general information about private delimiters
- Shows the XRB layouts for setting, reading, and clearing private delimiters
- Lists all private delimiter masks

Characteristics of Private Delimiters

Declaring a character as a private delimiter with the .SPEC directive overrides the existing ASCII code for the character. Thus, unlike a standard delimiter such as RETURN or LINE FEED, a private delimiter does not echo at the terminal. In addition, a special character no longer performs its normal function. For example, when the DELETE key is a private delimiter, it does not erase the last character typed.

A private delimiter has basically the same characteristics as a standard delimiter. Like a standard delimiter, it:

- Terminates a .READ on the terminal.
- Cannot be deleted. The DELETE key and CTRL/U do not affect private delimiters in the type-ahead buffer.
- Causes the system to awaken a sleeping job when typed at a terminal that the job has open or assigned. If the job cannot be awakened, the system stores the private delimiter character.

Once set, a private delimiter remains in effect for a terminal until one of the following occurs:

- The program clears it
- The job releases the terminal by deassigning it or by closing the I/O channel where the terminal is open
- The job terminates

In addition, the system clears private delimiters when a dial-up line gets hung up or the job controlling the terminal is killed.

Private delimiters change the way characters are processed in binary mode (MODE 1). When a terminal is open in binary mode and no private delimiter is in use, the system terminates a read after every character. However, if one or more private delimiters are in use, the system terminates a read only when a private delimiter is typed.

The system processes private delimiters after processing CTRL/S and CTRL/Q (if the STALL characteristic is set) and escape sequences (if the terminal is in ESC SEQ mode). This feature prevents a terminal from becoming permanently stalled, and it also lets you use private delimiters and escape sequences in the same program.

The system processes private delimiters before all other characters, including control characters such as CTRL/C. Thus, when you use a standard delimiter character as a private delimiter, it does not echo on the terminal.

Programming Hint

By combining escape sequences with private delimiters, you can define your own function keys without using single character I/O. First, make sure the keypad is in the right mode for your application. Define each function as the PF1 key followed by a character. Then define each character as a private delimiter so it does not echo on the terminal. For example, you might define PF1 + A as one function and PF1 + M as another function.

Data Passed (Terminal — Private Delimiters)

XRB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1		= octal 11 for private delimiters		0	XRLEN
3		0 or byte count when XRB+XRMOD = 1 or 2		2	XRBC
5		0 or address when XRB+XRMOD = 1 or 2		4	XRLOC
7	XRBLKM	TTYHND (=octal 2)	channel number *2	6	XRCI
11		flag byte	KB number	10	XRBLK
13		must = 0		12	XRTIME
15		subfunction code = 0, 1, or 2		14	XRMOD

XRB+XRLEN — Function code; set to 11_8 for private delimiter mask.

XRB+XRBC — For XRB+XRMOD equals 0, this word must be set to 0.

For XRB+XRMOD equals 1 (set private delimiter mask), this word is the byte count for the private delimiter bit mask.

For XRB+XRMOD equals 2 (read private delimiter mask), this word is the buffer length for a buffer into which the mask is to be read.

For XRB+XRMOD equals 1 or 2, the value of XRB+XRBC must be greater than or equal to 40_8.

XRB+XRLOC

For XRB+XRMOD equals 0, this word must be set to 0.

For XRB+XRMOD equals 1, this word is the address of the private delimiter mask. The mask itself can be up to 40_8 bytes long (40_8 bytes = 256_{10} bits). Each bit in the mask represents an ASCII character, as indicated by Table 3–4. Setting a bit indicates that the associated ASCII character is to serve as a private delimiter.

For XRB+XRMOD equals 2, this word is the address of a buffer into which the terminal's private delimiter mask is to be read.

XRB+XRCI

Channel number times two. When this byte equals 0, it indicates the job's keyboard. When this byte does not equal 0, a keyboard must be open on the indicated channel.

XRB+XRBLKM

Handler index for terminals; TTYHND (octal value is 2).

XRB+XRBLK

Specifies the keyboard number that the subfunction is to take place on. If this byte and the byte at XRB+11 equal zero, then the subfunction is performed on the terminal open on the channel indicated by XRB+XRCI.

XRB+11

Flag byte. If bit 7 of this byte is set (byte equals 200_8), it indicates that the keyboard number at XRB+XRBLK is real. (When XRB+XRBLK is zero, this bit set indicates KB0: is the desired terminal. If this bit is cleared and XRB+XRBLK is zero, it indicates that the keyboard open on the channel indicated at XRB+XRCI is the desired terminal.) All other bits must equal zero.

XRB+XRTIME

This word must be set to zero.

XRB+XRMOD

Subfunction code:

0 = Clear private delimiter mask.
1 = Set private delimiter mask.
2 = Read private delimiter mask.

Once set, private delimiters remain in effect for a terminal until cleared (XRB+XRMOD = 0) or until implicitly cleared by:

1. Deassigning or closing the terminal.
2. Killing the job or hanging up the line.

XRB+XRMOD (continued)

3. Keyboard monitor read (negative wait time on .READ).

Note that clearing the delimiter mask to all zeros is not the same as issuing the clear call (XRB+XRMOD = 0). Use the clear call to free system resources used when any mask is set.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the private delimiter subfunctions of .SPEC.

Errors (Private Delimiters)

PRVIOL One of the following:

1. Job does not own the specified keyboard and is not privileged.
2. Subfunction code at XRB+XRMOD is not 0, 1, or 2.
3. Function code is not 0 through 7 or 11 (returned for all .SPEC calls for terminals).
4. An illegal terminal number is specified at XRB+XRBLK.
5. The device open on the channel specified at XRB+XRCI is not a terminal.

BSERR An illegal channel number is specified at XRB+XRCI.

NOTOPN The channel specified at XRB+XRCI is not open.

NOBUFS The monitor needs to expand its internal buffer space, but memory is not available. May succeed at a later time.

BADCNT One of the following:

1. Byte count at XRB+XRBC does not equal zero or is less than 40_8.
2. Invalid address at XRB+XRLOC.

NOSUCH You are trying to read the private delimiter mask, but no private delimiters are set.

Table 3–4: Private Delimiter Masks

Octal Value	ASCII Character	Byte, address +n	Bit	Byte Value	
				Octal	Decimal
0*	NUL	n=0	0	1	1
1	SOH	0	1	2	2
2	STX	0	2	4	4
3	ETX (CTRL/C)	0	3	10	8
4	EOT	0	4	20	16
5	ENQ	0	5	40	32
6	ACK	0	6	100	64
7	BEL	0	7	200	128
10	BS (backspace)	1	0	1	1
11	HT (horizontal tab)	1	1	2	2
12	LF (line feed)	1	2	4	4
13	VT (vertical tab)	1	3	10	8
14	FF (form feed)	1	4	20	16
15	CR (carriage return)	1	5	40	32
16	SO	1	6	100	64
17	SI (CTRL/O)	1	7	200	128
20	DLE	2	0	1	1
21	DC1 (CTRL/Q)	2	1	2	2
22	DC2	2	2	4	4
23	DC3 (CTRL/S)	2	3	10	8
24	DC4	2	4	20	16
25	NAK (CTRL/U)	2	5	40	32
26	SYN	2	6	100	64
27	ETB	2	7	200	128
30	CAN	3	0	1	1
31	EM	3	1	2	2
32	SUB (CTRL/Z)	3	2	4	4
33	ESC (ESCAPE)	3	3	10	8
34	FS	3	4	20	16

* Octal codes 0 and 200–377 can be used only for terminals opened in binary mode because the parity bit is stripped and nulls are ignored in normal mode.

(continued on next page)

Table 3–4: Private Delimiter Masks (Cont.)

Octal Value	ASCII Character	Byte, address +n	Bit	Byte Value	
				Octal	Decimal
35	GS	3	5	40	32
36	RS	3	6	100	64
37	US	3	7	200	128
40	SP (space)	4	0	1	1
41	!	4	1	2	2
42	"	4	2	4	4
43	#	4	3	10	8
44	$	4	4	20	16
45	%	4	5	40	32
46	&	4	6	100	64
47	'(apostrophe)	4	7	200	128
50	(	5	0	1	1
51	)	5	1	2	2
52	*	5	2	4	4
53	+	5	3	10	8
54	, (comma)	5	4	20	16
55	– (dash / minus)	5	5	40	32
56	. (period)	5	6	100	64
57	/	5	7	200	128
60	0	6	0	1	1
61	1	6	1	2	2
62	2	6	2	4	4
63	3	6	3	10	8
64	4	6	4	20	16
65	5	6	5	40	32
66	6	6	6	100	64
67	7	6	7	200	128
70	8	7	0	1	1
71	9	7	1	2	2

(continued on next page)

Table 3–4: Private Delimiter Masks (Cont.)

Octal Value	ASCII Character	Byte, address +n	Bit	Byte Value	
				Octal	Decimal
72	:	7	2	4	4
73	;	7	3	10	8
74	<	7	4	20	16
75	=	7	5	40	32
76	>	7	6	100	64
77	?	7	7	200	128
100	@	10(8)	0	1	1
101	A	10(8)	1	2	2
102	B	10(8)	2	4	4
103	C	10(8)	3	10	8
104	D	10(8)	4	20	16
105	E	10(8)	5	40	32
106	F	10(8)	6	100	64
107	G	10(8)	7	200	128
110	H	11(9)	0	1	1
111	I	11(9)	1	2	2
112	J	11(9)	2	4	4
113	K	11(9)	3	10	8
114	L	11(9)	4	20	16
115	M	11(9)	5	40	32
116	N	11(9)	6	100	64
117	O	11(9)	7	200	128
120	P	12(10)	0	1	1
121	Q	12(10)	1	2	2
122	R	12(10)	2	4	4
123	S	12(10)	3	10	8
124	T	12(10)	4	20	16
125	U	12(10)	5	40	32
126	V	12(10)	6	100	64

(continued on next page)

Table 3–4: Private Delimiter Masks (Cont.)

Octal Value	ASCII Character	Byte, address +n	Bit	Byte Value	
				Octal	Decimal
127	W	12(10)	7	200	128
130	X	13(11)	0	1	1
131	Y	13(11)	1	2	2
132	Z	13(11)	2	4	4
133	[	13(11)	3	10	8
134	\ (backslash)	13(11)	4	20	16
135	]	13(11)	5	40	32
136	^ or ↑	13(11)	6	100	64
137	_ or ←	13(11)	7	200	128
140	` (grave accent)	14(12)	0	1	1
141	a	14(12)	1	2	2
142	b	14(12)	2	4	4
143	c	14(12)	3	10	8
144	d	14(12)	4	20	16
145	e	14(12)	5	40	32
146	f	14(12)	6	100	64
147	g	14(12)	7	200	128
150	h	15(13)	0	1	1
151	i	15(13)	1	2	2
152	j	15(13)	2	4	4
153	k	15(13)	3	10	8
154	l	15(13)	4	20	16
155	m	15(13)	5	40	32
156	n	15(13)	6	100	64
157	o	15(13)	7	200	128
160	p	16(14)	0	1	1
161	q	16(14)	1	2	2
162	r	16(14)	2	4	4
163	s	16(14)	3	10	8

(continued on next page)

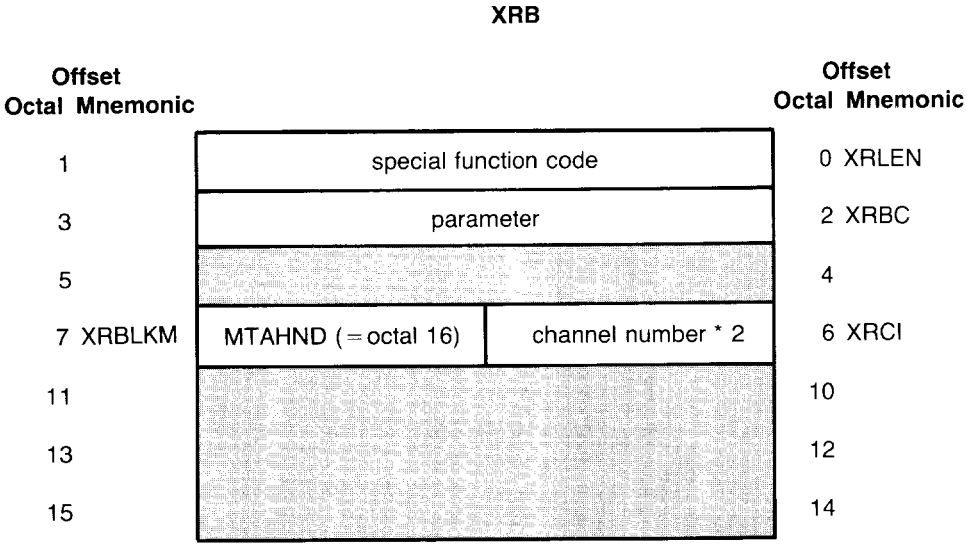
Table 3–4: Private Delimiter Masks (Cont.)

Octal Value	ASCII Character	Byte, address +n	Bit	Byte Value	
				Octal	Decimal
164	t	16(14)	4	20	16
165	u	16(14)	5	40	32
166	v	16(14)	6	100	64
167	w	16(14)	7	200	128
170	x	17(15)	0	1	1
171	y	17(15)	1	2	2
172	z	17(15)	2	4	4
173	{	17(15)	3	10	8
174	| (vertical)	17(15)	4	20	16
175	}	17(15)	5	40	32
176	~ (tilde)	17(15)	6	100	64
177	DEL (RUBOUT)	17(15)	7	200	128
200–377*					

* Octal codes 0 and 200–377 can be used only for terminals opened in binary mode because the parity bit is stripped and nulls are ignored in normal mode.

3.23.3 .SPEC for Magnetic Tape – Not Privileged

Data Passed (Magnetic Tape)



XRB+XRLEN — This word defines the special magnetic tape function to be performed. These codes are summarized in Table 3–5. For a detailed discussion of these functions, see the discussion of the MAGTAPE function in the *RSTS/E Programming Manual.*

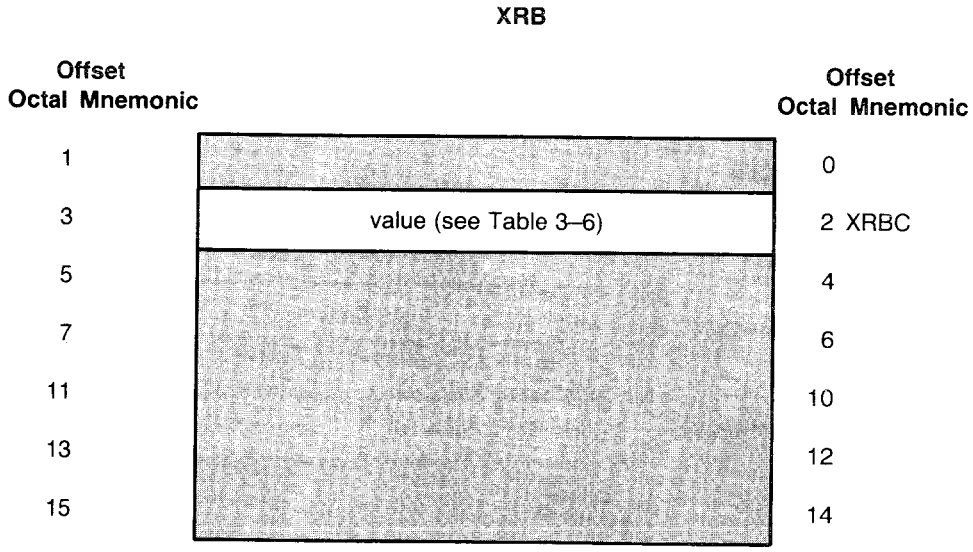
XRB+XRBC — The meaning of this word varies according to the special function code specified at XRB+XRLEN. Table 3–5 summarizes these values; for a detailed discussion, see the MAGTAPE function description in the *RSTS/E Programming Manual.*

XRB+XRCI — Channel number times two; defines the channel on which the tape is currently open.

XRB+XRBLKM — Handler index for magnetic tape: MTAHND (octal value equals 16).

.SPEC
(Magnetic Tape)

Data Returned (Magnetic Tape)



XRB + XRBC — The meaning of this word varies according to the value at XRB + XRLEN in the data passed. Table 3–6 summarizes these values. For a detailed discussion, see the MAGTAPE description in the *RSTS/E Programming Manual.*

Table 3–5: Special Functions for Magnetic Tape

Action	Function Code (octal)	Parameter	Value Returned
Rewind and off-line	0	unused	0
Write end-of-file	1	unused	0
Rewind	2	unused	0
Skip record	3	# records to skip	# records not skipped
Backspace over record	4	# records to backspace	# records not backspaced
Set density and parity	5	D+P+S (see below)	0
Tape status function	6	unused	status (see below)
File characteristics	7	unused	file characteristics (see below)
Rewind on close	10	unused	0

Note: Values below are given in octal except where noted.

Parameter word for function code 6:

D = Density

14_8 = 800_{10} BPI

400_8 = 1600_{10} BPI, phase-encoded

P = Parity (0 = odd, 1 = even)*

S = Stay

0 = Mode value specified in open does not stay on close

20000_8 = Mode value specified in open is retained after close

* DIGITAL recommends that you use odd parity. When you use even parity, you cannot write binary data. In addition, many operating systems and tape drives do not support even parity.

Table 3–6: Value Returned by .SPEC for Magnetic Tape

Value Returned at XRB+2 for Function Code 6 (Magnetic Tape Status Word)		
Bit	**Octal Value**	**Meaning**
15	100000	Last command caused an error.
14–13	If bit 3 = 0, these bits indicate density:	
	00000	Reserved
	20000	Reserved
	40000	800_{10} BPI
	60000	Reserved
	If bit 3 = 1, these bits indicate density:	
	00000	1600_{10} BPI
	20000	Reserved
	40000	Reserved
	60000	Reserved
12	00000	9–track tape
	10000	Reserved
11	0000	Odd parity
	4000	Even parity
10	2000	Magnetic tape is physically write-locked.
9	1000	Tape is beyond end-of-tape marker.
8	400	Tape is at beginning-of-tape (Load Point).
7	200	Last command detected an EOF.
6	100	The last command was .READ and the record read was longer than the I/O buffer size (that is, part of the record was lost).
5	40	Unit is nonselectable (off-line).
4	00	Unit does not accept 1600_{10} BPI.
	20	Unit accepts 1600_{10} BPI.
3	00	See values for bits $14–13_{10}$.
	10	See values for bits $14–13_{10}$.
2–0		Indicates last command issued: 0 = Off-line 1 = Read 2 = Write 3 = Write EOF 4 = Rewind 5 = Skip record 6 = Backspace record

(continued on next page)

Table 3–6: Value Returned by .SPEC for Magnetic Tape (Cont.)

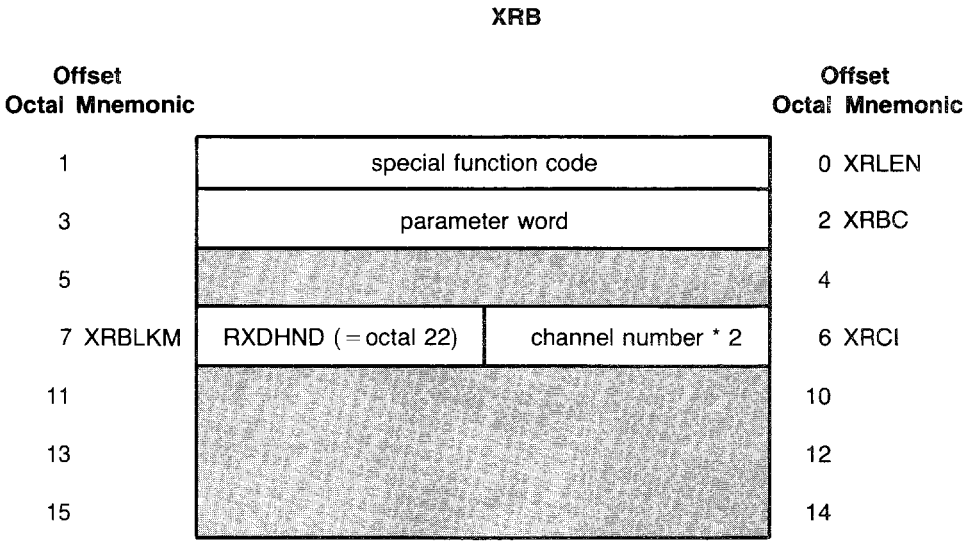
Value Returned at XRB+2 for Function Code 7 (File Characteristics Word)		
word = 0, DOS format or ANSI U (undefined) format ≠ 0, ANSI format, with bit meanings as defined below:		
Bit	**Octal Value**	**Meaning**
15–14	40000	F (fixed-length)
	100000	D (variable-length)
	140000	S (spanned)*
13–12	00000	Carriage control embedded 'M'
	10000	FORTRAN carriage control 'A'
	20000	Implied LF/CR before record ' '
11–0	—	For Format F, this value is the record length, in bytes. For Format D, this value is the maximum record length, in bytes.
* ANSI format S is not supported by RSTS/E systems.		

3.23.4 .SPEC for RX01/RX02 Flexible Diskette – Not Privileged

For RX01 and RX02 flexible diskette devices, the .SPEC directive lets you (1) obtain the density (single or double) of the current flexible diskette, (2) mount a new flexible diskette and recompute the density, and (3) reformat an RX02 flexible diskette for a desired density. Because the RX02 flexible diskette drive supports single and double density flexible diskettes, the .SPEC function is especially useful for programmed flexible diskette operations. For example, .SPEC allows you to mount a series of single and double density flexible diskettes without having to close and reopen the device for each mount. That is, the driver computes density once: during the initial open. If you insert a second flexible diskette that is incompatible with the initially computed density, a read or write operation will fail. .SPEC permits you to include an instruction in your program that causes the driver to recompute the density. In addition, for RX02 flexible diskette drives, .SPEC permits you to specify a density reformat operation.

.SPEC can require as much as 20 seconds to reformat the density of the RX02 flexible diskette and cannot be interrupted with CTRL/C. Note that if the operation is interrupted (by power failure or catastrophic error), the flexible diskette is rendered unusable. That is, the flexible diskette will contain both single and double density. To recover, you must reformat the flexible diskette.

Data Passed (Flexible Diskettes)



XRB+XRLEN — Function code specifying the desired operation:

- = 0 Returns density of currently mounted flexible diskette. (The parameter word at XRB+XRBC must be 0.)
- = 1 Recomputes density and returns density. (The parameter word at XRB+XRBC must also be 0.) This code must be issued prior to any I/O operation on the flexible diskette.
- = 2 Reformats the current flexible diskette to the density specified in the parameter word at XRB+XRBC. Allowed only on RX02 drives.

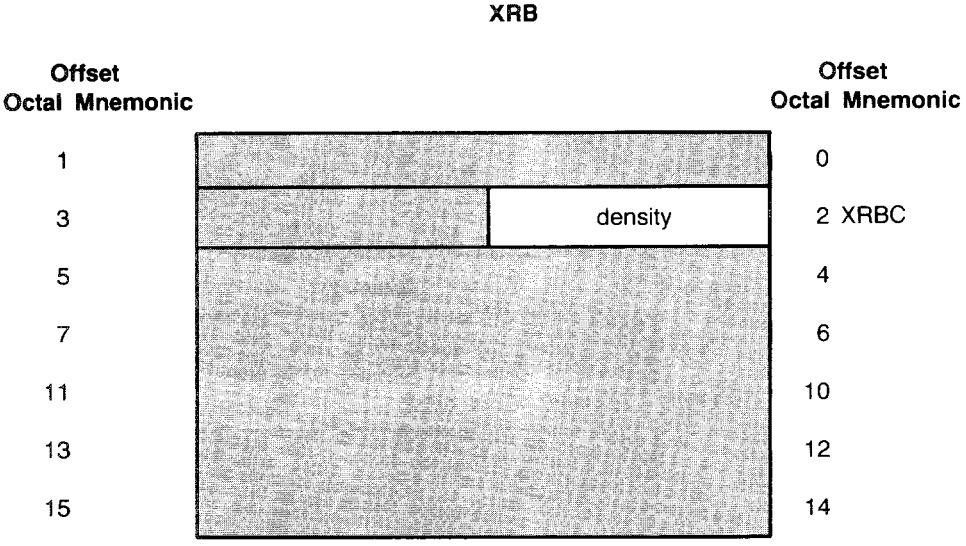
XRB+XRBC — Parameter word must equal 0 when function code equals 0 or 1. Otherwise, when function code equals 2:

- = 1 Reformats as single-density (one sector equals 128_{10} bytes)
- = 2 Reformats as double-density (one sector equals 256_{10} bytes)

XRB+XRCI — Channel number times 2; defines the channel on which the flexible diskette is currently open.

XRB+XRBLKM — Handler index for flexible diskette: RXDHND (octal value equals 22).

Data Returned



XRB+XRBC — Density, returned when XRB+XRLEN in data passed is 0 or 1. Equal to 1 for single-density (sector equals 128_{10} bytes) or 2 for double-density (sector equals 256_{10} bytes).

Errors

HNGDEV	A hardware error occurred. This can often be a transient condition. Retry the operation.
ERRERR	You tried to reformat on an RX01 flexible diskette drive. You can use .SPEC to reformat flexible diskette density only on RX02 drives.

3.23.5 .SPEC for Pseudo Keyboards – Not Privileged

For pseudo keyboards, the .SPEC function lets you:

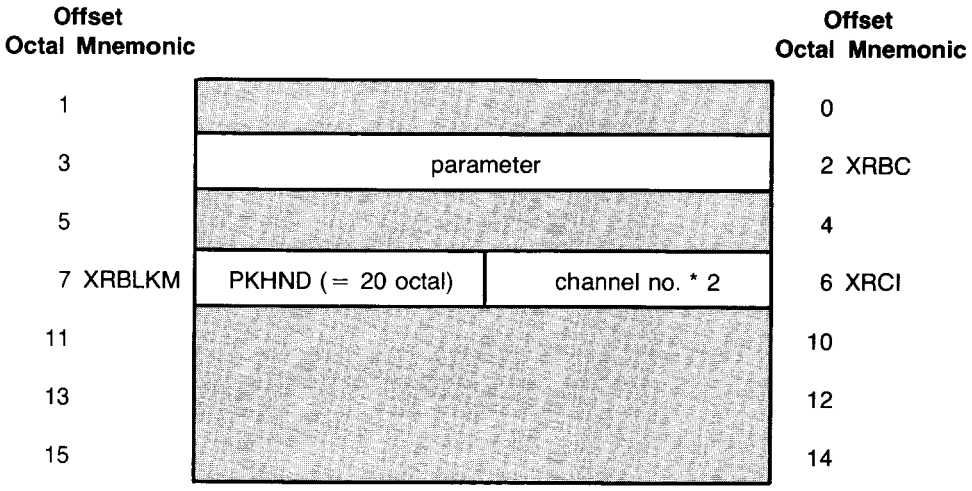
1. Disable and enable echo at the controlled job's keyboard (that is, the KB side of the pseudo keyboard).
2. Read a flag word that tells you whether echo is on or off at the controlled job's keyboard.

A pseudo keyboard receives two kinds of output from a controlled job: character echo, which is done by the RSTS/E monitor, and program output, which occurs when a program writes to the controlled job's keyboard. The .SPEC function affects only character echo, not program output.

Character echo is enabled by default. However, in some pseudo keyboard applications, it is more convenient to disable character echo. For example, in a pseudo keyboard application that uses both a terminal and a pseudo keyboard, you get character echo from the terminal. You also get character echo and program output from the pseudo keyboard. You can use this function to disable character echo at the pseudo keyboard.

Data Passed

XRB



XRB+XRBC — This word defines the action to be performed.

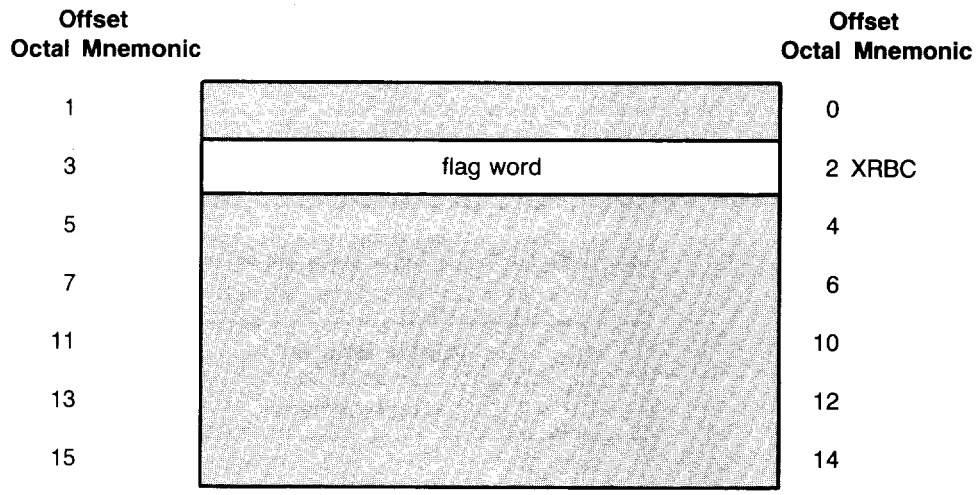
If = 0, read the flag word
= 377_8 (255_{10}), enable echo
= 177777_8 (-1_{10}), disable echo

XRB+XRCI — Channel number times two; defines the channel on which the pseudo keyboard is open.

XRB+XRBLKM — Handler index for pseudo keyboard PKHND (octal value = 20).

Data Returned

XRB



XRB+XRBC — If bit 5 = 0 Keyboard echo is enabled.
= 1 Keyboard echo is disabled.

Errors

No errors are possible with the pseudo keyboard subfunction of .SPEC.

3.24 .STAT — Return Job Statistics – Privilege and Not Privileged

Form

```
.STAT
```

Function

The .STAT directive returns current statistics on the job to the XRB.

Data Returned

XRB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1		current job image size, in K words	0	XRLEN
3		current run-time system size, in K words	2	XRBC
5		current private memory max, in K words	4	XRLOC
7		maximum job image size, in K words	6	XRCI
11		current project-programmer number	10	XRBLK
13		current run priority	12	XRTIME
15		current run burst	14	XRMOD

XRB+XRLEN — The current size of the user job image for this job, in K words.

XRB+XRBC — The size of the current run-time system for this job, in K words.

XRB+XRLOC — The current private memory maximum for the user job image, in K words. If the job has an unlimited memory maximum or if its private memory maximum is larger than the possible maximum size allowed by its current run-time system, then the value returned here is the maximum size possible for the current run-time system, in K words. If the job's private maximum is less than the run-time system minimum, then the value returned here is the run-time system's minimum size. (See .CORE, Section 3.6, for a discussion of these values.)

In all cases, this value represents the maximum size of the user job image under its current run-time system.

XRB+XRCI — The maximum job image size possible (under the current run-time system), in K words.

XRB+XRBLK The job's current project-programmer number is returned here. The programmer number is returned as a binary value in the low byte (XRB+XRBLK), the project number as a binary value in the high byte (XRB+11). If the job is not logged in, a value of 0 is returned here.

XRB+XRTIME The job's current run priority (returned to privileged users only; 0 returned to nonprivileged users). Run priority may range from -128_{10}, indicating a suspended job, to $+127_{10}$, the highest priority. The monitor schedules jobs for time-shared execution according to this priority. When a user first logs in to RSTS/E, LOGIN is run with priority 0. LOGIN sets the user's job run priority to –8. Only in unusual cases should the run priority be changed. It can be changed by a privileged user for any job with the UTILTY system program. It can also be changed with the UU.PRI subfunction of the .UUO directive (Section 3.32.33). It can be modified by the job by one-half step with the .SET and .CLEAR directives.

The special-case value of -128_{10} indicates that the job is never scheduled to run; it is suspended.

XRB+XRMOD The job's current run burst (returned to privileged users only; 0 returned to nonprivileged users). The run burst is the amount of time that the job will be allowed to execute compute-bound before the next job in the schedule is given control. The units of run burst are 1/60ths or 1/50ths of a second, depending on the clock in use and/or the line frequency. (Systems running with the KW11–P clock at crystal speeds, rather than at line frequency, have a run burst unit of 1/50th of a second. If the system is operating off a 60 Hz power line, one run burst unit equals 1/60th of a second.) The range of values for run burst is from 1 to 127_{10} inclusive. When a job is created, the monitor sets the run burst to a value of 6. This value can be modified by the system manager for a particular job with the UTILTY system program. It can also be modified with the UU.PRI subfunction of the .UUO directive (see Section 3.32.33).

Errors

No errors are possible on this directive.

Example

No data is passed to the monitor with the .STAT directive. Therefore, the call is simply:

```
.STAT
```

3.25 .TIME — Return Timing Information – Not Privileged

Form

```
.TIME
```

Function

The .TIME directive returns job timing information: elapsed CPU time, elapsed time connected to a user terminal (channel 0), elapsed device time, and memory utilization.

Data Passed

No data is passed with the .TIME directive.

Data Returned

XRB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1		low 16 bits of elapsed CPU time, .1 sec.	0	XRLEN
3		elapsed connect time to channel 0, min.	2	XRBC
5		low 16 bits of memory utilization, KCTs	4	XRLOC
7		elapsed device time, min.	6	XRCI
11		high 16 bits of elapsed CPU time, .1 sec.	10	XRBLK
13			12	
15		high 16 bits of memory utilization, KCTs	14	XRMOD

XRB + XRLEN — This word contains the low-order 16 bits of the job's elapsed CPU time, in tenths of a second of CPU utilization.

XRB + XRBC — This word contains the elapsed time that the job has been connected to a channel 0 terminal, in minutes.

XRB + XRLOC — This word contains the low-order 16 bits of the job's memory utilization, in kilo-core-ticks (KCTs). A kilo-core-tick is the use of 1K of memory for one-tenth of a second.

XRB + XRCI — This word contains the job's elapsed device time. Device time is the use of an assignable device for one minute. If a job owns two devices for one elapsed minute, then two units of device time are accrued.

XRB+XRBLK This word contains the high-order 16 bits of the job's elapsed CPU time (see XRB+XRLEN).

XRB+XRMOD This word contains the high-order 16 bits of the job's memory utilization (see XRB+XRLOC).

Errors

No errors are possible with the .TIME directive.

Example

Since no data is passed with this directive, the call is simply:

```
.TIME
```

3.26 .TTAPE — Enter Tape Mode – Not Privileged

Form

```
.TTAPE
```

Function

The .TTAPE directive enters "tape mode" on the job's terminal (channel 0). This directive is useful when it is necessary to read data from the low-speed paper tape reader available on some terminals. Make sure the terminal is open on channel 0, and give this call before the .READ. Three things happen:

1. No incoming characters are echoed, preventing needless output at the terminal while the tape is being read.
2. The DELETE character (ASCII code 177 octal) is ignored, rather than deleting the previous character.
3. A LINE FEED character (ASCII code 012 octal) is not automatically appended to an incoming RETURN (ASCII code 015 octal).

The .TTECH directive (Section 3.28) returns character processing to normal on channel 0.

Data Passed

No data is passed with the .TTAPE directive.

Data Returned

No data is returned with the .TTAPE directive.

Errors

DETKEY Channel 0 is not currently available for this job; it is running detached.

Example

Since no data is passed (or returned), the call is simply:

```
.TTAPE
```

3.27 .TTDDT — Disable Full-Line Buffering – Not Privileged

Form

```
.TTDDT
```

Function

The .TTDDT directive disables the monitor's usual practice of buffering a full line of data from the user's terminal (channel 0) before passing it on to the job on a .READ.

The monitor accepts data typed at a user terminal and stores it in a buffer until the job associated with the terminal reads the data. Normally, a .READ causes the monitor to pass a line to the job's buffer. (A line is any number of characters ending with a RETURN, LINE FEED, ESCAPE, FORM FEED, or CTRL/D combination.) If a full line is not in the monitor's buffer, the monitor stalls the job until it gets a delimiter and then awakens the job and passes the line on to the job's buffer.

The .TTDDT directive tells the monitor that, when the next .READ on channel 0 is issued, it is to pass on whatever is currently in the monitor's buffer, whether or not a delimiter has been typed. If no characters are in the monitor's buffer, the job is stalled until at least one character has been typed.

.TTDDT is a "one-shot" directive; it affects only the next .READ on channel 0. If you want to do this type of input consistently, you must execute a .TTDDT before each .READ.

This type of input is useful when you want to respond to each character that a user types. (Note that more than one character may be in the monitor's buffer. If you really want only one character, use .TTDDT before each .READ, and define a 1–character input buffer for the .READ.) For example, the Octal Debugging Tool (ODT) routine (see the *RSTS/E System Manager's Guide*) uses this capability to accept commands without requiring that you type a delimiter. This type of input puts a high load on the system and is not recommended except in unusual circumstances.

Data Passed

No data is passed with the .TTDDT directive.

Data Returned

No data is returned with the .TTDDT directive.

Errors

DETKEY Channel 0 is not currently available for this job; the job is running detached.

Example

Since no data is passed, the call is simply:

```
.TTDDT
```

3.28 .TTECH — Undo .TTAPE or .TTNCH – Not Privileged

Form

```
.TTECH
```

Function

The .TTECH directive causes the monitor to resume normal character input processing on channel 0 when it has been disabled with either a .TTAPE directive (see Section 3.26) or a .TTNCH directive (see Section 3.29).

Data Passed

No data is passed with the .TTECH directive.

Data Returned

No data is returned with the .TTECH directive.

Errors

DETKEY Channel 0 is not available for this job; the job is running detached.

Example

Since no data is passed, this call is simply:

```
.TTECH
```

3.29 .TTNCH — Stop Echo – Not Privileged

Form

```
.TTNCH
```

Function

The .TTNCH directive disables terminal echo on the job's terminal (channel 0). That is, whatever the user types is accepted, but it is not echoed back for display on the terminal. Otherwise, all normal character processing occurs. (The .TTECH directive, Section 3.28, returns character processing to normal on channel 0.)

Data Passed

No data is passed with the .TTNCH directive.

Data Returned

No data is returned with the .TTNCH directive.

Errors

DETKEY Channel 0 is not available for this job; the job is running detached.

Example

Since no data is passed or returned, the call is simply:

```
.TTNCH
```

3.30 .TTRST — Restart Output – Not Privileged

Form

```
.TTRST
```

Function

The .TTRST directive restarts program output to the user's terminal when such output has been stopped by the user's typing a CTRL/O or CTRL/C.

Terminal service on a RSTS/E system (a driver within the monitor itself) maintains a "discard all program output" indicator for each terminal. When this indicator is set, the driver ignores a .WRITE directive to channel 0 rather than display the data at the user's terminal. When the indicator is clear, a .WRITE to channel 0 is processed normally. The .TTRST directive clears this indicator.

The driver sets the indicator when the user types a CTRL/C combination. It reverses the indicator when the user types a CTRL/O combination. A .READ directive for channel 0 clears the indicator.

One use of .TTRST is in run-time systems with keyboard monitors. By issuing .TTRST before displaying any prompt, you can ensure that the prompt is actually displayed at the user's terminal.

Data Passed

No data is passed with the .TTRST directive.

Data Returned

No data is returned with the .TTRST directive.

Errors

DETKEY Channel 0 is not currently available for this job; the job is running detached.

Example

Since no data is passed, the call is simply:

```
.TTRST
```

3.31 .ULOG — Assign/Reassign/Deassign Device or Assign/Deassign User Logical

Form

```
.ULOG
```

Function

The .ULOG directive has three subfunctions. You select the particular action desired by setting a function field in the FIRQB (at offset FQFUN). The subfunctions are described in the following subsections; a summary is given below.

FQFUN Value (Octal)	Mnemonic	Action Performed
12	UU.ASS	Assign/reassign a device or assign user logical
13	UU.DEA	Deassign a device or user logical
14	UU.DAL	Deassign all devices and user logicals

3.31.1 UU.ASS (Assign/Reassign a Device, or Assign User Logical) – Privileged and Not Privileged

Form

```
MOVB      #UU.ASS,FIRQB+FQFUN
          .
          .
          .
 (set up FIRQB and XRB)
          .
          .
          .
.ULOG
```

Function

The UU.ASS subfunction of .ULOG allows you to do one of three things:

1. Assign a device to a job.
2. Reassign a device to another job.
3. Enter a user logical: This feature allows you to:

   a. Assign a logical name to a device. The monitor will then use this assignment for logical-to-physical device translation by the .FSS directive (Section 3.10).

   b. Associate a project-programmer number with a particular logical name. This "user logical ppn" will be used if the associated logical name is found in a string parsed by .FSS, but no ppn is found in the string. This feature is useful if you wish to override a system-wide logical name with an associated ppn. The logical name LB, for example, is commonly associated with a disk on the public structure and some specific project-programmer number (usually [1,1]). With this feature, you can set up a different device and ppn for the logical name LB.

   c. Assign a project-programmer number to be substituted for an at sign character (@) encountered in a file specification string parsed by an .FSS directive.

   d. Assign a protection code to be used as a default if no protection code is specified in a file specification string translated by an .FSS directive.

NOTE

Assigning user logicals assigns values to the USRPPN, USRPRT, and USRLOG areas in the low 1000 bytes of virtual memory (Section 2.4). If these values are in a non-standard location, you must specify where they are by setting the XRB. For all other uses of .ULOG, the XRB should be cleared to zeros.

Data Passed — Assign/Reassign Device

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3	FQFUN	UU.ASS (= octal 12)	2	
5			4	
7		(must = 0)	6	FQPPN
11		(must = 0) / 0=assign; ≠0=job	10	FQNAM1
13			12	
15		DOS or ANS (1 word in RAD50 format)	14	FQEXT
17			16	
21			20	
23		100001 for snagging assign/reassign; else 0	22	FQMODE
25			24	
27		(must = 0)	26	FQPFLG
31		device name (2 ASCII characters)	30	FQDEV
33		≠0, unit no. real / device unit number	32	FQDEVN
35			34	
37			36	

FIRQB+FQFUN The function code UU.ASS (octal value = 12).

FIRQB+FQNAM1 For assigning a device, the two bytes beginning here must equal 0.

For reassigning a device, this byte is the job number to which the device is to be reassigned. The byte at FIRQB+FQNAM+1 must equal 0. If you are nonprivileged, you can reassign a device only to a job that is logged in to the same account as your current account.

FIRQB+FQEXT For assigning or reassigning a magtape device, this word is either DOS or ANS (in RAD50 format) to indicate DOS or ANSI label format for the magtape drive.

FIRQB+FQMODE For assigning or reassigning a device, setting this field to 100001 octal indicates a "snagging" assign or reassign. That is, it will assign or reassign the device even if it is currently assigned to another job. This feature can be issued only from a privileged account. Furthermore, the target device must not be open, and the current owner cannot be performing a directory lookup on that device with the UU.DIR subfunction of the .UUO directive (see Section 3.32.) If you do not want a snagging assign or reassign, set this word to 0.

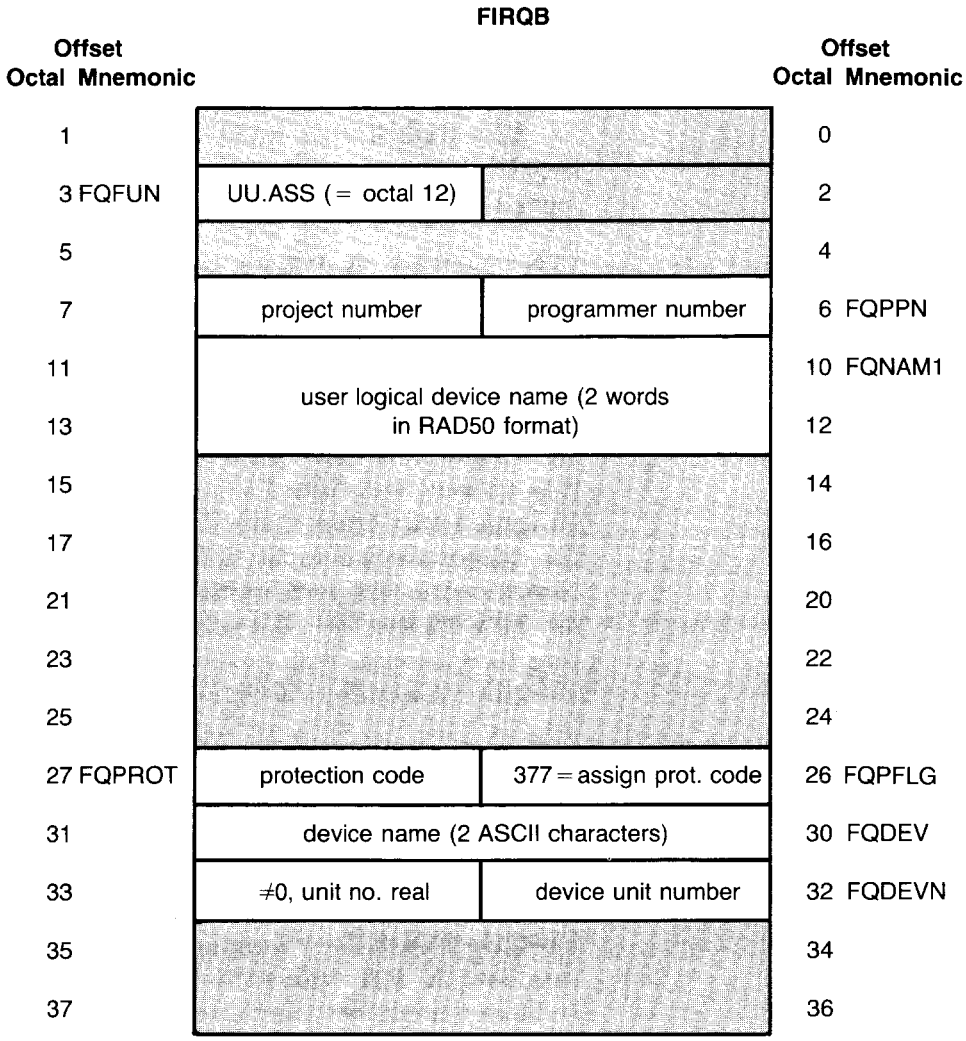
FIRQB+FQDEV Device name to be assigned or reassigned, specified as two ASCII characters. If this word is zero, the public disk structure is assumed.

FIRQB+FQDEVN Device unit number, passed as a binary value in byte FIRQB+FQDEVN. A nonzero value in byte FIRQB+FQDEVN+1 indicates an explicit device unit number. A zero value in byte FIRQB+FQDEVN+1 indicates no device unit number.

NOTE

The XRB should be cleared to zeros for assign/reassign device.

Data Passed — Enter User Logical



FIRQB + FQFUN — The function code UU.ASS (octal value = 12).

FIRQB + FQPPN — This word is the project-programmer number for those features that use a ppn (see items 3b and 3c in the function discussion at the beginning of this section). For a user-assignable ppn (case 3c), this is the project-programmer number to be used to replace an @ sign in a string parsed by an .FSS directive. In this case (3c), the two bytes at FIRQB + FQNAM1 and FIRQB + FQNAM1 + 1 must be 0.

If the four bytes FIRQB + FQNAM1 through FIRQB + FQNAM1 + 3 contain a logical device name, and a ppn is to be associated with that name (case 3b), specify the ppn here. If no ppn is to be associated with the logical device name, this word should be set to 0.

FIRQB+FQNAM1 To assign a user logical name to a device, set the two words beginning here to the logical name, in RAD50 format. Otherwise, set these bytes to zero.

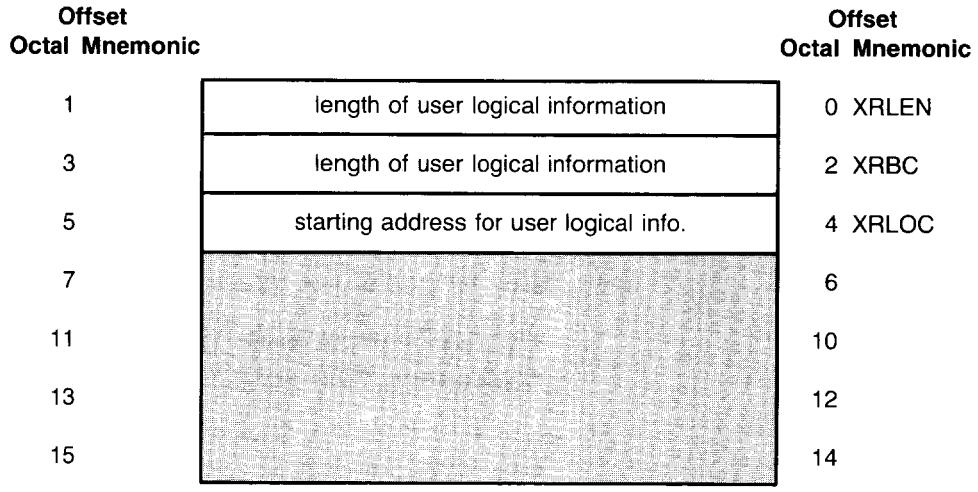
FIRQB+FQPFLG To assign a user-assignable default protection code, set this byte nonzero. (See FIRQB+FQPROT.)

FIRQB+FQPROT This byte is the protection code to be used as a default if no protection code is specified in a string scanned by an .FSS directive (case 3d). The byte at FIRQB+FQPFLG must be nonzero, and the two words at FIRQB+FQPPN and FIRQB+FQNAM1 must be zero.

FIRQB+FQDEV Device name for assigning a logical name to be associated with a device (case 3a and 3b). Specify the device as two ASCII characters. If this word is zero, the public disk structure is assumed.

FIRQB+FQDEVN Device unit number for assigning a logical name to be associated with a device (case 3a and 3b). The device unit number is passed as a binary value in byte FIRQB+FQDEVN. A nonzero value in byte FIRQB+FQDEVN+1 indicates an explicit device unit number. A zero value in byte FIRQB+FQDEVN+1 indicates no device unit number.

XRB



XRB+XRLEN — If the user logical information is in its standard location (USRPPN, USRPRT, and USRLOG), this word is passed as 0. If some nonstandard set of locations is being used, then specify the length of that information, in bytes, here. Thus, when the user logical information is in a nonstandard location, this word must be at least 4 (for user-assignable ppn and default protection code). The space for user logical names is optional. If no space is reserved for logical names in the nonstandard location, no such assignments will be allowed.

XRB+XRBC — This word also contains the length of the information (same as the word at XRB+XRLEN).

XRB+XRLOC — If the word at XRB+XRLEN is nonzero, then this word defines the starting location for the user logical information (the default ppn). You must specify an even value. The order and format of the information created by .ULOG is described in Section 2.4; see USRPPN, USRPRT, and USRLOG.

Data Returned

Except for a possible error code in byte 0 of the FIRQB, no meaningful data is returned by the UU.ASS subfunction of .ULOG.

Errors

INUSE — For assigning or reassigning a device, the specified device is currently open or has an open file. For assigning a user logical, no space is currently available for storing the information.

NODEVC — The device name specified at FIRQB+FQDEVN is a logical device name for which a physical device is not currently assigned.

NOTAVL — The device specified exists on the system, but the operation failed for one of the following reasons:

1. The device is currently reserved by another job (see description of FIRQB+FQMODE).
2. Ownership of the device requires privilege that the user does not have. For example, a nonprivileged user tried to assign a device that is currently assigned to another user.
3. The device or its controller is disabled.
4. The device is a keyboard line for a pseudo keyboard only.

BDNERR — An attempt was made to reassign a device to a nonexistent job. This error can occur only for a reassign call.

PRVIOL — You are not privileged and tried to either:

1. Reassign a device that requires privilege to assign.
2. Reassign a device to a job that is logged in to an account other than your current account.

BADCNT — This error has two possible causes:

1. The length specified at XRB+XRLEN and XRB+XRBC must be 4 + 8n, where n = 0, 1, 2, 3, or 4. If it is not, this error is returned. (Other lengths are not valid user logicals.)
2. The value at XRB+XRLOC is odd.

Example

The following code assigns the user logical name "OUT" to the public disk structure. (Assume that the FIRQB and XRB have been previously cleared to zero.)

```
MOVB    #UU.ASS,FIRQB+FQFUN          ;SET FUNCTION CODE
MOV     #^ROUT,FIRQB+FQNAM1          ;SET LOGICAL NAME AS
MOV     #^R   ,FIRQB+FQNAM1+2        ;TWO WORDS RAD50
.ULOG
```

3.31.2 UU.DEA — Deassign a Device or User Logical – Not Privileged

Form

```
MOVB     #UU.DEA,FIRQB+FQFUN
         .
         .
         .
(set up FIRQB)
         .
         .
         .
.ULOG
```

Function

The UU.DEA subfunction of .ULOG deassigns a device from the current job (releases it for use by other jobs) or deassigns a user-logical assignment.

Data Passed for Deassign Device

FIRQB

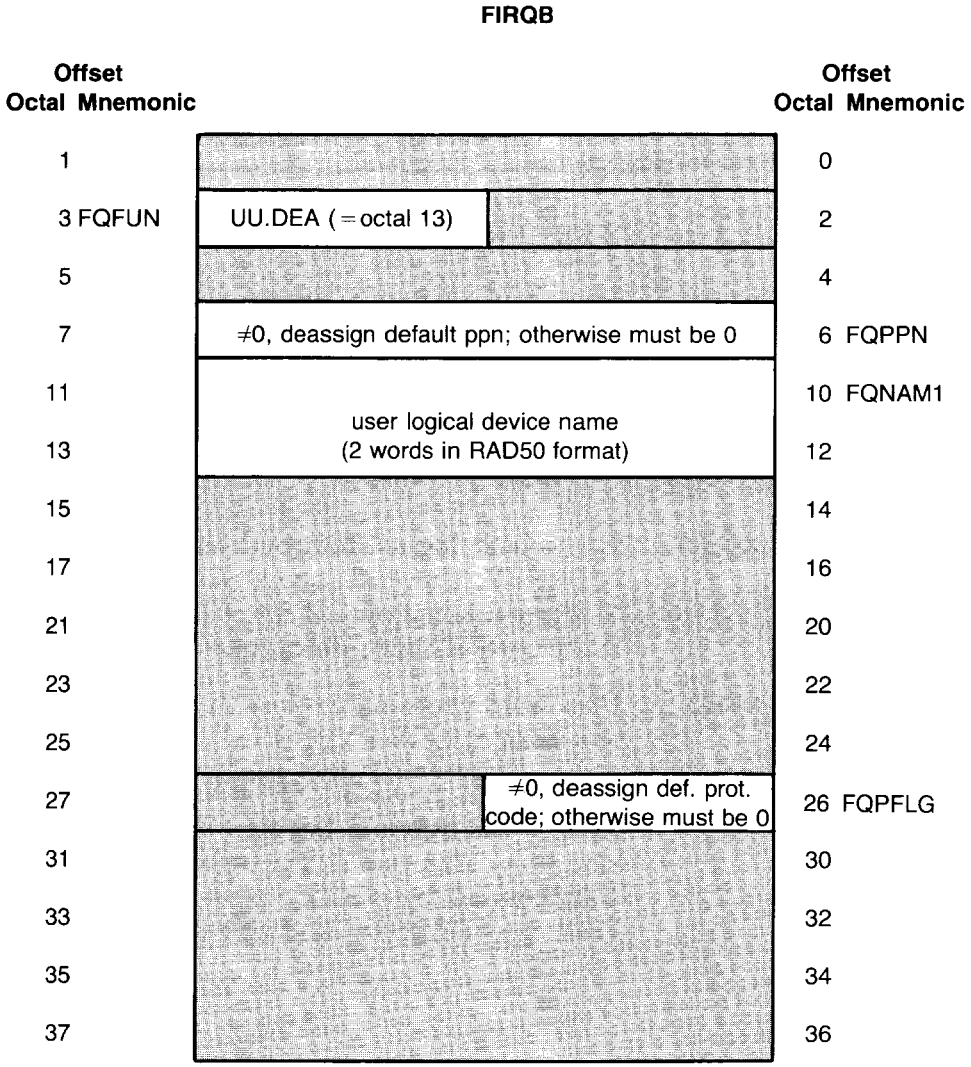
Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3	FQFUN	UU.DEA (=octal 13)	2	
5			4	
7		(must = 0)	6	FQPPN
11		(must = 0)	10	FQNAM1
13			12	
15			14	
17			16	
21			20	
23			22	
25			24	
27		(must = 0)	26	FQPFLG
31		device name (2 ASCII characters)	30	FQDEV
33		≠0, unit number real / device unit number	32	FQDEVN
35			34	
37			36	

FIRQB+FQFUN	The UU.DEA function code (octal value equals 13).
FIRQB+FQDEV	The name of the device to be deassigned, as two ASCII characters.
FIRQB+FQDEVN	The device unit number is passed in this byte, in binary. A nonzero value in FIRQB+FQDEVN+1 indicates an explicit device unit number. A zero value in FIRQB+FQDEVN+1 indicates no device unit number.

NOTE

The XRB should be cleared to zeros for deassigning a device.

Data Passed for Deassign User Logical



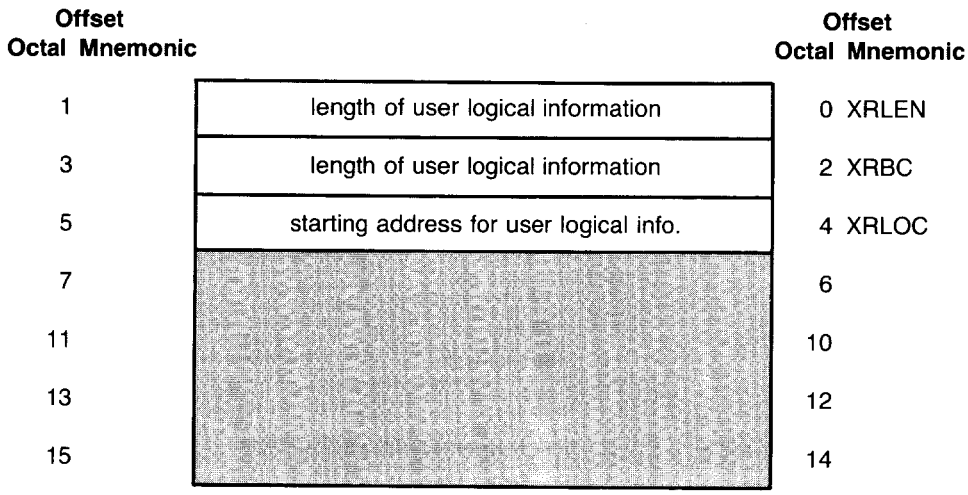
FIRQB+FQFUN — The UU.DEA function code (octal value equals 13).

FIRQB+FQPPN — Any nonzero value deassigns any previously assigned default project-programmer number. (In this case, set the word at FIRQB+FQNAM1 to zero.) If deassigning default protection code, set this word to zero.

FIRQB+FQNAM1 — The logical device name to be deassigned, as two words in RAD50 format. If deassigning a default ppn or protection code, set the word at FIRQB+FQNAM1 to zero.

FIRQB+FQPFLG — Set this byte to nonzero if deassigning a protection code.

XRB



XRB+XRLEN — If the user logical information is in its standard location (USRPPN, USRPRT, and USRLOG), this word is passed as 0. If some nonstandard set of locations is being used, then the length of that information, in bytes, is specified here.

XRB+XRBC — This word contains the length of the information (same as the word at XRB+XRLEN).

XRB+XRLOC — If the word at XRB+XRLEN is nonzero, then this word defines the starting location for the user logical information (the default ppn). The order and format of the information created by .ULOG is described in Section 2.4; see USRPPN, USRPRT, and USRLOG.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the UU.DEA subfunction of .ULOG.

Errors

NODEVC — The device or its type specified at the locations FIRQB+FQDEV and FIRQB+FQDEVN is not part of your system configuration.

BADCNT — This error has two possible causes:

1. The length specified at XRB+XRLEN and XRB+XRBC must be 4+8n, where n=0, 1, 2, 3, or 4. If it is not, this error is returned. (Other lengths are not valid user logicals.)
2. The value at XRB+XRLOC is odd.

Example

The following code deassigns MT0: from the current job:

```
MOVB    #UU.DEA,FIRQB+FQFUN         ;SET FUNCTION CODE
CLR     FIRQB+FQNAM1                ;CLEAR NAME
MOV     #"MT,FIRQB+FQDEV            ;SET DEVICE
MOVB    #0,FIRQB+FQDEVN             ;SET UNIT NUMBER
MOVB    #377,FIRQB+FQDEVN+1         ;UNIT NUMBER REAL
.ULOG
```

3.31.3 UU.DAL — Deassign All Devices and User Logicals – Not Privileged

Form

```
MOVB      #UU.DAL,FIRQB+FQFUN
          .
          .
          .
 (set up FIRQB and XRB)
          .
          .
          .
.ULOG
```

Function

The UU.DAL function of .ULOG deassigns all devices and user logicals for the calling program.

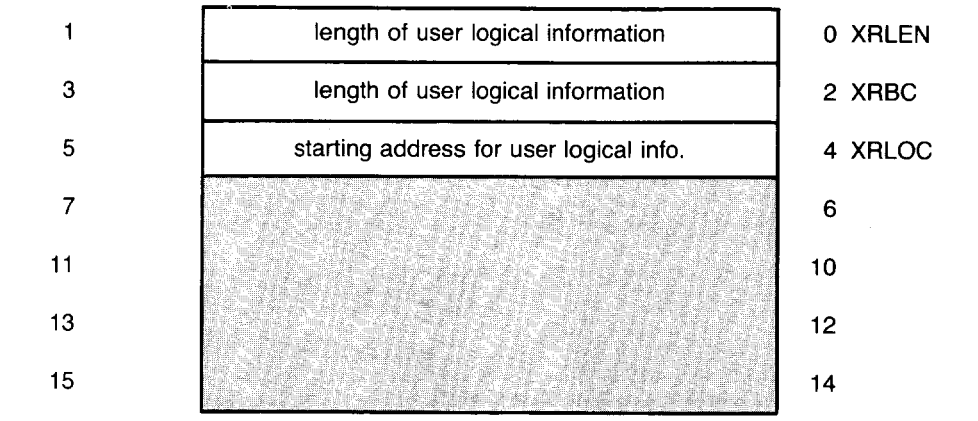
Data Passed

FIRQB

Offset Octal	Mnemonic		Offset Octal
1			0
3	FQFUN	UU.DAL (=octal 14)	2
5			4
7			6
11			10
13			12
15			14
17			16
21			20
23			22
25			24
27			26
31			30
33			32
35			34
37			36

FIRQB+FQFUN The function code UU.DAL (octal value equals 14).

XRB



XRB+XRLEN If the user logical information is in its standard location (USRPPN, USRPRT, and USRLOG), set this word to 0. If some nonstandard set of locations is being used, then specify the length of that information, in bytes, here.

XRB+XRBC This word contains the length of the information (same as the word at XRB+XRLEN).

XRB+XRLOC If the word at XRB+XRLEN is nonzero, then this word defines the starting location for the user logical information (the default ppn). The order and format of the information created by .ULOG is described in Section 2.4; see USRPPN, USRPRT, and USRLOG.

Data Returned

No data is returned by the UU.DAL subfunction of .ULOG.

Errors

BADCNT
1. The length specified at XRB+XRLEN and XRB+XRBC must be 4+8n, where n=0, 1, 2, 3, or 4. If it is not, this error is returned. (Other lengths are not valid user logicals.)
2. The value at XRB+XRLOC is odd.

Example

The following code deassigns all devices and user logicals for the current job. The user logical information is in its standard location.

```
MOVB    #UU.DAL,FIRQB+FQFUN
CLR     XRB+XRLEN
CLR     XRB+XRBC
CLR     XRB+XRLOC
.ULOG
```

3.32 .UUO — Execute BASIC–PLUS SYS Call

Form

```
.UUO
```

Function

The .UUO directive allows a MACRO program to execute the BASIC–PLUS SYS calls to the File Processor (FIP). The SYS calls are described in the *RSTS/E Programming Manual*. For the MACRO programmer's convenience, the FIRQB formats for the calls are shown here; for detailed descriptions, however, refer to the *RSTS/E Programming Manual*.

Table 3–7 summarizes the FIP SYS calls; note that some of the calls are handled by directives other than .UUO. (.FSS, for example, handles the file name string scan in MACRO.) The mnemonic subfunction names are provided by the COMMON.MAC prefix file; the decimal values are the BASIC–PLUS FIP call subfunction codes and the .UUO subfunction codes that COMMON.MAC relates to the mnemonics listed.

The rest of this section gives the FIRQB formats for data passed and returned for the .UUO subfunctions; the subfunctions are organized in alphabetical order of the mnemonics provided by COMMON.MAC.

Table 3–7: .UUO Subfunctions — Calls to the File Processor (FIP)

Mnemonic	BASIC–PLUS SYS Call Code (Decimal)	Privileged Status	Function	Section
UU.TB3	–29	No	Get monitor tables — Part III	3.32.43
UU.SPL	–28	No	Spooling	3.32.37
UU.DMP	–27	Yes	Snap shot dump	3.32.18
UU.FIL	–26	Both	File placement and modification	3.32.21
UU.ATR	–25	No	Read or write attributes	3.32.3
UU.CCL	–24	Yes	Add/delete CCL command	3.32.7
(.FSS)	–23	No	Terminating file name string scan	3.10
(.SET)	–22	Yes	Set special run priority	3.21
(.SET/ .CLEAR)	–21	Yes	Drop/regain temporary privilege	3.21, 3.5
(.SET/ .CLEAR)	–20	Yes	Lock/unlock job in memory	3.21, 3.5
UU.LOG	–19	Yes	Set number of logins	3.32.25
UU.RTS	–18	Yes	Add run-time system	3.32.35
		Yes	Remove run-time system	
		Yes	Load run-time system	
		Yes	Unload run-time system	
		Yes	Declare default keyboard monitor	
		Yes	Add resident library	
		Yes	Remove resident library	
		Yes	Load resident library	
		Yes	Unload resident library	
UU.NAM	–17	No	Name run-time system	3.32.28
UU.DIE	–16	Yes	System shutdown	3.32.15
UU.ACT	–15	Yes	Accounting dump	3.32.1
UU.DAT	–14	Yes	Change system date/time	3.32.12
UU.PRI	–13	Yes	Change priority/run burst/job size	3.32.33

(continued on next page)

Table 3–7: .UUO Subfunctions — Calls to the File Processor (FIP) (Cont.)

Mnemonic	BASIC–PLUS SYS Call Code (Decimal)	Privileged Status	Function	Section
UU.TB2	–12	No	Get monitor tables — Part II	3.32.42
UU.BCK	–11	Yes	Change file backup statistics	3.32.5
(.FSS)	–10	No	File name string scan	3.10
UU.HNG	–9	Yes	Hangup a dataset	3.32.22
UU.FCB	–8	No	Get open channel statistics	3.32.20
(internal to BASIC–PLUS)	–7	No	CTRL/C trap enable	—
UU.POK	–6	Yes*	Poke memory	3.32.31
(.SPEC)	–5	Yes	Broadcast to terminal	3.23
(.SPEC)	–4	Yes	Force input to terminal	3.23
UU.TB1	–3	No	Get monitor tables — Part I	3.32.41
UU.NLG	–2	Yes	Disable logins	3.32.30
UU.YLG	–1	Yes	Enable logins	3.32.45
UU.PAS	0	Yes	Create user account	3.32.30
UU.DLU	1	Yes	Delete user account	3.32.17
UU.CLN	2	Yes	Obsolete. Use ONLCLN.	—
UU.MNT	3	Yes	Disk pack status	3.32.27
UU.LIN	4	Yes	Login	3.32.24
UU.BYE	5	Both	Logout	3.32.6
UU.ATT	6	Both Both No	Attach Reattach Swap console	3.32.4
UU.DET	7	Yes	Detach	3.32.14
UU.CHU	8	Yes Yes Yes	Change password/quota Kill job Disable terminal	3.32.9

* Poke memory can be executed only from account [1,1].

(continued on next page)

Table 3–7: .UUO Subfunctions — Calls to the File Processor (FIP) (Cont.)

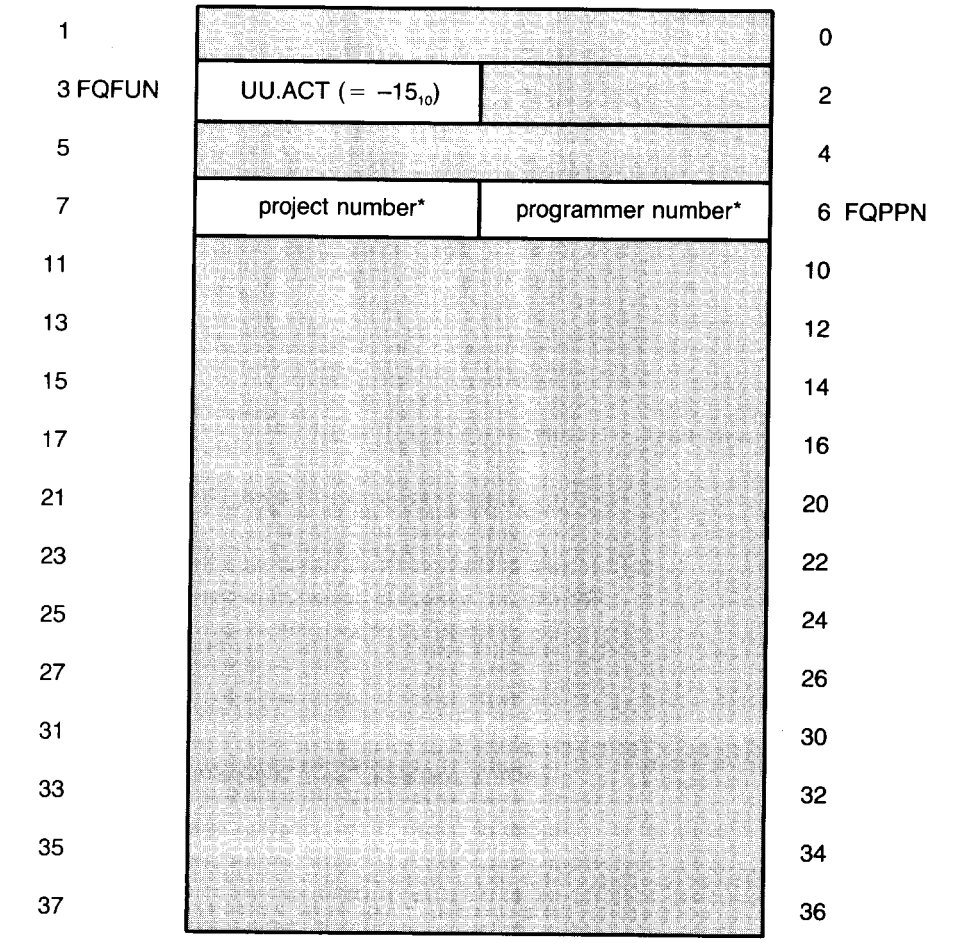
Mnemonic	BASIC–PLUS SYS Call Code (Decimal)	Privileged Status	Function	Section
UU.ERR	9	No	Return error message	3.32.19
UU.ASS	10	Both	Assign/reassign device**	3.32.2
UU.DEA	11	No	Deassign device**	3.32.13
UU.DAL	12	No	Deassign all devices**	3.32.11
UU.ZER	13	Both	Zero a device	3.32.46
UU.RAD	14	Both	Read or read-and-reset accounting data	3.32.34
UU.DIR	15	Both No	Directory lookup on index Special magnetic tape directory lookup	3.32.16
UU.TRM	16	Both	Set terminal characteristics	3.32.44
UU.LOK	17	Both Both	Disk directory lookup on file name Disk wildcard directory lookup	3.32.26
	18		Obsolete	
UU.CHE	19	Yes	Enable/disable disk caching	3.32.8
UU.CNV	20	No	Date and time conversion	3.32.10
UU.SLN	21	Yes	System logical names	3.32.36
(.MESAG)	22	Both	Message send/receive	3.12
UU.SWP	23	Yes	Add, remove, list system files	3.32.39
UU.JOB	24	Both	Create job	3.32.23
UU.PPN	25	No	Wild card PPN lookup	3.32.32
UU.STL	29	Yes	Stall/unstall system	3.32.38
UU.SYS	26	Both	Return job status information	3.32.40

** To assign or deassign user logicals, use .ULOG, described in Section 3.31.

3.32.1 UU.ACT (Accounting Information Dump) – Privileged

Data Passed

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no meaningful data is returned with the UU.ACT subfunction of .UUO.

Errors

NOSUCH The account specified does not exist.

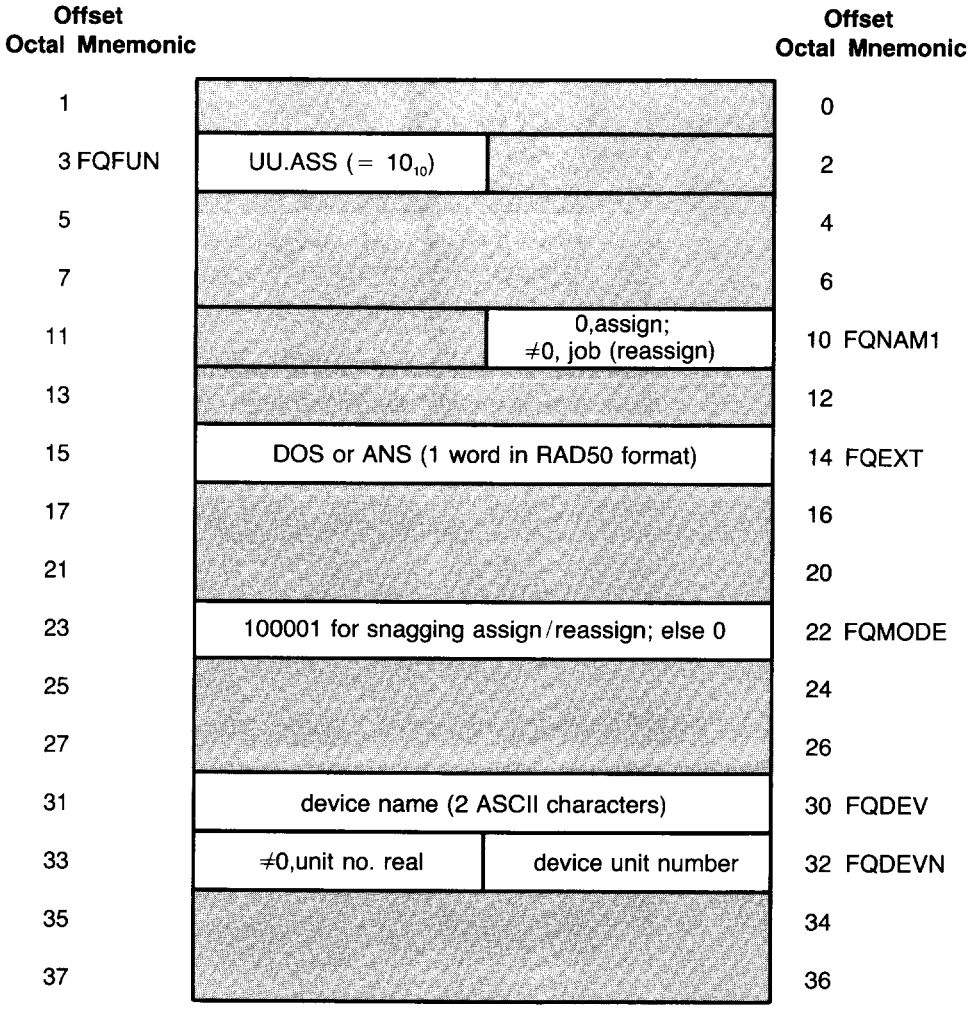
---

* 0 in both bytes means the current account.

3.32.2 UU.ASS (Assign/Reassign Device) – Privileged and Not Privileged

Data Passed

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no meaningful data is returned with the UU.ASS subfunction of .UUO.

Errors

INUSE For a reassign, the specified device is open or has an open file. For an assign, more than four user logical assignments are made.

NODEVC Device name at FIRQB + FQDEV is a logical name for which no physical device is assigned.

NOTAVL The device specified exists on the system, but the operation failed for one of the following reasons:

1. The device is currently reserved by another job (see description of FIRQB + FQMODE).
2. Ownership of the device requires privilege that the user does not have. For example, a nonprivileged user tried to assign a device that is currently assigned to another user.
3. The device or its controller is disabled.
4. The device is a keyboard line for a pseudo keyboard only.

PRVIOL You are nonprivileged and tried to either:

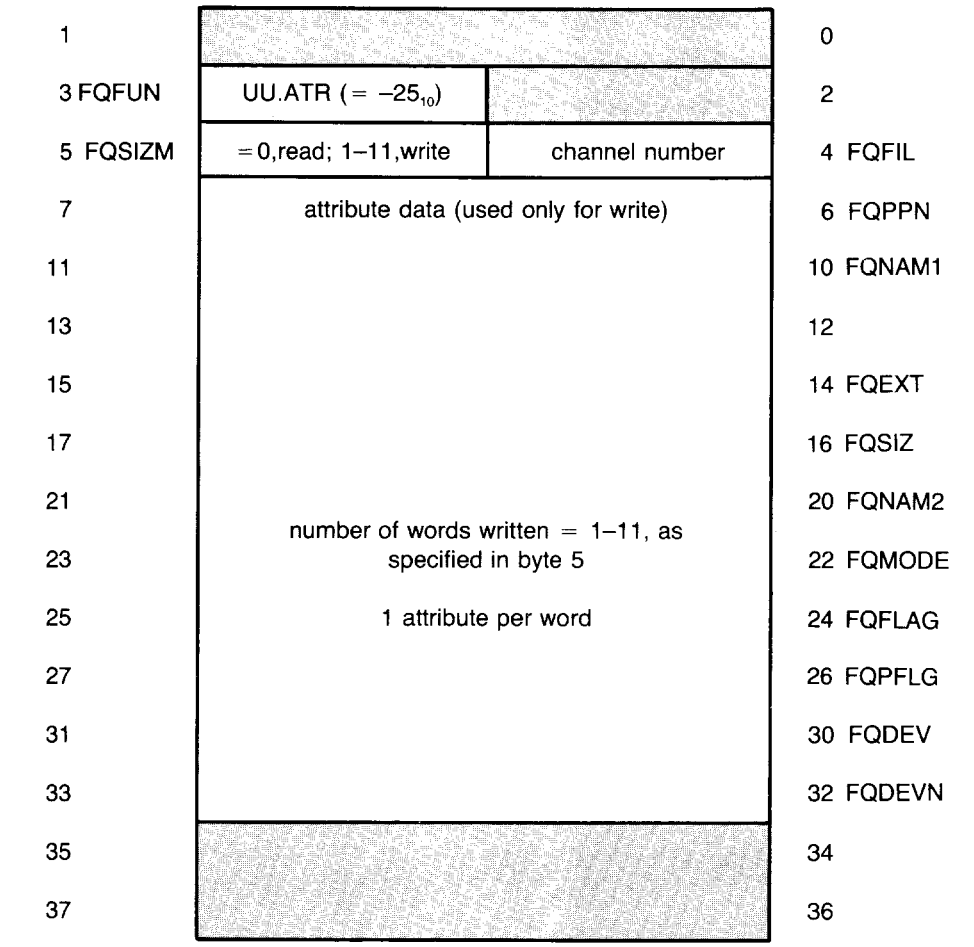
1. Reassign a device that requires privilege to assign.
2. Reassign a device to a job that is logged in to an account other that you current account.

BDNERR Your program attempted to reassign a device to a nonexistent job.

3.32.3 UU.ATR (Read/Write File Attributes) – Not Privileged

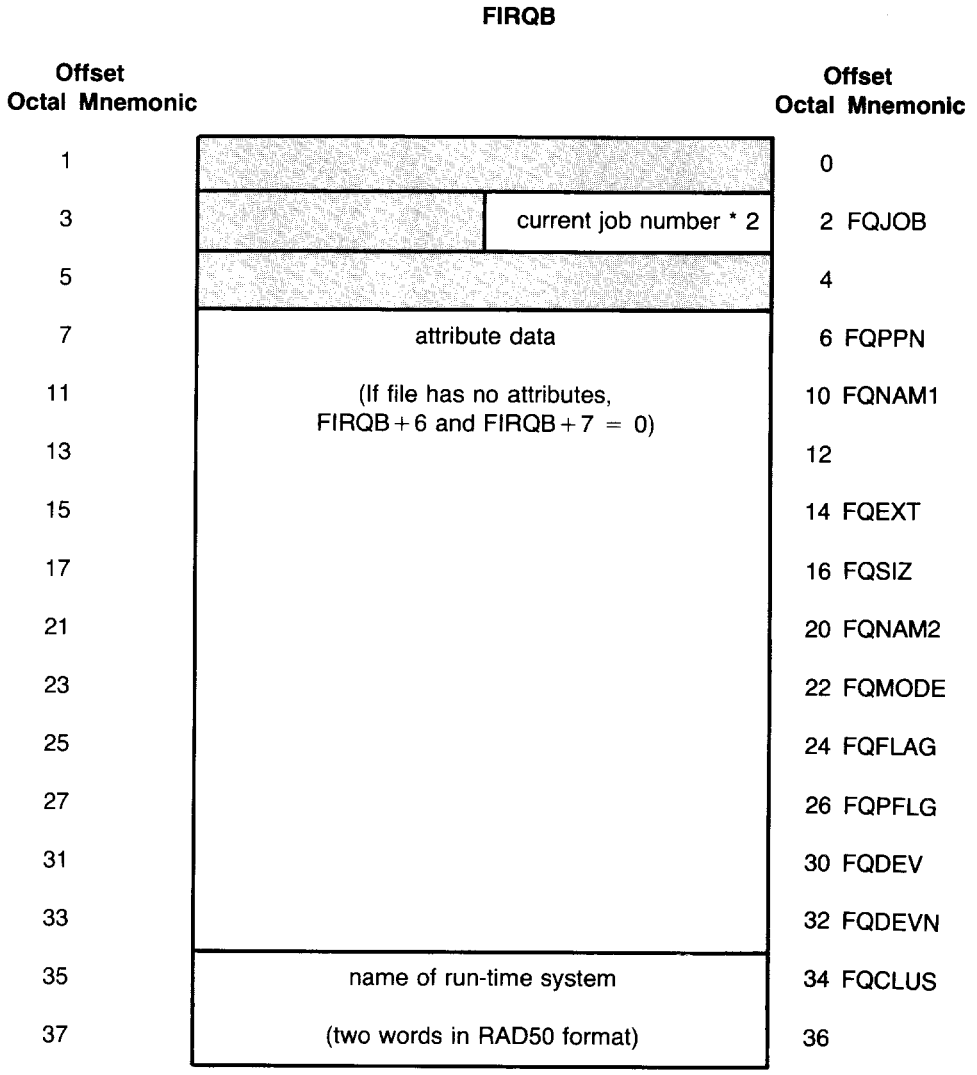
Data Passed

FIRQB



Data Returned

Other than a possible error in byte 0 of the FIRQB, data is returned on a "read" only (byte 5 of data passed equals 0).



Errors

NOROOM Occurs only on a write. The User File Directory (UFD) of the account is full. Some files must be deleted to free entries for attributes.

NOTOPN Channel specified at FIRQB + FQFIL must have file open.

PRVIOL Job does not have read (or write) access to the file open on the channel, or a UFD is open on the channel.

DEVNFS Device on which file is open must be disk.

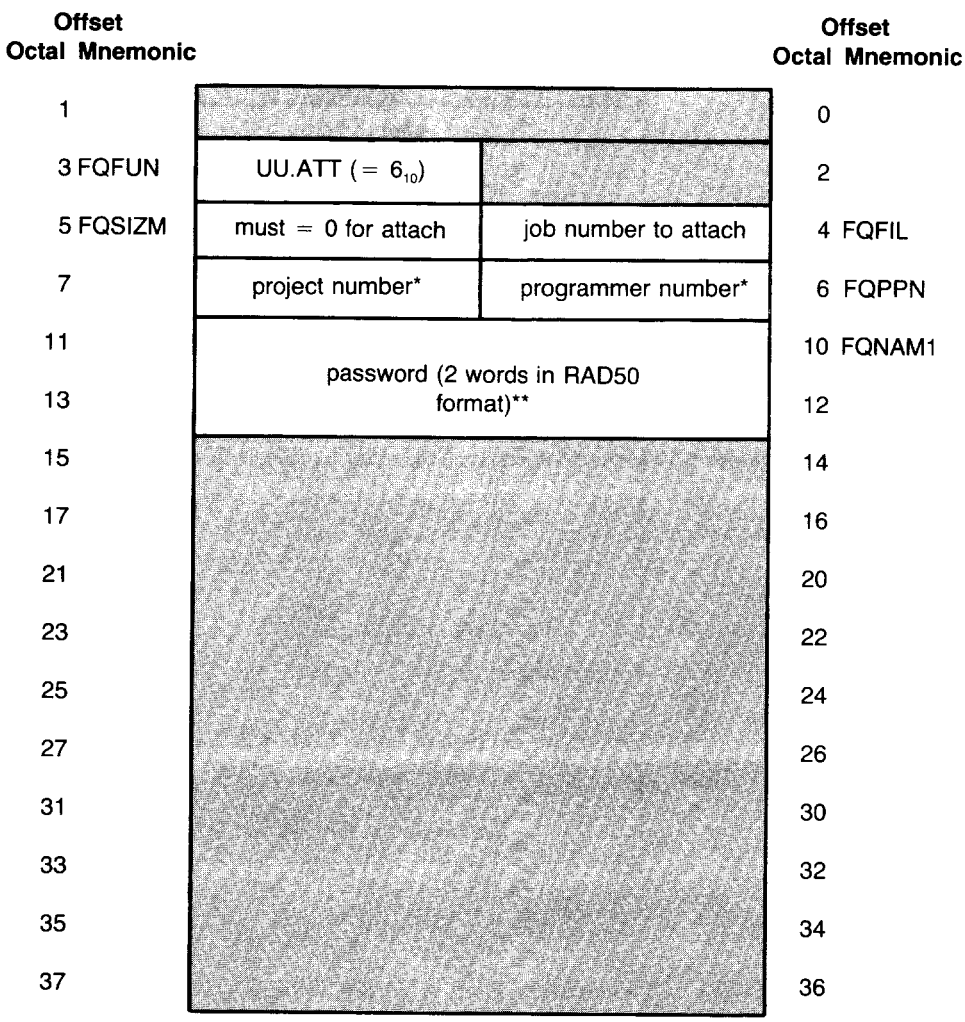
BADCNT Write only. No value greater than 11 can be specified at FIRQB + FQFIL + 1.

BSERR Attributes can be written only on channels 1 through 15.

3.32.4 UU.ATT (Attach/Reattach Job/Swap Console) – Privileged and Not Privileged

Data Passed — Attach

FIRQB



* 0 in both bytes means your project-programmer number. You do not need to include a password. If you are nonprivileged, you can only specify your own project-programmer number.

** You must include a password if you specify a project-programmer number other than your own. Note that this use of the call requires privilege.

Data Returned

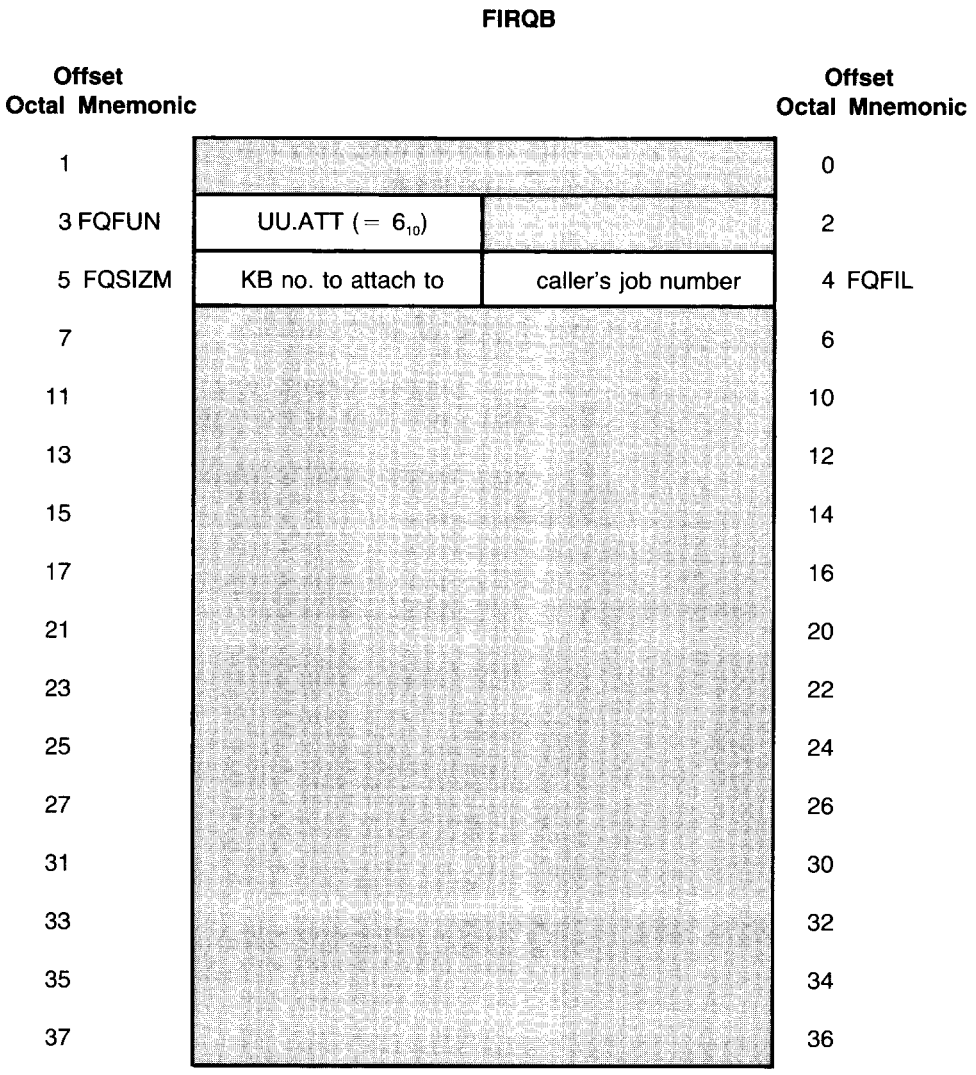
Other than a possible error in byte 0 of the FIRQB, no data is returned by the attach function.

Errors

BADFUO

1. Job executing the call has an open channel.
2. Job executing the call is detached.
3. Caller is a source (BAS) program rather than a compiled (BAC) program.
4. Job number at FIRQB + FQFIL is not detached.
5. Project-programmer number (ppn) specified does not match ppn of job to attach.
6. Caller is nonprivileged, and the job to attach has a ppn different from that of the caller.
7. Password is not valid or contains nonalphanumeric characters.

Data Passed — Reattach



Data Returned

Other than a possible error in byte 0 of the FIRQB, no data is returned by the reattach function.

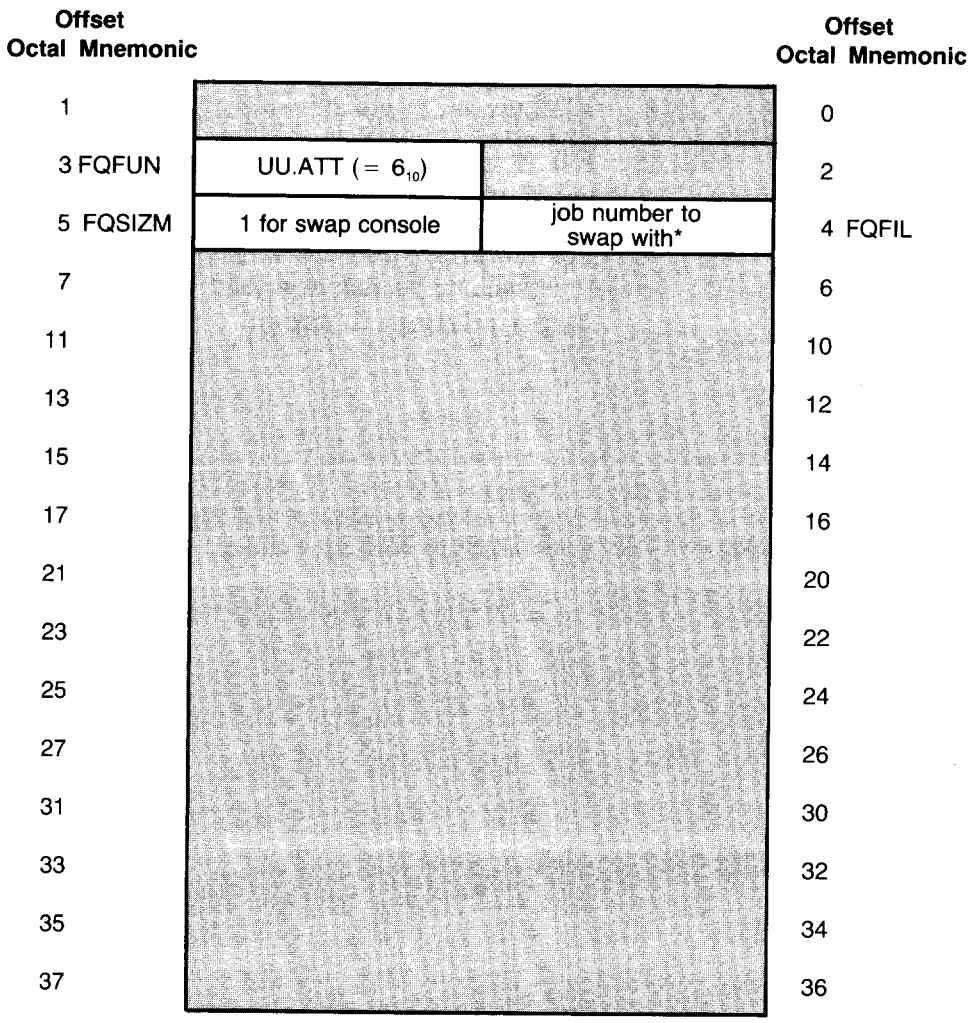
Errors

BADFUO

1. Job number at FIRQB+FQFIL is less than 1 or greater than JOB MAX on system.
2. Caller is not detached.
3. KB number is out of range.
4. KB is currently assigned, opened, or the console keyboard of some job other than the calling job.
5. For nonprivileged users, this error also occurs if the KB to attach to is not assigned to any job, and the KB requires privilege to assign.

Data Passed — Swap Console

FIRQB



* If the calling job is attached, this job must be detached. If the calling job is detached, this job must be attached. Both the calling job and this job must be running under the same project-programmer number.

Data Returned

Other than a possible error in byte 0 of the FIRQB, no data is returned by the swap console function.

Errors

BADFUO

1. Both the calling job and the job specified at FIRQB+FQFIL are detached, or neither job is detached.
2. The job specified at FIRQB+FQFIL has a project-programmer number different from the caller's.
3. The value at FIRQB+FQSIZM is neither 0 nor 1.

3.32.5 UU.BCK (Change File Statistics) – Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.BCK (= -11_{10})		2	
5	FQSIZM	LSB of last access*	channel no. (1_{10}–15_{10})	4	FQFIL
7		LSB of creation date*	MSB of last access*	6	FQPPN
11		LSB of creation time*	MSB of creation date*	10	FQNAM1
13			MSB of creation time*	12	
15				14	
17				16	
21				20	
23				22	
25				24	
27				26	
31				30	
33				32	
35				34	
37				36	

Data Returned

Other than a possible error in byte 0 of the FIRQB, no data is returned by the UU.BCK subfunction.

Errors

BADFUO The file open on the channel specified is not a disk file or is a user file directory.

---

* The system internal format for dates has the form:

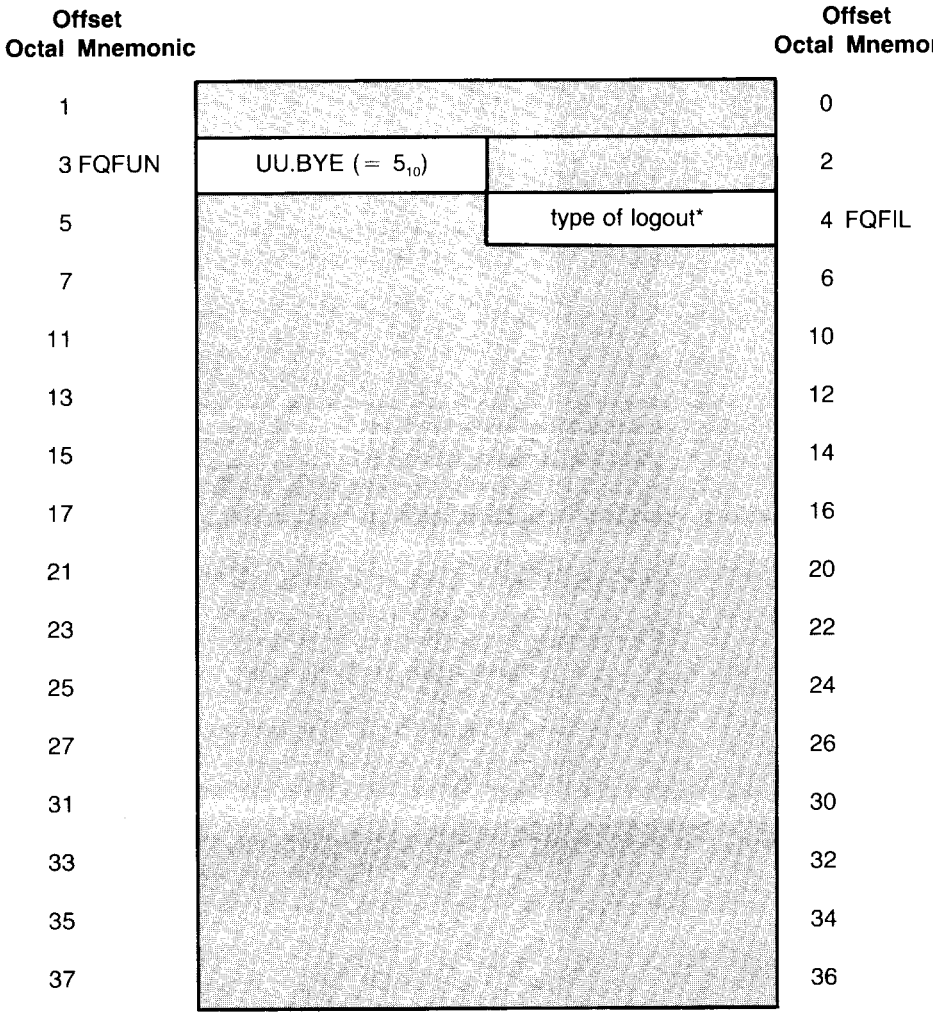
[(year – 1970) * 1000_{10}] + day-within-year

Time is specified in minutes until midnight, where 1440 equals midnight. See the .DATE directive for a discussion of these formats.

3.32.6 UU.BYE (Logout) – Privileged and Not Privileged

Data Passed

FIRQB



* For privileged users, bits 0 and 1 specify the type of logout:

Bit 0 = 0 Close I/O channels, deassign devices, remove receivers, and dismount disks mounted /NOSHARE before performing logout.

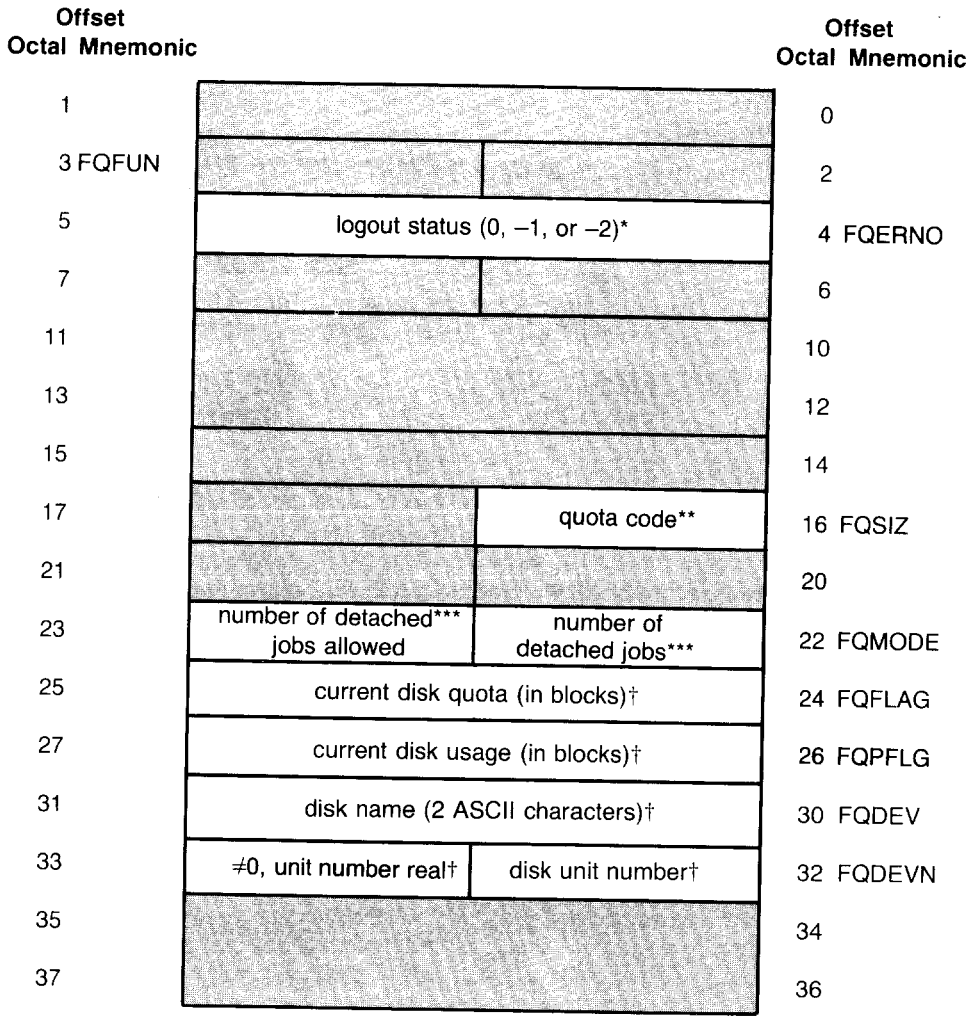
= 1 Just perform logout; do not close channels, deassign devices, remove receivers, or dismount disks.

Bit 1 = 0 Check quotas on all mounted disks before performing logout.

= 1 Perform logout without checking disk quotas.

For nonprivileged users, the system forces bits 0 and 1 to 0.
Bits 2 through 7 are reserved; set them to 0.

Data Returned



Errors

No errors are possible with the UU.BYE subfunction.

---

* 0 = Neither disk quota nor nonprivileged detached job quota is exceeded. If you are privileged, the system returns control to your program. If you are nonprivileged, the system kills your job after performing necessary clean-up functions.

–1 = A quota is exceeded; your job is still logged in.

–2 = A quota is exceeded, but your job is logged out. If you are privileged, the system returns control to your program. If you are nonprivileged, the system kills your job after performing necessary clean-up functions.

** Indicates which quota is exceeded. (This value is valid only if FIRQB + FQFIL is –1 or –2.)

0 = Disk quota

1 = Nonprivileged detached job quota.

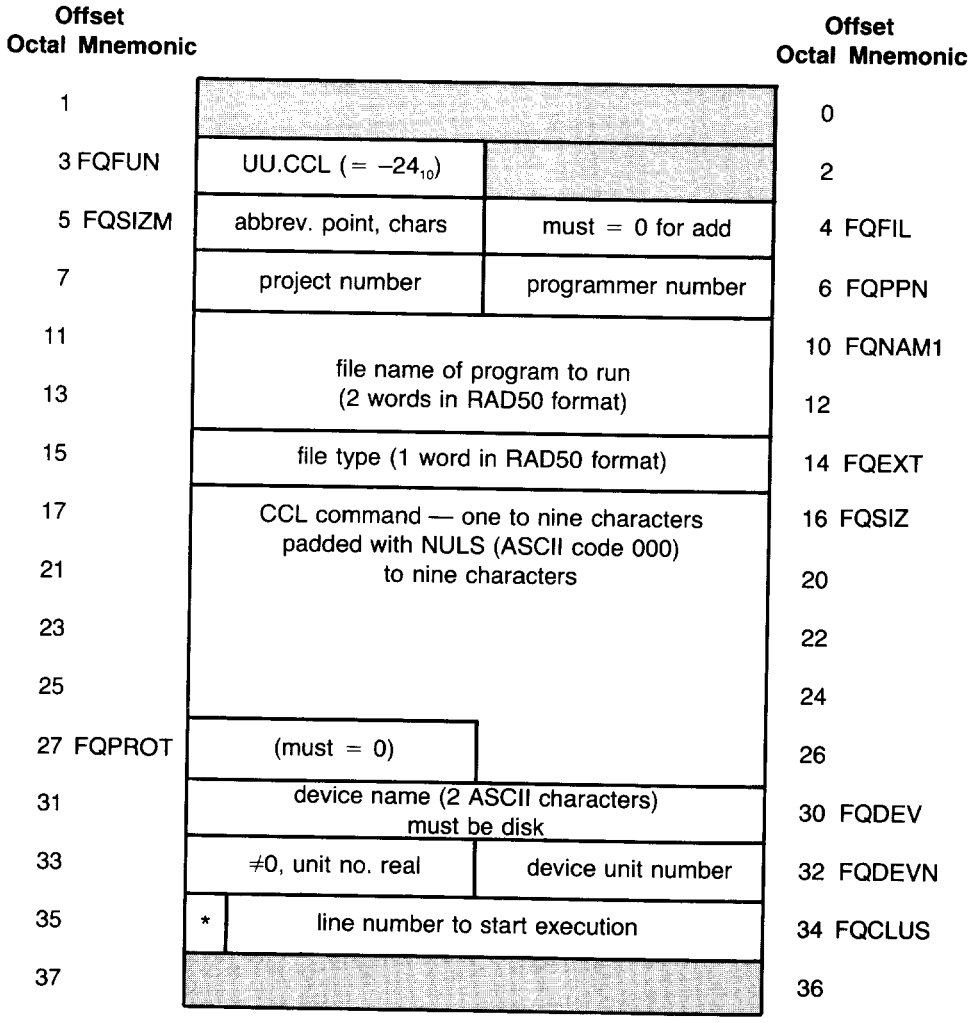
*** Returned if nonprivileged detached job quota is exceeded. FIRQB+FQMODE contains the number of detached jobs in the current account. FIRQB+FQMODE+1 contains the number of detached jobs currently allowed.

† Returned if disk quota is exceeded.

3.32.7 UU.CCL (Add/Delete CCL Command) – Privileged

Data Passed — Add

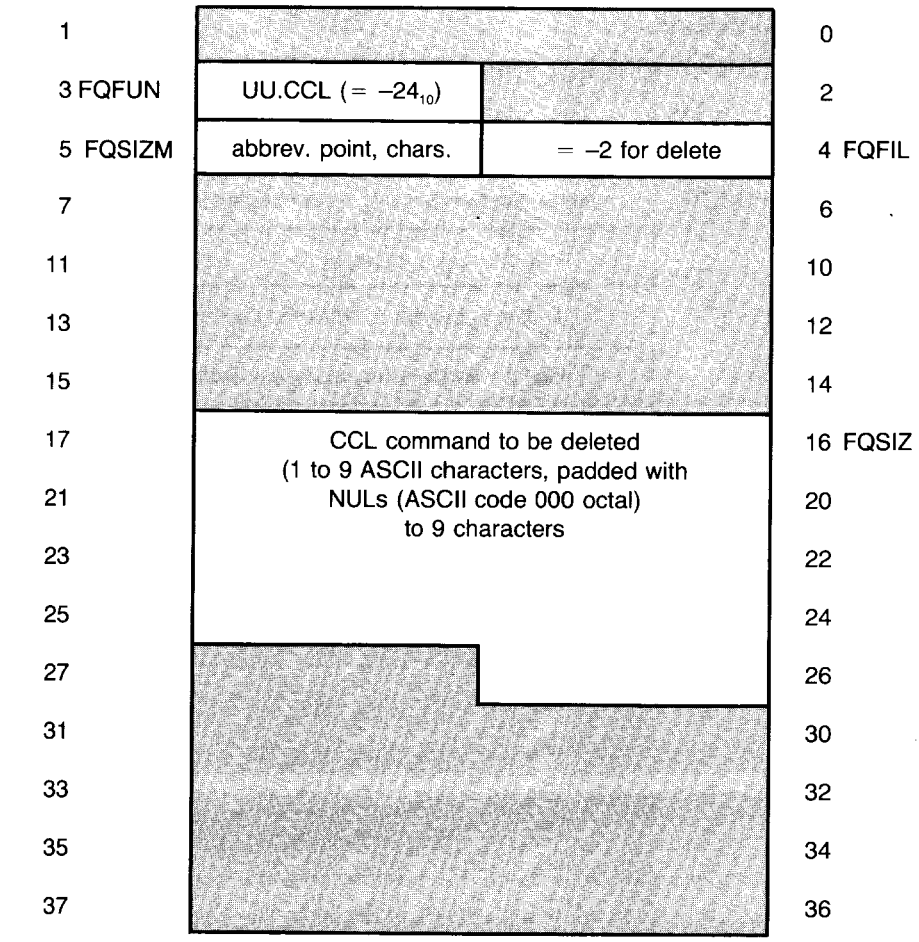
FIRQB



* Privilege indication in bit 15.

Data Passed — Delete

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.CCL.

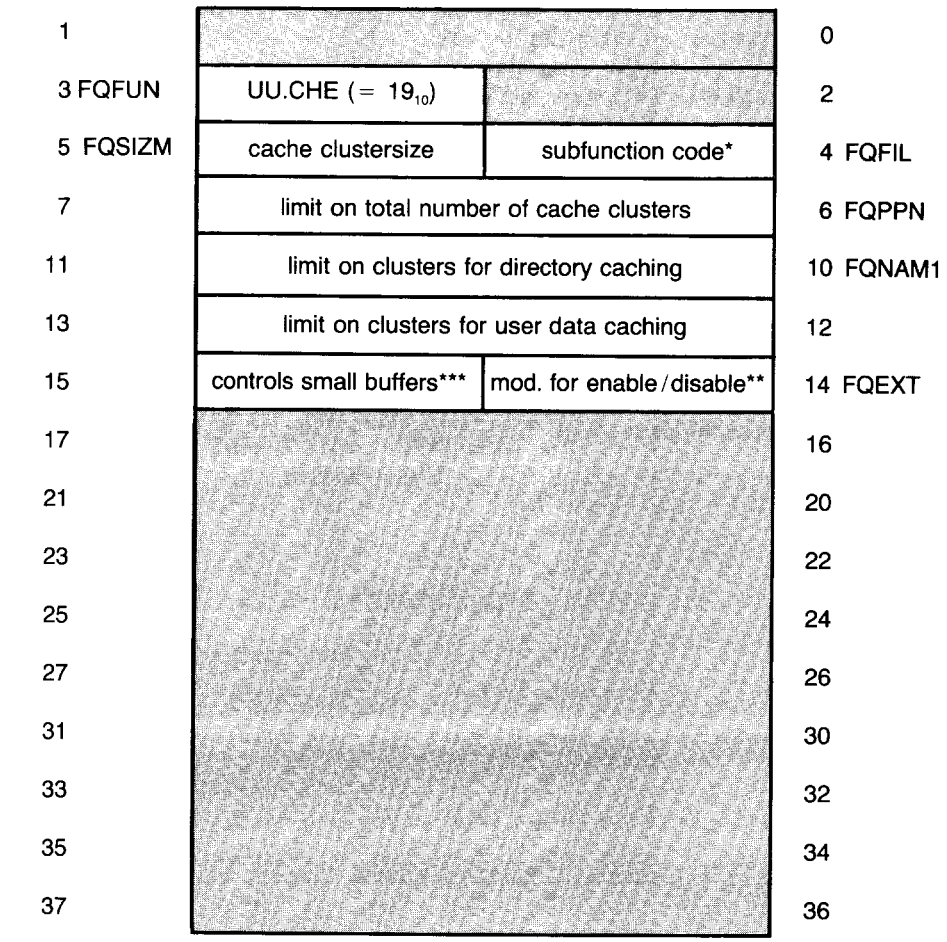
Errors

BADNAM	For add only. The CCL command either begins with a number or contains an otherwise unacceptable character.
INUSE	For add only. The CCL command is already defined.
NODEVC	For delete only. The CCL command does not exist.

3.32.8 UU.CHE (Enable/Disable Disk Caching) – Privileged

Data Passed

FIRQB



---

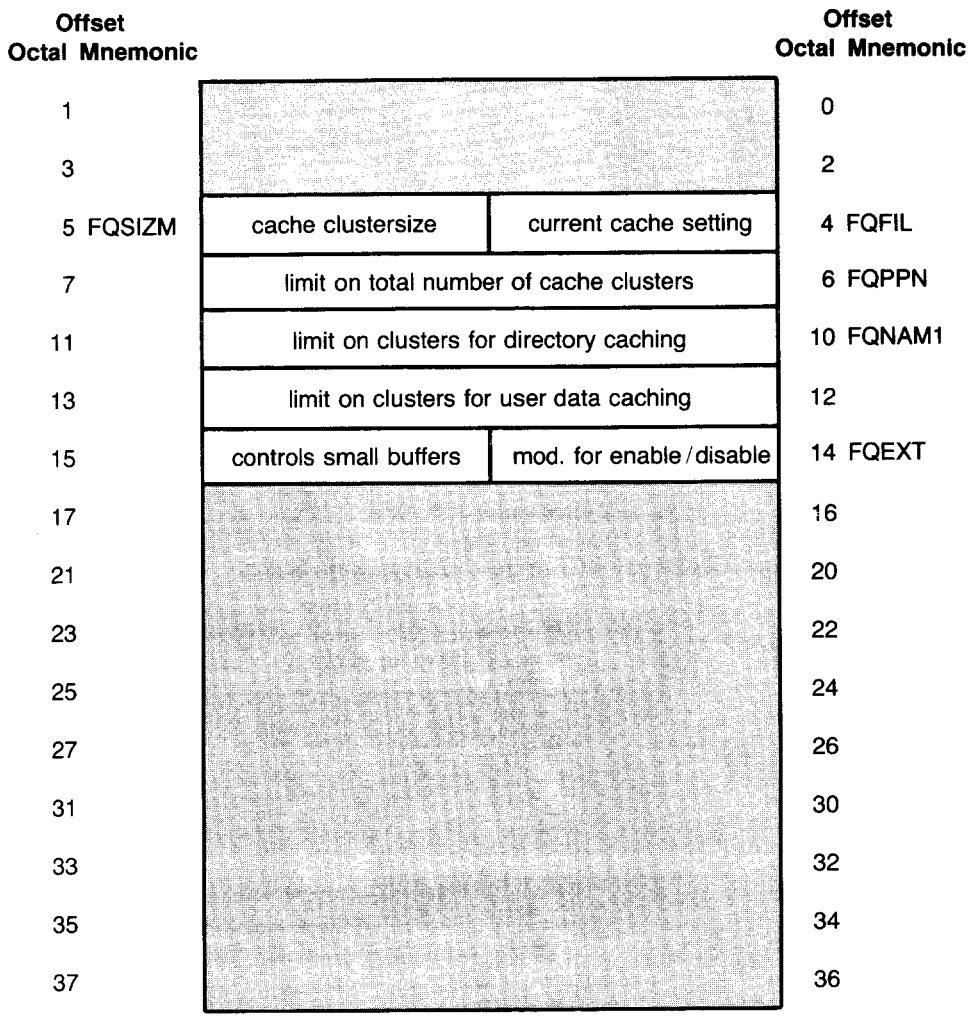
* 0 = Enable directory and data caching
1 = Disable all caching
200 = Return current caching parameters

** 0 = Use current setting
1 = Enable caching as specified in file open mode or UFD setting
100 = Cache all data transfers regardless of the file open mode or UFD setting
200 = Disable all caching

*** 0 = Use current setting
1 = Allow use of small buffer pool
200 = Do not use small buffer pool

Data Returned

FIRQB



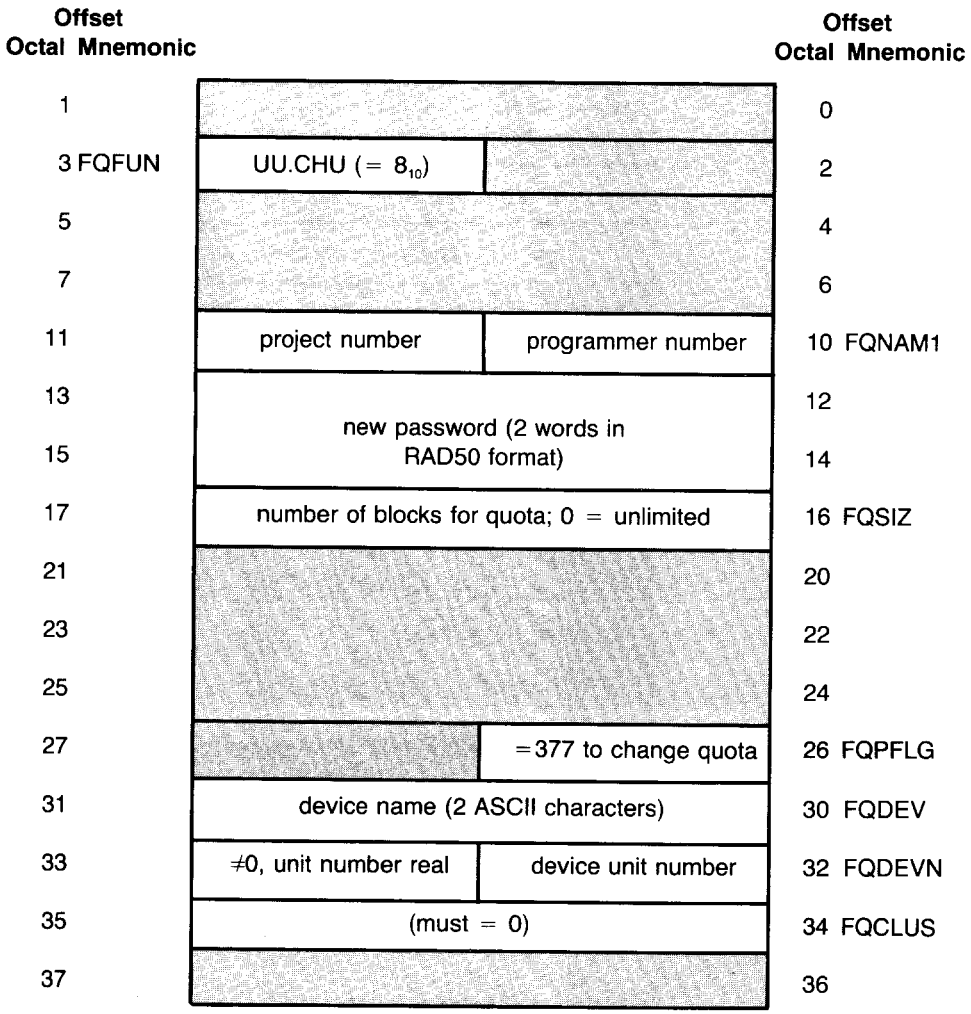
Errors

INUSE	All of the clusters allotted to the cache are in use.
NOROOM	Not enough XBUF space to enable data caching. System manager must allocate at least 2K words.
NOTAVL	Tried to change cluster size while cached file disk transfer in progress. Retry.
PRVIOL	Current user is not privileged.
ERRERR	Caching not enabled during system generation.

3.32.9 UU.CHU (Change Password/Quota, Disable Terminal, Kill Job) – Privileged

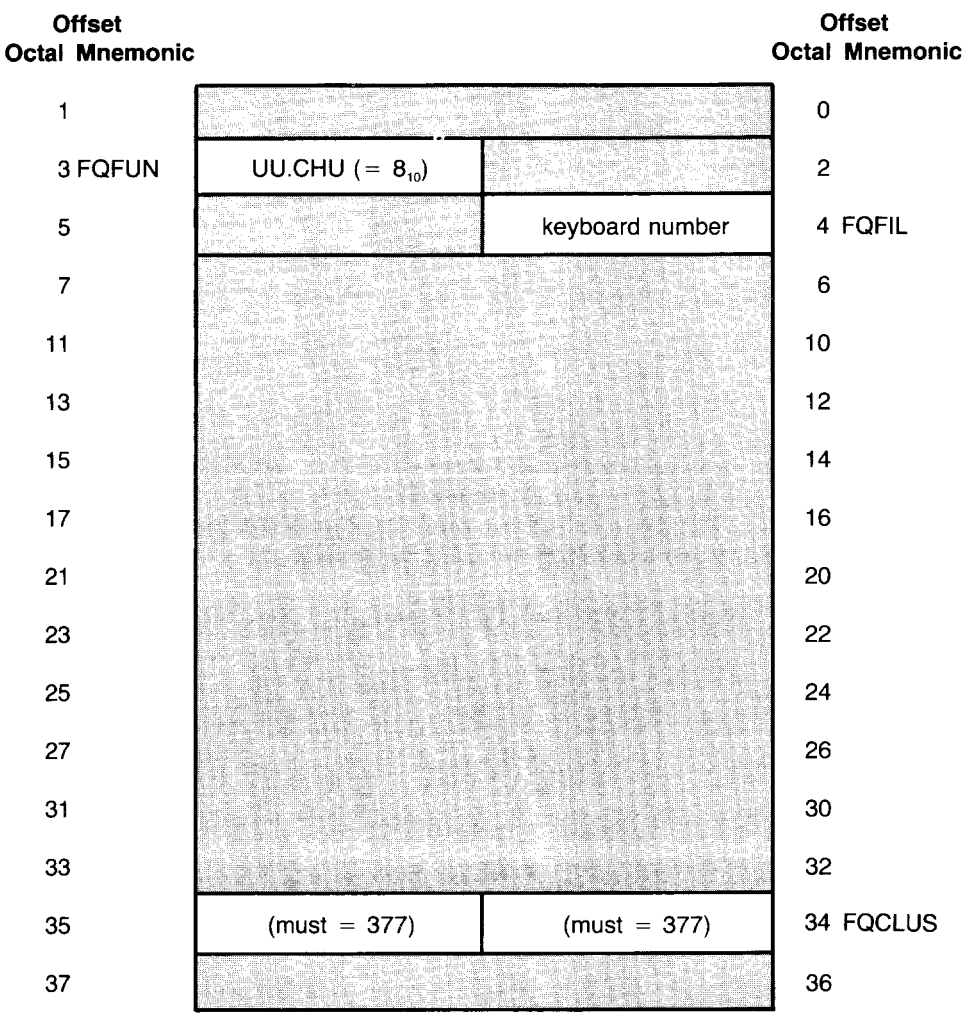
Data Passed (Change Password/Quota)

FIRQB



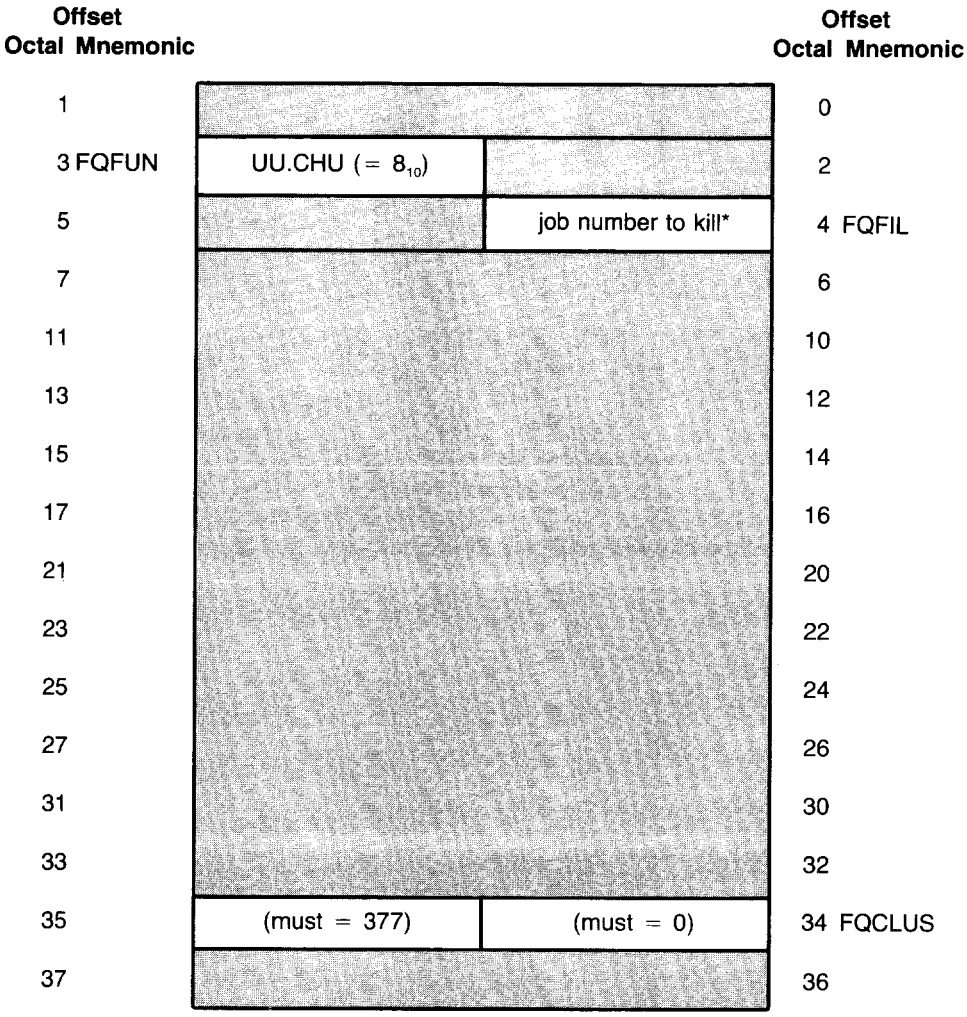
Data Passed (Disable Terminal)

FIRQB



Data Passed (Kill Job)

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the UU.CHU subfunction.

Errors

NOSUCH For change password/quota. Account not present on disk specified.

NODEVC For change password/quota. Device specified is not on the system.

---

* Specify 0 to kill the current job.

BADFUO

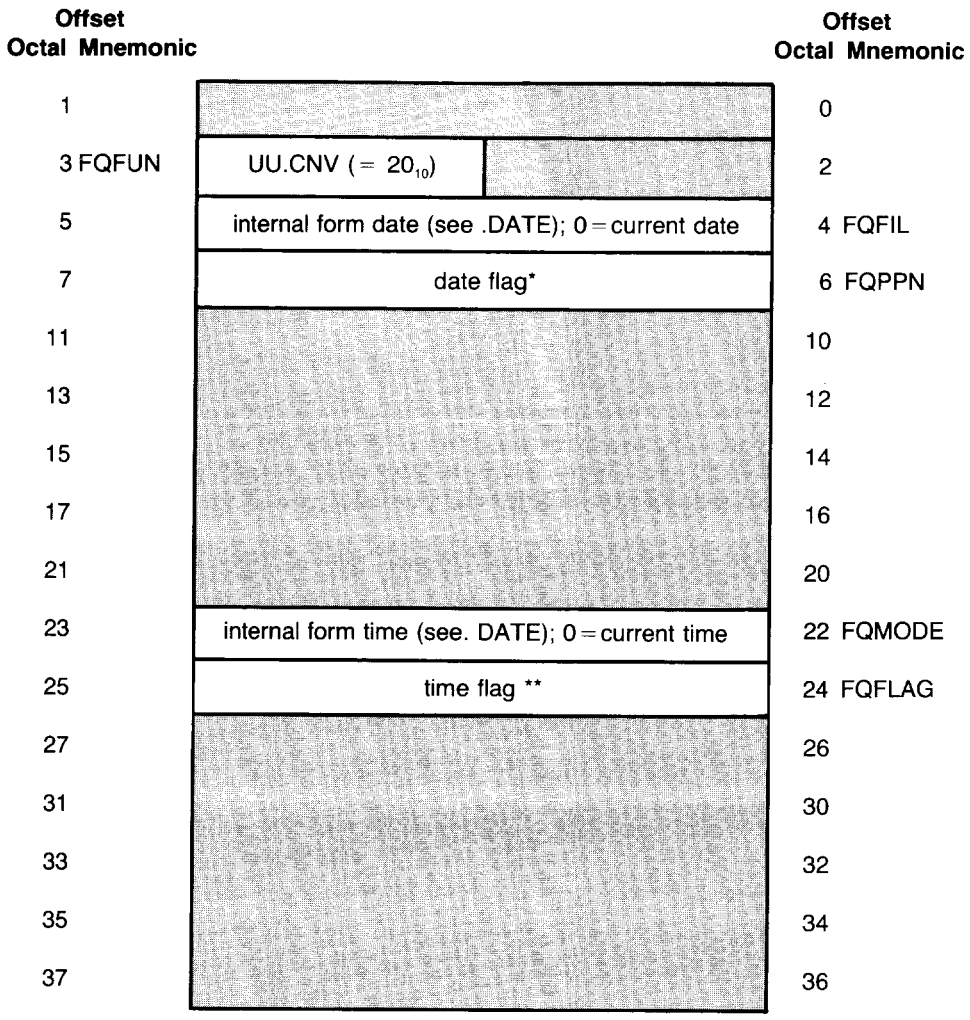
1. For change password/quota. Device specified is not a disk.
2. For kill job. Job number is 0 or greater than JOB MAX.
3. For disable terminal. Keyboard number:

   a. Is greater than number of keyboards on system
   b. Is a pseudo keyboard
   c. Is currently opened or assigned by a job

3.32.10 UU.CNV (Date and Time Conversion) – Not Privileged

Data Passed

FIRQB

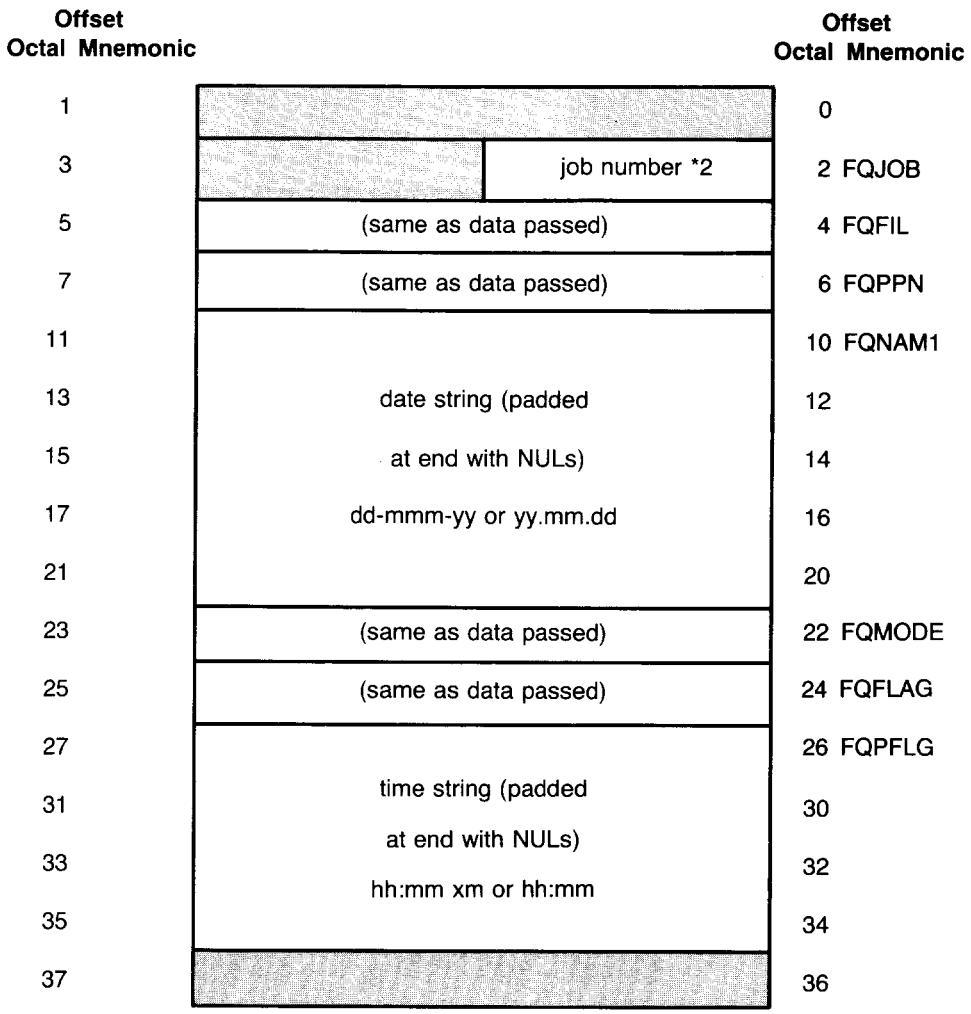


---

* date flag =0 Use system default format
<0 Use alphabetic date format
>0 Use ISO numeric date format

** time flag =0 Use system default format
<0 Use AM/PM time format
>0 Use 24–hour time format

Data Returned

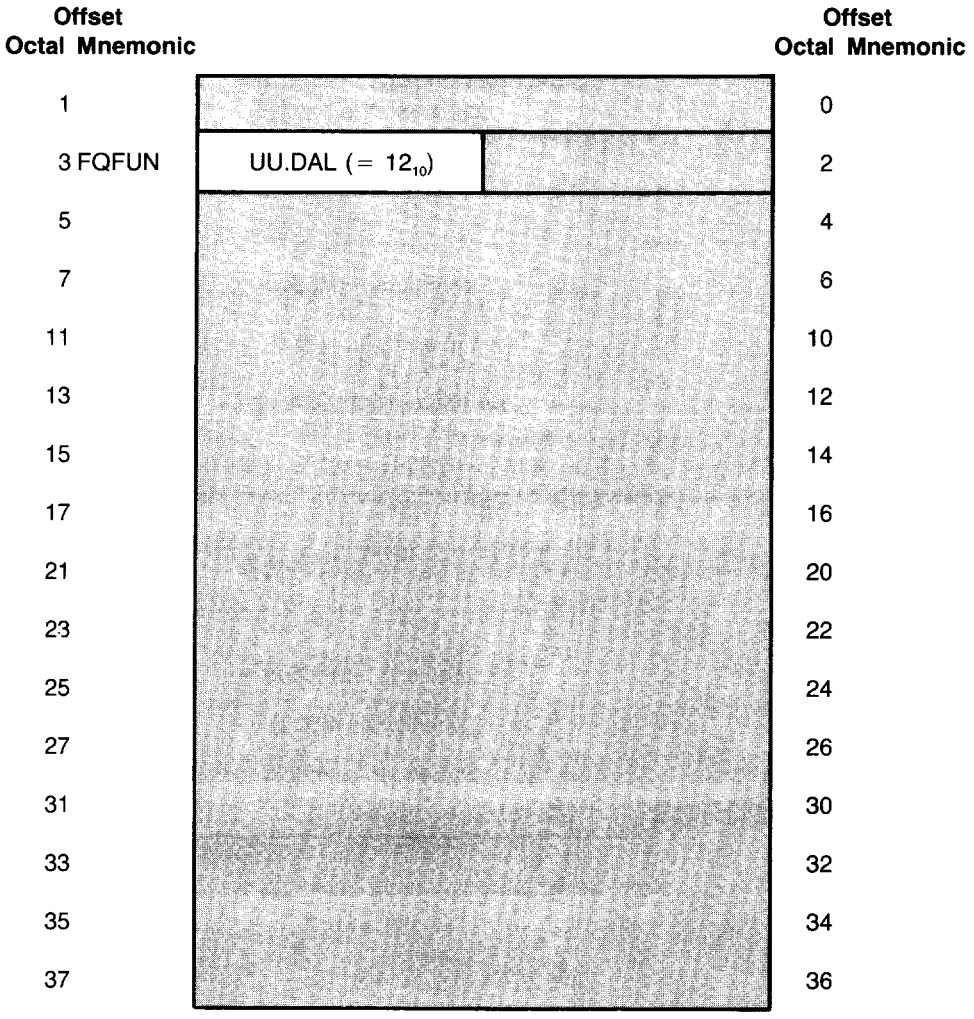


Errors

No errors are possible; however, if the date or time passed is illegal, random output is generated.

3.32.11 UU.DAL (Deassign All Devices) – Not Privileged

Data Passed



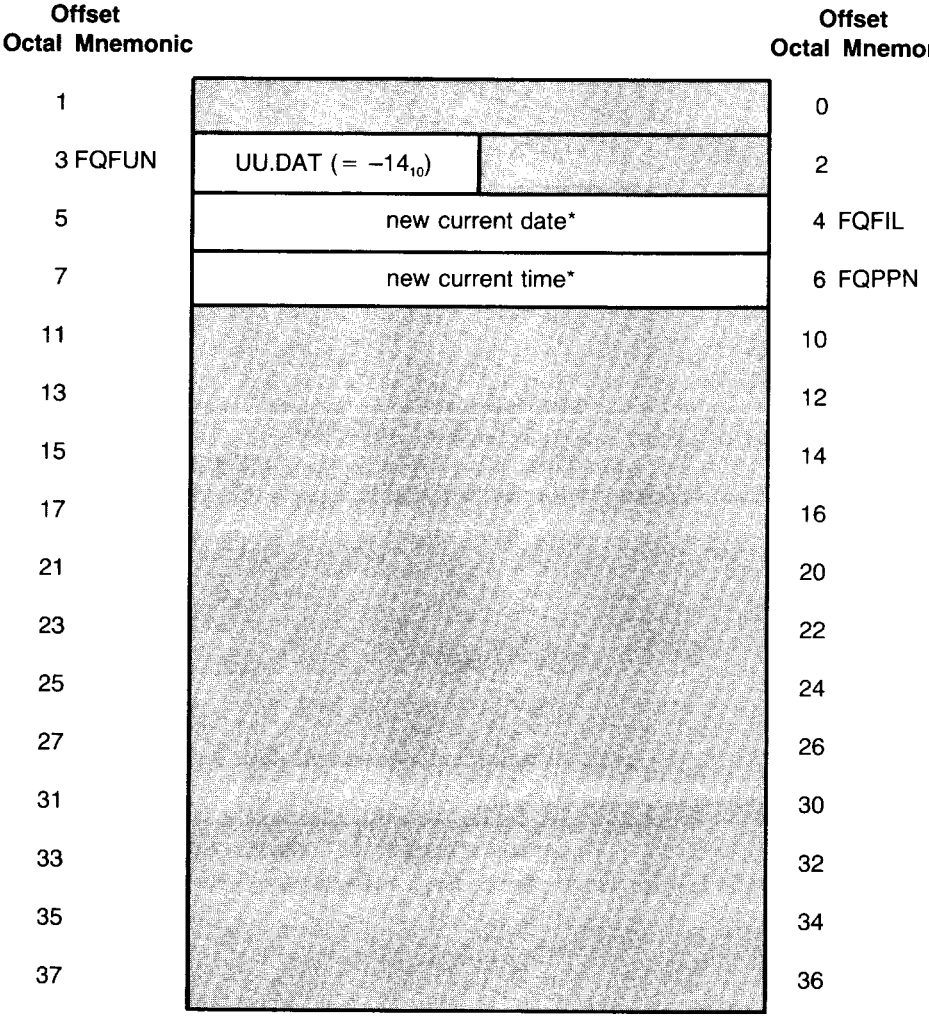
Data Returned

No data is returned by the UU.DAL subfunction; no errors are possible.

3.32.12 UU.DAT (Change System Date/Time) – Privileged

Data Passed

FIRQB



Data Returned

No data is returned by the UU.DAT subfunction; no errors are possible.

---

* The system internal format for date is:

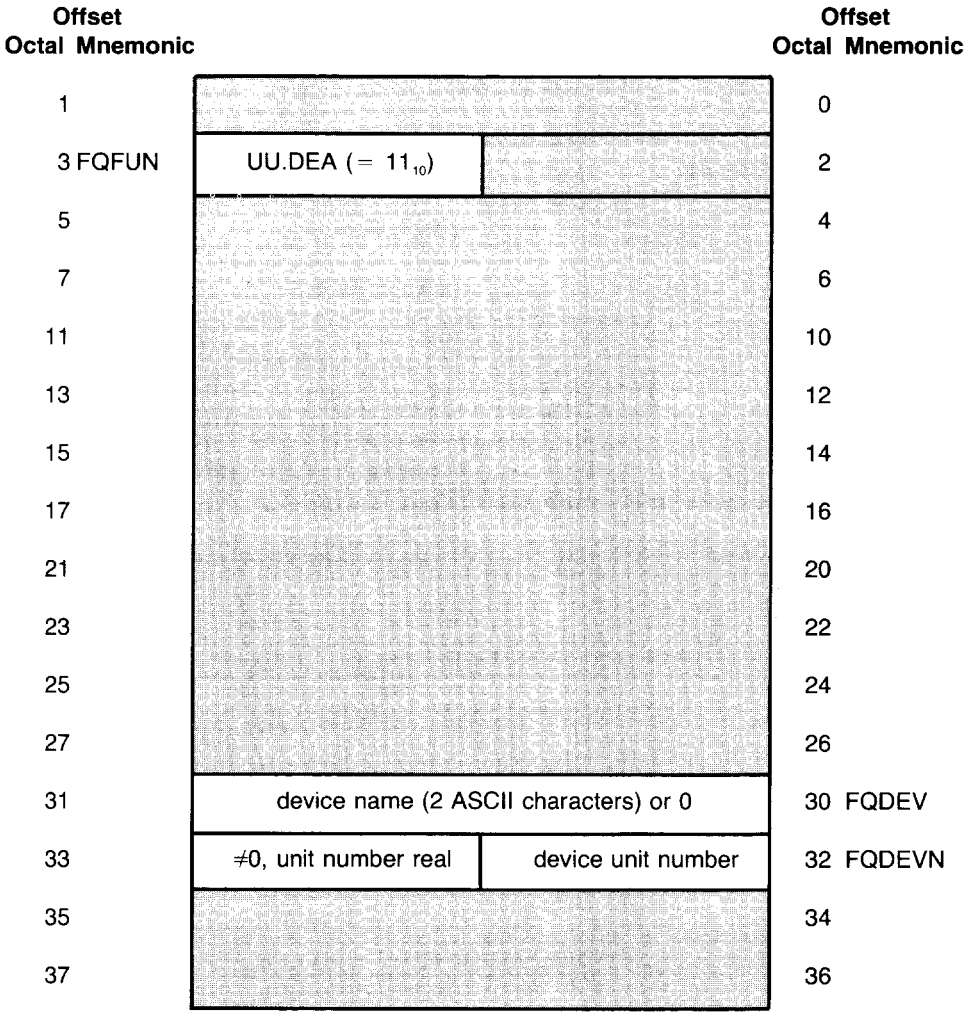
[(year – 1970) * 1000_{10}] + day-within-year

Time is expressed as minutes until midnight, where 1440 equals midnight. A value of 0 in either of these fields means no change is to be made. See the .DATE directive for a discussion of these formats.

3.32.13 UU.DEA (Deassign Device) – Not Privileged

Data Passed

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the UU.DEA subfunction.

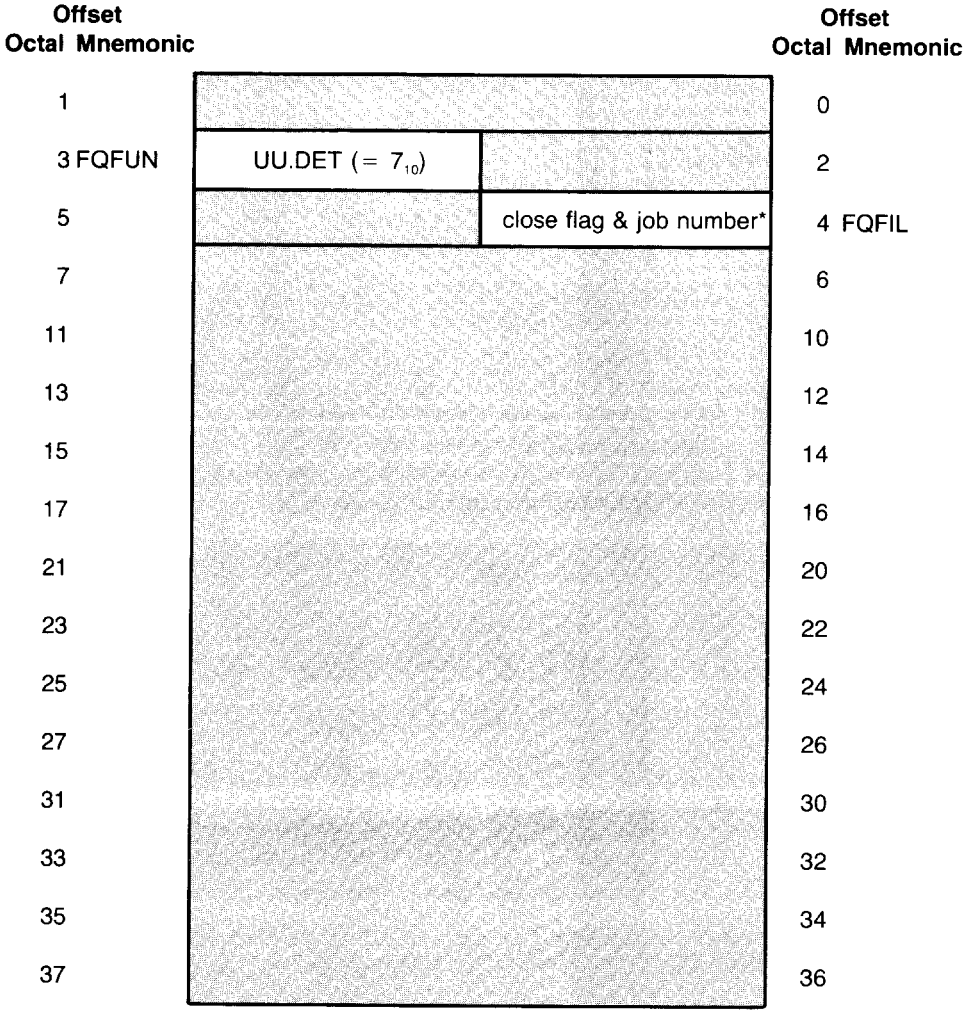
Errors

NODEVC The device or device type in the two words at FIRQB + FQDEV is not on the system.

3.32.14 UU.DET (Detach) – Privileged

Data Passed

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the UU.DET subfunction.

Errors

BADFUO The current job is already detached.

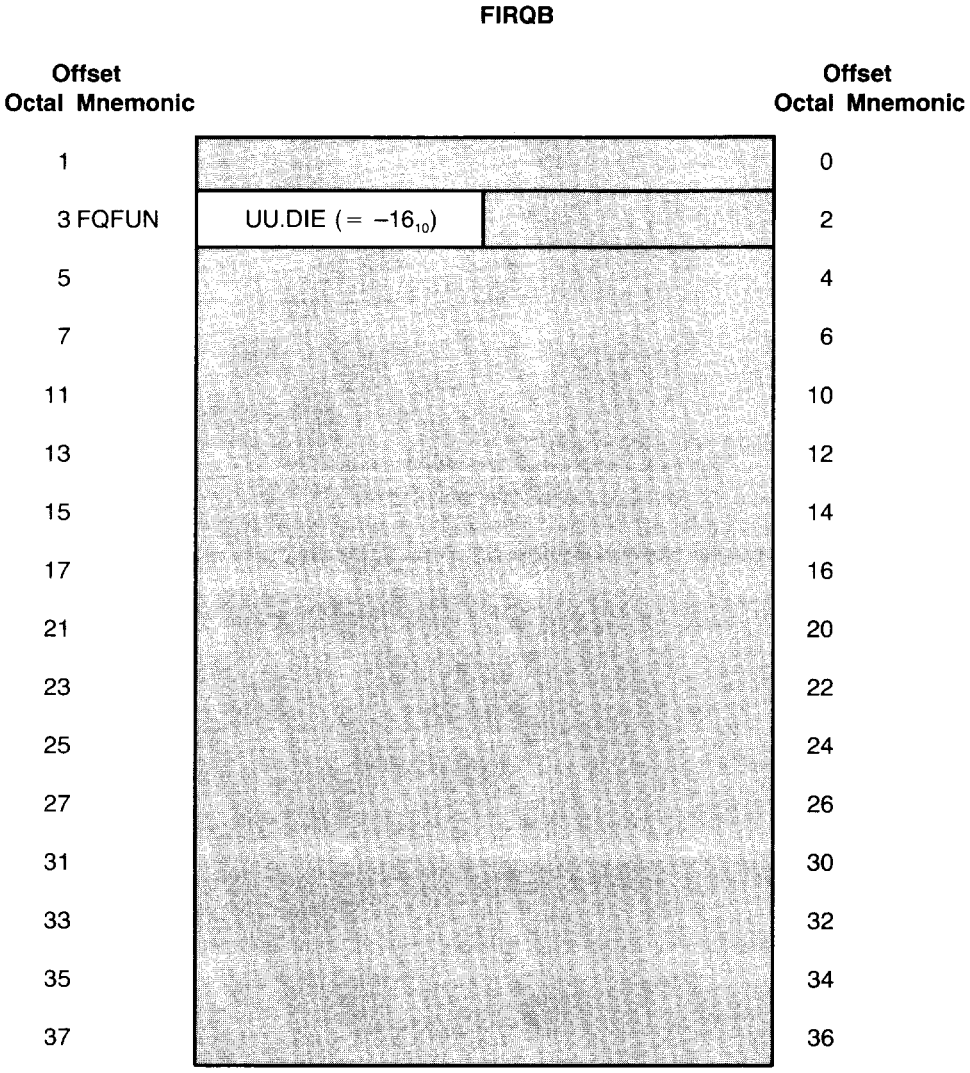
---

* Bit 7 is the "close flag."
  = 0 Do not close terminal channels or deassign terminal.
  = 1 Close all channels on which terminal is open and deassign terminal, if assigned.

Bits 0–6 = Job number to detach. If 0, detach calling job.

3.32.15 UU.DIE (System Shutdown) – Privileged

Data Passed



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.DIE.

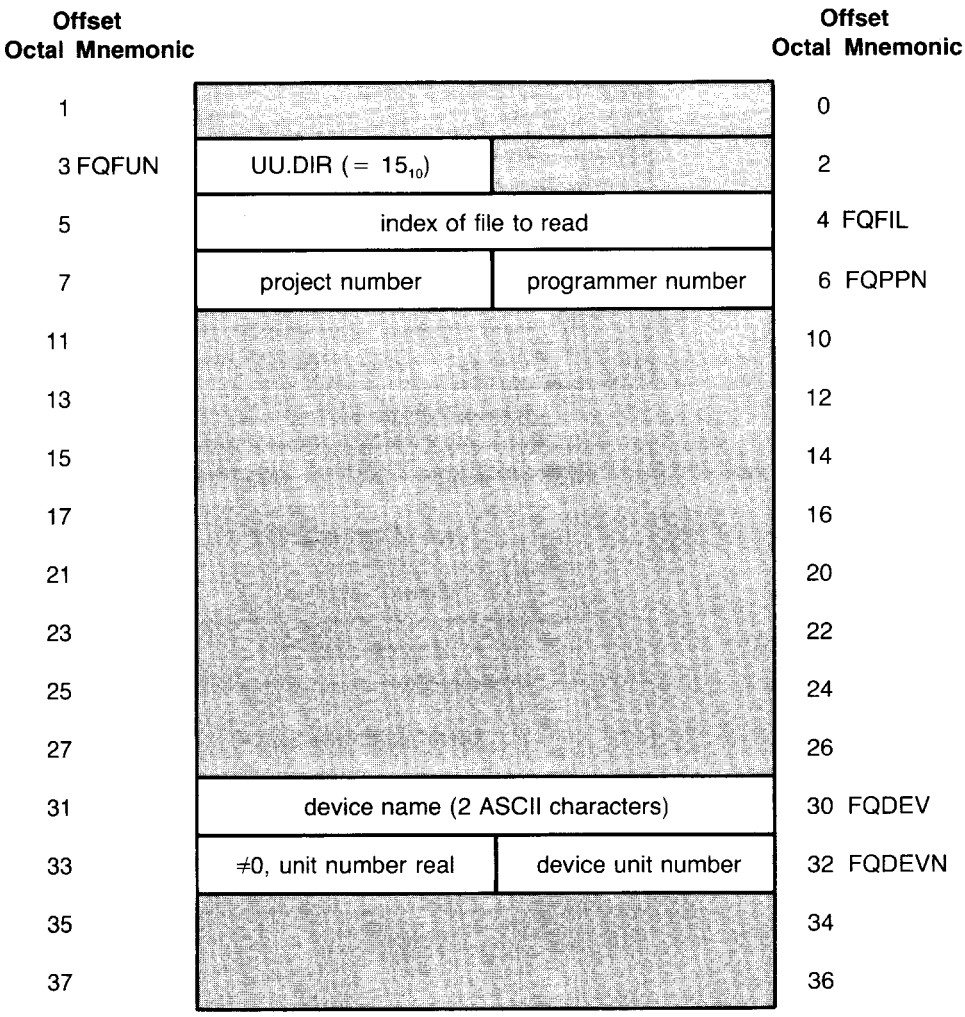
Errors

BADFUO For this call to succeed, only one job can be running, logins must be disabled, no disks except the system disk can be mounted, and no files can be open on the system disk. One of these conditions has not been met.

3.32.16 UU.DIR (Directory Lookup) – Privileged and Not Privileged

Data Passed — Directory Lookup on Index

FIRQB



Data Returned — Directory Lookup on Index

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3		current job number * 2	2	FQJOB
5	(same as data passed)		4	FQFIL
7	project number	programmer number	6	FQPPN
11	file name (2 words in RAD50 format)		10	FQNAM1
13			12	
15	file type (1 word in RAD50 format)		14	FQEXT
17	LSB (least significant bits) of file size		16	FQSIZ
21	MSB of file size	protection code	20	FQNAM2
23	date of last access*		22	FQMODE
25	date of creation*		24	FQFLAG
27	time of creation*		26	
31	(same as data passed)		30	FQDEV
33			32	FQDEVN
35	file cluster size		34	FQCLUS
37	USTAT byte**	number entries returned	36	FQNENT

Errors

NOSUCH	The account specified does not exist on the device or no more files exist on the account.
DEVNFS	The device specified is not file-structured.
xxxxxx	The call also returns device-dependent errors such as NOTMNT (disk pack not mounted).

---

* System internal format for date is:

[(year –1970) * 1000_{10}] + day-within-year

Time is returned as minutes before midnight, where 1440 equals midnight. See the .DATE directive for a discussion of these formats.

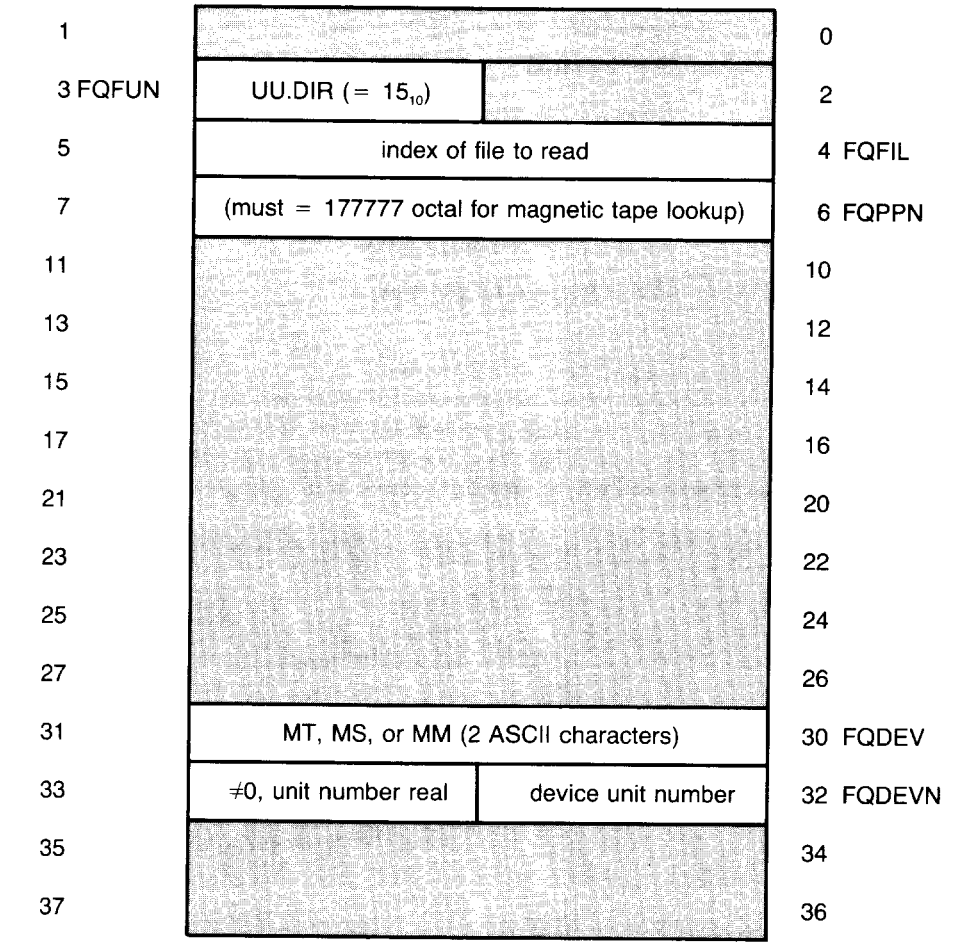
** The USTAT byte from the UFD Name entry.

This byte contains the following internal flag information (for disk only):

Bit Value	Meaning
002	File is placed.
004	Some job has write access now.
010	File is open in update mode.
020	File is contiguous; no extend available.
040	No delete or rename allowed.
200	File is marked for deletion.

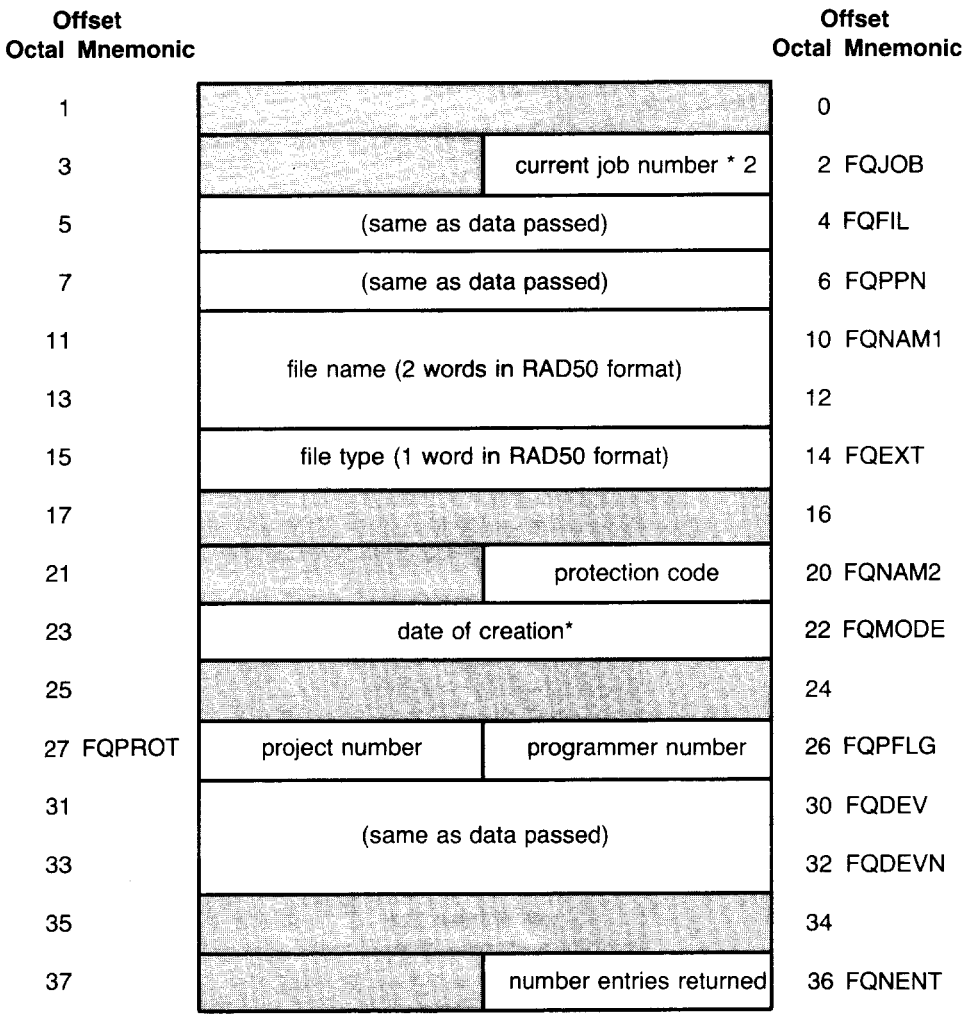
Data Passed — Special Magnetic Tape Directory Lookup

FIRQB



Data Returned — Special Magnetic Tape Directory Lookup

FIRQB



Errors

NOSUCH No more files exist on the tape.

DEVNFS The device specified in the two words at FIRQB + FQDEVN is not file-structured.

---

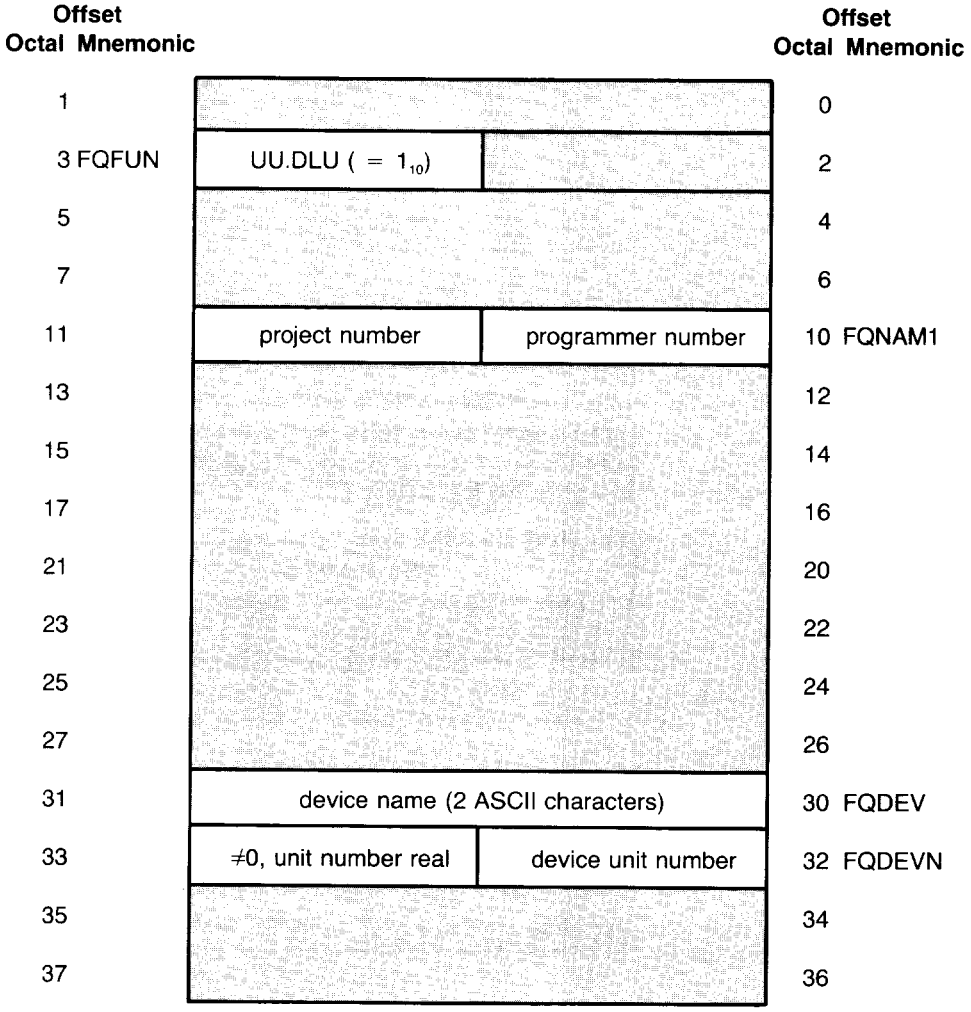
* System internal format for date is:

[(year –1970) * 1000_{10}] + day-within-year

3.32.17 UU.DLU (Delete User Account) – Privileged

Data Passed

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the UU.DLU subfunction.

Errors

INUSE The account contains files (has not been zeroed), or a user is currently logged in to the system under the account.

NOSUCH The specified account does not exist.

PRVIOL	Account specified is either [0,0] or [0,1]. If you are using a disk with the RDS0 (pre-version 8.0) file structure, you also get this error if you specify account [1,1]. (See the *RSTS/E System Manager's Guide* for more information about RSTS/E file structures.)
DEVNFS	Device is not disk or is a disk open in non-file-structured mode.

3.32.18 UU.DMP (Snap Shot Dump) – Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3	FQFUN	UU.DMP (= -27_{10})	2	
5			4	
7			6	
11			10	
13			12	
15			14	
17			16	
21			20	
23			22	
25			24	
27			26	
31			30	
33			32	
35			34	
37			36	

Data Returned

Except for a possible error in byte 0 of the FIRQB, no meaningful data is returned with the UU.DMP subfunction of .UUO.

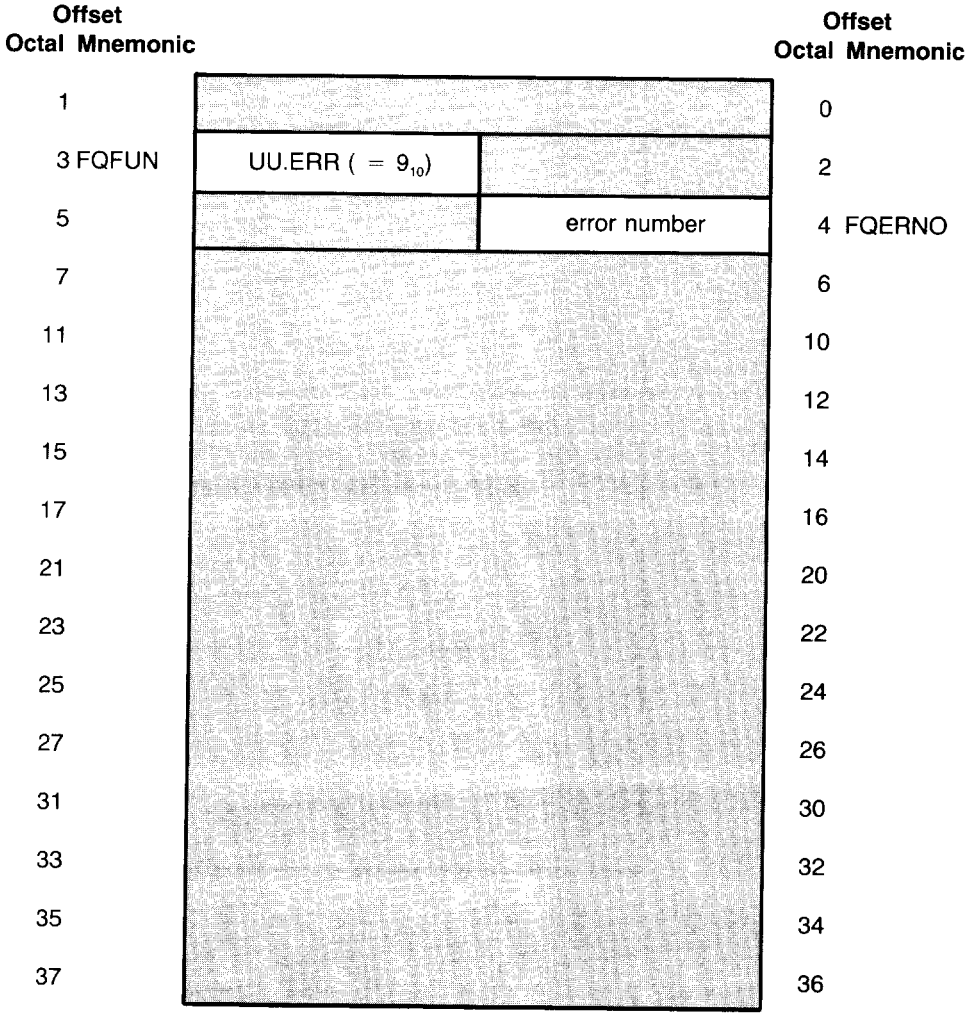
Errors

NOSUCH	The call tried to write data to the file CRASH.SYS, but crash dump was not enabled. To enable crash dump, the system manager must answer "yes" to the CRASH DUMP question during system initialization.
BADFUO	A nonprivileged user attempted to execute this call.
xxxxxx	The call also returns device-dependent errors such as HNGDEV or NOTMNT.

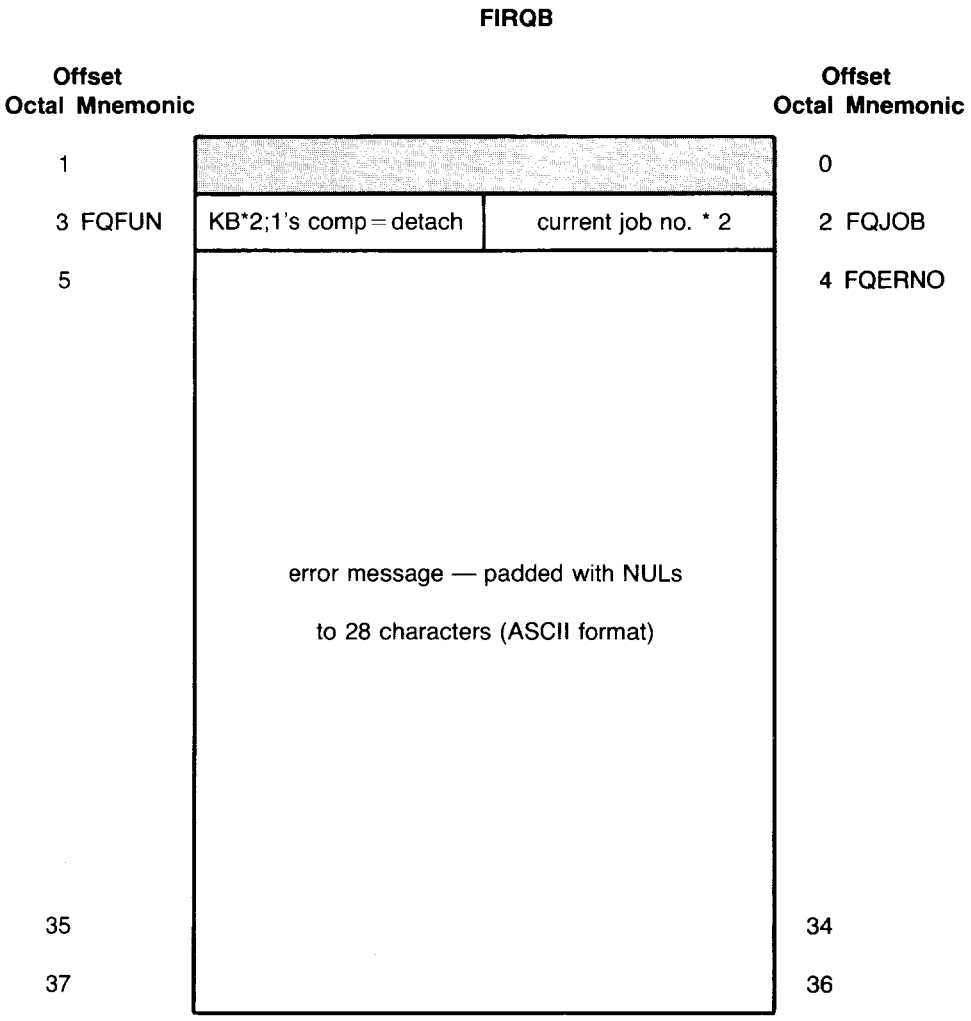
3.32.19 UU.ERR (Return Error Messages) – Not Privileged

Data Passed

FIRQB



Data Returned



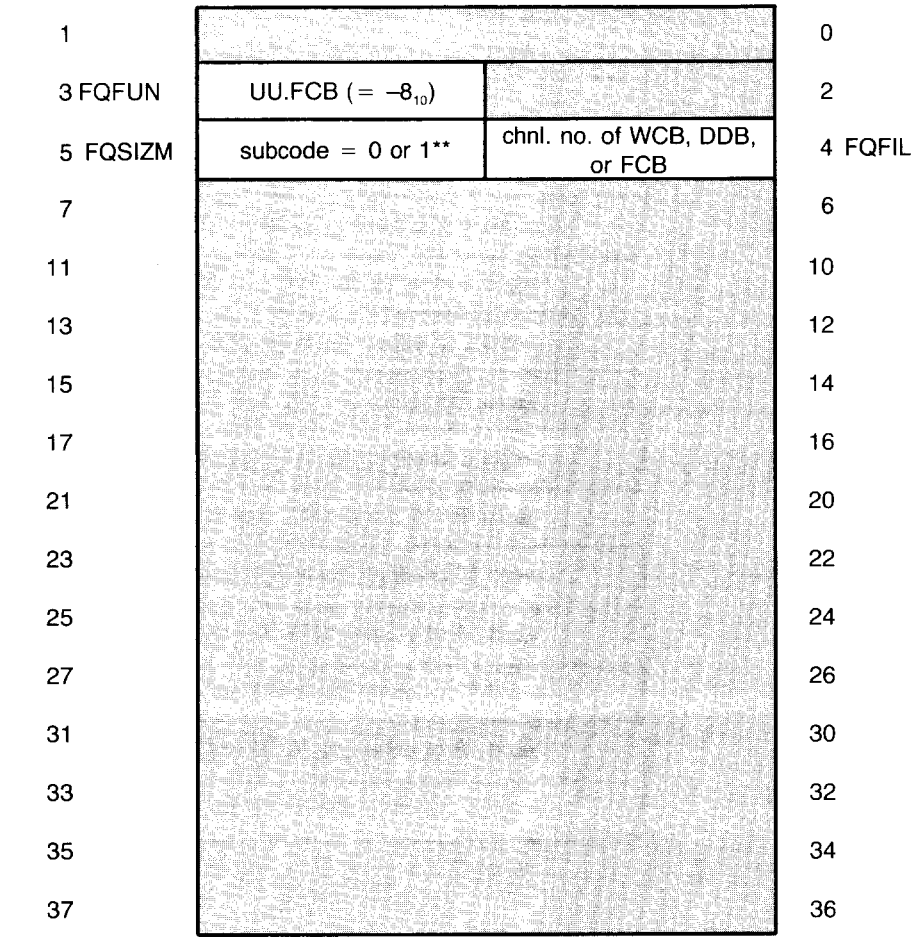
Errors

No errors are possible with the UU.ERR subfunction.

3.32.20 UU.FCB (Get Open Channel Statistics (WCB/DDB/FCB*)) – Not Privileged

Data Passed

FIRQB



---

* The WCB is the window control block for the file system. The DDB is the device data block for nondisk devices. The FCB is the file control block for the file system. The WCB, DDB, and FCB are internal RSTS/E structures that are subject to change at any time.

** 0 returns WCB or DDB information.
1 returns FCB information.

Data Returned (for subcode = 0)

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3		current job number * 2	2	FQJOB
5		word 1 of either the WCB or DDB	4	FQFIL
7		word 2 of either the WCB or DDB	6	FQPPN
11		word 3 of either the WCB or DDB	10	FQNAM1
13		word 4 of either the WCB or DDB	12	
15		word 5 of either the WCB or DDB	14	FQEXT
17		word 6 of either the WCB or DDB	16	FQSIZ
21		word 7 of either the WCB or DDB	20	FQBUFL
23		word 8 of either the WCB or DDB	22	FQMODE
25		word 9 of either the WCB or DDB	24	FQFLAG
27		word 10 of either the WCB or DDB	26	FQPFLG
31		word 11 of either the WCB or DDB	30	FQDEV
33		word 12 of either the WCB or DDB	32	FQDEVN
35		word 13 of either the WCB or DDB	34	FQCLUS
37		word 14 of either the WCB or DDB	36	FQNENT

Data Returned (for subcode = 1)

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job number * 2	2	FQJOB
5	FQSIZM	no. users with file open read regardless	no. users with file open other than read regardless	4	FQFIL
7		MSB of file size	Status byte*	6	FQPPN
11		LSB of file size		10	FQNAM1
13				12	
15				14	
17				16	
21				20	
23				22	
25				24	
27				26	
31				30	
33				32	
35				34	
37				36	

Errors

NODEVC You requested FCB information (FIRQB + FQSIZM = 1), but the file open on the I/O channel specified at FIRQB + FQFIL is not a disk file.

NOTOPN The I/O channel specified at FIRQB + FQFIL is not open.

BADFUO The subfunction code passed in FIRQB + FQSIZM is a value other than 0 or 1.

BSERR The channel number specified at FIRQB + FQFIL is outside the range 0 through 15_{10}.

* This byte contains the following internal flag information:

Bit Value	Meaning
002	File is placed.
004	Some job has write access now.
010	File is open in update mode.
020	File is contiguous; no extend available.
040	No delete or rename allowed.
100	File is a UFD.
200	File is marked for deletion.

3.32.21 UU.FIL (File Placement and Modification) – Privileged and Not Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.FIL ($=-26_{10}$)		2	
5	FQSIZM	function code**	channel number* (1 to 15_{10})	4	FQFIL
7		project number	programmer number	6	FQPPN
11		file name (2 words in RAD50 format)		10	FQNAM1
13				12	
15		file type (1 word RAD50)		14	FQEXT
17		least significant bits of block number***		16	FQSIZ
21		MSB of block number	placed/cache/seq.flags†	20	FQNAM2
23		new date of last access		22	FQMODE
25		new date of creation		24	FQFLAG
27		new time of creation		26	FQPFLG
31		device name (2 ASCII characters) must be disk		30	FQDEV
33		≠0, unit number real	device unit number	32	FQDEVN
35				34	
37				36	

* You can also place a 0 in this byte and specify the file by file specification (device, project-programmer number, file name, and type). The system ignores the values in bytes 6 through 15 and 30 through 33 if you use a nonzero channel number.

** function code:

= 1	Set or reset file's placed bit (cannot be used with code 10_8).
= 2	Modify code 20_8 to return 0 as device cluster number if file's placed bit is not set.
= 4	Change file's backup statistics (privileged only).
= 10_8	Change file's run-time system name field.
= 20_8	Return file's retrieval information.
= 40_8	Reset file's contiguous bit.
= 100_8	Enable/disable sequential mode caching if file is cached (cannot be used with code 10_8; privileged only).
= 200_8	Enable/disable data caching on the file (cannot be used with code 10_8; privileged only).

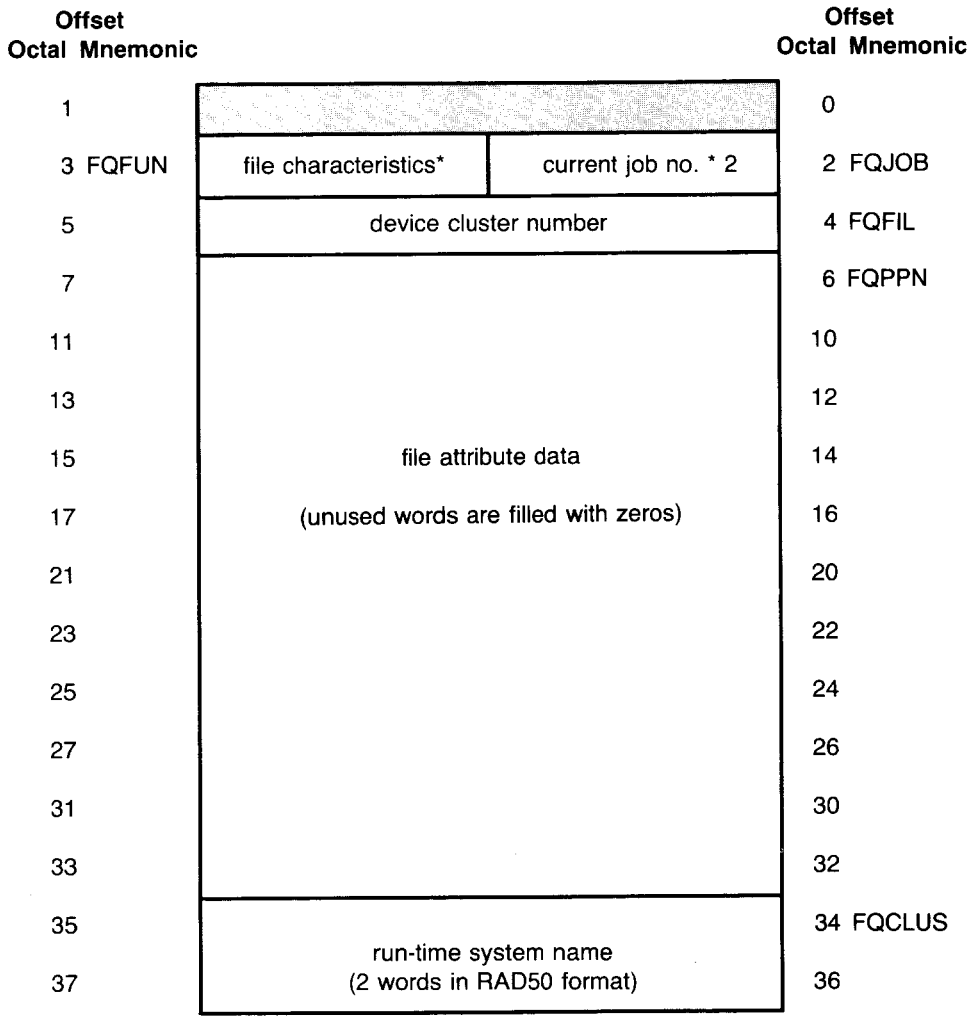
*** If bit 3 of FIRQB+FQSIZM is set, the two words beginning at FIRQB+FQSIZ are the new run-time system name in RAD50.

† flags

= 2	New value for placed bit if bit 1 of FIRQB+FQSIZM is set.
= 4	New value for sequential bit if bit 6 of FIRQB+FQSIZM is set.
= 200_8	New value for cached bit if bit 7 of FIRQB+FQSIZM is set.

Data Returned

FIRQB



Errors

NOSUCH	File or account is not present on disk.
NOTOPN	Channel is not open.
PRVIOL	File open is not a disk file or job lacks necessary privilege.
BADFUO	File open is not a disk file or is a user file directory.

---

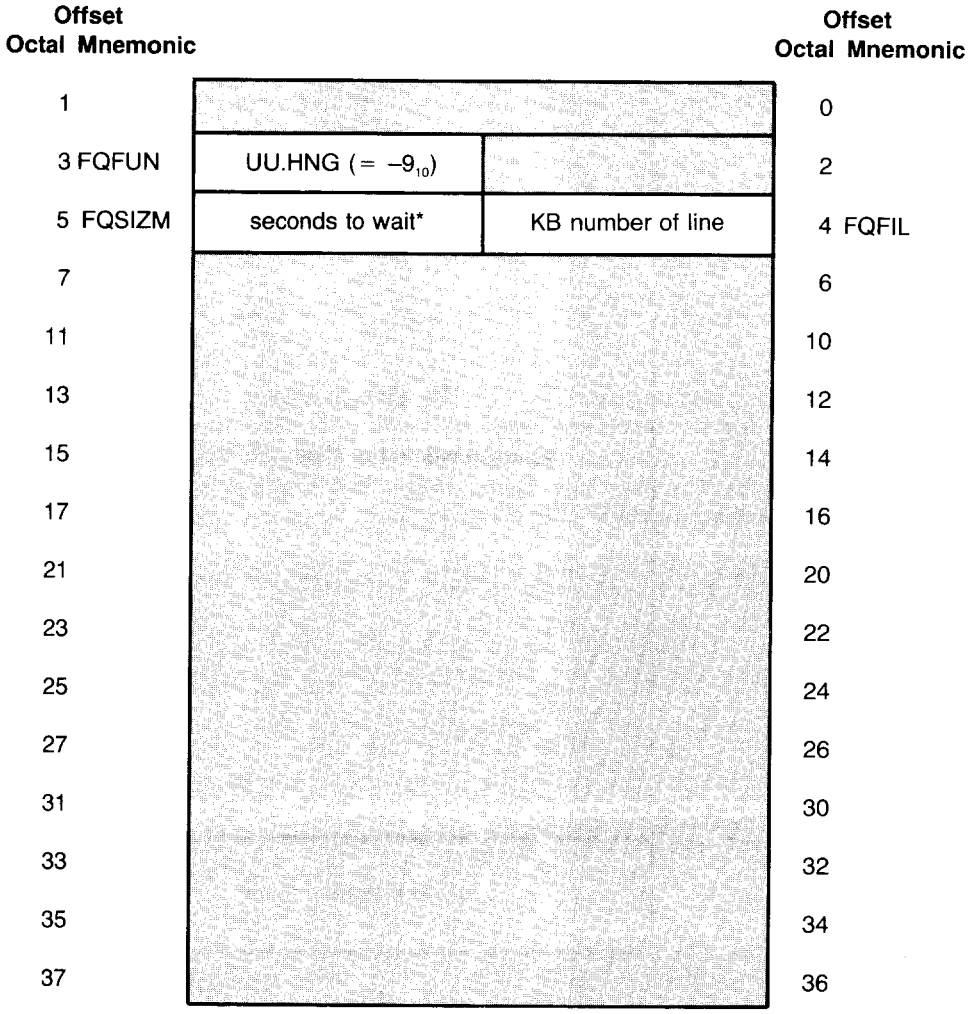
* file characteristics:

= 2	File is placed.
= 4	File will be cached sequentially, if at all.
= 20_8	File is contiguous.
= 200_8	File will be cached when open.

3.32.22 UU.HNG (Hang Up a Dataset) – Privileged

Data Passed

FIRQB



Data Returned

No data is returned with a UU.HNG subfunction; no errors are possible.

---

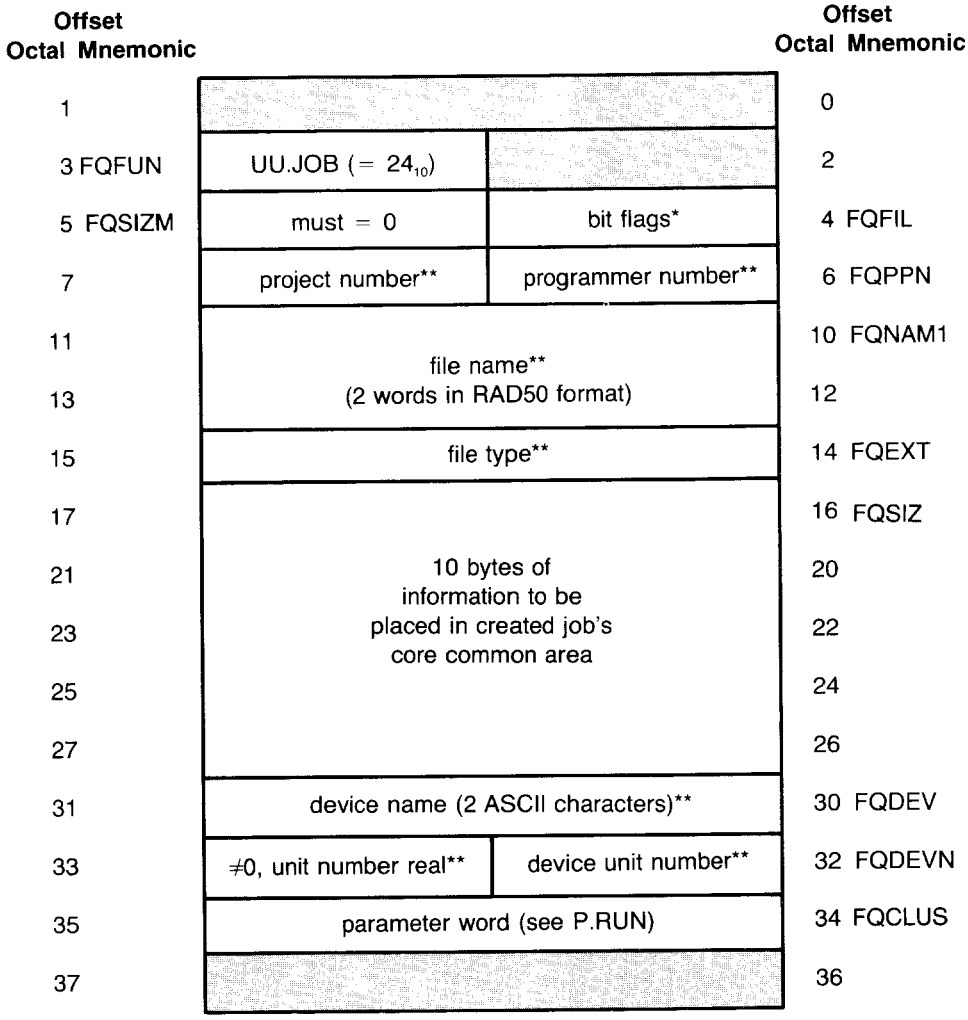
* Use one of the following values:

−1	Set "Data Terminal Ready" to permit a modem connected to a RSTS/E system to dial out. If a connection is not made in 30 seconds, perform an automatic hang-up of the dataset. (An optional patch is available to increase this time limit; see the *RSTS/E Maintenance Notebook.)*
0	Hang up in two seconds.
$1–127_{10}$	Hang up in one to 127 seconds.

3.32.23 UU.JOB (Create Job) – Privileged and Not Privileged

Data Passed — Logged-out job (privileged only)

FIRQB



---

* The following bit flag is valid for a logged-out job:

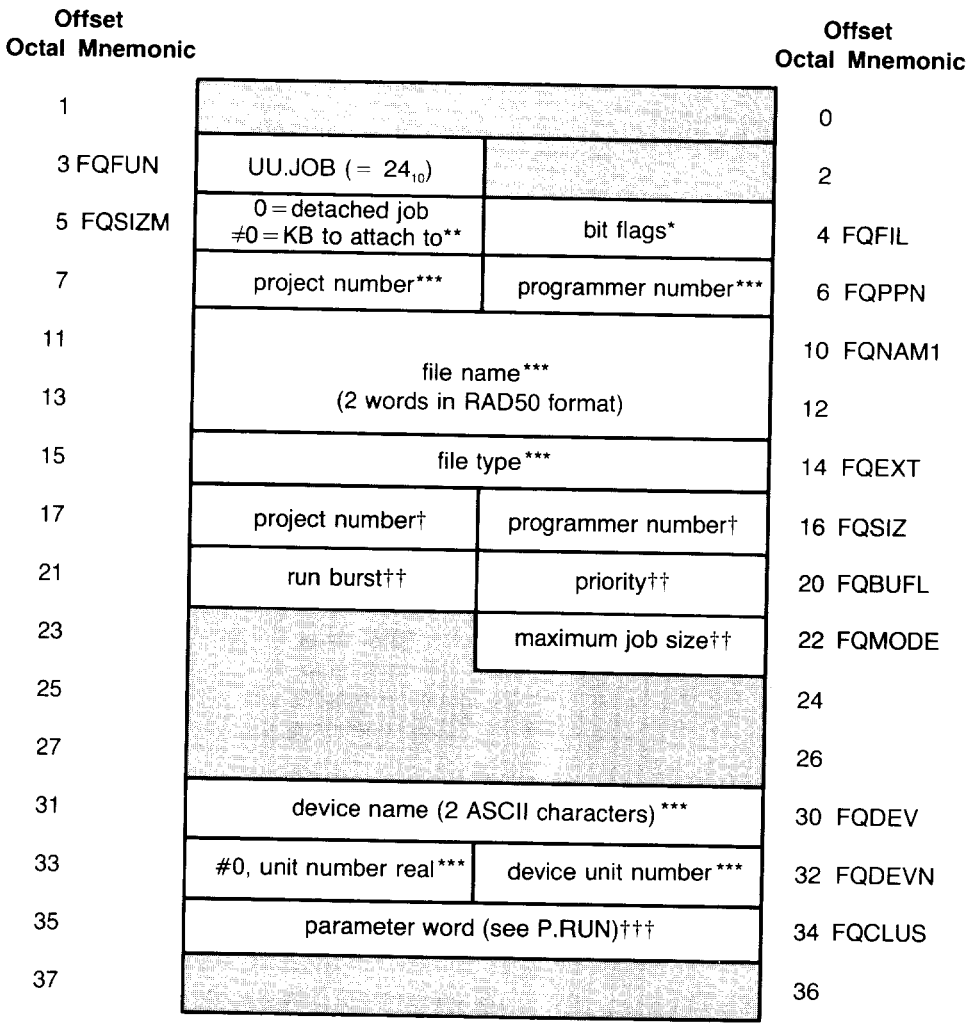
200_8 Create job even if logins are disabled. When this bit is not set, the job is created only if logins are enabled.

All other bits must be 0.

** File specification of program to run.

Data Passed — Logged-in job to run a program (privileged and nonprivileged)

FIRQB



NOTE

When you create a logged-in job, the system passes your current job's core common and user logicals to the new job, regardless of privilege. The system temporarily stores this data in XBUF.

* The following bit flags are valid when you create a logged-in job to run a program:

  40_8 Create job to run under account specified at FIRQB+FQSIZ (privileged users only).

  100_8 Create logged-in job. You must set this bit for a logged-in job. The new job will run under the caller's account unless you also set bit 40_8.

  200_8 Create job even if logins are disabled (privileged users only). When this bit is not set, the job is created only if logins are enabled.

  Set all bits you do not use to 0.

** Bits 0–6 indicate keyboard number to attach to. To specify KB0:, use 200_8.

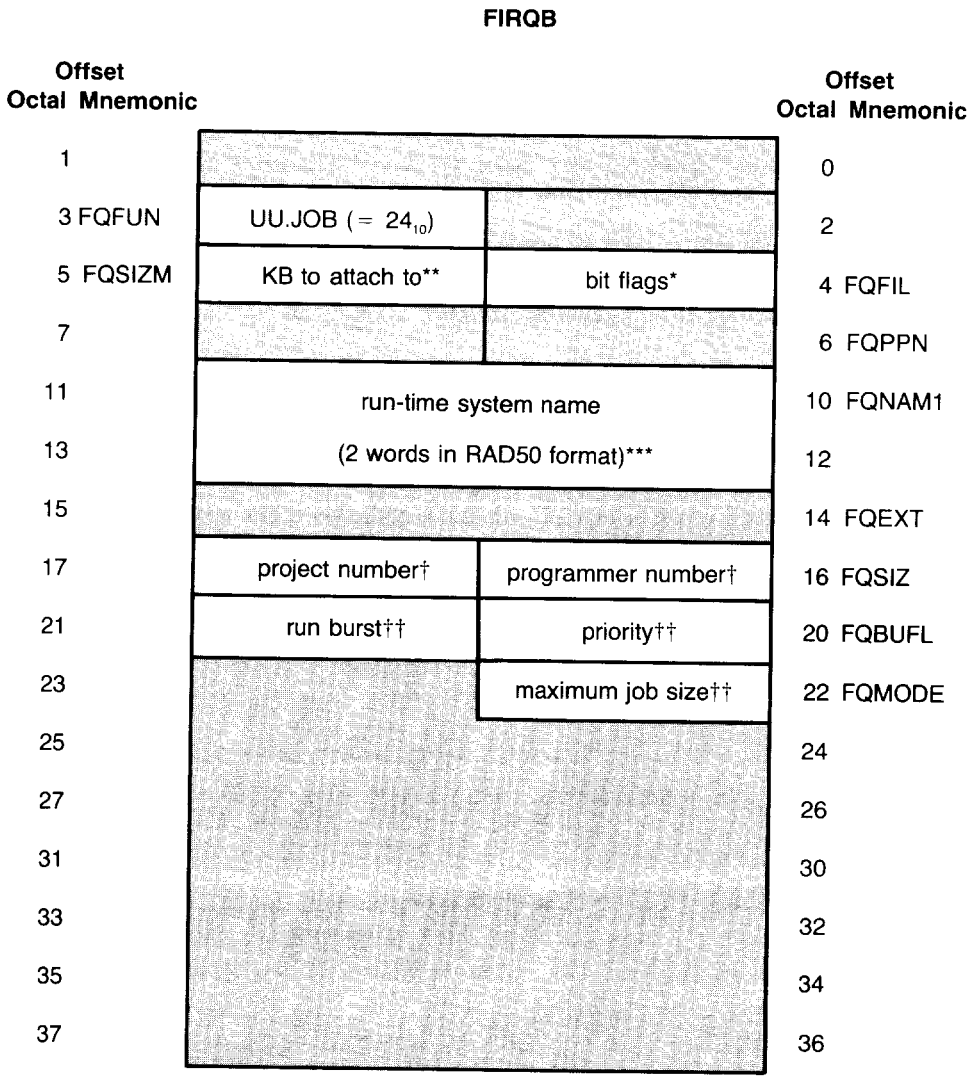
*** File specification of program to run.

† Account under which job will run (privileged users only; bit 40_8 at FIRQB+FQFIL must be set). If you are nonprivileged, set both bytes to 0.

†† Only privileged users can specify these values. If you specify 0, the system uses the caller's values. Use –1 to explicitly specify priority 0. If you are nonprivileged, set all three bytes to 0.

††† The system clears bit 15 in the parameter word if you are nonprivileged, as for a .RUN or .CHAIN. See the description of P.RUN in Section 2.5.4 for more information.

Data Passed — Logged-in job to enter a keyboard monitor at P.NEW (privileged and nonprivileged)



NOTE

When you create a logged-in job, the system passes your current job's core common and user logicals to the new job, regardless of privilege. The system temporarily stores this data in XBUF.

---

* The following bit flags are valid when you create a logged-in job to enter a keyboard monitor:

  20_8 Enter keyboard monitor instead of running a program. (You must set this bit.)

  40_8 Create job to run under account specified at FIRQB + FQSIZ (privileged users only).

  100_8 Create logged-in job. (You must set this bit.) The new job will run under the caller's account unless you also set bit 40_8.

  200_8 Create job even if logins are disabled (privileged users only). When this bit is not set, the job is created only if logins are enabled.

  Set all bits you do not use to 0.

** Bits 0–6 indicate keyboard number to attach to. To specify KB0:, use 200_8.

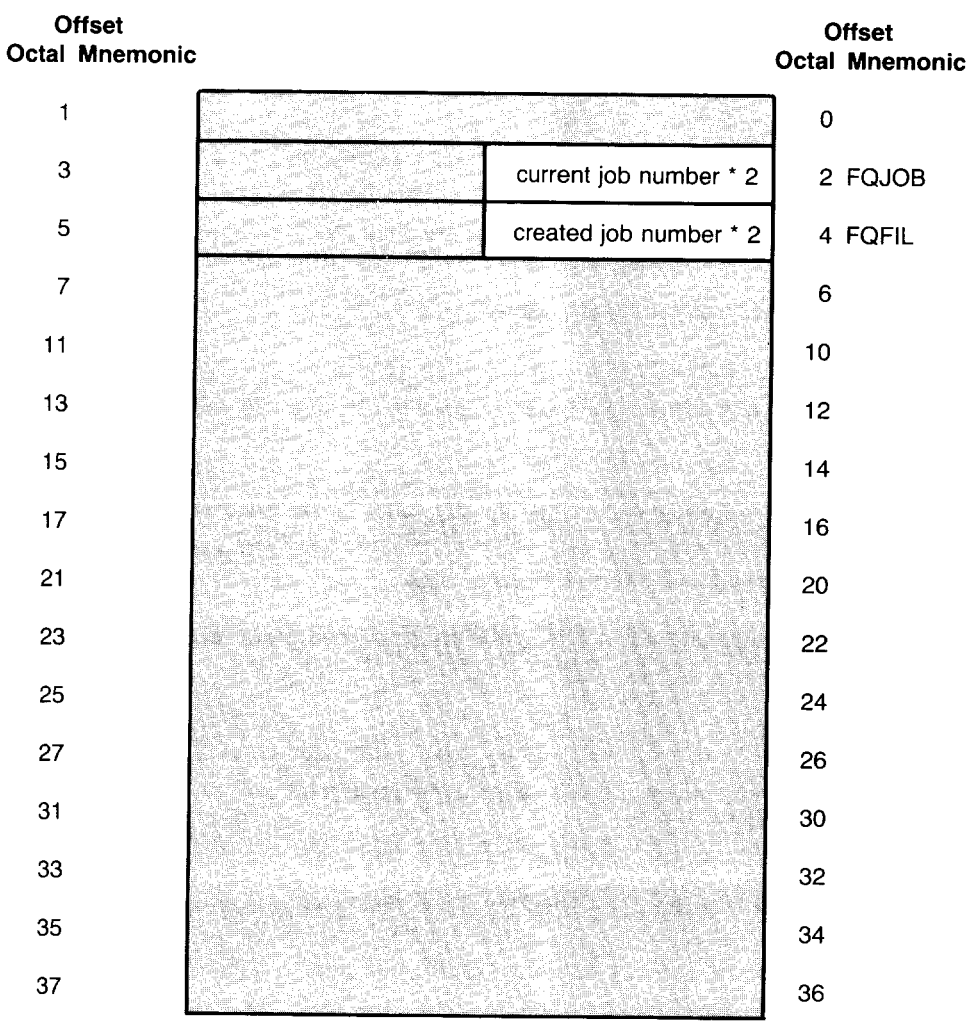
*** The run-time system you specify must be installed and must be a keyboard monitor. Specify 0 for the system's default keyboard monitor.

† Account under which job will run (privileged users only; bit 40_8 at FIRQB + FQFIL must be set). If you are nonprivileged, set both bytes to 0.

†† Only privileged users can specify these values. If you specify 0, the system uses the caller's values. Use –1 to explicitly specify priority 0. If you are nonprivileged, set all three bytes to 0.

Data Returned (all types of jobs)

FIRQB



Errors

NOROOM Job cannot be created. Probable causes:

1. Logins are disabled and bit 7 (value = 200) at FIRQB + FQFIL is clear.
2. The system's job or swap slots are currently full.

PRVIOL You are nonprivileged and tried to:

1. Create a job when logins were disabled.
2. Create a logged-out job.
3. Create a job to run under an account other than the current account.
4. Create a detached job that would cause you to exceed your detached job quota. (This quota is set by your system manager; its default value is 0.)

NOSUCH You are privileged and are trying to create a logged-in job, but the system cannot log the job in. Possible causes are that you specified a nonexistent account or an account that cannot be logged in to (password is ??????).

BADFUO You specified a keyboard monitor at FIRQB + FQNAM1, but did not supply a keyboard number at FIRQB + FQSIZM.

NODEVC The keyboard number at FIRQB + FQSIZM is invalid.

NOTAVL The keyboard specified at FIRQB + FQSIZM is open, is in use, or is not assigned to the calling job. A nonprivileged user can also get this error if the system manager has restricted the device to privileged users.

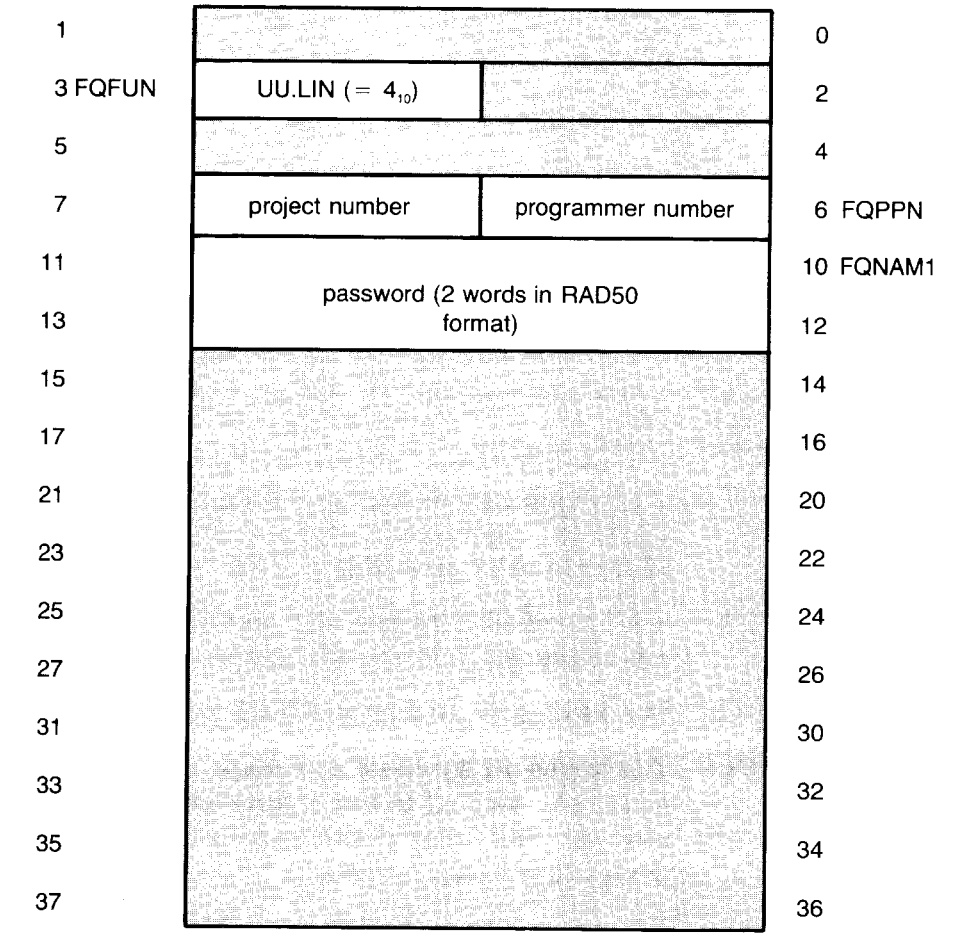
NOBUFS You are trying to create a logged-in job, but not enough XBUF is available for temporary storage of your current job's core common and user logicals.

Note that when you create a logged-in job, you can also get any error that can occur for the UU.LIN (Login), .RUN, and .RTS directives, because the monitor executes these directives for the new job. If an error occurs, the monitor kills the new job and returns the error to your job instead.

3.32.24 UU.LIN (Login) – Privileged

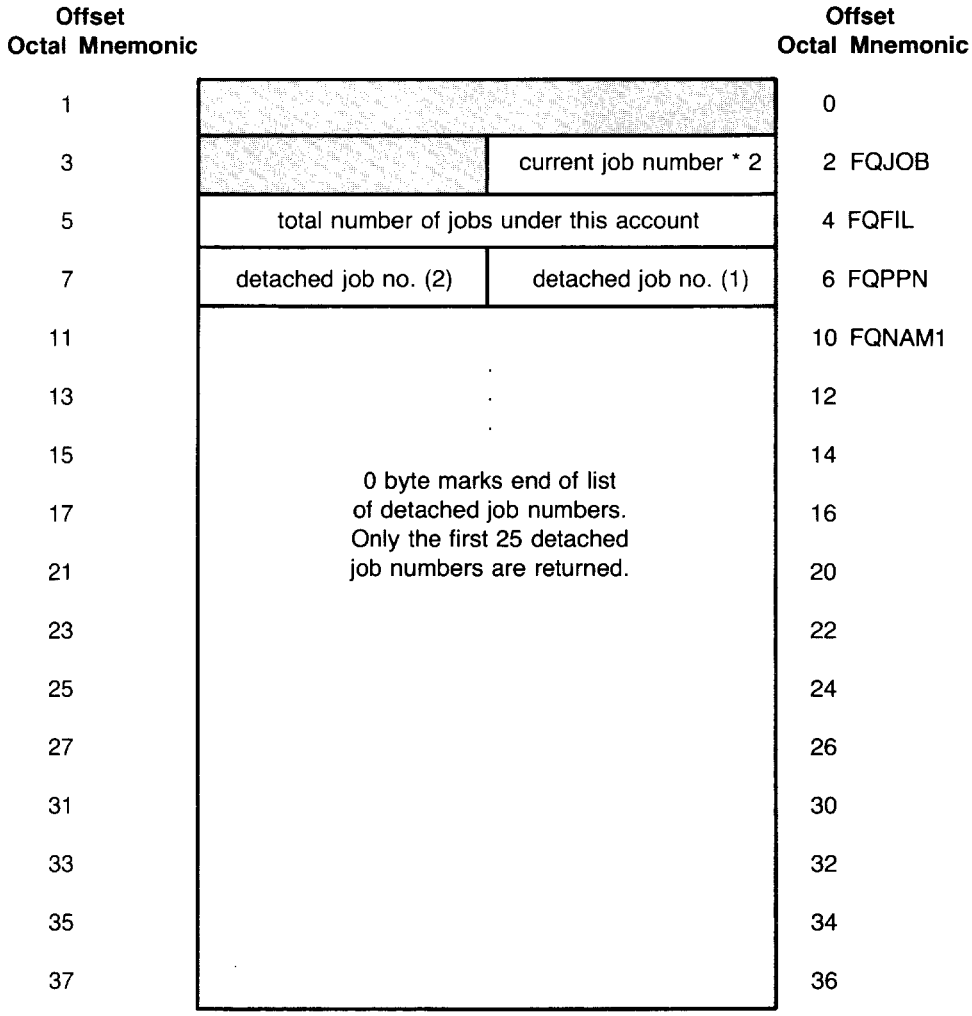
Data Passed

FIRQB



Data Returned

FIRQB



Errors

NOSUCH

1. The project-programmer number specified at FIRQB+FQPPN is [0,1] or it does not exist.
2. The password specified at FIRQB+FQNAM1 does not match the password of the account on the system or contains nonalphanumeric characters.

3.32.25 UU.LOG (Set Number of Logins) – Privileged

Data Passed

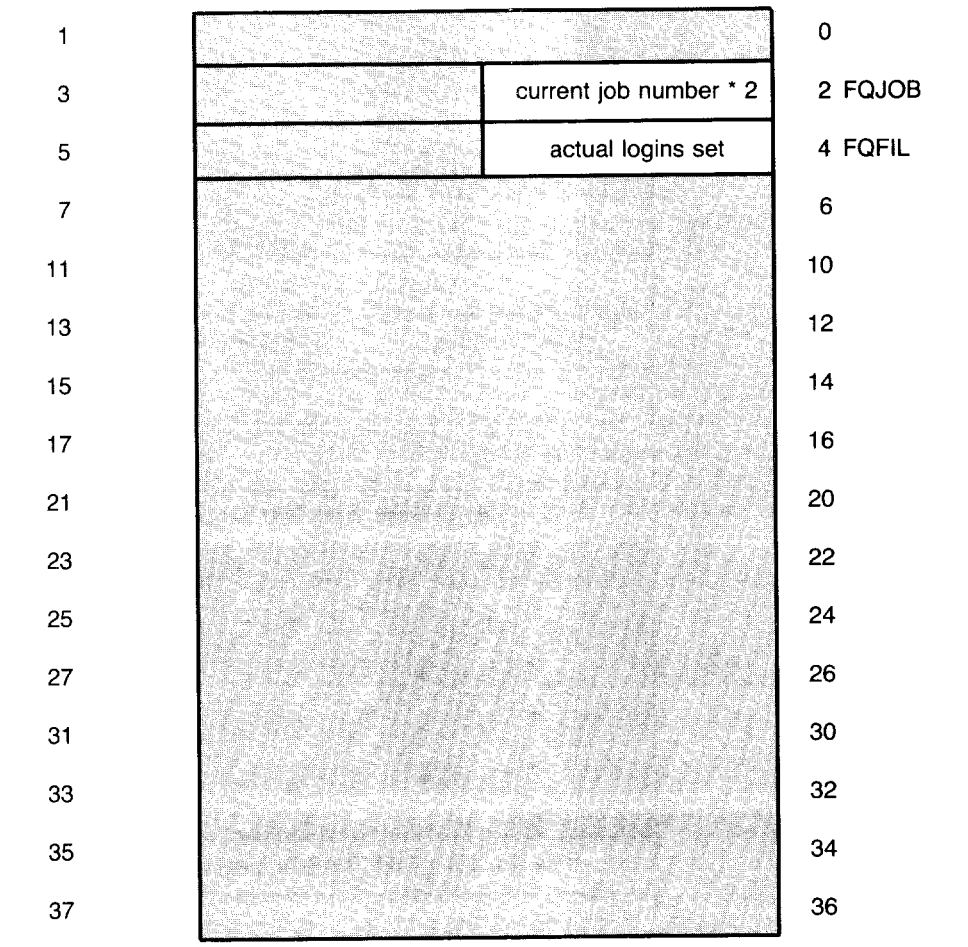
FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.LOG (= -19_{10})		2	
5			number of allowed jobs*	4	FQFIL
7				6	
11				10	
13				12	
15				14	
17				16	
21				20	
23				22	
25				24	
27				26	
31				30	
33				32	
35				34	
37				36	

* A value of 0 sets the number of allowed jobs to 1. The upper limit for the number of logins is either the system JOB MAX or the number of jobs that can currently be swapped, whichever is lower. If you specify a larger value, the system sets the number of logins to the upper limit. You do not receive an error.

Data Returned

FIRQB



Errors

No errors are possible with the UU.LOG subfunction.

3.32.26 UU.LOK (Disk Directory Lookup by File Name/Wildcard Lookup) – Privileged and Not Privileged

Data Passed — Look up by File Name

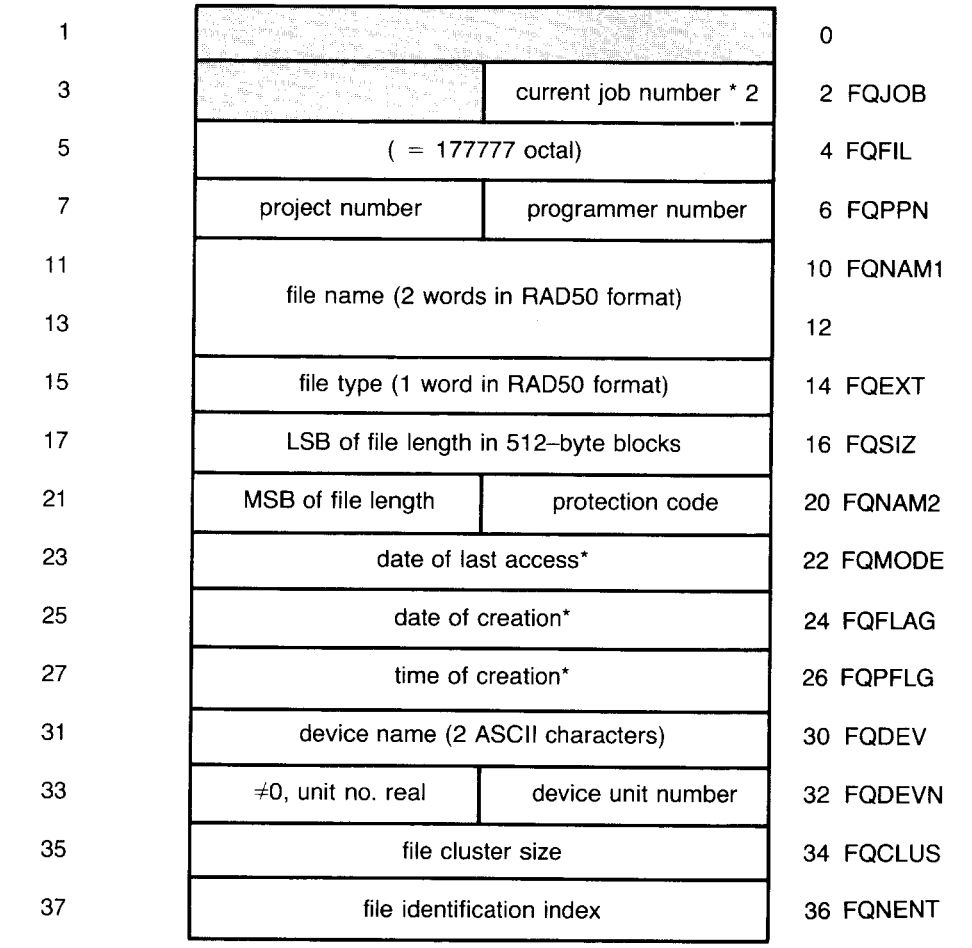
FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3	FQFUN	UU.LOK (= 17_{10})	2	
5		(must = 177777 octal)	4	FQFIL
7		project number* / programmer number*	6	FQPPN
11		file name (2 words in RAD50 format)	10	FQNAM1
13			12	
15		file type (1 word in RAD50 format)	14	FQEXT
17			16	
21			20	
23			22	
25			24	
27			26	
31		device name (disk) (2 ASCII characters)	30	FQDEV
33		≠0, unit number real / device unit number	32	FQDEVN
35			34	
37			36	

* 0 in both bytes means the current account.

Data Returned — Look up by File Name

FIRQB



Errors

BADNAM The file name at FIRQB+FQNAM1 is missing.

NOSUCH Device specified at FIRBQ+FQDEV is not a disk, or the file specified does not exist on the disk. This error also occurs when a nonprivileged user does not have read access to the specified file.

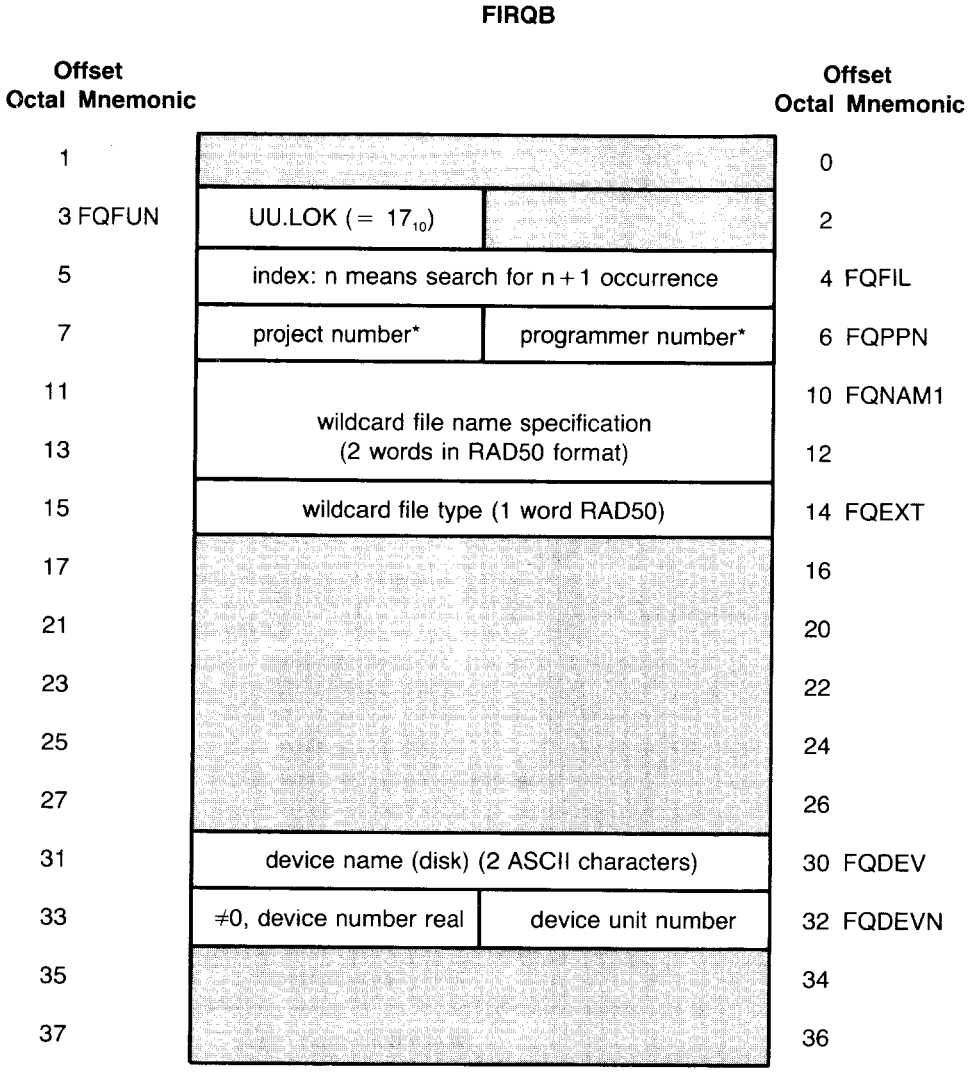
---

* System internal format for dates is:

[(year –1970) * 1000_{10})] + day-within-year

Time is in minutes until midnight, where 1440 equals midnight. See the .DATE directive for a discussion of these formats.

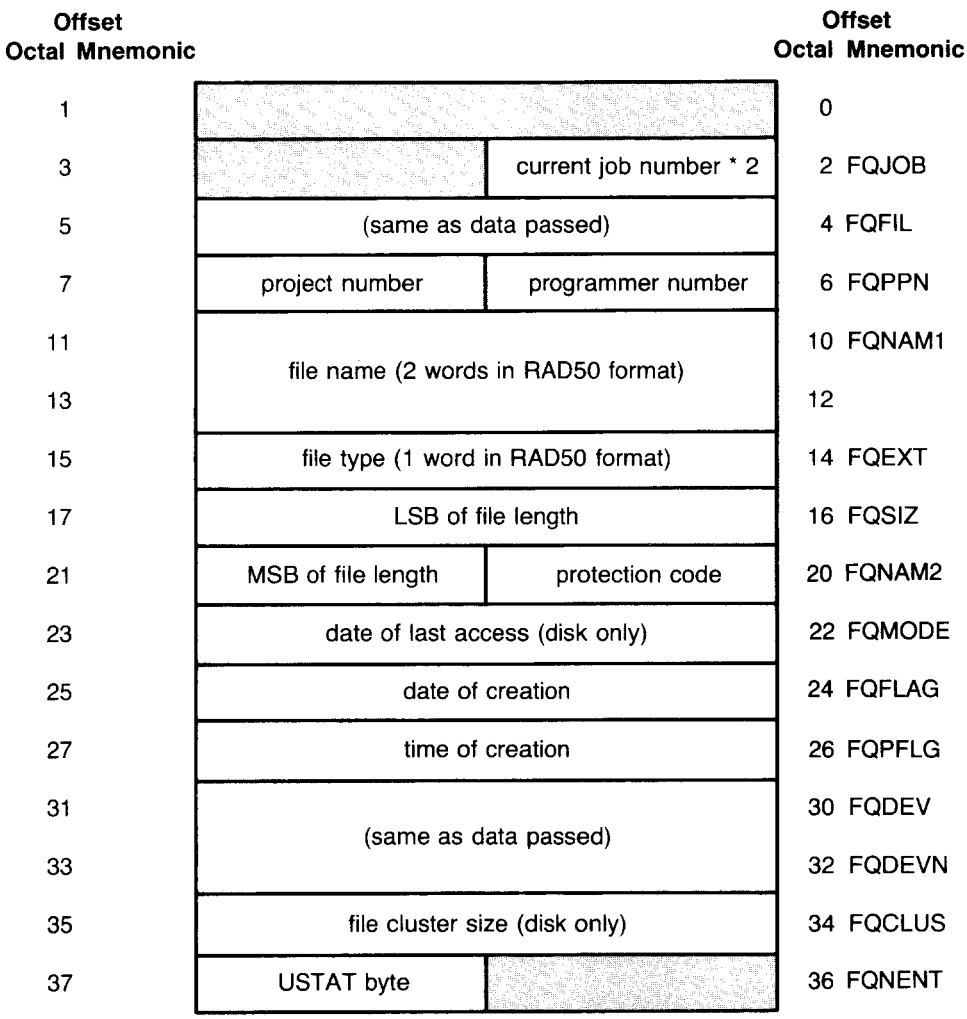
Data Passed — Disk Wildcard Directory Lookup



* 0 in both bytes means the current account.

Data Returned — Disk Wildcard Directory Lookup

FIRQB



Errors

BADNAM — No file name appears at FIRQB+FQNAM1.

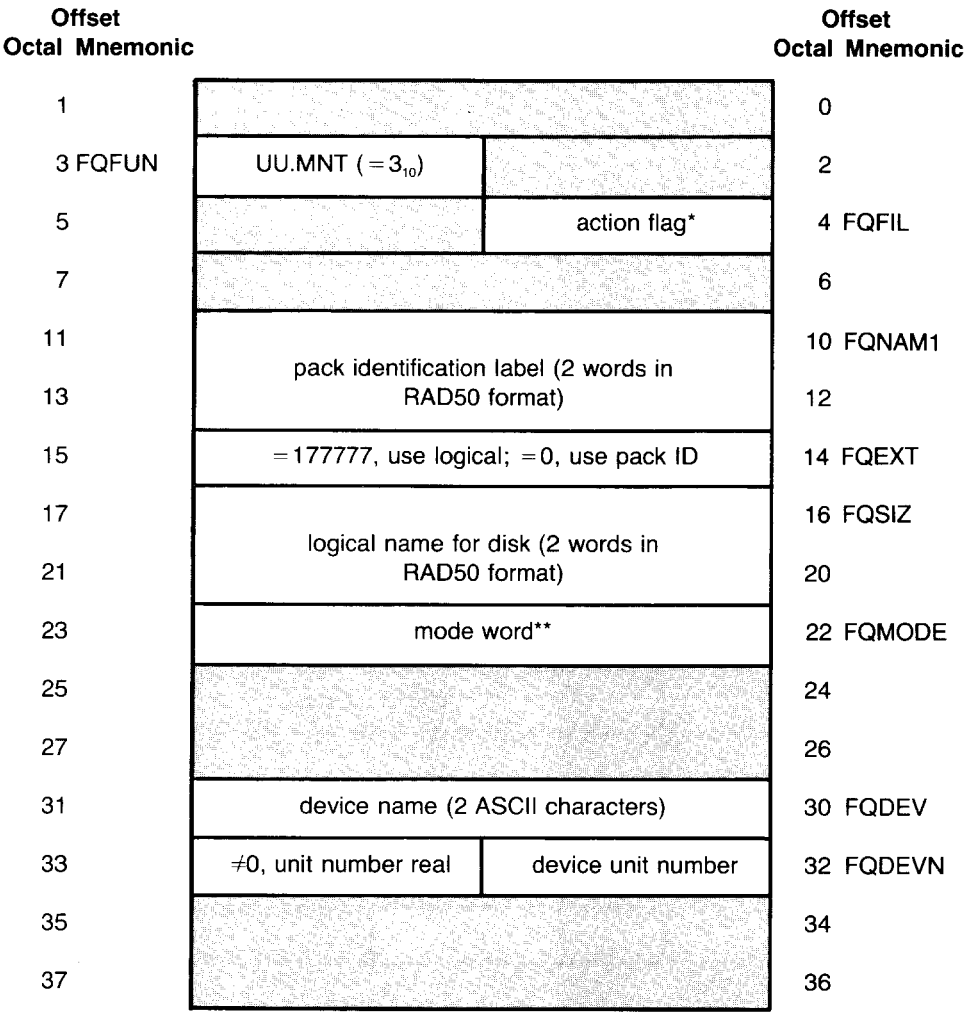
NOSUCH — Device specified is not a disk or no file corresponds to the index given.

PAKLCK — The disk is locked and the account under which the call is executed is not privileged.

3.32.27 UU.MNT (Disk Pack Status) – Privileged

Data Passed

FIRQB



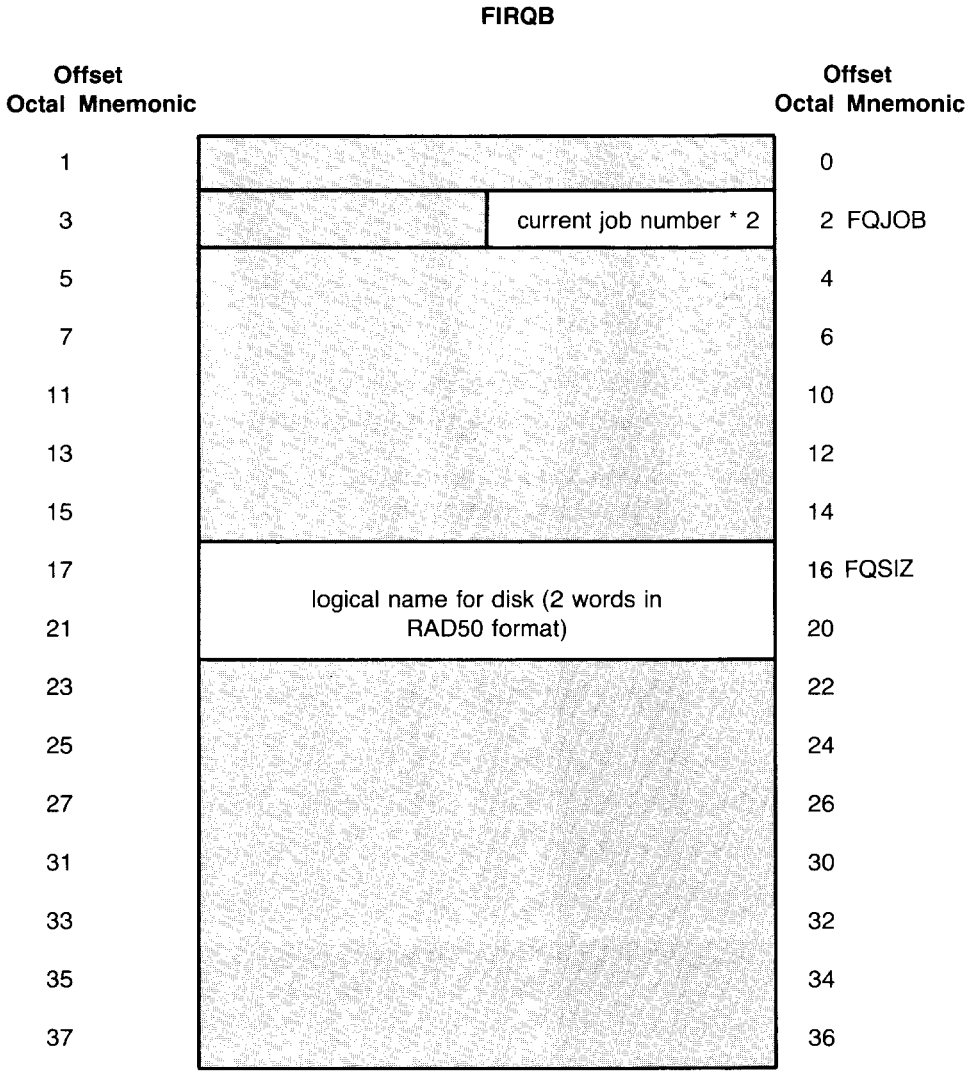
---

* byte 4 = 0 Mount a disk pack or cartridge.
= 2 Dismount a disk pack or cartridge.
= 4 Lock out a disk pack or cartridge.
= 6 Unlock a disk pack or cartridge.

** mode word = 2000_8 Mount with lookup of pack ID.
4000_8 Mount disk for use by single user (/NOSHARE).
10000_8 Mount disk read-write, even if disk was initialized as read only.
20000_8 Mount disk read-only.
40000_8 Mount as a private disk (/PRIVATE).

You can specify one value or combine the values. You must also set the sign bit in the mode word when you use any of these values.

Data Returned



Errors

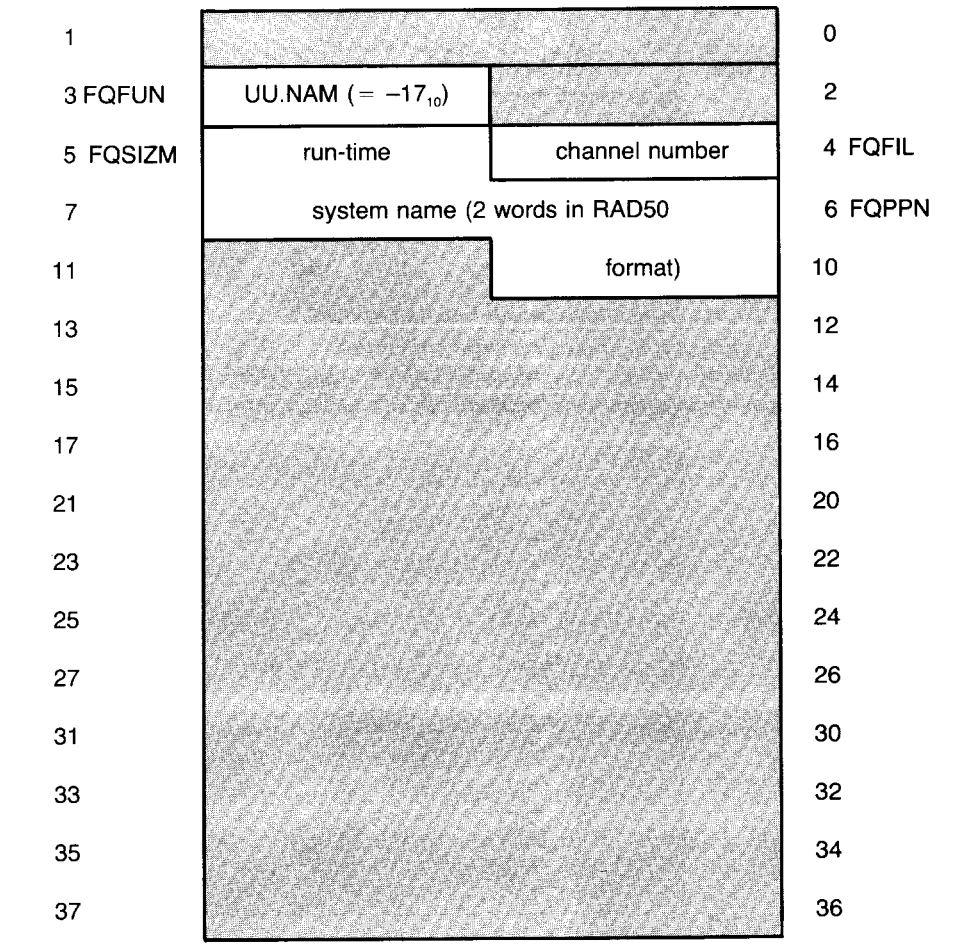
INUSE	Attempted to dismount a disk with an open file.
NODEVC	Device type or unit at FIRQB + FQDEV is not on the system.
PRVIOL	Tried to mount a disk that does not contain the RSTS/E file structure. Use the INITIALIZE command, the online DSKINT program, or the DSKINT initialization option to initialize the disk.
HNGDEV	Tried to mount a disk that is not write-enabled.
BADFUO	Tried to mount a disk that is already mounted or that resides in a nondismounted drive, or drive specified is the system disk.

WRGPAK	Tried to mount a disk with an incorrect pack label.
NOTMNT	Tried to lock, unlock, or dismount a disk that is not mounted.
INTPAK	The storage allocation table on the disk needs to be rebuilt because the disk was not properly dismounted when it was last used. Before using the disk, use the ONLCLN program to rebuild the storage allocation table. Note that when this error occurs, the mount succeeds, but the disk is always mounted read-only with the "dirty" bit set. (You can see if this bit is set by looking at the SYSTAT program's disk status report. SYSTAT reports that the disk is "dirty.")
BADPAK	Disk is beyond recovery. For example, the cluster size is larger than 16 or the storage allocation table is unreadable.
DEVNFS	Tried to lock, unlock, or dismount a disk currently open in non-file-structured mode.

3.32.28 UU.NAM (Associate a Run-Time System with a File) – Not Privileged

Data Passed

FIRQB



Data Returned

Other than a possible error in byte 0 of the FIRQB, no data is returned by the UU.NAM subfunction.

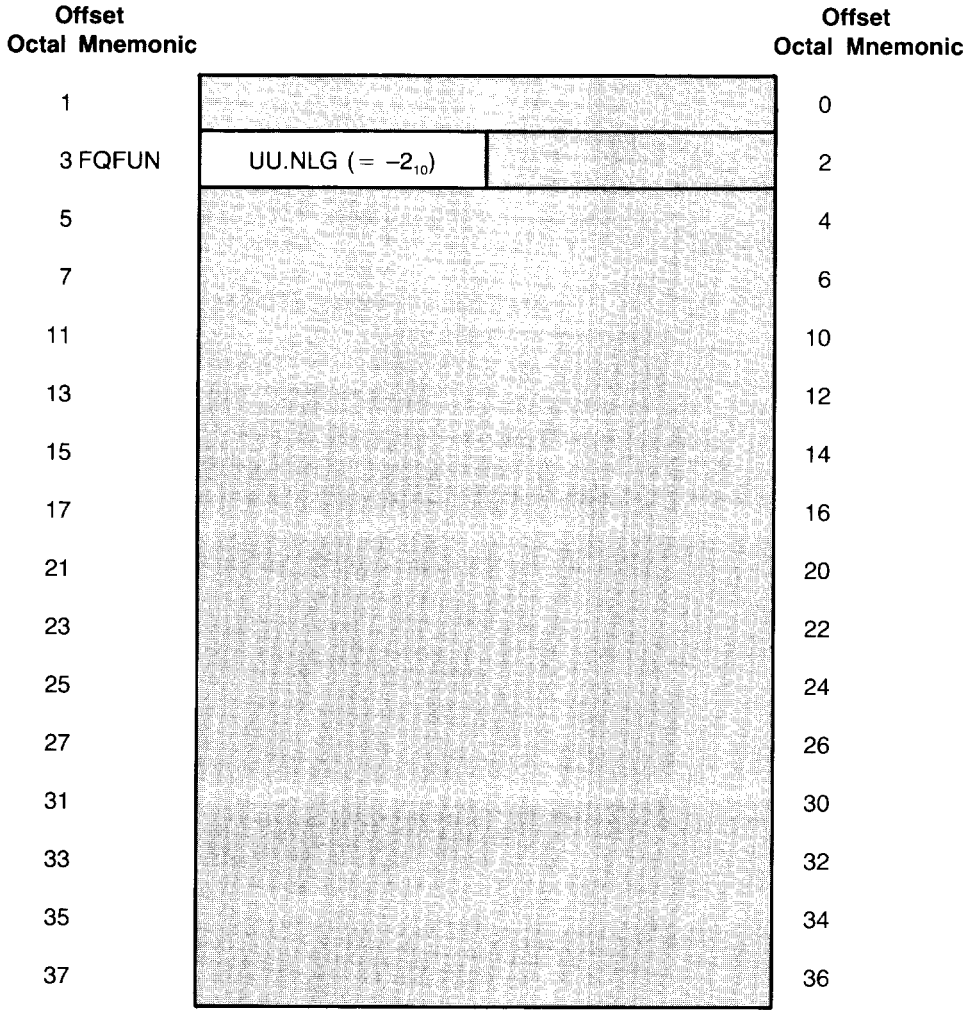
Errors

NOTOPN	Channel is not open.
PRVIOL	File open on channel is not a disk file or the job executing the call does not have write access.

3.32.29 UU.NLG (Disable Further Logins) – Privileged

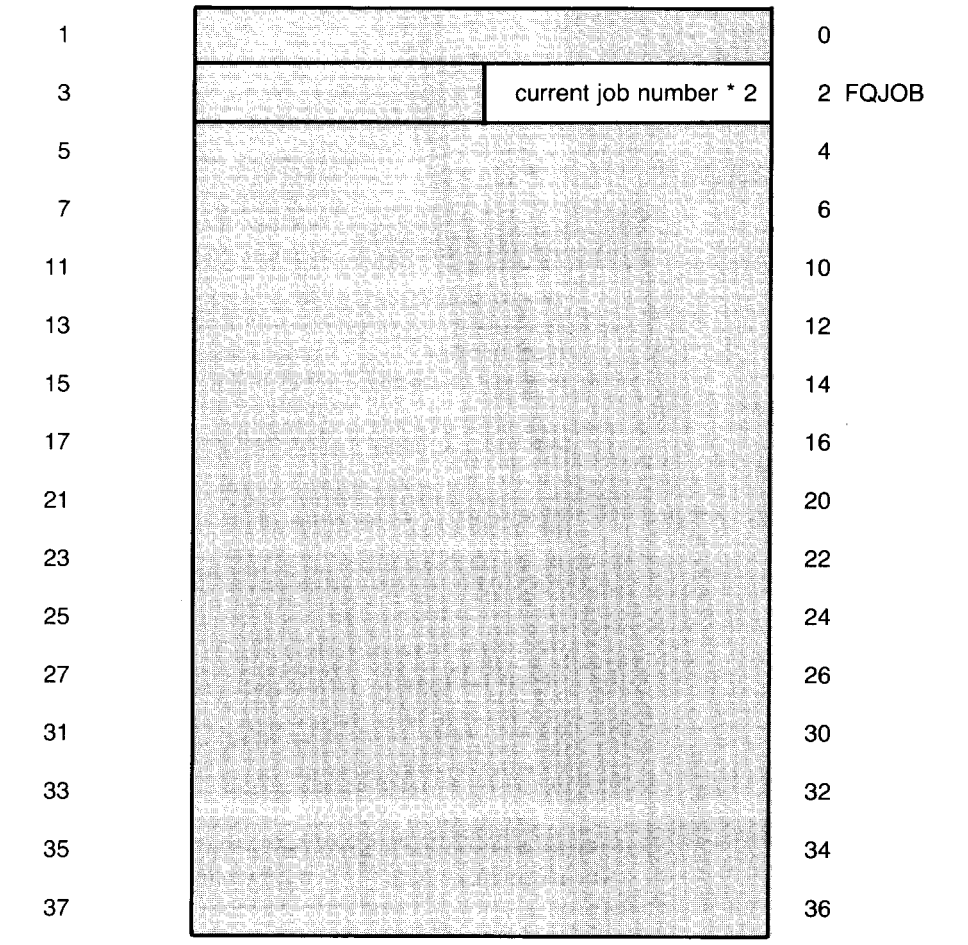
Data Passed

FIRQB



Data Returned

FIRQB



Errors

No errors are possible with the UU.NLG subfunction.

3.32.30 UU.PAS (Create User Account) – Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.PAS ($=0_{10}$)		2	
5			number of clusters to preextend UFD*	4	FQFIL
7		device cluster number for UFD or –1**		6	FQPPN
11		project number	programmer number	10	FQNAM1
13		password (2 words in RAD50 format)***		12	
15				14	
17		disk quota (maximum number of blocks)†		16	FQSIZ
21				20	
23				22	
25				24	
27				26	
31		device name (2 ASCII characters)		30	FQDEV
33		≠0, unit number real	device unit number	32	FQDEVN
35		UFD cluster size (0 = use pack cluster size)		34	FQCLUS
37				36	

* 0 = preextend one cluster
1–7 = preextend specified number of clusters

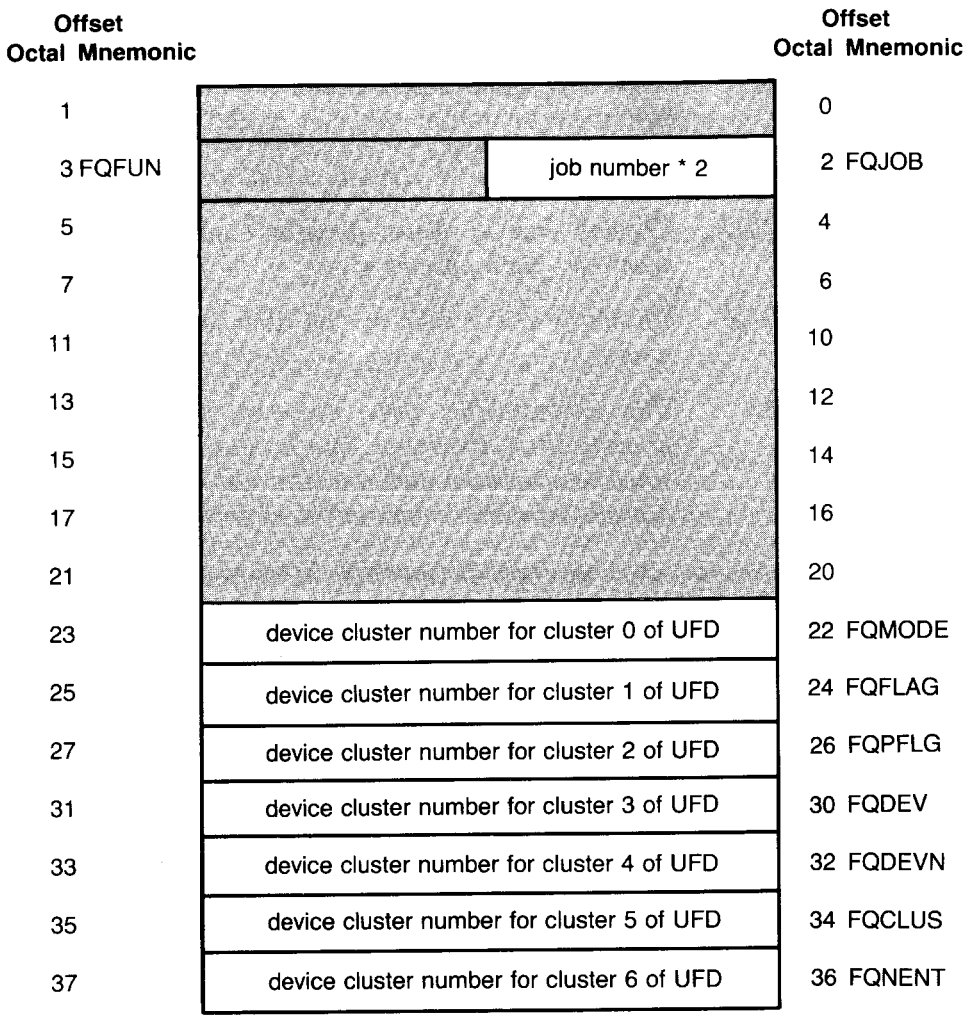
** –1 means place UFD in the middle of the disk.

*** Specify 0 to create an account that no one can log in to (same as password "??????").

† Use 0 for an unlimited quota.

Data Returned

FIRQB



Errors

NOROOM The monitor cannot allocate one cluster for the UFD you are creating because the disk is full.

PRVIOL Project-programmer number is [0,0], or either the project number or programmer number is 255.

FIEXST Account specified already exists on device.

BADCLU Cluster size is either greater than 16 or is nonzero and less than the pack cluster size.

DEVNFS Device is not a disk or the disk is open in non-file-structured mode.

ABORT The account has been entered, and the directory has been preextended. However, the account has not been given a password or quota because an internal consistency check has failed. Submit an SPR if you get this error, and include:

1. A copy of the disk. (Use the SAVE/RESTORE IMAGE function to make the copy.)
2. A snap shot dump. (Use the UTILTY SNAP command to copy the contents of memory to CRASH.SYS, then run the ANALYS program to save the information. Submit a listing that contains the ANALYS program's output.)

See the *RSTS/E System Manager's Guide* for more information on SAVE/RESTORE, UTILTY, and ANALYS.

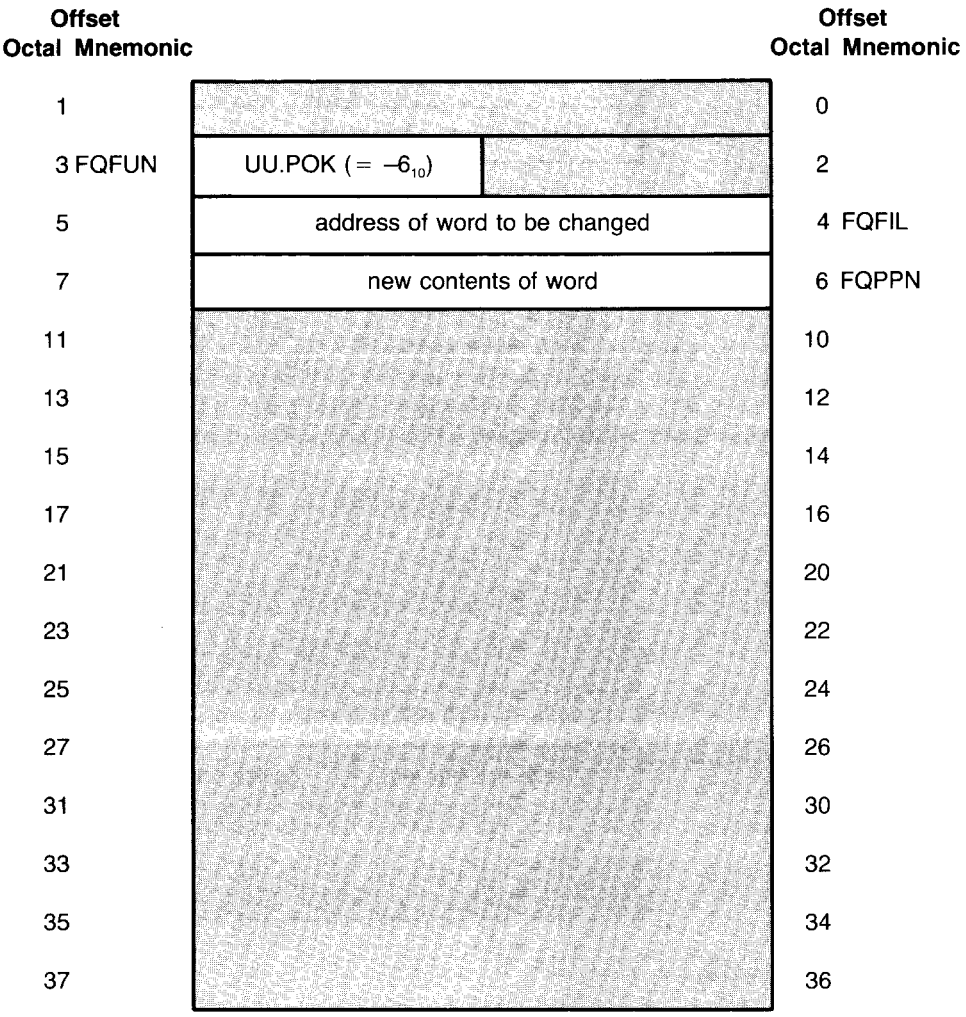
BADCNT
1. The number of clusters specified at FIRQB+FQFIL is less than 0 or greater than 7.
2. The number of clusters specified at FIRQB+FQFIL plus the device cluster number specified at FIRQB+FQPPN is greater than the disk size. (See Appendix B for disk size information.)

3.32.31 UU.POK (Poke Memory) – Privileged

The UU.POK subfunction can be executed only by a job running on account [1,1].

Data Passed

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the UU.POK subfunction.

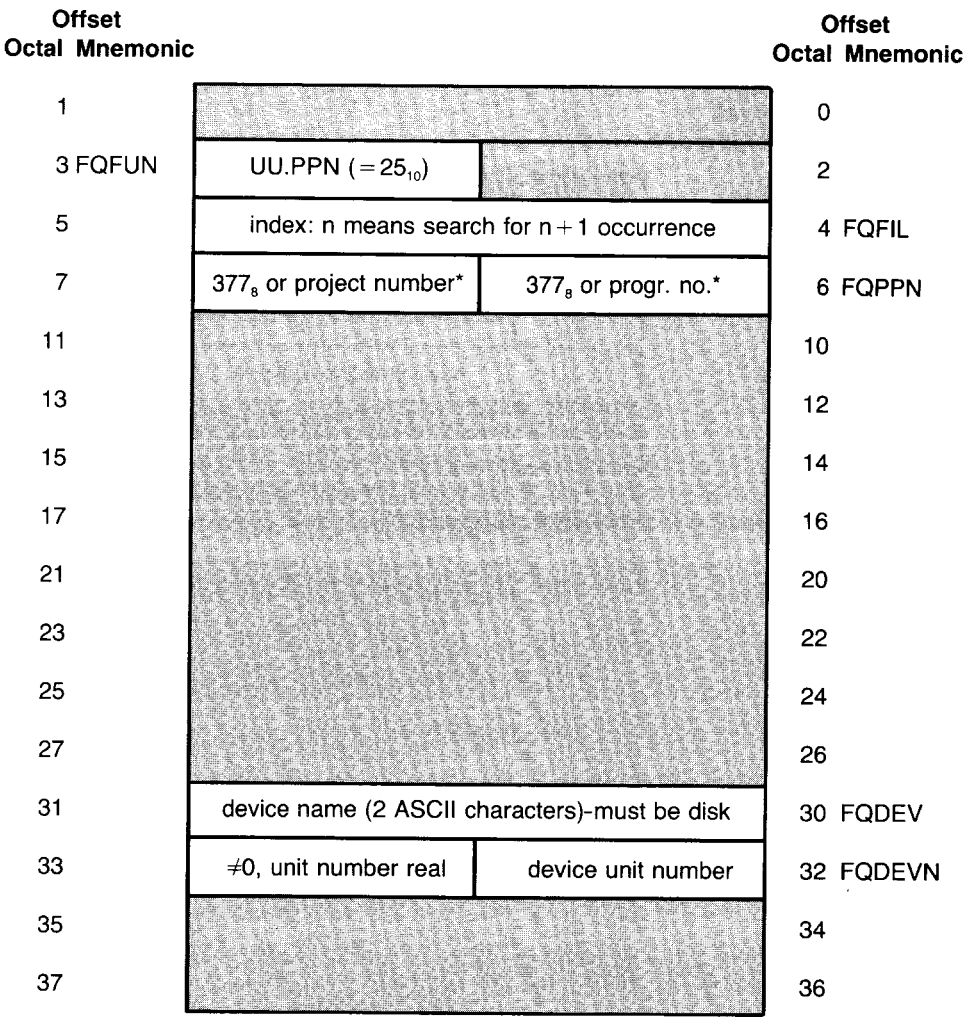
Errors

PRVIOL The job executing the call is not operating under account [1,1] or the address specified is odd.

3.32.32 UU.PPN (Wildcard PPN Lookup) – Not Privileged

Data Passed

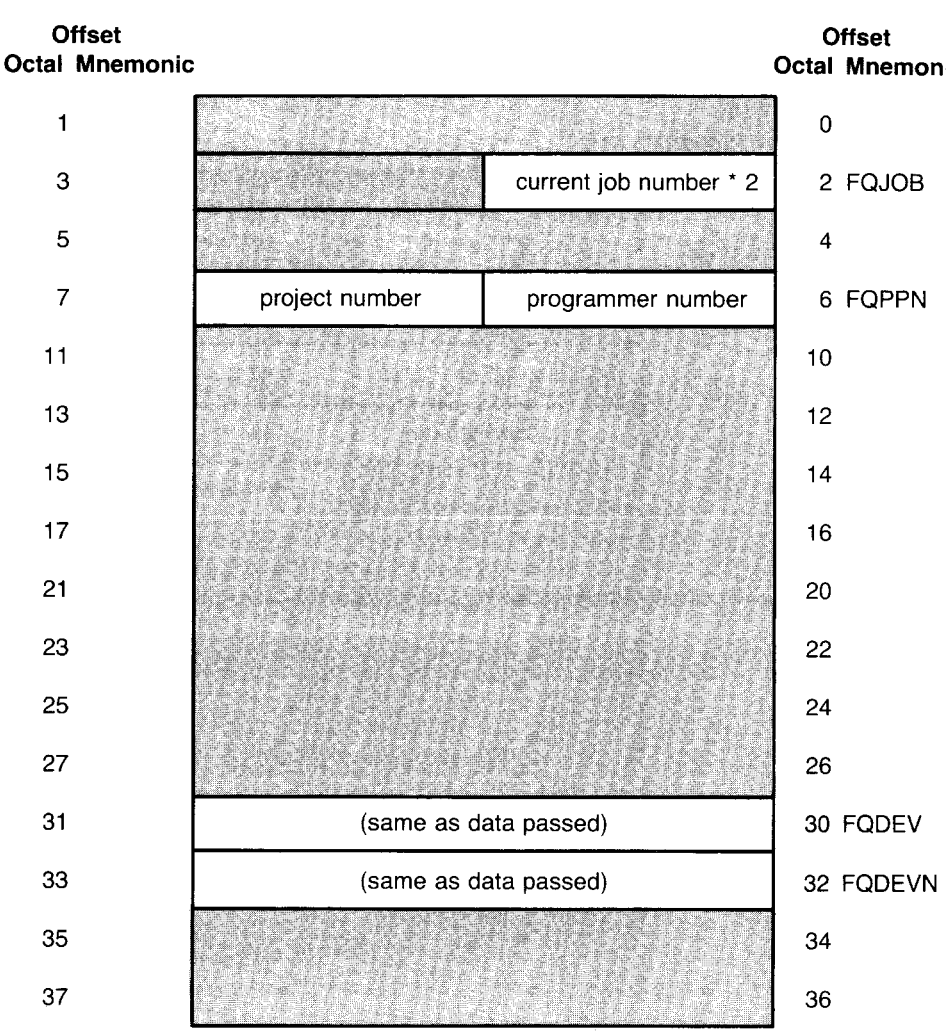
FIRQB



* 377_8 represents a wildcard. Use 0 in both bytes to indicate the current account.

Data Returned

FIRQB



Errors

NOSUCH Device is not a disk, or no match exists for the specified index.

PAKLCK The disk is locked out. Execute the call under a privileged account to override this.

3.32.33 UU.PRI (Change Priority/Run Burst/Job Size) – Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.PRI ($= -13_{10}$)		2	
5	FQSIZM	≠0, change run priority	job number; 377 = caller	4	FQFIL
7		≠0, change run burst	new run priority*	6	FQPPN
11		≠0, change size	new run burst**	10	FQNAM1
13			maximum size***	12	
15				14	
17				16	
21				20	
23				22	
25				24	
27				26	
31				30	
33				32	
35				34	
37				36	

* Run priority values range from -128_{10} to $+120_{10}$ in steps of 8. The value -128_{10} suspends the job.

** The run burst should be a value from 1 to 127_{10}. If you use a value outside this range, the monitor sets the run burst to 6.

*** The maximum size can range from 1 to 255_{10} (1024–word units). The system uses SWAP MAX if you exceed that value.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned with UU.PRI.

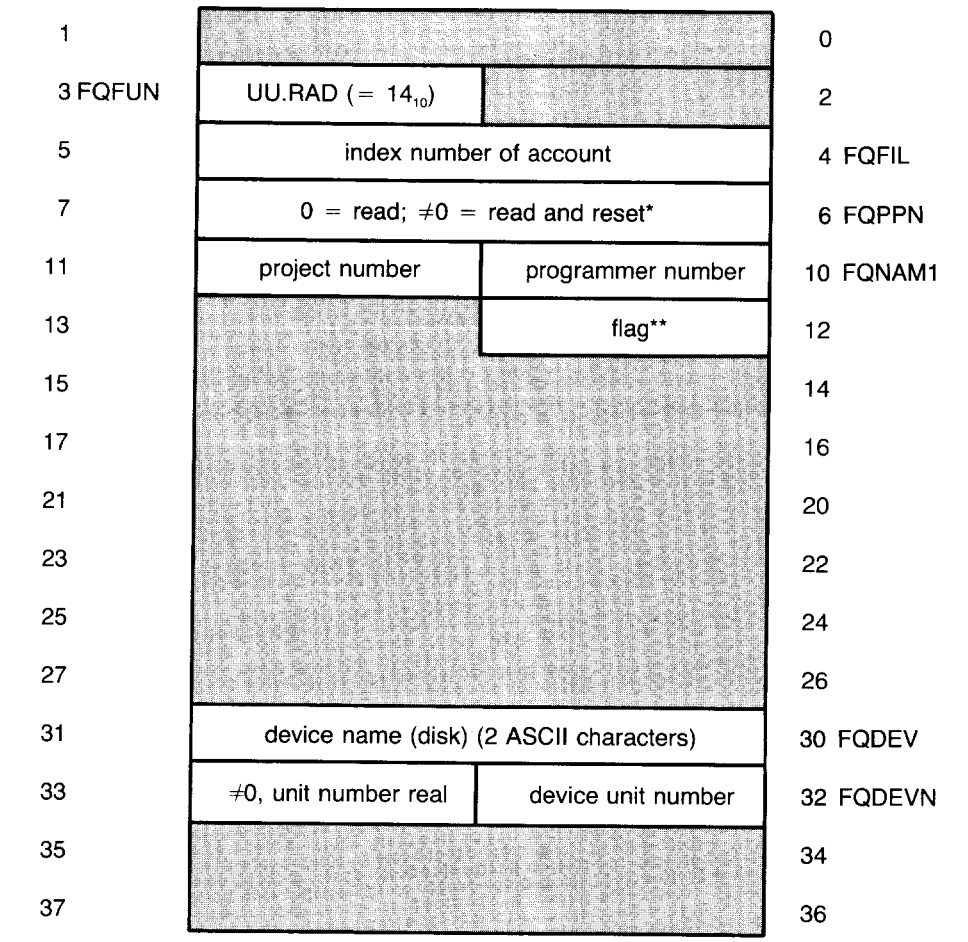
Errors

BADFUO	The job number specified at FIRQB + FQFIL does not exist.

3.32.34 UU.RAD (Read or Read-and-Reset Accounting Data) – Privileged and Not Privileged

Data Passed

FIRQB



---

* The system ignores this word for a nonprivileged caller and performs a read operation only.

** Bit 0 = 0 Implies the call returns blocks.
= 1 Implies no data. Note that setting this bit to 1 can save considerable execution time if you do not need the disk usage information otherwise returned to the word at FIRQB + FQPPN.
Bit 1 = 0 Project-programmer number at FIRQB + FQNAM1 corresponds to a real account, if index value at FIRQB + FQFIL is 0. (If index value is not 0, the system uses the index value and ignores any ppn at FIRQB + FQNAM1.)
= 1 Project-programmer number at FIRQB + FQNAM1 contains a wildcard (377_8).

Data Returned

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3		current job number * 2	2	FQJOB
5		(same as data passed)	4	FQFIL
7		number of blocks owned by account read*	6	FQPPN
11		ppn of account read	10	FQNAM1
13		password of account read (2 words in RAD50 format) if caller privileged	12	
15			14	FQEXT
17		low-order 16 bits of CPU time (.1 secs)	16	FQSIZ
21		connect time (minutes)	20	FQBUFL
23		low-order 16 bits of KCTs	22	FQMODE
25		device time (minutes)	24	FQFLAG
27	FQPROT	MSB of CPU time / MSB of KCTs	26	FQPFLG
31		(same as data passed)	30	FQDEV
33		(same as data passed)	32	FQDEVN
35		disk quota in blocks; 0 = unlimited	34	FQCLUS
37		user file directory cluster size	36	FQNENT

Errors

NOSUCH The project-programmer number specified does not exist on the disk, or the index specified is greater than the number of accounts on the disk.

BADFUO Device specified is not a disk.

---

* This word is zero if the byte at FIRQB + 12 in the data passed is 1.

3.32.35 UU.RTS (Add/Remove/Load/Unload Run-Time System or Resident Library) – Privileged

Data Passed — Add a Run-Time System

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.RTS (= -18_{10})		2	
5			flag*	4	FQFIL
7		project number	programmer number	6	FQPPN
11		run-time system name (2 words of RAD50)		10	FQNAM1
13				12	
15		=n, n*1K starts load; =0, monitor decides		14	FQEXT
17		max. user job image size, K words (P.SIZE)		16	FQSIZ
21		min. user job image size, K words (P.MSIZ)		20	FQBUFL
23		stay flag**	linked-list position	22	FQMODE
25		flag word (P.FLAG)		24	FQFLAG
27	FQPROT	dflt. exec. file type (P.DEXT), RAD50		26	FQPFLG
31		device name (disk) where stored		30	FQDEV
33		≠0, unit number real	device unit number	32	FQDEVN
35				34	
37				36	

NOTE

The word at FIRQB+FQEXT must be nonzero for a read/write run-time system.

---

* flag = 0 Values for P.SIZE, P.MSIZ, P.FLAG, and P.DEXT symbols are in the call.
= 200_8 Obtain values for these symbols from the .RTS file.

** stay flag = 200_8 Run-time system is permanently resident.
= 0 Memory occupied by this run-time system can be released when usage count goes to 0.

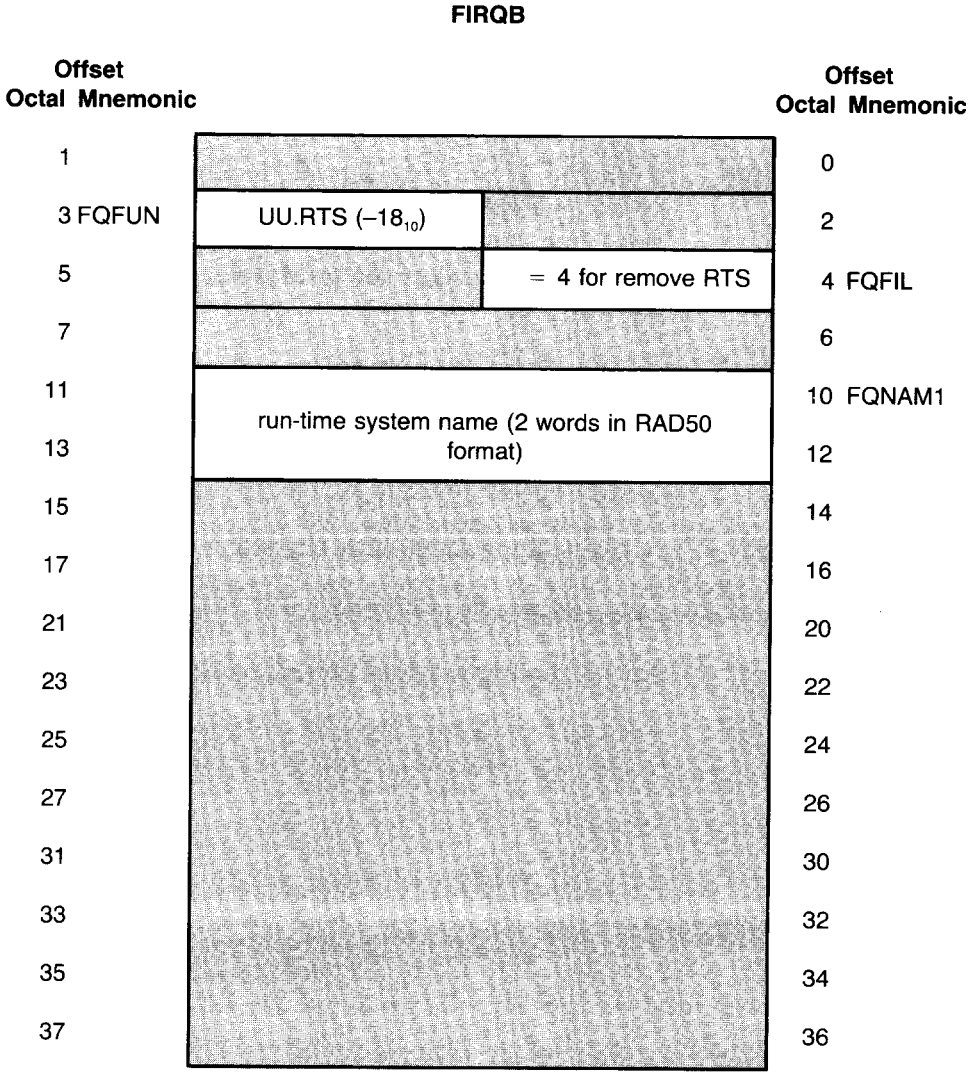
Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

Errors

NOROOM	If loaded at address specified, memory would be fragmented and a swapping violation would occur. See the discussion of assigning and allocating memory in the *RSTS/E System Generation Manual* for guidelines on how to avoid fragmenting memory.
NOSUCH	No file with the name given and a type of .RTS can be found in the account at FIRQB + FQPPN on the device specified.
PRVIOL	The file to be added has a bad format. For example, it is not contiguous or has illegal entries in the SIL index.
FIEXST	A run-time system with the same name already exists.
BADCNT	The range of memory starting at the load address at FIRQB + FQEXT is not available. See the SYSTAT memory status report to select an available range of memory.
NOBUFS	Adding a run-time system description block requires a small buffer and none is currently available.

Data Passed — Remove a Run-Time System



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

Errors

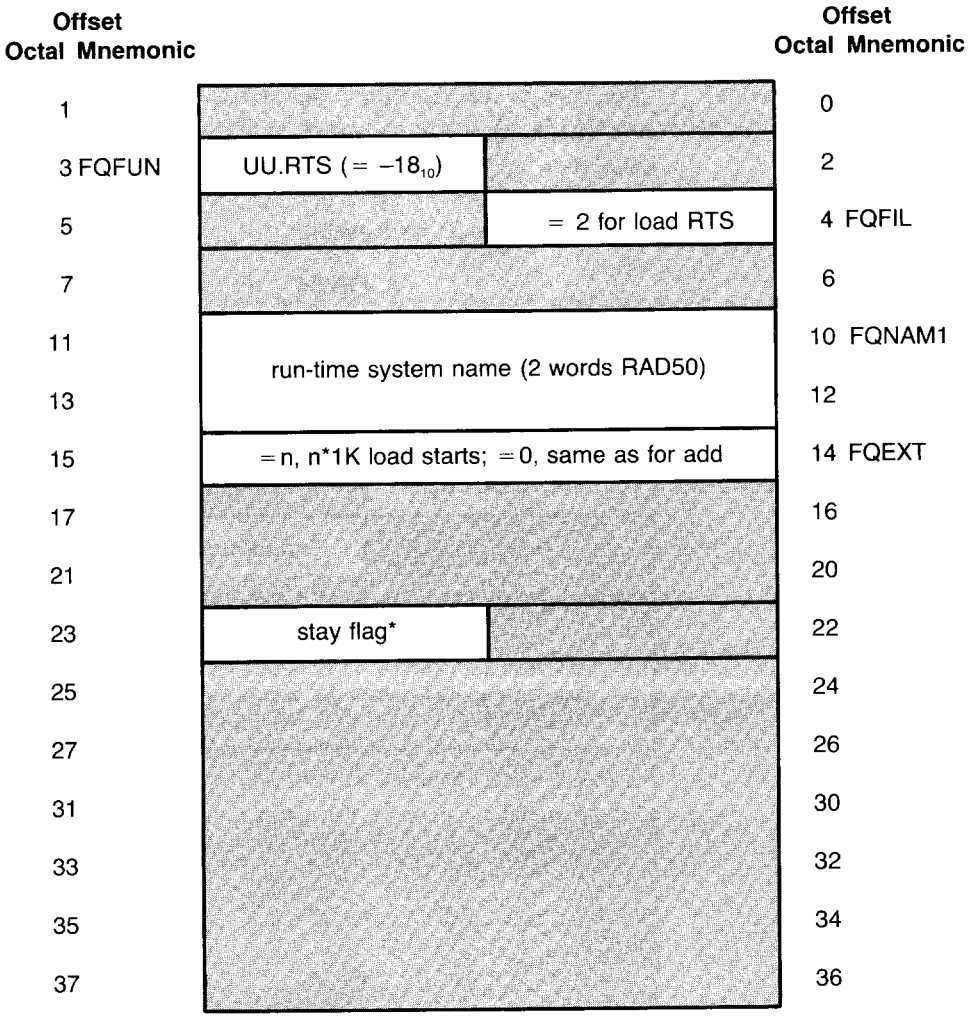
INUSE — This run-time system is currently being loaded into memory or is resident and in use. It cannot be removed until the usage count is zero.

NOSUCH — The run-time system is not currently defined.

PRVIOL — The run-time system is the primary run-time system and cannot be removed with this call. Use the DEFAULT initialization option to change the primary run-time system before starting timesharing.

Data Passed — Load a Run-Time System

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

---

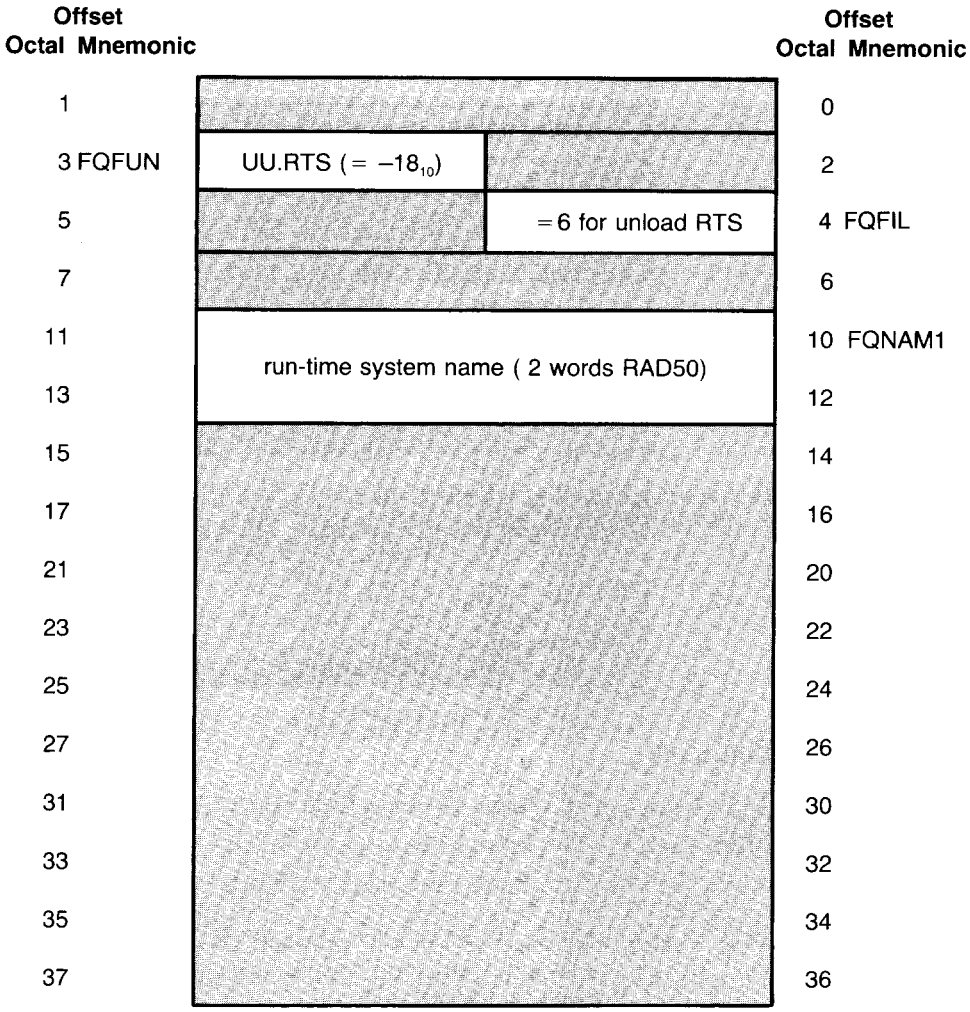
* stay flag = 200_8 Run-time system is permanently resident.
= 0 Memory occupied by this run-time system can be released when usage count goes to 0.

Errors

NOROOM	If the run-time system were loaded at the address specified, memory would be fragmented and a swapping violation would occur.
NOSUCH	The run-time system is not currently defined.
PRVIOL	The run-time system is the primary run-time system and has a preset loading address. Use the DEFAULT initialization option to relocate.
BADCNT	The range of memory starting at FIRQB + FQEXT is not available. See the memory status report of a display program to select an available range of memory.

Data Passed — Unload a Run-Time System

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

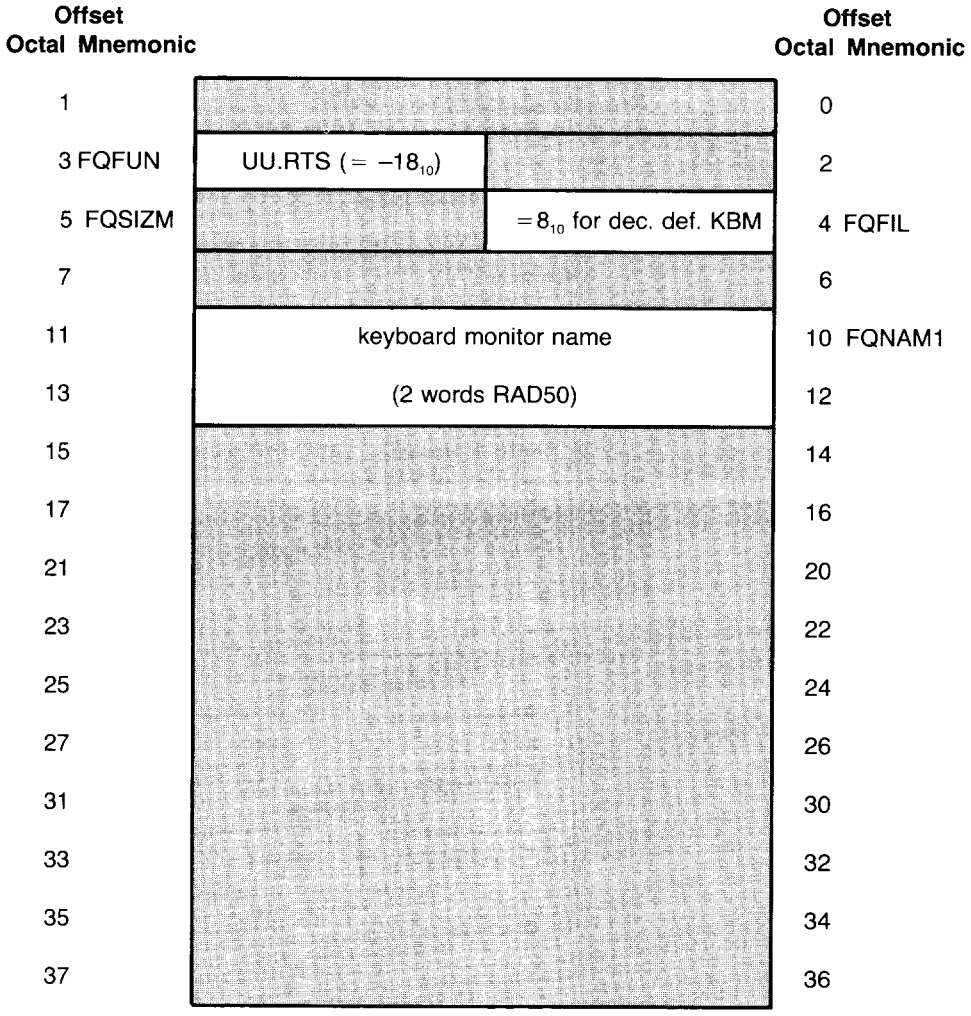
Errors

INUSE The run-time system is currently being loaded or is resident and in use. It cannot be unloaded until usage count is zero.

NOSUCH The run-time system is not currently defined.

Data Passed — Declare Default Keyboard Monitor

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

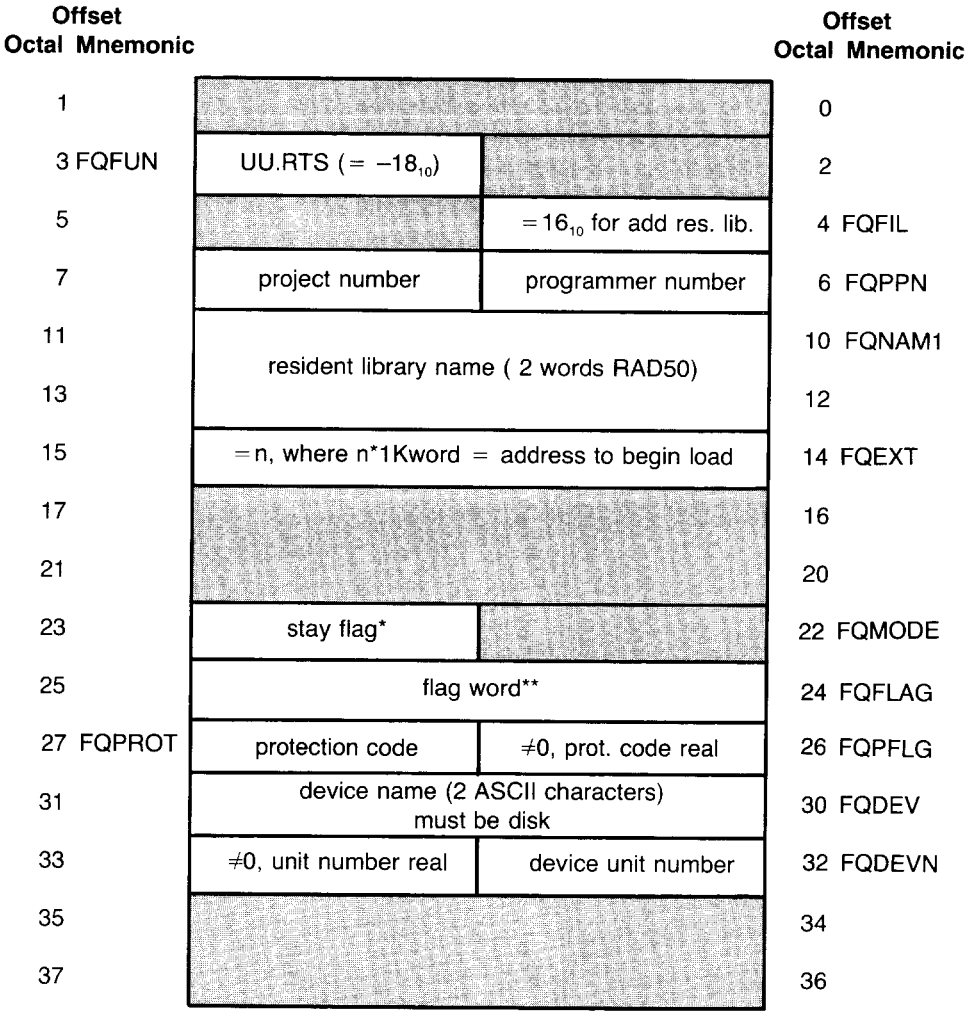
Errors

NOSUCH The file name in FIRQB+FQNAM1 has not been added as a run-time system.

PRVIOL The file name specified is not a keyboard monitor.

Data Passed — Add a Resident Library

FIRQB



* stay flag = 200_8 Resident library is permanently resident.
= 0 Resident library can be removed from memory when usage count goes to zero.

** flag word bit settings indicate resident library characteristics:

Bit 9 = 1 (Octal value = 1000) The resident library is to be accessed by only one user. That is, the library is not to be shared.
= 0 The resident library can be shared by more than one user.

Bit 10 = 1 (Octal value = 2000) Read/write access is allowed.
= 0 Read-only access.

Bit 12 = 1 (Octal value = 4000) Errors occurring within code in the resident library are not to be recorded in the system error log.
= 0 Errors may be recorded in the system error log.

Bit 13 = 1 (Octal value = 10000) The resident library is immediately removed from memory when its usage count goes to zero.
= 0 When the usage count for this library goes to zero, the monitor will remove it from memory when the space it takes is needed for something else.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

Errors

NOROOM If loaded at address specified, memory would be fragmented and a swapping violation would occur. See the discussion of assigning and allocating memory in the *RSTS/E System Generation Manual* for guidelines on how to avoid fragmenting memory.

NOSUCH No file with the given name and a type of .LIB can be found for the given account and device.

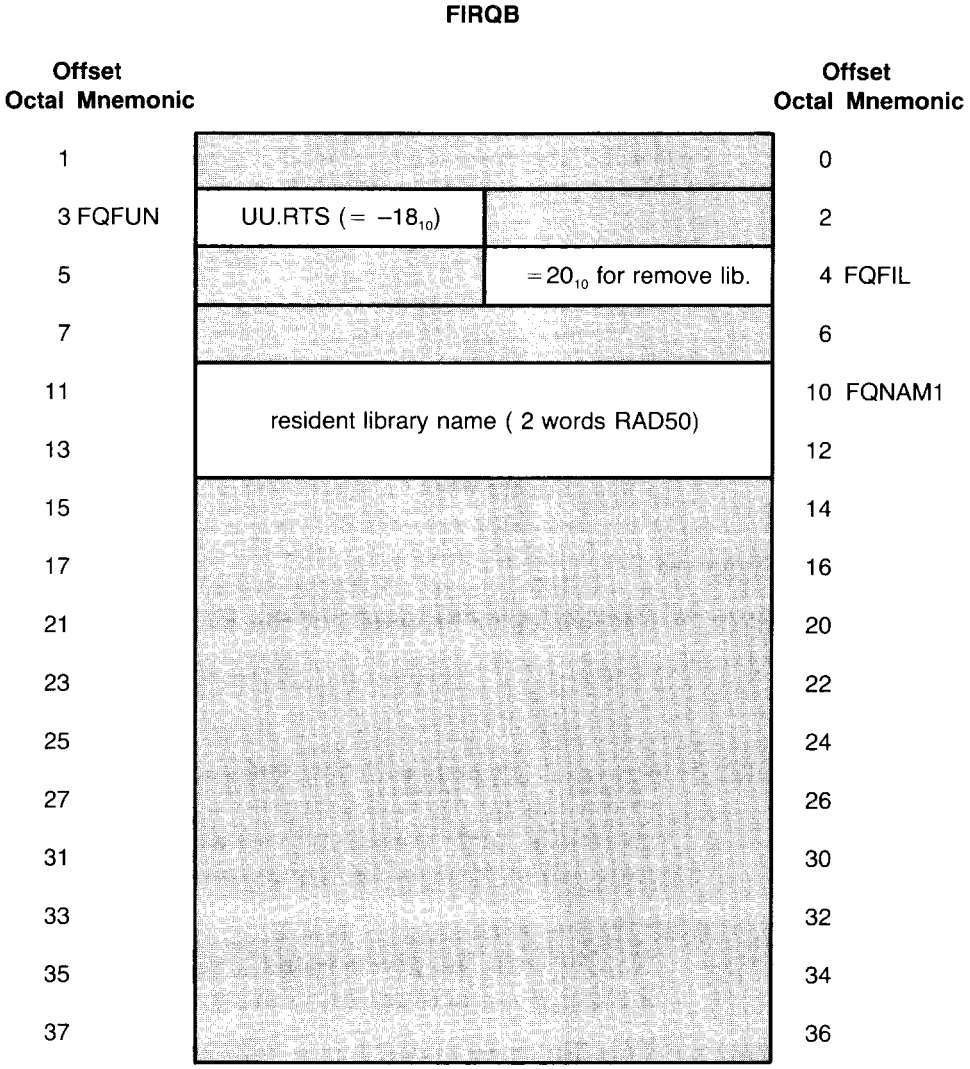
PRVIOL The file to be added has a bad format. For example, it is not contiguous or has illegal entries in the SIL index.

FIEXST You specified the file name of a resident library that already exists.

BADCNT You did not specify a load address, or the one specified is not available. See the memory status report of a display program to find an available range of memory.

NOBUFS Adding a resident library description block requires a small buffer and none is currently available.

Data Passed — Remove a Resident Library



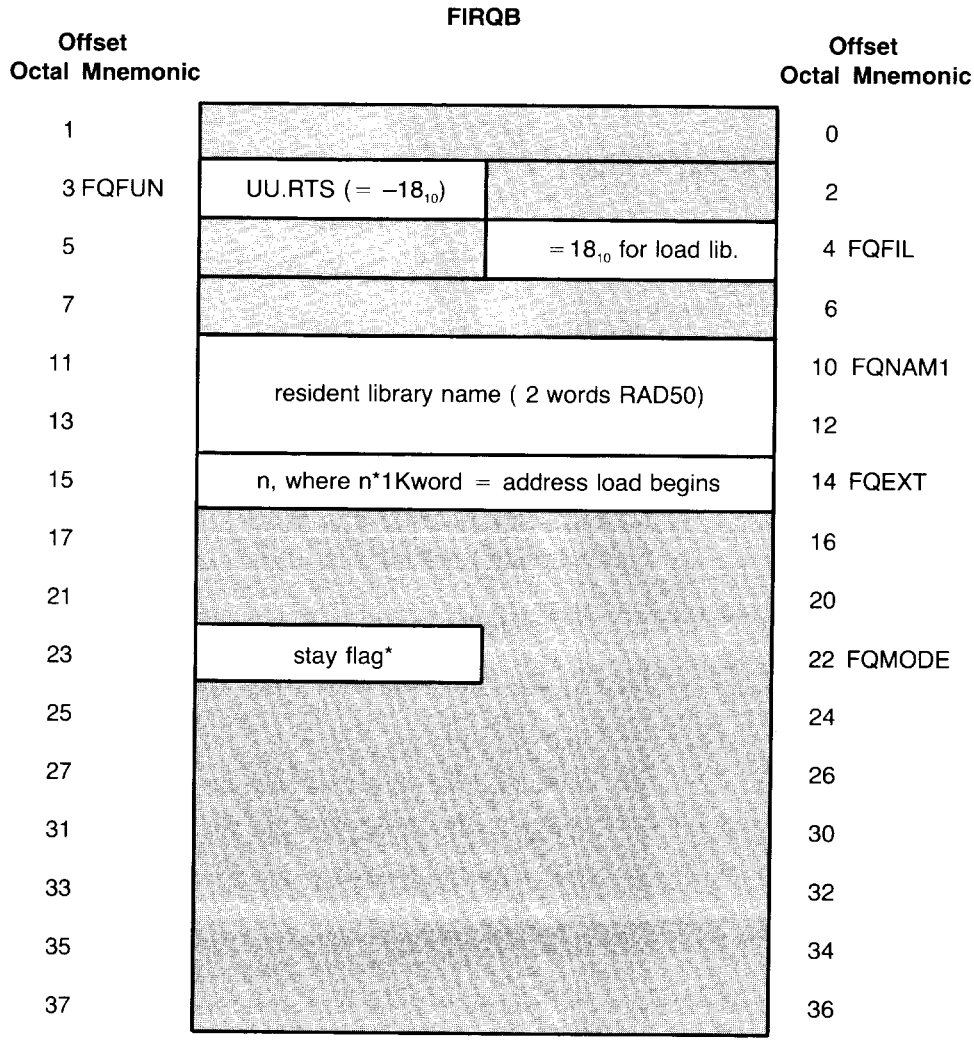
Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

Errors

INUSE	You tried to remove a library that is being loaded into memory or is in use. A resident library cannot be removed while a job is attached to it.
NOSUCH	You specified a resident library that is not currently defined.

Data Passed — Load a Resident Library



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

Errors

NOROOM — If loaded at the address specified, memory would be fragmented and a swapping violation would occur. See the discussion of assigning and allocating memory in the *RSTS/E System Generation Manual* for guidelines on how to avoid fragmenting memory.

---

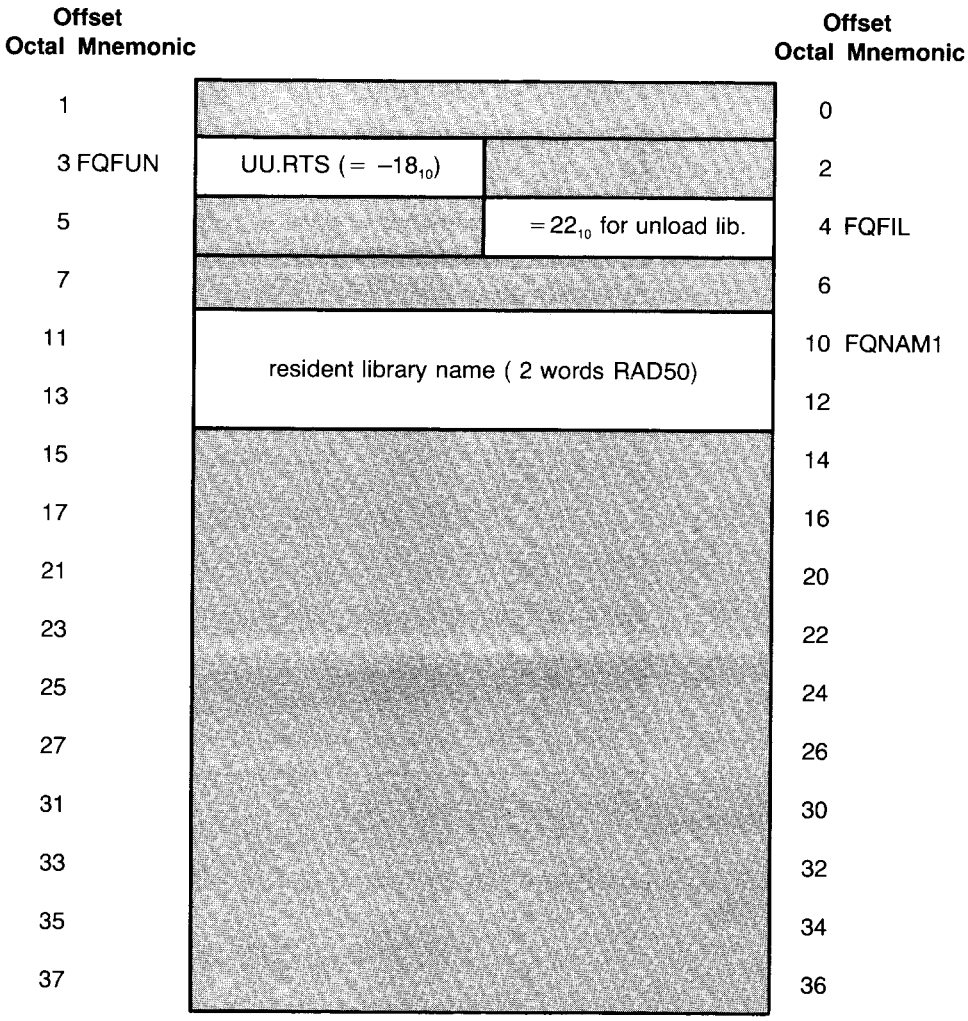
* stay flag = 200_8 Resident library is permanently resident.
= 0 Resident library can be removed from memory when usage count goes to zero.

NOSUCH You specified a resident library that is not currently defined.

BADCNT Load address specified is not available. See the memory status report of a display program to find an available range of memory.

Data Passed — Unload a Resident Library

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.RTS.

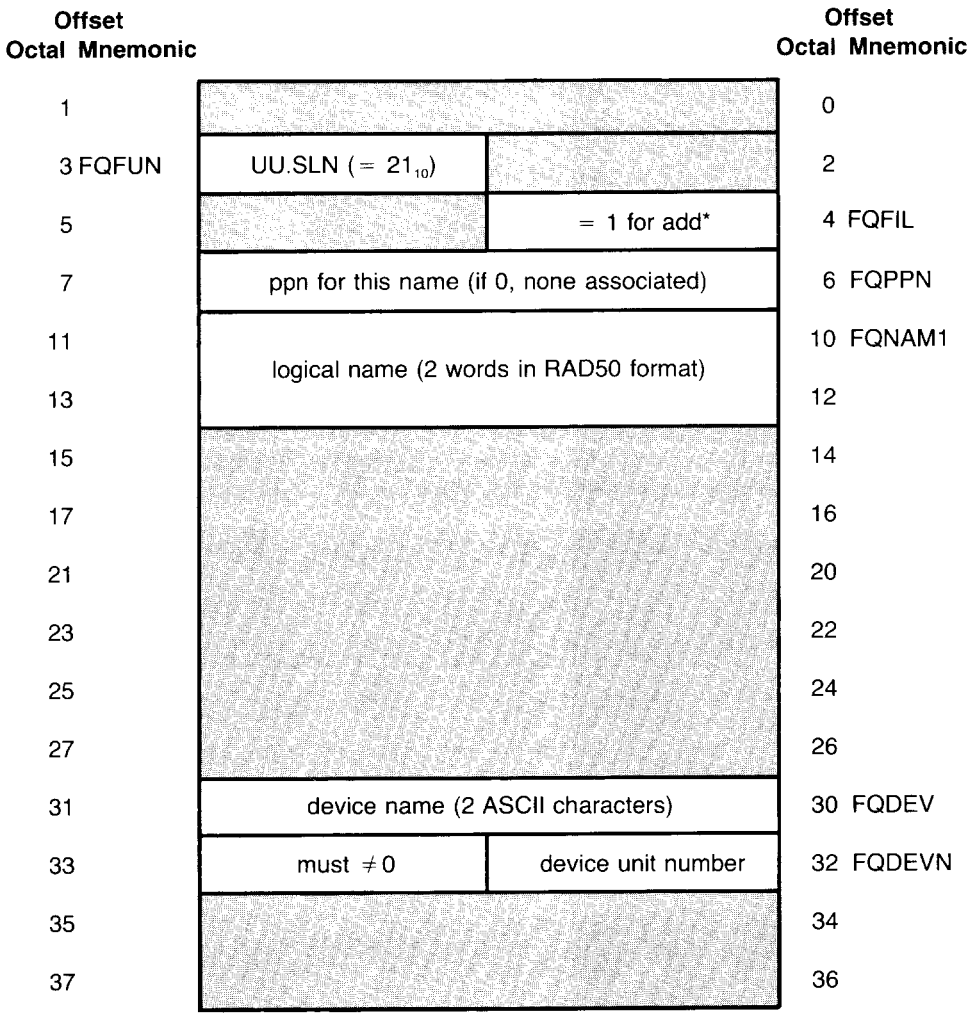
Errors

INUSE	You tried to unload a resident library that is now being loaded or is in use. A library cannot be unloaded while a job is still attached to it.
NOSUCH	You specified a resident library that is not currently defined.

3.32.36 UU.SLN (System Logical Names) – Privileged

Data Passed — Add New Names

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no meaningful data is returned.

Errors

BADNAM No logical name was given or the name contains nonalphanumeric characters.

---

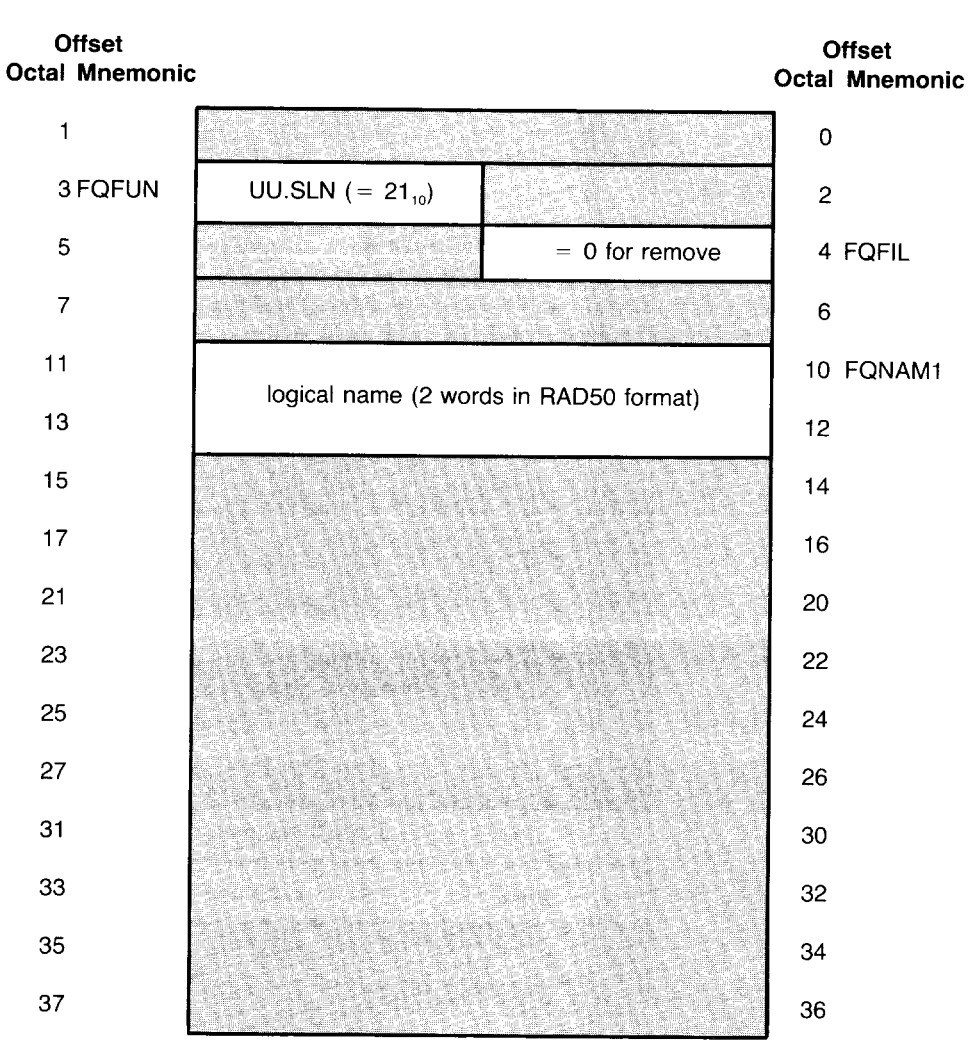
* This code adds a new entry to the second part of the system logical name table.

INUSE The logical name given duplicates one already in the first or second part of the table.

NOROOM All entries in the logical name table are in use. To free an entry, use the remove logical name call.

NODEVC The device specification is illegal or is not on the system.

Data Passed — Remove Logical Names



Data Returned

Except for a possible error in byte 0 of the FIRQB, no meaningful data is returned.

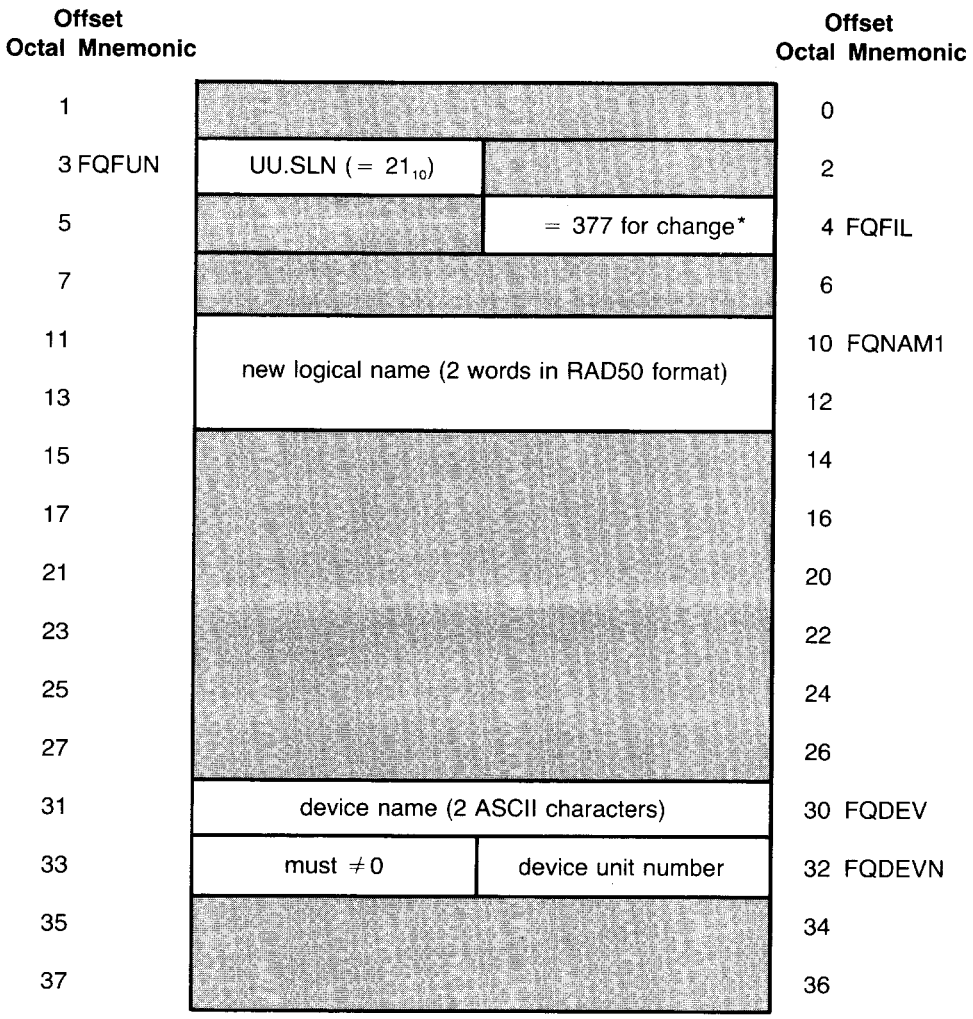
Errors

BADNAM — No logical name was given or the name contains nonalphanumeric characters.

NOSUCH — The logical name given is not currently defined as a logical name.

Data Passed — Change Disk Logical Name

FIRQB



* This code changes the logical name associated with a disk in the first part of the system logical name table.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no meaningful data is returned by the UU.SLN subfunction.

Errors

BADNAM	No logical name was given, or the name contains nonalphanumeric characters.
INUSE	The logical name given duplicates one already in the first or second part of the table.
NOSUCH	The disk specified is not on the system.
NODEVC	The device specified is illegally formatted or is not a disk.

3.32.37 UU.SPL (Spooling) – Not Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.SPL ($=-28_{10}$)		2	
5				4	
7		project number	programmer number	6	FQPPN
11		file name (2 words RAD50) can include wildcards		10	FQNAM1
13				12	
15		file type (1 word RAD50) can include wildcards		14	FQEXT
17		device name to spool to (ASCII); 0 => LP*		16	FQSIZ
21		≠0, unit number real	device unit number	20	FQBUFL
23		must = 0		22	FQMODE
25		flag word**		24	FQFLAG
27				26	
31		device name where file is (disk)		30	FQDEV
33		≠0, unit number real	device unit number	32	FQDEVN
35				34	
37				36	

* RSTS/E currently has two spooling packages: the small (micro–RSTS) spooling package and the large (standard RSTS/E) spooling package. When you execute the UU.SPL directive, the system first looks for the small spooling package's receiver, QMAN. If the device name you specify at FIRQB+FQSIZ is 0 or "LP," the system sends the request to the small spooler. If the device name at FIRQB+FQSIZ is anything other than 0 or "LP," the system sends the request to the large spooler. (UU.SPL works the same way on all systems, no matter which spooling package is installed.) Note that the system manager can install an optional patch that forces spooling requests to the large spooler, even if the small spooler is installed.

** The flag word has a different meaning for the small and large spooling packages.

You can set the following bits for the small spooling package:

4	Delete after spooling; same as DCL PRINT command's /DELETE qualifier.
40_8	No header; same as DCL PRINT command's /NOFLAG_PAGES qualifier.

You can set the following bits for the large spooling package:

1	File spooled with FORTRAN carriage control; same effect as QUE/TYP:FTN.
2	Restart; same as QUE/RE.
4	Delete after spooling; same as QUE/DE or DCL PRINT command's /DELETE qualifier.
10_8	Binary file; same as QUE/BI.
20_8	End; same as QUE/END.
40_8	No header; same as QUE/NH or DCL PRINT command's /NOFLAG_PAGES qualifier.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no meaningful data is returned with the UU.SPL subfunction.

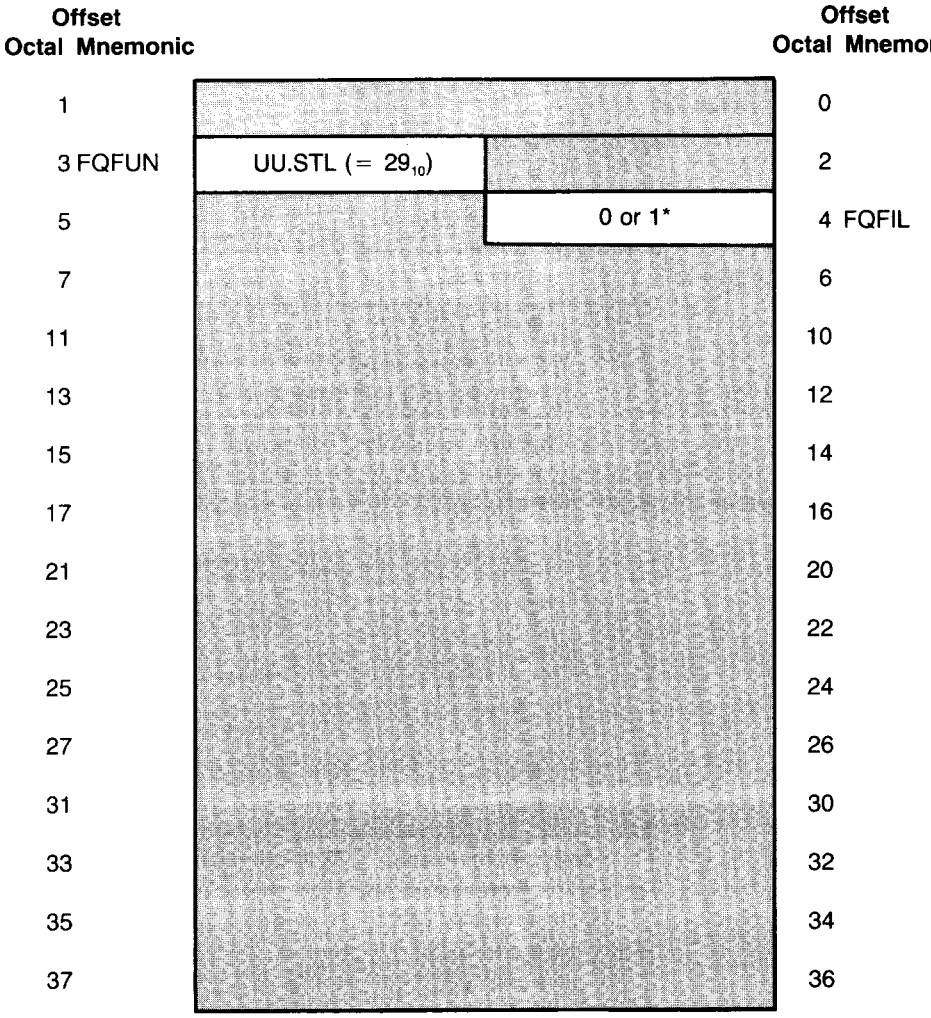
Errors

NOROOM	The number of messages pending for the queue is at its declared maximum. This may be a transient condition; retry the operation.
NOSUCH	The account specified at FIRQB+FQPPN does not exist on the device specified, the file name or type specified at FIRQB+FQNAM1 cannot be found, or neither QMAN (small spooler) nor QUEMAN (large spooler) is installed as a message receiver.
NODEVC	An attempt was made to spool a file to a spooling device that had a unit number greater than 7, or the file to be spooled is contained on an invalid device.
PRVIOL	An attempt was made to queue a file to which the user did not have read access or to queue a compiled file. This error also occurs if a nonprivileged user attempts to queue a file that has the privileged <128> bit set.
HNGDEV	This error is caused by a hardware condition. For example, the specified disk could not be accessed.
NOTMNT	The specified disk device is not mounted; logically mount the disk with UTILTY, UMOUNT, or INIT.
PAKLCK	The disk is in a locked state. Execute the call under a privileged account to override this condition.
DEVNFS	The device specified at FIRQB+FQDEV of the call is not a file-structured device.
NOBUFS	System buffers are not currently available to store this message. This may be a transient condition; retry the operation.

3.32.38 UU.STL (Stall/Unstall System) – Privileged

Data Passed

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the UU.STL subfunction.

Errors

INUSE	You tried to stall the system, but it is already stalled.
BADFUO	You specified "unstall," but the system is not stalled.

---

* 0 = Return the system to normal ("unstalled") state.
1 = "Stall" – suspend all currently active jobs except for the calling job.

3.32.39 UU.SWP (Add, Remove, and List System Files) – Privileged

Data Passed — Add System File

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.SWP (= 23_{10})		2	
5	FQSIZM	= 1 for add	file to add*	4	FQFIL
7				6	
11		for [0,1] file with type of .SYS, file name as 2 words RAD50. If both words = 0, non-file-structured disk.		10	FQNAM1
13				12	
15				14	
17				16	
21				20	
23				22	
25				24	
27				26	
31		device name (disk)	(2 ASCII characters)	30	FQDEV
33		≠0, unit number real	device unit number	32	FQDEVN
35				34	
37				36	

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the add option of the UU.SWP subfunction.

---

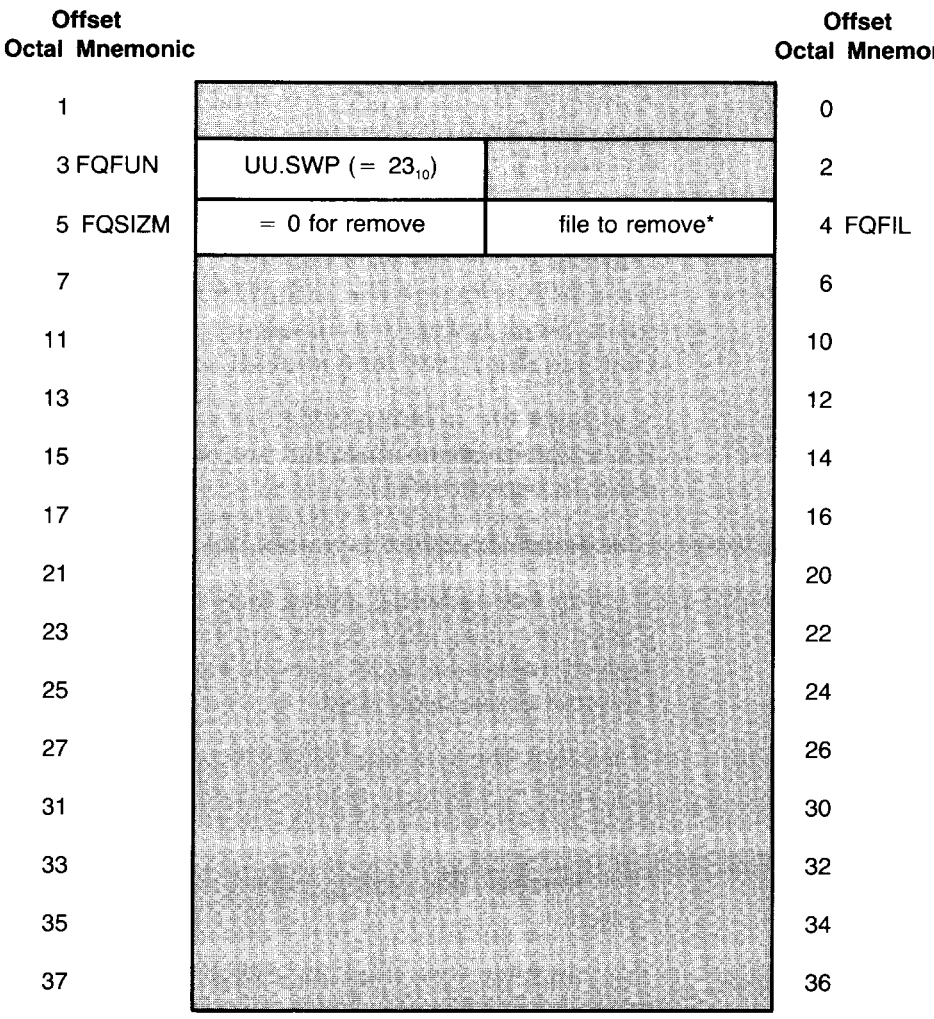
* byte 4 = 0 Swap file 0.
1 Swap file 1.
2 Illegal — generates error, as slot 2 is always added.
3 Swap file 3.
4 Overlay file.
5 Error message file.
6 DECnet/E system file.

Errors

BADNAM	No file name is specified when an overlay, an error message, or a DECnet/E system file is being added, or the name specified contains nonalphanumeric characters.
INUSE	A swap file is being added to a non-file-structured disk but the disk is currently mounted (that is, it is being used as a file-structured device).
NOROOM	If an overlay or error file is being added, this error indicates that the file is not long enough. (The overlay file should be at least 64 blocks and the error file at least 16 blocks.) If a swap file is being added to a file-structured device, this error means that the file is not long enough to store even one job.
NOSUCH	A system file is being added to a file-structured disk, but the file with the name specified and with a .SYS file type does not exist in account [0,1].
NODEVC	The device specified is a disk but is not on this system.
NOTAVL	A swap file is being added to a non-file-structured disk, but either the disk unit or its controller has been disabled. The system manager must use an initialization option to enable the unit or its controller.
PRVIOL	A system file is being added to a file-structured disk. Either the unit is logically write locked, or the file specified is bad (that is, it is not contiguous or is currently open).
FIEXST	The system file being added is already installed on the system.
BADFUO	The number specified at FIRQB+FQFIL is either 2 or is greater than 6. The swap file for file 2 must exist on the system disk and cannot be added during timesharing. System files to be added are defined only by the values 0, 1, 3, 4, 5, and 6.
NOTMNT	A system file is being added to a file-structured disk but that disk is not currently mounted. Use either INIT or the UTILTY MOUNT command to logically mount the disk before the file is added.
DEVNFS	The device specified is not a disk device.

Data Passed — Remove System File

FIRQB



Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by the remove option of the UU.SWP subfunction.

---

* byte 4 = 0 Swap file 0.
= 1 Swap file 1.
= 2 Illegal — generates error, as slot 2 is always added.
= 3 Swap file 3.
= 4 Overlay file.
= 5 Error message file.
= 6 DECnet/E system file.

Errors

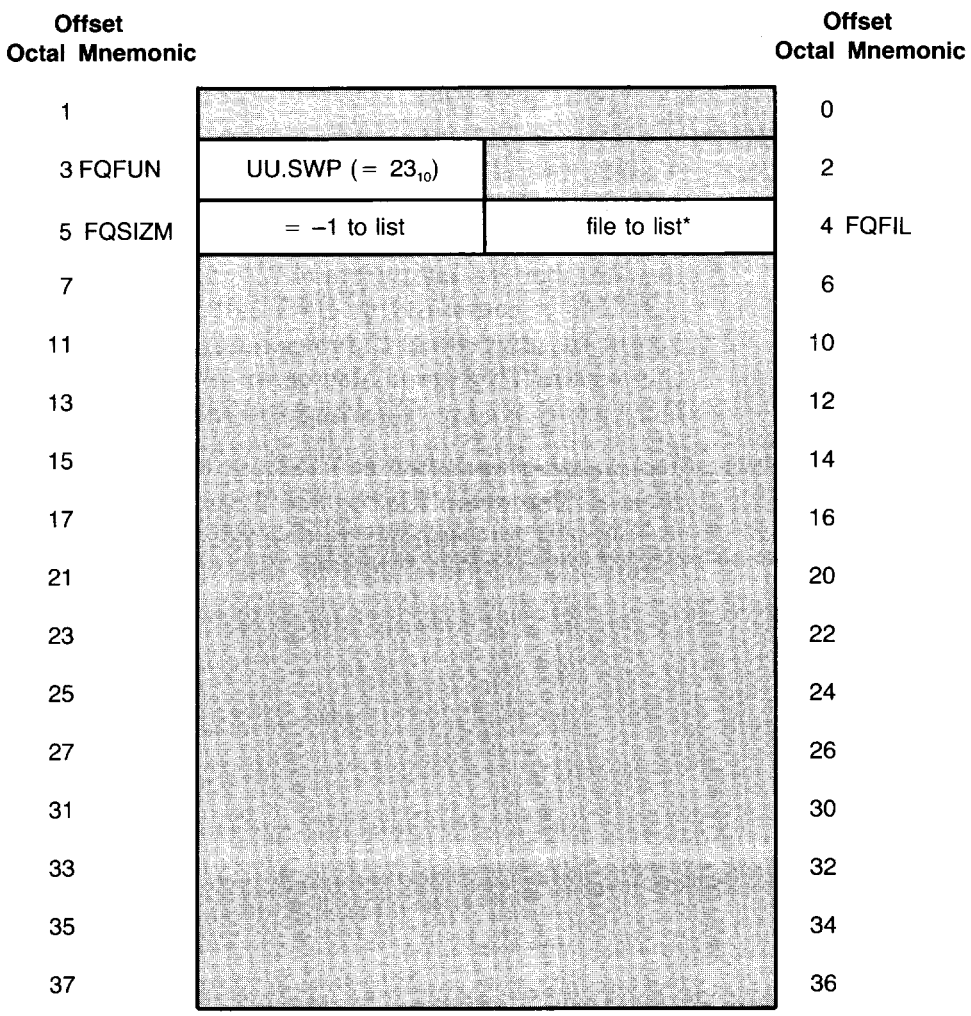
INUSE The swap file to be removed can be properly removed but currently contains one or more swapped out jobs. The system will lock the file and begin swapping jobs to other files. Retry the call at a later time when the swapped out jobs are no longer in this file.

PRVIOL A swap file is to be removed but its removal will decrease the swap file space below the limit required to store the maximum number of jobs on the system. To remove the swap file, decrease the number of logins currently allowed (by either the SET LOGINS x command or SYS call), wait until the number of logged in jobs falls to the maximum, and try the removal operation again. This error also occurs if you attempt to remove the DECnet/E system file when DECnet/E is still on.

BADFUO The number specified at FIRQB+FQFIL is either 2 or greater than 6. The swap file for file 2 must exist on the system disk and cannot be removed during time sharing. System files to be removed are defined only by the values 0, 1, 3, 4, 5 and 6.

Data Passed — List System File

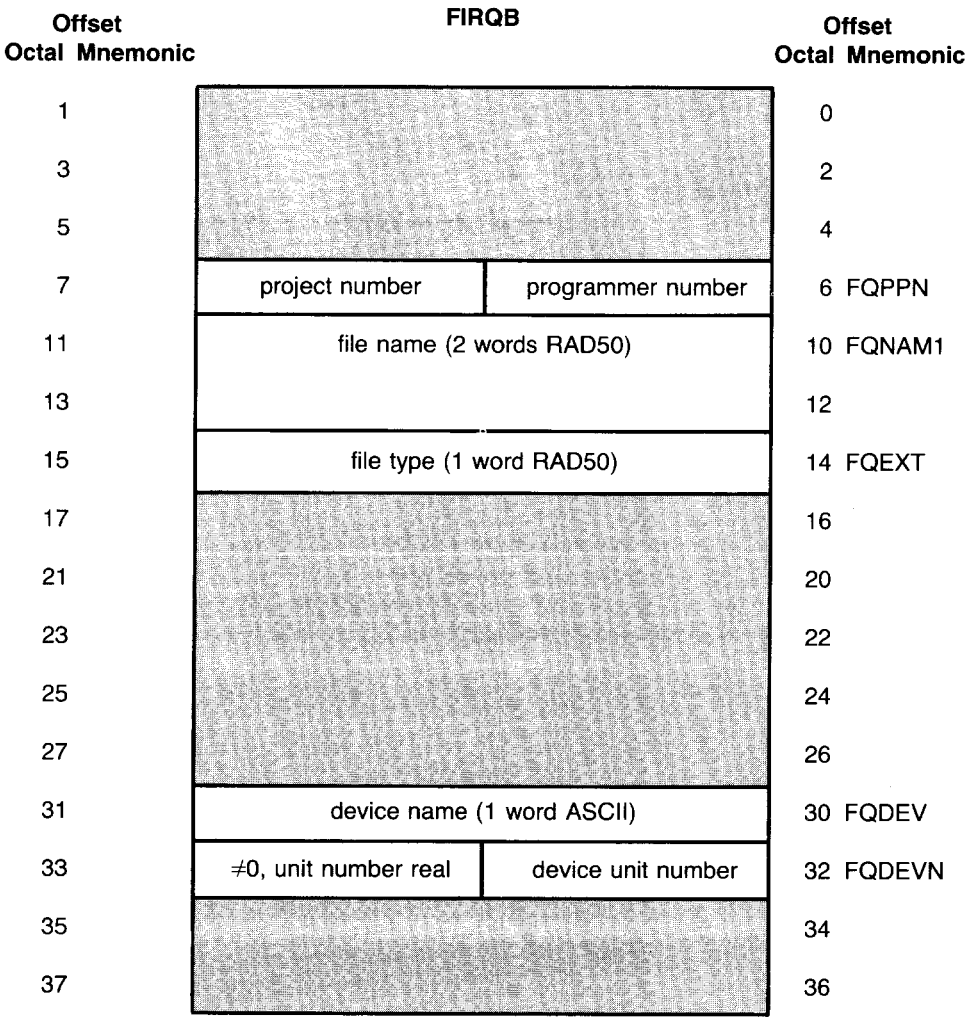
FIRQB



---

* byte 4 = 0 Swap file 0.
= 1 Swap file 1.
= 2 Swap file 2.
= 3 Swap file 3.
= 4 Overlay file.
= 5 Error message file.
= 6 DECnet/E system file.

Data Returned — List System File



Errors

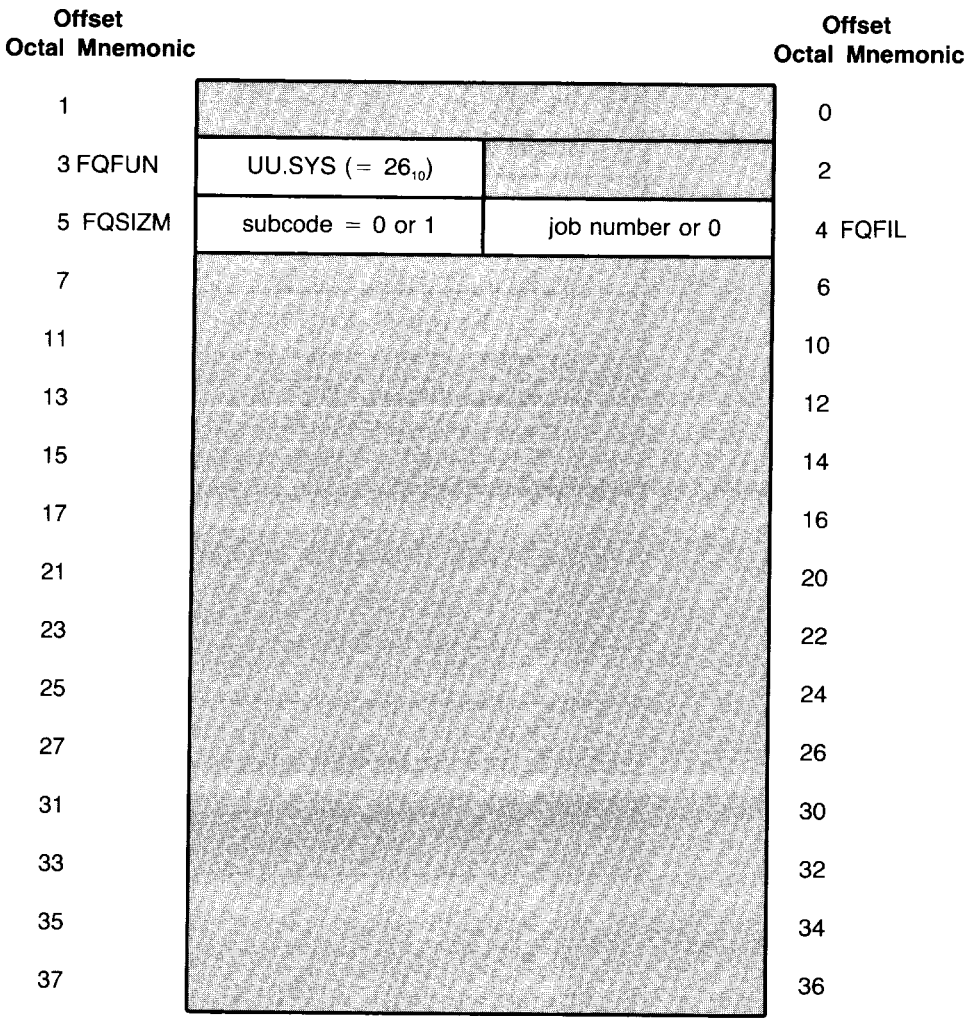
NOSUCH The number specified at FIRQB + FQFIL refers to a file that is not currently installed.

BADFUO The number specified at FIRQB + FQFIL is less than 0 or greater than 6.

3.32.40 UU.SYS (Return Job Status Information) – Privileged and Not Privileged

Data Passed

FIRQB



Data Returned (for subcode = 0)

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			calling job no. * 2	2	FQJOB
5	FQSIZM	KB no. of job's console*	status job no *2	4	FQFIL
7		swap slot location	controlling job no. *2 +1**	6	FQPPN
11		least significant word of CPU time, .1 sec.		10	FQNAM1
13		job's current connect time, in mins.		12	
15		least significant word of KCTs		14	FQEXT
17		job's accumulated device time, mins.		16	FQSIZ
21		MSB of CPU time	MSB of KCTs	20	FQBUFL
23		job name (2 words RAD50)		22	FQMODE
25				24	
27	FQPROT	project number	programmer number	26	FQPFLG
31		job keyboard monitor (2 words RAD50)		30	FQDEV
33				32	
35		current run-time system (2 words RAD50)		34	FQCLUS
37				36	

* If this value is negative, the job is detached, and the value is the one's complement of the keyboard number to which the job was previously attached.

** Returned only if job is attached to a pseudo keyboard.

Data Returned (for subcode = 1)

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			calling job no * 2	2	FQJOB
5	FQSIZM	KB no. of job's console*	status job number * 2	4	FQFIL
7		current flag word		6	FQPPN
11		curr. info. posting	current IOSTS	10	FQNAM1
13		current JBSTAT word		12	
15		current JBWAIT word		14	FQEXT
17		mem. control sub-block	size, in K words	16	FQSIZ
21		current physical addr., 32–word increm.		20	FQBUFL
23		run burst, in .1 sec**	priority**	22	FQMODE
25		value at offset 6	max. memory size	24	FQFLAG
27		(depends on JBWAIT and JBSTAT)		26	FQPFLG
31		read or write state		30	FQDEV
33		pointer to Job Data Block		32	FQDEVN
35		pointer to second Job Data Block		34	FQCLUS
37		pointer to first Receiver ID Block		36	FQNENT

Errors

PRVIOL The job specified at FIRQB + FQFIL does not exist.

BADFUO The job number specified at FIRQB + FQFIL is less than zero or greater than JOBMAX.

Neither error can occur for a nonprivileged caller. Information is returned on the calling job, and the number at FIRQB + FQFIL is ignored.

---

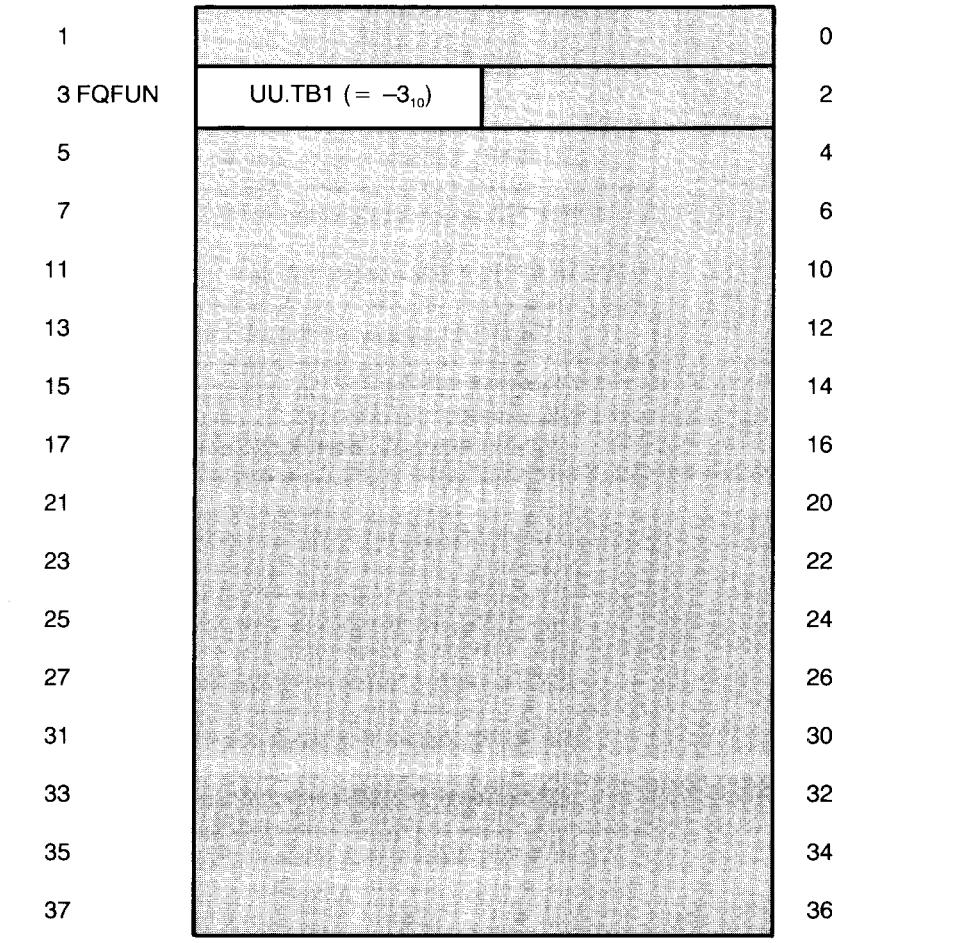
* If this value is negative, the job is detached, and the value is the one's complement of the keyboard number to which the job was previously attached.

** Returned only if the caller is privileged.

3.32.41 UU.TB1 (Get Monitor Tables, Part I) – Not Privileged

Data Passed

FIRQB



Data Returned

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job no. * 2	2	FQJOB
5	FQSIZM	MAXCNT(max. job no.)	CNT.KB–1(max. KB no.)	4	FQFIL
7		(DEVCNT) address of max. unit no. table		6	FQPPN
11		(DEVPTR) address of pointers to dev. DDBs		10	FQNAM1
13		(MEMLST) root link word in 1st mem. ctrl. sublk.		12	
15		(JOBTBL) address of the job table		14	FQEXT
17		(JBSTAT) address of job status table		16	FQSIZ
21		(JBWAIT) address of job wait flag table		20	FQBUFL
23		(UNTCLU) add. of unit cluster/err cnt tbl		22	FQMODE
25		(UNTCNT) address of disk status table		24	FQFLAG
27		(SATCTL) address of free-block table		26	FQPFLG
31		(JSBTBL) add. of job stat. ordered by drv.		30	FQDEV
33		(SATCTM) add. of free block count table		32	FQDEVN
35		current date in system internal format*		34	FQCLUS
37		(UNTOWN) add. of disk owner/option table		36	FQNENT

Errors

No errors are possible with UU.TB1.

---

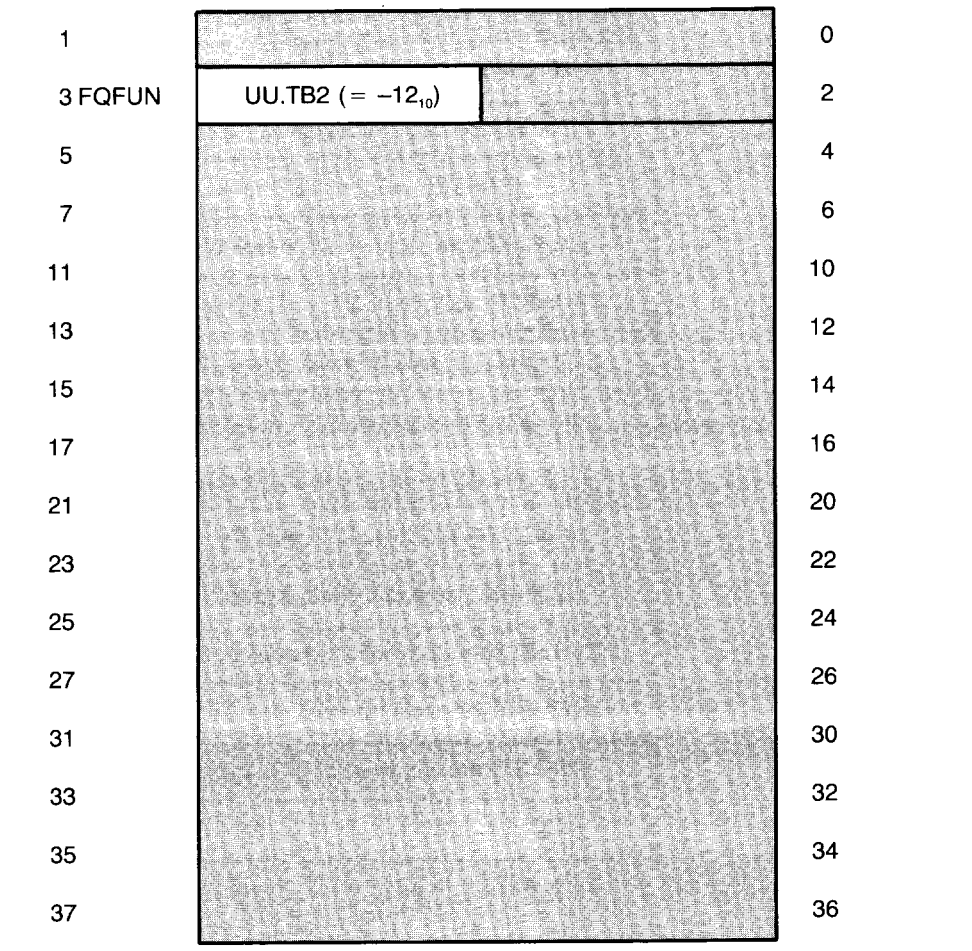
* System internal format for date is:

$[(\text{year} - 1970) * 1000_{10}] + \text{day-within-year}$

3.32.42 UU.TB2 (Get Monitor Tables, Part II) – Not Privileged

Data Passed

FIRQB



Data Returned

FIRQB

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1			0	
3		current job no. * 2	2	FQJOB
5		(FREES) add. of free buffer info table	4	FQFIL
7		(DEVNAM) add. of device name table	6	FQPPN
11		(CSRTBL) add. of CSR table	10	FQNAM1
13		(DEVOKB) no. disks in DEVNAM times 2	12	
15		(TTYHCT) no. hung term. errs. since strtup	14	FQEXT
17		(JOBCNT) no. jobs now/no. logins allowed	16	FQSIZ
21		(RTSLST) rt. lnk.word in RTS dsc.blk.list	20	FQBUFL
23		(ERLCTL) error logging control data	22	FQMODE
25		(SNDLST) list of eligible msg. rec. jobs	24	FQFLAG
27		(LOGNAM) table of system logical names	26	FQPFLG
31		(DEVSYN) start of synonym names in DEVNAM	30	FQDEV
33		(MEMSIZ) physical memory size,Kwrds*32	32	FQDEVN
35		(CCLLST)root link word of CCL desc. blks.	34	FQCLUS
37		(FCBLST) add. of table of roots of FCBs	36	FQNENT

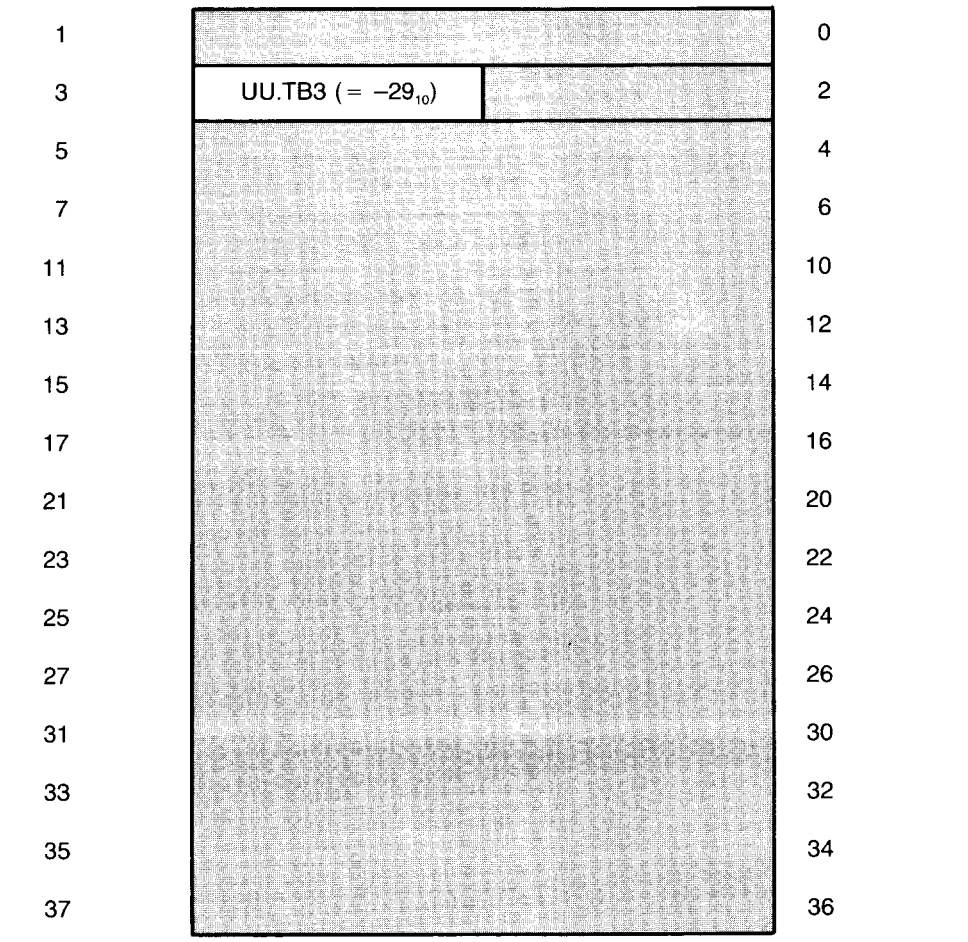
Errors

No errors are possible with UU.TB2.

3.32.43 UU.TB3 (Get Monitor Tables, Part III) – Not Privileged

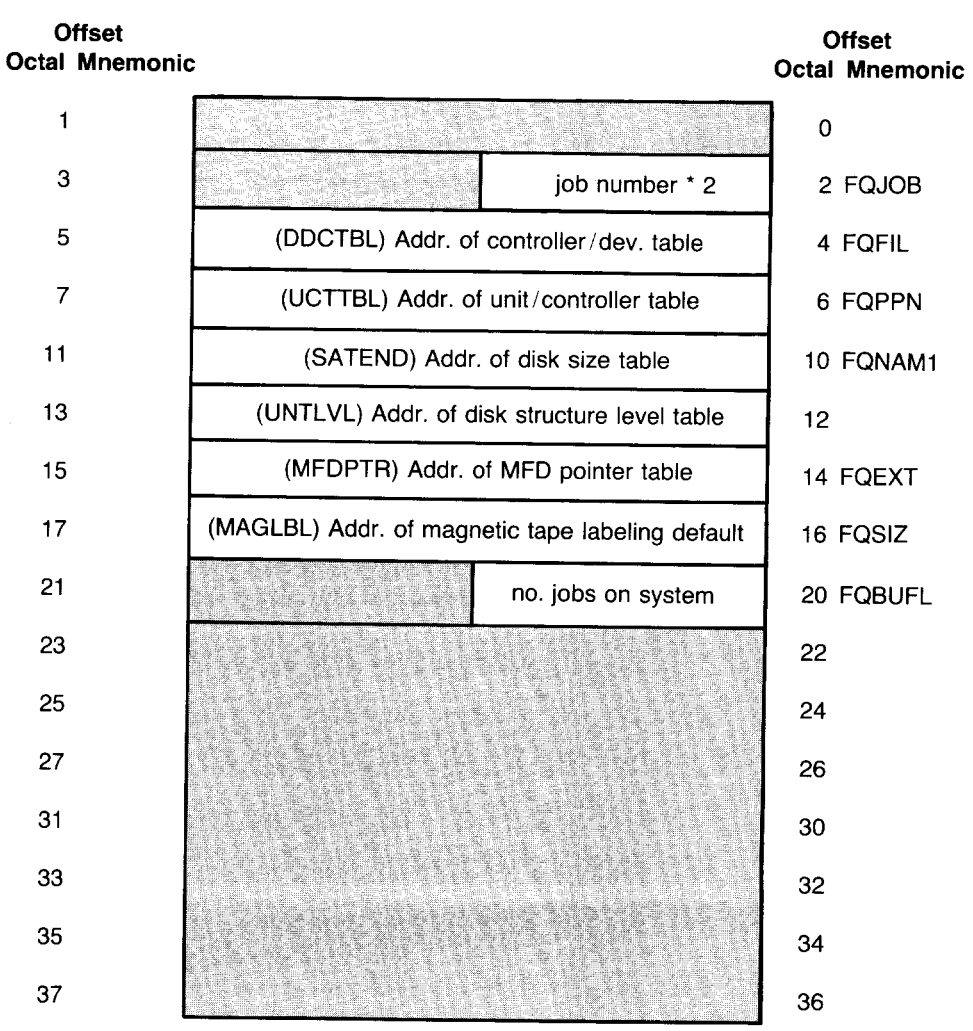
Data Passed

FIRQB



Data Returned

FIRQB



Errors

No errors are possible with UU.TB3.

3.32.44 UU.TRM (Set Terminal Characteristics) – Privileged and Not Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.TRM (= 16_{10})		2	
5	FQSIZM	377(current) or KB no		4	
7		Tab/No Tab 200 377	Width: 0 or width+1	6	FQPPN
11		LC Outpt/No LC Outpt 200 377	No Form/Form 200 377	10	FQNAM1
13		Full Dplx/Local Echo 200 377	No XON/XON 200 377	12	
15		No LC Inpt/LC Input 200 377	No Scope/Scope* 200 377	14	FQEXT
17		rcv baud rt;377=2741	No Fill/Fill 1 fill factor+1	16	FQSIZ
21		xmt baud rt;377=2741	No Parity/Even/Odd 1 2 3	20	FQBUFL
23		Up Arrow/No Up Arrow 200 377	No Stall/Stall 200 377	22	FQMODE
25		parameters**		24	FQFLAG
27	FQPROT	No Esc Seq/Esc Seq 200 377	377= /Ring	26	FQPFLG
31		No Esc/Esc 200 377	Delimiter/Delim x 200 200+ASCII code	30	FQDEV
33		Resume CTRL–C/Any 200 377	CTRL/R,CTRL/T disable=200 enable=377	32	FQDEVN
35		Enable/disable broadcast 200 377	BREAK = null/CTRL/C 200 377	34	FQCLUS
37				36	

* Setting "No Scope" automatically sets "No Stall."
Setting "Scope" automatically sets "Stall."

** Value = 10 + DATA + STOP + PARITY

where:

DATA is	0	for 5–bit characters.
	1	for 6–bit characters.
	2	for 7–bit characters.
	3	for 8–bit characters.
STOP is	0	for 1 stop bit per character.
	4	for 2 stop bits per character or 1.5 bits if DATA = 0.
PARITY is	0	for no parity bit.
	20	for even parity format.
	60	for odd parity format.

This byte applies only to interfaces that support DATA/STOP/PARITY features. When this byte is used with these interfaces, it overrides the setting of byte 20. In addition, when this byte is used, byte 17 must be nonzero.

Data Returned (Current Keyboard Characteristics)

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3			current job no. * 2	2	FQJOB
5	FQSIZM	377(current) or KB no	KB status*	4	FQFIL
7		Tab/No Tab	Width: 0 or width + 1	6	FQPPN
11		LC Outpt/No LC Outpt	No Form/Form	10	FQNAM1
13		Full Dplx/Local Echo	No XON/XON	12	
15		No LC Inpt/LC Input	No Scope/Scope	14	FQEXT
17		rcv baud rt;377 = 2741	No Fill/Fill	16	FQSIZ
21		xmt baud rt;377 = 2741	No Parity/Even/Odd	20	FQBUFL
23		Up Arrow/No Up Arrow	No Stall/Stall	22	FQMODE
25			interface for line	24	FQFLAG
27	FQPROT	No Esc Seq/Esc Seq		26	
31		No Esc/Esc	Delimiter/Delim x	30	FQDEV
33		Resume CTRL–C/Any	CTRL/R,CTRL/T flag	32	FQDEVN
35		Enable/disable broadcast	BREAK = null/CTRL/C	34	FQCLUS
37				36	

---

* Bit 0 = 1 Keyboard is disabled or is a pseudo keyboard that is not in use.
Bits 1–6 = 0 No job owns keyboard.
≠ 0 Job number * 2 of job that owns keyboard.
Bit 7 Reserved.

Errors

BADFUO One of two possibilities:

1. Keyboard number is out of range of valid numbers.
2. Current keyboard is specified but is detached.

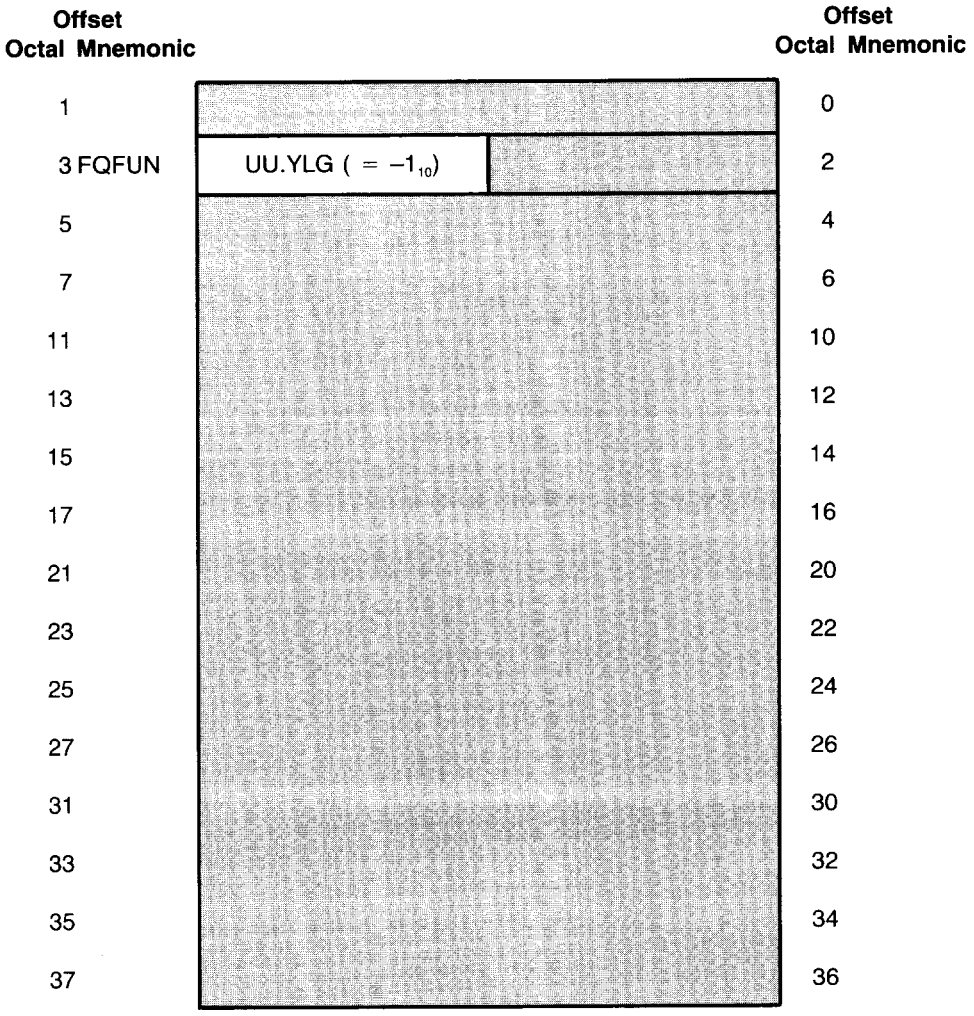
PRVIOL You are nonprivileged and tried to either:

1. Read the characteristics of a terminal not opened or assigned to your job.
2. Change the characteristics of a terminal other than your job's console terminal (KB:).
3. Change the speed setting for your terminal. (Byte 17 or 21 is nonzero.)
4. Set the ring list entry for a dial-up terminal line. (Byte 26 is nonzero.)

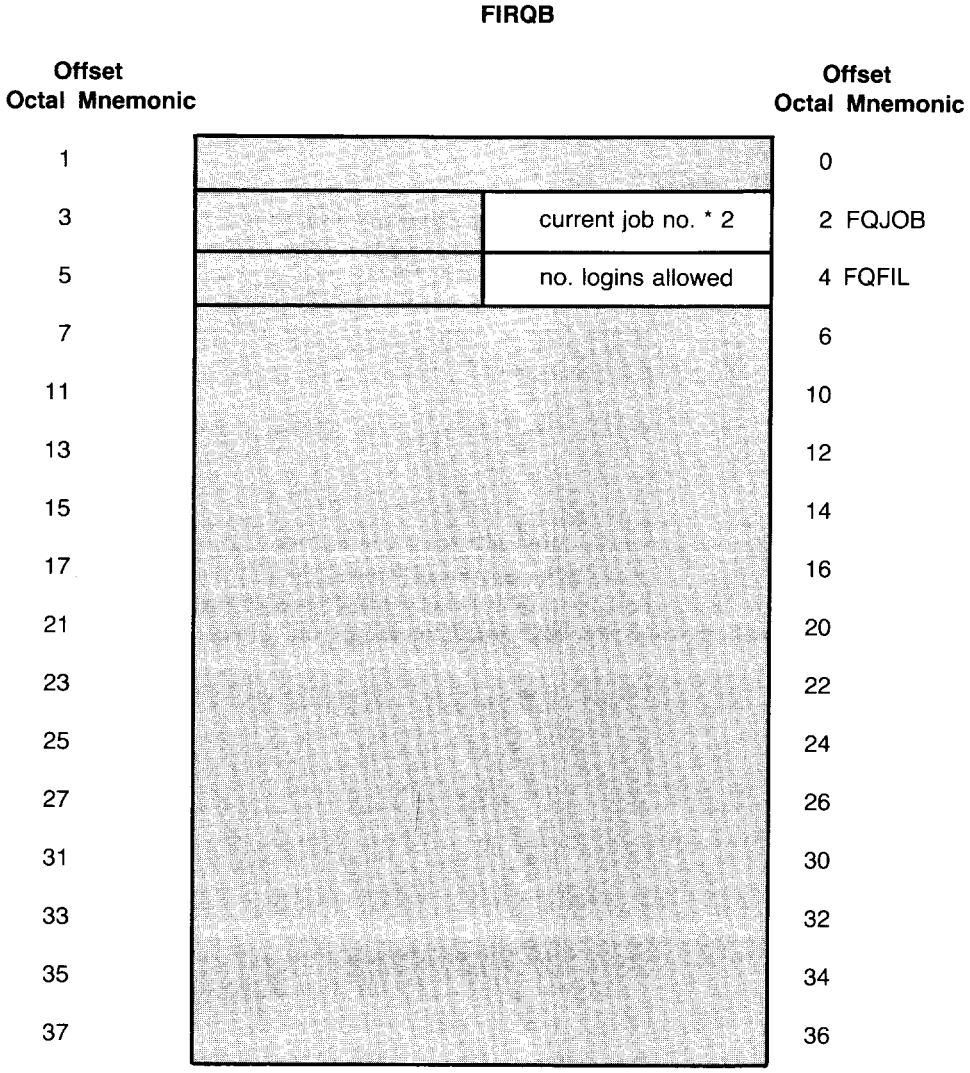
3.32.45 UU.YLG (Enable Logins) – Privileged

Data Passed

FIRQB



Data Returned



Errors

No errors are possible with UU.YLG.

3.32.46 UU.ZER (Zero Device) – Privileged and Not Privileged

Data Passed

FIRQB

Offset Octal	Mnemonic			Offset Octal	Mnemonic
1				0	
3	FQFUN	UU.ZER (= 13_{10})		2	
5			flag (disk only)*	4	FQFIL
7		project number**	programmer number**	6	FQPPN
11		vol. ID for label (ANSI format mag. tape only) (2 words in RAD50 format)		10	FQNAM1
13				12	
15				14	
17				16	
21				20	
23				22	
25				24	
27				26	
31		device name (disk, magtape, or DECtape)		30	FQDEV
33		≠0, unit number real	device unit number	32	FQDEVN
35				34	
37				36	

---

* flags = 0 Delete all files except those write-protected against owner; retain UFD clusters.
= –1 Delete all files regardless of protection code; release UFD clusters (privileged users only).

Nonprivileged users can use this flag to delete all files regardless of their protection codes, but cannot release UFD clusters.

** Nonprivileged users can specify the current account only. Be sure to include the project-programmer number; the system returns an error if both bytes are zero.

Data Returned

Except for a possible error in byte 0 of the FIRQB, no data is returned by UU.ZER.

Errors

BADNAM The specified device is ANSI magnetic tape, and the volume ID is either missing or invalid.

NOSUCH The account specified does not exist on the device specified.

NODEVC The device specified is not on the system.

NOTAVL The device specified exists on the system, but you cannot zero it for one of the following reasons:

1. A file is currently open on the device.
2. The device is currently reserved by another job.
3. The device or its controller is disabled.

BADFUO No project-programmer number was specified.

PRVIOL The unit is write-locked, the unit is locked and the caller is not privileged, or the caller is not privileged and FIRQB + FQPPN contains a project-programmer number other than the current account.

DEVNFS The specified device does not allow access by file name.

3.33 .WRITE — Write Data to File or Device – Not Privileged

Form

```
.WRITE
```

Function

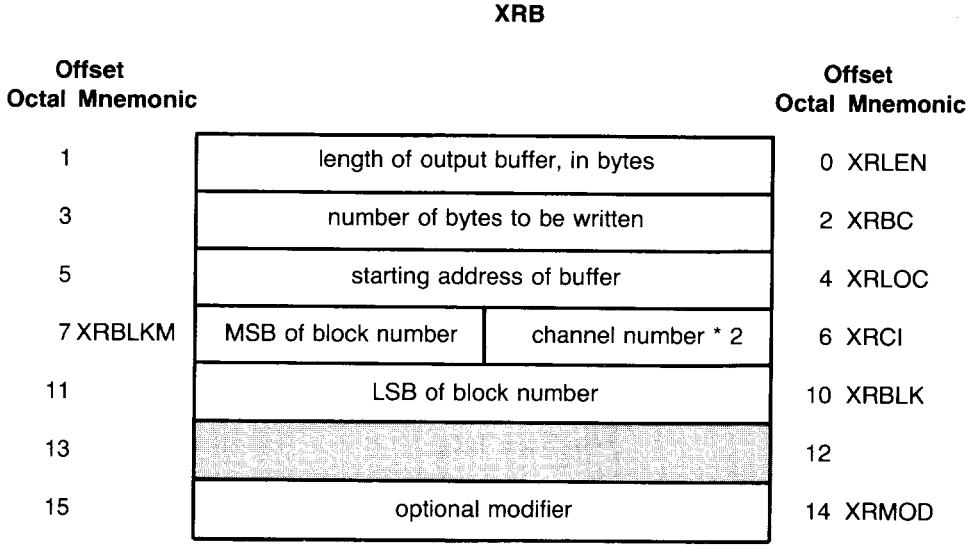
The .WRITE directive writes a specified number of bytes of data from a user buffer (defined in the XRB) to a file or device currently open on a channel. Unless an error occurs, the .WRITE always transfers the number of bytes specified. The number of bytes specified must be less than or equal to the size of the buffer. Specific details for each device are given in Table 3–8. The "best guess" buffer sizes (returned by the monitor at FIRQB+FQBUFL when the device was opened) are also shown in Table 3–8 for comparison.

Table 3–8: Data Output with .WRITE

Device	Block Size, in Bytes (Decimal)	"Best Guess*", in Bytes (Decimal)	Restrictions on Number of Bytes Written
Byte-Oriented Devices (FLGFRC = 1, FLGRND = 0)			
Keyboard (Terminal)	N/A	128	None
Pseudo-Keyboard	N/A	128	None
Paper Tape Punch	N/A	128	None
Line Printer	N/A	128	None
Block-Sequential Devices (FLGFRC = 0, FLGRND = 1)			
Magnetic Tape	18 to 30,000	512	14 – 32767 bytes
DECtape (file-structured)	510	510	Must be ≤ 510 bytes; if < 510, automatically filled with NULs to 510 bytes.
DMC/DMR	1 to 8000	512	None
Block-Random Devices (FLGFRC = 0, FLGRND = 0)			
Disk	512	512	Must be multiple of 512 bytes. (Must be 512 if writing UFD.)
Flexible Diskette (RX01 and RX02)	512 (block mode) 128 (RX01 or RX02 single-density sector mode) or 256 (RX02 double-density sector mode)	512	If not a multiple of block/sector size, 0 fill to end of block/sector.
DECtape (non-file-structured)	512	512	Must be ≤ 512 bytes; if < 512, automatically filled with NULs to 512 bytes.

* Returned at FIRQB + FQBUFL when file or device was opened.

Data Passed



XRB+XRLEN — Length of the output buffer, in bytes. The value of this word must be nonzero.

XRB+XRBC — The number of bytes to be written. The value of this word must be nonzero and less than or equal to the buffer size, as specified at offset XRB+XRLEN. For disk, this must be some multiple of 512_{10}.

XRB+XRLOC — Starting address of the output buffer. For disk, flexible diskette, and magtape devices, this address must be on a word (even) boundary. For all other devices, the buffer can begin on an odd address.

The buffer, as defined by XRLOC for its start and XRLOC+XRLEN–1 for its last byte, must lie wholly within either the user job image (low segment) or the run-time system's address space (high segment).

If the buffer is in the low segment, the address defined by the contents of XRB+XRLOC must be greater than 170 (octal) to avoid destroying job-context information used in swapping the job (Section 2.4).

If the buffer is in the high segment, it must not fall within the pseudo-vector region. That is, it must not fall above the location P.OFF (Section 2.5). In addition, the run-time system must currently be mapped read/write (see PF.RW bit description in P.FLAG word, Section 2.5).

XRB+XRCI Channel number times two; defines the channel on which the data is to be written, as previously defined in an open (OPNFQ, CREFQ, CRTFQ, CRBFQ functions of CALFIP, Section 3.2).

XRB+XRBLKM For random files greater than 65,535 bytes in length, this byte contains the most significant bits of the starting block number where the write is to begin. This byte is combined with the word at XRB+XRBLK to form a 24–bit field.

XRB+XRBLK Starting block number where the write is to begin. Performs the same action as the BLOCK option for disk or the RECORD option for flexible diskette and non-file-structured DECtape, as described in the *RSTS/E Programming Manual*. This parameter is ignored if the device is not a random-access device. If the device is random-access, and this field is nonzero, it is interpreted as the block number where the write is to start (1 to n, where n is the length of the file in 512–byte blocks). If zero, the next sequential block is written.

For disk devices opened as non-file-structured, this value is the starting device cluster number where the write is to begin.

XRB+XRMOD Output operation modifier; significant only for line printer, terminal, and DMC/DMR devices. (The monitor informs you with the FLGMOD bit of the flag word whether or not the device will accept modifiers.) Except for modifier 20000_8, which is useful to MACRO programmers only, this parameter performs the same action as the BASIC–PLUS RECORD modifier for these devices, as described in the *RSTS/E Programming Manual*.

Modifier 20000_8 provides a "no stall" option for line printer or terminal output. This modifier causes the monitor to return control to your program if an output stall is to occur on the device. You can determine the number of bytes still to be written by checking the contents of XRB+XRBC.

This modifier is useful for programs that must perform several different functions with optimal performance, such as a line printer spooler that performs message send/receive and prints files at the same time. When an output stall does occur, the program can perform other processing before trying to write the remaining bytes to the line printer or terminal. You may also find this modifier useful in multiterminal service applications.

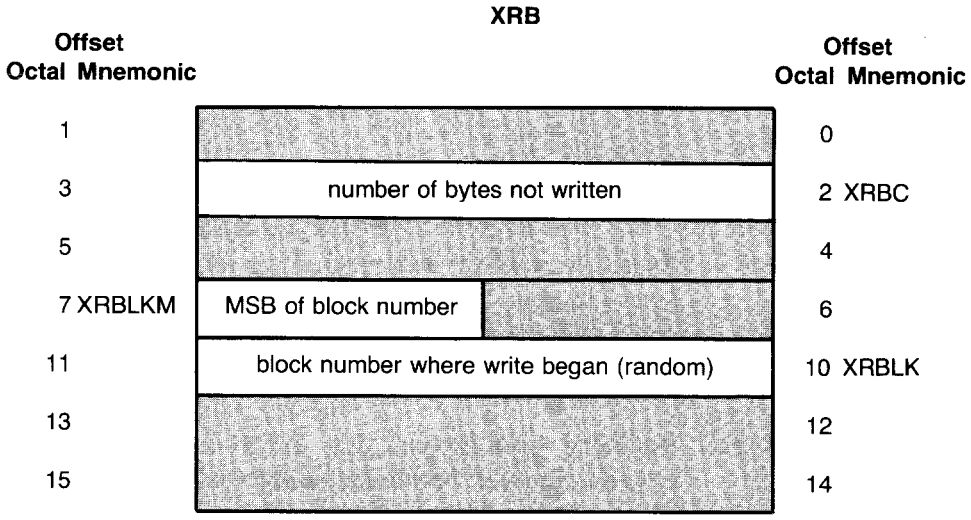
XRB+XRMOD (Cont.)

When you use the "no stall" option on a line printer, you can perform a special test to see whether the line printer is busy without causing your program to stall. To perform the test, write a null character and set modifier bits 20000_8 and 4. When both bits are set, the system will return control to your program instead of stalling it. The value at XRB+XRBC tells you the status of the line printer:

XRB+XRBC = 0	The line printer buffers are empty (that is, there are no characters still to print).
XRB+XRBC ≠ 0	The line printer buffers still contain one or more characters to print. Repeat the test until the system returns 0 at XRB+XRBC.

For more information on modifier 4 for line printers, see Appendix B and the *RSTS/E Programming Manual.* The *RSTS/E Programming Manual* also contains more information on special programming techniques for line printers.

Data Returned



XRB+XRBC — The number of bytes not written in this .WRITE operation. Except for terminals and line printers, this value is generally 0.

XRB+XRBLKM — For large disk files (greater than 65,535 bytes), the most significant bits of the block number of the block just written. (Combined with XRB+XRBLK to form a 24–bit field.)

XRB+XRBLK — For random-access devices, this word contains the block number of the block just written. Block numbers range from 1 to n, where n is the length of the file, in 512–byte blocks. They define the order in which the file is written.

For disk devices opened as non-file-structured, this is the device cluster number of the cluster just written.

Errors

BADCNT — The first three words of the XRB that describe the output buffer are illegal (an illegal byte count).

BSERR — The specified channel number is illegal; that is, it is not in the range $0 - 30_{10}$ (since channel is specified as channel * 2), or it is an odd number.

NOTOPN — No device is currently open on the specified channel.

PRVIOL — The file or device currently open on the specified channel is read-only, or the open for this file or device did not specify write access.

All other errors are device-dependent. Common errors are:

DATERR — Some data error occurred. This error is issued when a parity error occurs, and so forth.

HNGDEV — Some hard device I/O error occurred. For example, the specified device is off-line, is physically write-locked, and so forth.

NOROOM — No more room is available on the device. For example, the end-of-tape marker has been encountered, no more available disk space, and so forth.

Example

The following code writes the contents of a buffer to the file open on channel 5:

```
MOV             #1024.,XRB+XRLEN        ;SET BUFFER LENGTH
MOV             #1024.,XRB+XRBC         ;SET BYTES TO WRITE
MOV             #BUFFER,XRB+XRLOC       ;SET BUFFER ADDRESS
MOVB            #5*2,XRB+XRCI           ;SET CHANNEL = 5
.WRITE
```

PART III
RSX and RT11 Emulator Directives

Chapter 4
RSX Run-Time System Environment

4.1 Introduction

Chapter 1 describes the RSX run-time system as emulating the RSX operating system environment on RSTS/E. This is only partially true; the RSX run-time system does not provide a real-time, multiprocessing environment to users of a RSTS/E time-sharing system. However, the RSX run-time system does process directives that are identical in form, and similar in objective, to a subset of the RSX–11M monitor directives described in the *RSX–11M Executive Reference Manual.* The part of the RSX run-time system that handles these directives is called the RSX emulator.

If you use the RSX directives described in Chapter 5, you must assemble your program with the MAC assembler and link the modules with the task builder (TKB). The program can then be run as a user job image under the control of the RSX run-time system or its derivatives.

4.1.1 Advantage: Transportable Code

The RSX emulator directives are useful if you are coding a program to be run under both the RSX–11M and RSTS/E operating systems. Not all RSX–11M executive directives are emulated, however, and the meanings of those that are emulated are fitted to the RSTS/E environment.

So, to design a program to be run under both operating systems, you need to know how the directives work under RSX–11M and under RSTS/E. This manual describes what the directives do under RSTS/E. Some comparison is made here to RSX–11M, but you must refer to the *RSX–11M Executive Reference Manual* for a complete description of how these directives work in the RSX–11M environment.

Another benefit of using compatible directives is that programs may be executed on VAX/VMS under the RSX–11M Application Migration Executive (AME), which is similar in function to the RSX emulator. AME emulates most RSX–11M executive directives.

4.1.2 General Services

Besides transportable code, the RSX emulator directives also provide:

1. Non-File-Structured Input/Output. The RSX emulator directives handle only non-file-structured I/O (for example, terminal I/O). For this type of I/O, using the directives takes less memory than using RMS*. RMS routines are available to MACRO programmers through resident libraries, or they can be linked as part of the user job image. In either of these cases, the RMS routines take space in the job area. The RSX emulator directives that do I/O take no extra space, since the code to handle the processing is a part of the run-time system.

2. Trap Handling. The RSX emulator includes processing to handle synchronous and asynchronous system traps. In most cases, the emulator aborts the program when such traps occur and prints an error message at the job's terminal. With certain of the directives, you can request that this processing be bypassed, to handle such traps in your program. For example, you can specify an address to which control is to be passed when a Floating-Point Processor (FPP) exception trap occurs.

4.1.3 RSX Directive Emulation Within RSTS/E Monitor

There is an option at system generation to put RSX emulation capability into the RSTS/E monitor. This allows task images to execute without a run-time system, because the monitor handles the RSX emulation directly. As a result, task images may expand to be as large as (32K–32) words, or else a resident library (for example, RMS) may be used which occupies the address space otherwise taken by the run-time system.

Task images that execute using the RSX emulator in the monitor must be built to run with the RSX run-time system. This run-time system performs the initial task image loading, sets up the low-core parameters needed for RSX emulation, and then invokes the RSTS/E monitor with the .RSX directive. Since RSTS/E jobs are never without a run-time system, the monitor associates the job with a zero-length run-time system called "...RSX" for the duration of the task's execution. When the task ends, the job exits to its default keyboard monitor.

If your system does not have the RSX emulator in the monitor, then the RSX emulation is done by the run-time system.

---

*See the *RSTS/E RMS–11 MACRO Programmer's Guide* for a description of RMS (Record Management Services) capabilities for MACRO programmers.

4.2 System Macro Library

The RSX emulator directives that you code into your program are macro calls; they must be expanded into executable code at assembly time. The macro expansions for the RSX emulator directives are contained in the system macro library — LB:RSXMAC.SML. This library file can be named in the input-file list when the assembly is done. For example:

```
MAC OBJ,OBJ=SRC1,SRC2,LB:RSXMAC.SML/ML
```

However, the MAC assembler searches the library automatically to resolve undefined symbols, so this is not really necessary.

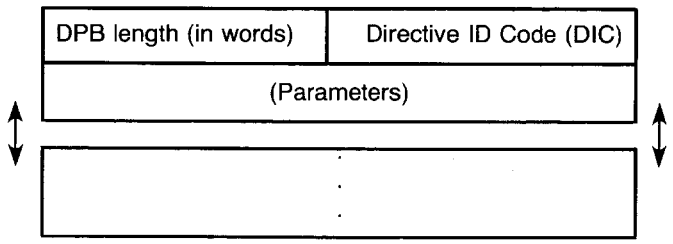
In your code, though, you must use the MACRO–11 .MCALL directive to define the directives you use as external macros needed to assemble the source program. The .MCALL must appear before the first directive is called. For example:

```
.MCALL  ALUN$,QIO$
        .
        .
        .
ALUN$   2,TT,5
        .
        .
        .
```

4.3 Directive Processing

At assembly time, the RSX directives are expanded to code that defines the action to be done and the parameters to be used in a Directive Parameter Block (DPB). Figure 4–1 shows the general form of a DPB.

Figure 4–1: General Form of the Directive Parameter Block (DPB)



The Directive Identification Code (DIC) in the low byte of the first word of the DPB defines the particular function to be performed. The RSX emulator examines the DIC to determine what it is supposed to do. The high byte of the first word of the DPB gives the length of the DPB in words (including the DIC and length byte). The remaining words contain parameters;

usually these are values specified by the user in the directive call, expanded through MACRO–11 data definition macros such as .BYTE, .WORD, .RAD50, and so forth.

The directive expansions also include executable code to tell the emulator where the DPB is, an EMT 377 instruction to pass control to the emulator and, optionally, transfer control to an error processing routine specified by the user.

At execution time, either the DPB address or the DPB itself is pushed on the stack. The choice depends on the form of the directive, as described in Section 4.4. The EMT 377 instruction passes control to the RSTS/E monitor. The monitor recognizes the EMT 377 and passes control to the RSX run-time system*. The RSX run-time system examines the DPB and processes the directive. It then pops the DPB address or the entire DPB off the stack and either returns control to the program that executed the directive or, if the directive was an exit of some kind, to whatever location is appropriate to the exit.

When control is returned in-line, the carry condition code in the program status word (PSW) indicates that the directive executed successfully (carry code = 0) or failed (carry code = 1). Further information on the success or failure of the directive is given in the Directive Status Word. You can refer to the Directive Status Word in your program with the mnemonic $DSW; this variable is defined and its location is resolved automatically at link time by the task builder (TKB). Other mnemonics are similarly related to the possible values of $DSW, so that the MACRO programmer can test for error conditions using symbols such as IS.SUC for successful completion. The symbols appropriate to each directive are listed under the heading "DSW Return Codes" in Chapter 5.

4.4 Directive Forms — $, $C, $S — and Their Expansions

The library file RSXMAC.SML contains three different expansions for each of the RSX–11M directives listed in Chapter 5. The form of the directive name you use determines which of the three expansions will be inserted in your code. Directive names consist of up to four letters, followed by a dollar sign ($) and, optionally, a C or an S. Figure 4–2 shows an example of the three forms for the ALUN$ directive (assign logical unit).

*Unless the system was generated with RSX emulation in the monitor.

Figure 4–2: Example of RSX Directive Forms

Your Code	Expansion	Description
;		
;	A: .BYTE 7,4	Create DPB at assembly
;	.WORD 5	time, in data section
$ FORM A: ALUN$ 5,TT,0	.ASCII /TT/	of program
;	.WORD 0	
;	;	
;	;	
DIR$ #A,ERR	MOV #A,-(SP)	Push DPB address on stack
	EMT 377	Pass control to RSX emulator
;	BCC .+6	Branch if no error
;	JSR PC,ERR	Jump to ERR on error
;	;	
	;	
DIR$ #A,ERR	(same as above)	(same as above)
;	.PSECT $DPB$$	Create DPB at assembly
;	$$$=.	time and automatically put
;	.BYTE 7,4	in separate PSECT called
$C FORM ALUN$C 5,TT,0,PSEC1,ERR	.WORD 5	$DPB$$
;	.ASCII /TT/	
;	.WORD 0	
;	.PSECT PSEC1	Reenter specified PSECT
	MOV #$$$,-(SP)	Push DPB address on stack
	EMT 377	Pass control to RSX emulator
	BCC .+6	Branch if no error
	JSR PC,ERR	Jump to ERR on error
;	MOV #0,-(SP)	Create DPB at execution
;	MOV #"TT,-(SP)	time and push entire DPB on
;	MOV #5,-(SP)	stack
$S FORM ALUN$S #5,#"TT,#0,ERR	MOV (PC)+,-(SP)	
;	.BYTE 7,4	
;	EMT 377	Pass control to RSX emulator
;	BCC .+6	Branch if no error
	JSR PC,ERR	Jump to error routine

4.4.1 $ Form (and DIR$ Directive)

As shown in Figure 4–2, the $ form of an RSX emulator directive generates a DPB in-line. The DIR$ directive expands to code that pushes the address of the DPB on the stack and passes control to the RSX emulator. The $ form/DIR$ combination is useful when the same directive is to be executed repeatedly. One DPB is defined and then invoked by as many DIR$ directives as needed. This saves memory. Also, you can modify the individual parameters in the DPB when the same directive is used many times with varying parameters.

$ Form

The $ form of an RSX emulator directive expands to code that defines the directive parameter block (DPB). The expansions consist of MACRO–11 data storage directives (for example, .BYTE, .WORD, .RAD50); they are not executable. Therefore, use the $ form of an RSX emulator directive only in a data section of your program.

Finally, use parameters that fit the MACRO–11 data storage directive in which they will appear in the expanded code. For example, the ALUN$ directive below expands to the four instructions that follow:

```
ALUN$ 5,TT,0
```

Expansion:

```
.BYTE     7,4
.WORD     5
.ASCII    /TT/
.WORD     0
```

Although the two-character second parameter (TT) is used in a MACRO–11 .ASCII directive, you do not need to specify the enclosing slashes; the expansion does that for you.

Note that the first word of the expansion (the first word of the DPB) consists of two bytes. The octal 7 in the low byte is the directive identification code (DIC) byte for the ALUN$ directive. The octal 4 in the high byte indicates that the DPB is 4 words long.

DIR$ Directive

The general form of the DIR$ directive is:

DIR$ [*adr*] [,*err*]

where:

adr is the address of the first word of the DPB. As shown in the expansion below, this address must be a valid source address for a MOV instruction. (The *adr* parameter can be omitted. In this case, the entire DPB must be on the stack when the DIR$ is executed.)

err is an optional error address. Control passes to this address if the directive does not execute successfully. If this parameter is omitted, control returns in-line with the C–bit set in the program status word (PSW).

For example, the DIR$ directive below expands to the four instructions that follow:

```
DIR$ #DPB,ERROR
```

Expansion:

```
MOV    #DPB,-(SP)
EMT    377
BCC    .+6
JSR    PC,ERROR
```

The address of the DPB is placed on the stack, and the EMT transfers control to the RSX emulator. The emulator processes the directive according to the parameters specified in the DPB, pops the DPB address off the stack, and (for ALUN$) returns control in-line. The BCC instruction branches around the JSR to ERROR if the C–bit is not set (no error occurred).

Other Features of $ Form/DIR$ Combination

The $ form of the RSX emulator directives generates local offsets that you can use to refer to parameters in the DPB.* For example, in the ALUN$ directive, you specify a logical unit number, physical device name, and device unit number:

```
LN5DEF:  ALUN$ 5,TT,0
```

When this directive is expanded, it includes code that allows you to refer to the parameters as:

`LN5DEF+A.LULU`	(logical unit number)
`LN5DEF+A.LUNA`	(device name)
`LN5DEF+A.LUNU`	(unit number)

In Chapter 5, these offset names are listed under "Local Symbol Definitions" for each directive. The number of bytes defined for each offset is given in parentheses following a description of the area referenced by the offset. For example:

A.LUNU — Logical unit number (2)

(You can also use offset names with the $C form of directives, but not with the $S form, since $S generates values that are pushed on the stack.)

The $ form expansions are also designed so that you can use them to simply set up the offset symbols to reference a DPB in another module. If you define the symbol $$$GLB in your program ($$$GLB = 0 will do), the $ form of any directive generates the symbolic offsets as global symbols and does not generate the DPB itself. Thus, for example, suppose another module is assembled with the code:

```
MYMY:  ALUN$ 5,TT,0
```

*The expansions use the MACRO–11 .NLIST directive so that the code that generates these offsets (not of interest to your program) does not show up in your assembly listings.

In the current module, define MYMY as a global symbol, define $$$GLB, specify an ALUN$, and you will be able to reference MYMY+A.LULU, and so forth.

4.4.2 $C Form

If you know the DPB parameters at assembly time and need to issue a directive only once, the $C form saves you the trouble of coding two directives: the $ form and DIR$. The $C form also saves execution time over the $S form, which uses MOV instructions to place the entire DPB on the stack (Section 4.4.3).

The $C form of the directives expands to code that:

1. Defines the DPB in a separate PSECT (called $DPB$$).
2. Returns to a user-specified PSECT. If no PSECT name is given, returns to the blank (unnamed) PSECT.
3. Pushes the DPB address on the stack.
4. Passes control to the RSX emulator.
5. On return, passes control to a user-specified error routine if the directive did not execute successfully. If no error routine is specified, no such code is generated.

The directive descriptions in Chapter 5 show, for the most part, the form and expansion for the $ form. To use the $C form, you must add the C to the directive name and, optionally, add a PSECT name and error address to the parameter list. For example, the ALUN$C directive below expands to the instructions that follow:

```
ALUN$C 5,TT,0,PART1,ERROR
```

Expansion:

```
          .PSECT   $DPB$$
$$$=.
          .BYTE    7,4
          .WORD    5
          .ASCII   /TT/
          .WORD    0
          .PSECT   PART1
          MOV      #$$$,-(SP)
          EMT      377
          BCC      .+6
          JSR      PC,ERROR
```

The DPB is placed in a separate PSECT called $DPB$$. The $$$ symbol is equated to the address of the DPB, which is then used in the MOV instruction that pushes the DPB address on the stack before control is passed to the emulator with the EMT 377. The last two instructions set up the test for error and transfer control to the specified error routine. If no error address had been specified in the call, these two instructions would not have been generated.

Other Features of $C Form

The offset mnemonics generated for the $ form are also generated for the $C form. You can save and use the $$$ address to form the base address for the offsets if you want to. The $$$GLB feature discussed for the $ form in Section 4.4.2 also works for the $C form of the directives.

4.4.3 $S Form

The $S form is useful if your code is reentrant and the DPB parameters can vary. The DPB is generated when the directive is executed: in other words, at run time. The $S form of the directives expands to code that:

1. Generates the DPB at execution time and pushes it on the stack.
2. Transfers control to the RSX emulator*.
3. On return, passes control to a user-specified error routine if the directive did not execute successfully. If no error routine is specified, no such code is generated.

To use the $S form of the directives, add S to the directive names shown in Chapter 5. You can also specify an error address as the last parameter in the line. For example, the ALUN$S directive below expands to the instructions that follow:

```
ALUN$S    #5,DEV,UN,ERR
```

Expansion:

```
MOV    UN,-(SP)
MOV    DEV,-(SP)
MOV    #5,-(SP)
MOV    (PC)+,-(SP)
.BYTE  7,4
EMT    377
BCC    +,6
JSR    PC,ERR
```

The DPB is generated at execution time and pushed on the stack. Control passes to the RSX emulator, and if control returns with an error (the C–bit is set on return), control transfers to the error routine. If no error address had been specified in the call, the last two instructions would not have been generated.

4.5 First 1000 Bytes of Low Segment for RSX

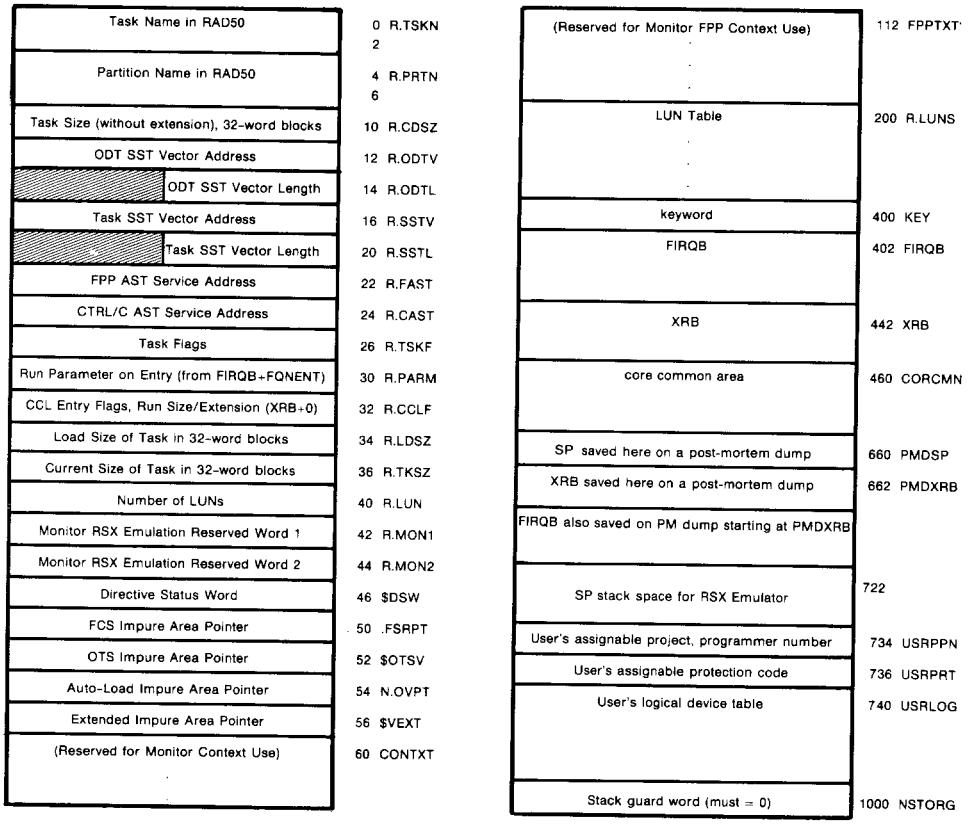
As noted in Section 2.4, the first 1000 (octal) bytes of virtual address space have special meaning to the monitor. The Task Builder (TKB), which links programs assembled or compiled under the RSX run-time system and its

*The RSX emulator checks the low byte of the first word on the stack to determine if it is a DPB address (even low byte) or a DIC (odd low byte). If the low byte is odd, then the emulator knows that the rest of the DPB is on the stack.

derivatives, generates a header for your executable program. When the RSX run-time system loads your program, it uses information from this header to load certain values into the low 1000 bytes of your program. Bytes 46–56 are loaded from the header information. Other mnemonics are assigned by COMMON.MAC.

Figure 4–3 shows the allocation of the first 1000 bytes of the low segment for RSX–type programs. Note that all locations are subject to change except for $DSW, KEY, FIRQB, XRB, CORCMN, USRPPN, USRPRT, and USRLOG.

Figure 4–3: First 1000 Bytes of Low Segment for RSX



Chapter 5
RSX Emulator Directives

5.1 Introduction

The RSX emulator directives:

1. Perform non-file-structured input/output
2. Let you specify your own trap routines
3. Control execution of the program
4. Return system information to the job
5. Provide access to resident libraries

5.1.1 Perform Non-File-Structured Input/Output

- ALUN$ assigns a logical unit number to a device and unit. The logical unit number then serves to define the device in I/O requests.
- GLUN$ returns information about a logical unit.
- QIO$ and QIOW$ open, read from, write to, and close logical units. Input/output in RSTS/E is "synchronous." That is, it is not "asynchronous" as in some systems such as RSX–11M where a user program can request I/O, do other processing, and get an interrupt when the I/O is complete. Thus, under RSTS/E, QIO$ (queue input/output) and QIOW$ (queue input/output and wait) do the same thing. Control returns to the calling program when the I/O is done.
- WTSE$ is provided for compatibility with RSX–11M, where QIO$ and WTSE$ can be used together to cause the program to wait until I/O is completed. Since I/O in RSTS/E is synchronous, the WTSE$ is implemented here as a "no-operation."

- GMCR$ returns the command line typed by the user to invoke the program currently running, if it was invoked by a RSTS/E CCL command.
- WSIG$ is provided for compatibility with RSX–11M, where some recoverable error conditions require waiting for some "significant event" to occur before retrying the operation that caused the error. In RSTS/E, such events are transparent to the user program. WSIG$ simply causes the program to sleep for one second.

5.1.2 Specify Trap Routines

- SFPA$ lets you specify an address to which control is to pass when an FPP exception trap occurs. (The FPP is the floating-point processor available on all PDP–11 processors that can run RSTS/E, except the PDP–11/35 and 40.)
- ASTX$ exits from an FPP trap-handling routine, restoring the directive status word (DSW), program counter (PC), and program status word (PSW) to the values they had before the trap occurred.
- SCCA$ defines an address to which control passes when a user at the job's terminal types a CTRL/C combination. The SCCA$ directive is unique to RSTS/E; it does not exist in the RSX–11M operating system.
- SVTK$ lets you specify a table of addresses to which control is to pass when synchronous system traps occur. Eight possible traps can be handled by the job, including PDP–11/35 and 40 floating-point-instruction-set (FIS) errors, TRAP instructions, breakpoint (BPT) instructions, IOT instructions, and T-bit traps.
- SVDB$ lets you specify a table of synchronous system trap addresses that are to override any specified with an SVTK$ directive. This is a useful capability for a debugging routine, to divert breakpoint traps, for example, to the debugging routine regardless of any SVTK$ directive in other modules of the job.

5.1.3 Control Program Execution

- EXIT$ returns control to the job's keyboard monitor (a normal exit).
- ABRT$ prints a "severe error" message at the job's terminal and returns control to the job's keyboard monitor.
- EXST$ prints a status message at the job's terminal and returns control to the job's keyboard monitor.
- EXTK$ increases or decreases the amount of memory allocated for execution of the user job image.
- SPND$ suspends execution of the program until the user types a delimiter on the keyboard (KB:).

5.1.4 Return System Information

- GTSK$ returns "task" parameters: run priority, project-programmer number, number of logical units assigned, address of SST vector table (set up the SVTK$ or SVDB$), and the system in which the job/task is running.
- GTIM$ returns the current day and time.
- GPRT$ returns "partition" parameters: general information about the job area and user job image area.

5.1.5 Access Resident Libraries

- ATRG$ attaches the job to a resident library, laying the groundwork for access to the library. If necessary, the resident library is loaded from disk.
- DTRG$ detaches the job from a resident library.
- CRAW$ creates a "window" of virtual addresses for reference to a library. Optionally, you can map the window to actual locations in a resident library.
- ELAW$ eliminates an address window; it releases the virtual addresses for other use, such as expanding the job image or creating another window.
- MAP$ relates a created address window to actual memory locations in an attached resident library. The user program can then refer to locations in the library using the virtual addresses defined in the window.
- UMAP$ unmaps a window from a library; you can then map the window to another portion of the library or to a different library.

The RSX emulator directives are described in alphabetical order in the remainder of this chapter.

5.2 ABRT$ — Abort

The ABRT$ directive terminates execution of the calling job. Any open files or devices are reset (that is, closed without the usual cleanup operations). On a RSTS/E system, ABRT$ acts the same as the EXST$ directive with a "severe error." Or, if EXST$ is not implemented, ABRT$ acts as a simple EXIT$.

Control is passed to the job keyboard monitor at the keyboard monitor entry point (P.NEW, Section 2.4).*

Macro Call

ABRT$ *name*

The *name* parameter is ignored in RSTS/E; the calling program can abort only itself.

Macro Expansion

```
ABRT$       ALPHA
.BYTE       83.,3       ;DIC=83., DPB SIZE = 3 WORDS
.RAD50      /ALPHA/     ;IGNORED IN RSTS/E
```

Local Symbol Definitions

A.BTTN name (4)

DSW Return Codes

IE.SDP DIC or DPB size is invalid. This does not occur unless your program changes the value of the first word of the DPB during execution. Control does not otherwise return in-line for the ABRT$ directive.

*The job keyboard monitor is the default keyboard monitor unless the job has run the SWITCH utility or executed the .RTS directive (Section 3.19) to establish a job keyboard monitor.

5.3 ALUN$ — Assign Logical Unit Number

The ALUN$ directive assigns a logical unit number (LUN) to a physical device unit. You can then use the logical unit number in QIO$ or QIOW$ directives to do I/O (open, read, write, or close the associated device).

The device name specified can be:

1. One of the following user logical device names: OV, LB, or CL. This name must be assigned by the user at a terminal with ASSIGN, as described in the *RSTS/E System User's Guide,* or by the job with the .ULOG UU.ASS subfunction (Section 3.31.1).
2. A standard RSTS/E device name (such as KB, PP, SY, or LP).
3. TI or TT, which are synonyms for KB.

The RSX emulator checks the name specified in the call against the user logical names; if a match is found, the logical unit number is assigned to the corresponding physical device. If the device name specified does not match any user logical device names, a similar search is performed for a match with a system-wide logical device name. If there is no match, the emulator checks the monitor's physical device table, and, if a match is found, assigns the logical unit number specified in the call to the actual physical device unit specified. Otherwise, the directive returns an error.

Macro Call

ALUN$ *lun,dev,unt*

where:

lun = Logical unit number; 1–14.

dev = Device name; two ASCII characters.

unt = Device unit number. –1 indicates no unit number, as opposed to a unit number of 0. RSTS/E makes a distinction between SY (meaning the public disk structure) and SY0 (meaning the disk from which the system was booted). Likewise, TI, TT, and KB all refer to the job's terminal, whereas KB0 and TT0 both refer to the system console terminal.

Macro Expansion

```
ALUN$       5,TT,0
.BYTE       7,4         ;ALUN$ MACRO DIC=7, DPB SIZE=4 WORDS
.WORD       5           ;LOGICAL UNIT NUMBER 5
.ASCII      /TT/        ;DEVICE NAME IS TT (TERMINAL)
.WORD       0           ;DEVICE UNIT NUMBER=0
```

ALUN$

Local Symbol Definitions

A.LULU Logical unit number (2)

A.LUNA Physical device name (2)

A.LUNU Physical device unit number (2)

DSW Return Codes

IS.SUC Successful completion

IE.IDU Invalid device and/or unit

IE.ILU Invalid logical unit number

IE.SDP DIC or DPB size is invalid

5.4 ASTX$ — AST Service Exit

In the RSTS/E environment, the ASTX$ directive can be used to terminate a routine that handles an asynchronous floating-point-processor trap, if the program has indicated its intention to handle such traps with an SFPA$ directive (Section 5.20). No other condition in RSTS/E causes an asynchronous trap from which an ASTX$ exit is useful. (The CTRL/C trap requires only an RTI instruction to exit properly. See SCCA$S, Section 5.19.)

When an ASTX$ is executed, the stack must be in the state:

SP → (DSW) at the time trap occurred
(PC) at the time trap occurred
(PS) at the time trap occurred
word SP pointed to before the trap

So, your routine must pop the Floating-Point Exception Code (FEC) and Floating-Point Address (FEA) pushed on the stack when the trap occurred, as described for the SFPA$ directive (Section 5.20).

When the ASTX$ directive is executed, the RSX emulator restores the DSW value to the DSW location and loads the PC and PS registers with their appropriate values. Thus, these values are popped from the stack, and control returns to the user program at the point where it left off when the trap occurred.

You can change the contents of the PC address in the stack before the ASTX$ is executed so that control is passed to some other point than where it left off. Be sure you know what you are doing, though, because it is hard to debug errors in an asynchronous trap service routine.

Macro Call

ASTX$ [*err*]

where:

err = Error routine address

Macro Expansion

```
ASTX$S    ERR
MOV       (PC)+,-(SP)       ;PUSH DPB ONTO THE STACK
,BYTE     115.,1            ;DIC = 115., DPB SIZE = 1 WORD
EMT       377               ;TRAP TO RSX EMULATOR
JSR       PC,ERR            ;JUMP TO ERR IF DIRECTIVE UNSUCCESSFUL
```

NOTE

The $S form is recommended for this directive because (1) only a one-word DPB is generated and (2) no BCC instruction is generated for the $S form (or the $C form). Thus, the $S form takes less space than the $form/DIR$ combination. And, since the one-word DPB requires only one MOV instruction, the $S form takes less space and no more execution time than the $C form.

Local Symbol Definitions

None

DSW Return Codes

IE.SDP DIC or DPB size invalid. This occurs only if your program changes the first word of the DPB at execution time.

5.5 ATRG$ — Attach Resident Library

The ATRG$ directive attaches the job to a resident library. The type of access (read-only or read/write) is specified in the call. If the calling job can access the library in that fashion*, the monitor sets up its own internal tables that lay the groundwork for the job to map windows to the library. Note, however, that the resident library does not take up space in the job area (virtual memory) with an attach. APRs are assigned — virtual memory in the job area is taken — when a window is created (CRAW$).

If the attach is successful, a resident library ID is returned to the job. This ID is used in other directives to detach the job from the library (DTRG$) or to map and unmap windows to and from the library (MAP$ and UMAP$). Once the job is attached to a library, the ATRG$ directive can be used simply to determine the library ID.

Up to five resident libraries can be attached to a job at any given time.

Macro Call

ATRG$ *adr*

where:

adr = Address of an 8–word area defining the library to be attached and the type of access desired. Information is also returned to this area by the ATRG$ directive. Two supplementary directives are available to define (RDBDF$) or define and fill (RDBBK$) such an 8–word area, called the resident library definition block or RDB. The RDBDF$ and RDBBK$ directives are described in Appendix C. The descriptions below show the offsets automatically defined by these directives.

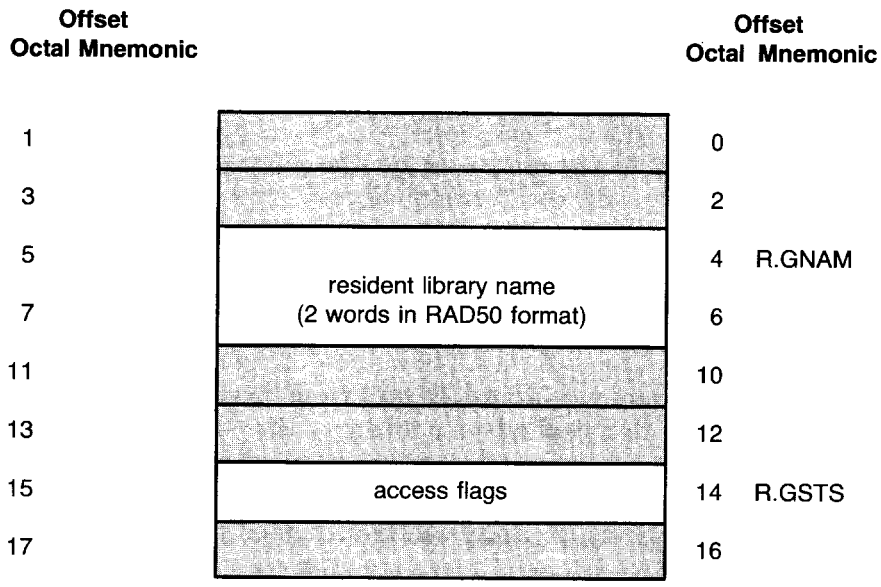
Macro Expansion

```
ATRG$   RDBLK
.BYTE   57.,2                   ;DIC = 57., DPB SIZE = 2 WORDS
.WORD   RDBLK                   ;ADDRESS OF RDB
```

*The job's ability to access the resident library depends upon the protection assigned by the system manager when the resident library was installed. The default protection grants read access to all users and denies write access to all users.

ATRG$

Data Passed in RDB



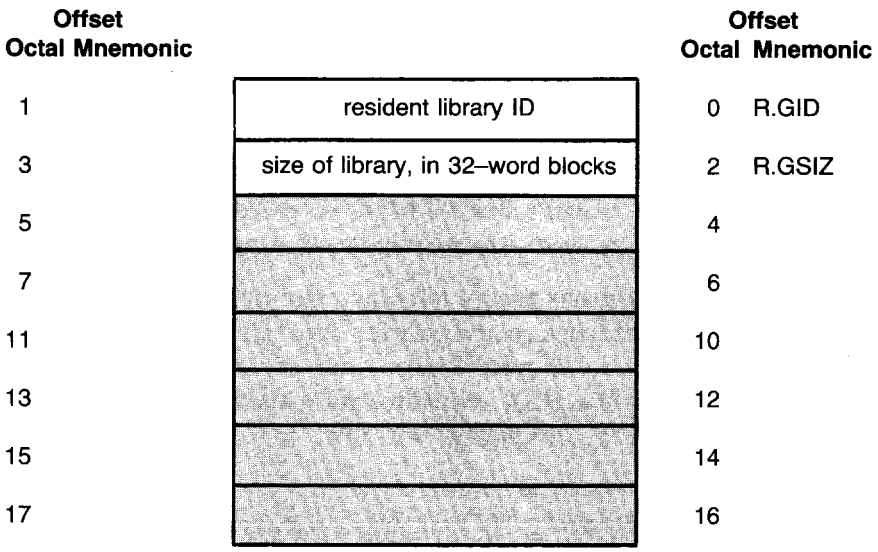
adr+R.GNAM The name of the resident library to which the job is to be attached, as two words of RAD50 data. (Resident libraries are made known to the RSTS/E monitor by the system manager with the ADD LIBRARY command of UTILTY (see the *RSTS/E System Manager's Guide*). With this command, the system manager defines a file (filename.LIB) as a resident library. The monitor regards "filename" as the resident library's name.)

adr+R.GSTS The low-order two bits in this word define the desired access to the library.

RDBBK$, RDBDF$ Mnemonic	Bit	Meaning
RS.RED	0 = 1	Read-only access is desired.
RS.WRT	1 = 1	Read/write access is desired.

Once a job is attached to a resident library, the access cannot be changed without detaching (DTRG$) and reattaching. Also, once the job is attached to a library, an ATRG$ with the same resident library name specified at *adr*+R.GNAM simply returns the assigned resident library ID.

Data Returned to RDB



adr+R.GID This word is an identifier which must be used in subsequent calls to identify a resident library, rather than the resident library name. Thus, you use this identifier to detach (DTRG$) and to map and unmap windows (MAP$ and UMAP$) to the library.

adr+R.GSIZ The size of the resident library, in 32–word blocks.

Local Symbol Definition

A.TRBA Resident library definition block address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.UPN Attaching a job to a resident library requires a small buffer and no small buffers are currently available.

IE.PRI The attach did not succeed because the caller's privileges do not allow the access requested. This could happen either because the access code specified is not compatible with the possible access defined when the library was installed by the system manager or because the protection code associated with the resident library file excludes the access requested by the user.

IE.PNS The resident library name specified is not known to the monitor. A resident library must be installed by the system manager before it can be used.

5.6 CRAW$ — Create Address Window

The CRAW$ directive can be used either to create a window (a range of virtual addresses) or to create a window and map it to a range of actual addresses in an attached resident library. You define the range of addresses by (1) naming a base APR, which defines the starting virtual address for the window, and (2) specifying the size of the window in 32–word blocks. Thus a window always begins on a 4K–word boundary in virtual memory. It may take more than 4K words, depending on the size of the window.

If the virtual address range overlaps the user job image, the call fails with an error. The address range cannot overlap the run-time system (high segment) unless it is the RSX run-time system on a system that has been generated with the RSX emulator as a part of the monitor. If the address range overlaps an existing window, the previously created window is eliminated.

The difference between (1) creating a window and (2) creating and mapping a window is best illustrated by example. By using create without map, you can define one window, which can be mapped to a library or portion of a library and then remapped using MAP$ to another portion of the same library or another library, as many times as desired. For example, suppose your program takes up 24K words, and you want to access a 24K–word resident library of data values. You can use create without map to set up a 4K–word window in APR 6. You can then map the window to the first 4K words of the library, process the data, map to the next 4K words of the library, and so forth.

If, on the other hand, you had a 4K–word program and still wished to access a 24K–word library, you could use CRAW$ to create a 24K–word window and map it to the entire library in APRs 1–6.

A job can create a maximum of seven windows. A window takes at least one APR. It may take more, depending on the size you specify for the window. Thus, the maximum of seven assumes seven 4K windows in APRs 1 through 7. APR 0 can never be used to create a window, since the user program takes at least this much space. As mentioned above, a window cannot overlap the user job image; thus, the size of the user job image determines the lowest base APR that can be used. If the program (user job image) is less than 4K words, APRs 1 and up (to the limit imposed by the run-time system boundary) can be used to create windows. If the user job image is between 4K words and 8K words, APRs 2 and up can be used to create windows, and so forth.

If you create a window that overlaps an already existing window, the old window is eliminated. For example, if you create a 6K–word window using a base APR of 5, the window uses APRs 5 and 6. If you then create a 4K–word window using a base APR of 6, the entire old window is eliminated. APR 5 is free for other use; APR 6 is used for the new window.

Macro Call

CRAW$ *adr*

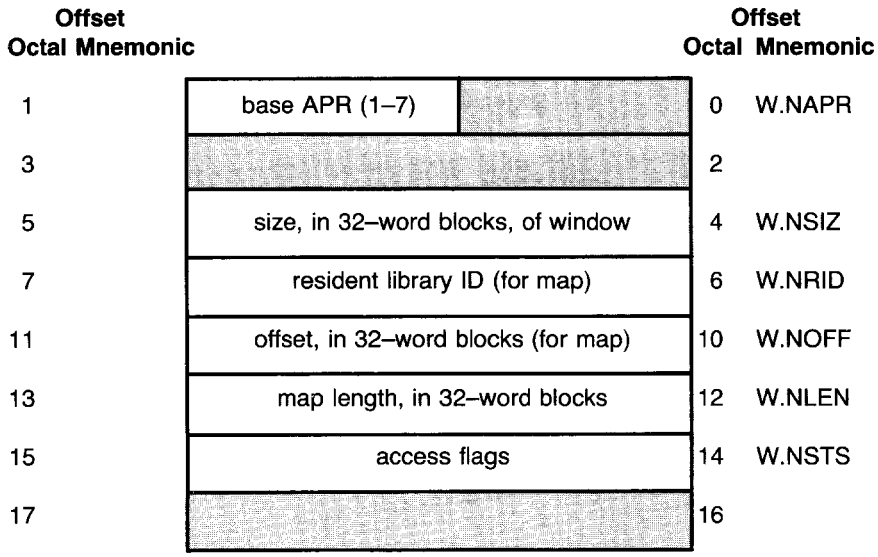
where:

adr = Address of an 8–word area defining the window and the resident library to which the window is to be mapped, if mapping is requested. Information is also returned to this area by the CRAW$ directive. Two supplementary directives are available to define (WDBDF$) or define and fill (WDBBK$) such an area, called the window definition block, or WDB. The WDBDF$ and WDBBK$ directives are described in Appendix C. The descriptions below show the offsets automatically defined by these directives.

Macro Expansion

```
CRAW$   WDB
.BYTE   117.,2          ;DIC = 117., DPB SIZE = 2 WORDS
.WORD   WDB             ;ADDRESS OF WDB
```

Data Passed in WDB



adr+W.NAPR The base APR of the window, 1–7. Implicitly defines the starting address of the window. This byte cannot be zero, nor can it name an APR already being used to map the user job image (see discussion above).

adr+W.NSIZ The desired size of the window, in 32–word blocks. For example, a value of 128_{10} = 4K words.

adr+W.NRID — The identifier of the resident library to which the window is to be mapped. (This is the value returned at *adr*+R.GID in the RDB for an ATRG$ directive.) This word is ignored for calls requesting a create without mapping (bit 7 at *adr*+W.NSTS, below, is zero).

adr+W.NOFF — The offset, in 32–word blocks, from the start of the library where the mapping is to begin. This word is ignored if no mapping is requested (bit 7 at *adr*+W.NSTS, below, is zero.) A value of zero for this word indicates the window is to be mapped beginning at the first byte of the library. A value of 1 indicates the window is to be mapped beginning at the 33rd word of the library (starting address + 64), and so forth.

adr+W.NLEN — The length, in 32–word blocks, of the area to be mapped (ignored if bit 7 at *adr*+W.NSTS, below, is zero). This value cannot be greater than the length of the window specified at *adr*+W.NSIZ. In addition, this value, combined with the offset specified at *adr*+W.NOFF, cannot indicate an address beyond the end of the library.

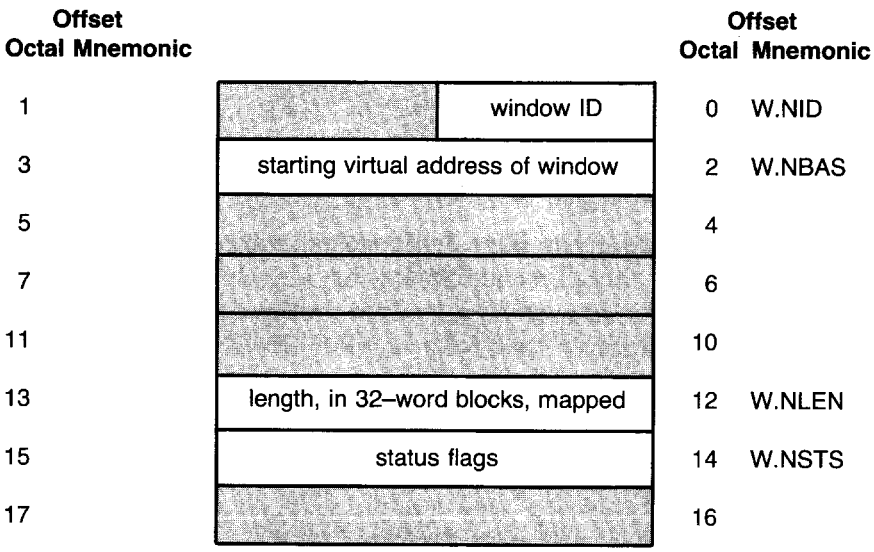
A value of zero defaults to either the size of the window or the space remaining in the library, whichever is smaller.

adr+W.NSTS — Two bits in this word define whether the window is to be mapped and whether read-only access or read/write access to the window is desired.

WDBDF$, WDBBK$

Mnemonic	Bit	Meaning
WS.MAP	7 = 1	The window is to be mapped.
	= 0	The window is not to be mapped
WS.WRT	1 = 1	Read/write access is desired.
	= 0	Read-only access is desired.

Data Returned in WDB



adr+W.NID Window ID, a value from 1–7. You can use this ID in later MAP$ calls to map the newly created window. You must use the ID in any ELAW$ calls to eliminate the window.

adr+W.NBAS The starting virtual address of the new window.

adr+W.NLEN The length, in 32–word blocks, actually mapped by the window.

adr+W.NSTS Status flags.

WDBDF$,
WDBBK$

Mnemonic	Bit	Meaning
WS.CRW	15 = 1	The window was created successfully.
	= 0	The window was not created.
WS.ELW	13 = 1	An existing window was eliminated because it overlapped the newly created window.
	= 0	No existing windows were eliminated by this create.
WS.UNM	14 = 1	An existing window was unmapped because it overlapped the newly created mapping.
	= 0	No existing windows were unmapped by this mapping.

Local Symbol Definitions

C.RABA Window definition block address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.ALG Either the base APR and window length specified were invalid or the offset and mapping length values specified were invalid. (For example, an offset indicating a starting address for the mapping that was beyond the end of the library returns this error.)

IE.NVR The library ID specified for mapping does not specify any library currently attached to the job.

IE.PRI The create was unsuccessful because the user privileges do not allow the access desired. At this point, since the library has been attached successfully with some access defined, this error means that the access requested in the CRAW$ is not allowed by the access requested in the ATRG$.

IE.WOV An attempt was made to create more than seven address windows.

IE.UPN Creating a window requires a small buffer; a small buffer is not currently available.

5.7 DTRG$ — Detach Resident Library

The DTRG$ directive detaches the job from a previously attached resident library. Any windows mapped to the library by the calling job are unmapped and eliminated. If no other jobs are currently attached to the library, and it is not a permanently resident library, the monitor removes the library from memory.

Macro Call

DTRG$ *adr*

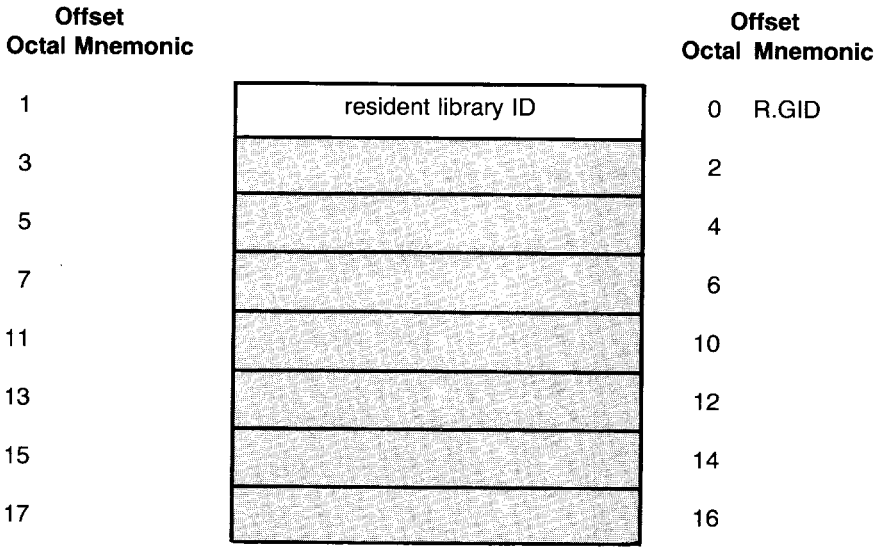
where:

adr = Address of an 8–word area defining the library to be detached. Information is also returned to this area by the DTRG$ directive. Two supplementary directives are available to define (RDBDF$) or define and fill (RDBBK$) such an 8–word area, called the resident library definition block, or RDB. The RDBDF$ and RDBBK$ directives are described in Appendix C. The descriptions below show the offsets automatically defined by these directives.

Macro Expansion

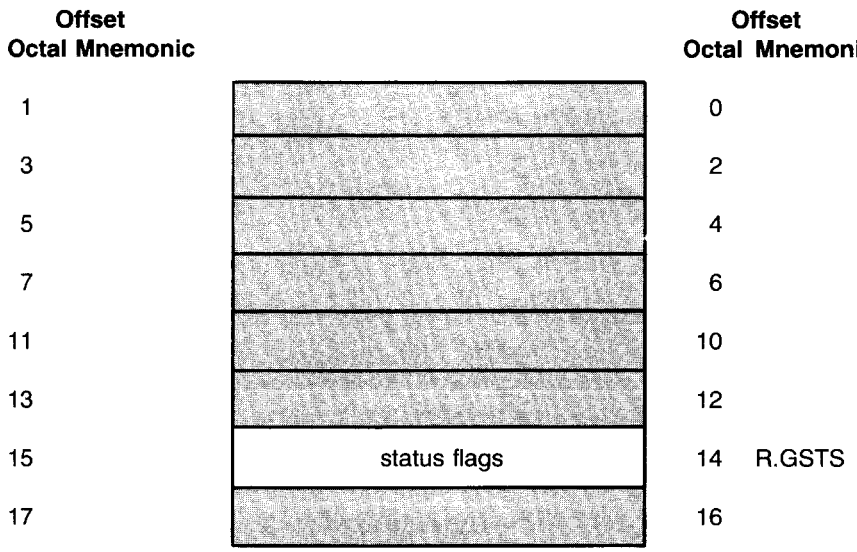
```
DTRG$   RDBLK
.BYTE   59.,2           ;DIC = 59., DPB SIZE = 2 WORDS
.WORD   RDBLK           ;ADDRESS OF RDB
```

Data Passed in RDB



adr+R.GID This word is the library identification returned (at the same position in the RDB) by the ATRG$ directive.

Data Returned in RDB



adr+R.GSTS Bit 14 = 1 (RS.UNM) if any windows were unmapped as a result of this detach.

Local Symbol Definitions

D.TRBA Resident library definition block address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.NVR The library ID specified does not identify any library currently attached to the job.

5.8 ELAW$ — Eliminate Address Window

The ELAW$ directive eliminates an address window that was created by the job, unmapping the window if necessary. ELAW$ frees the APRs used by the window and makes them available for creating another window or for expanding the user job image size.

Macro Call

ELAW$ *adr*

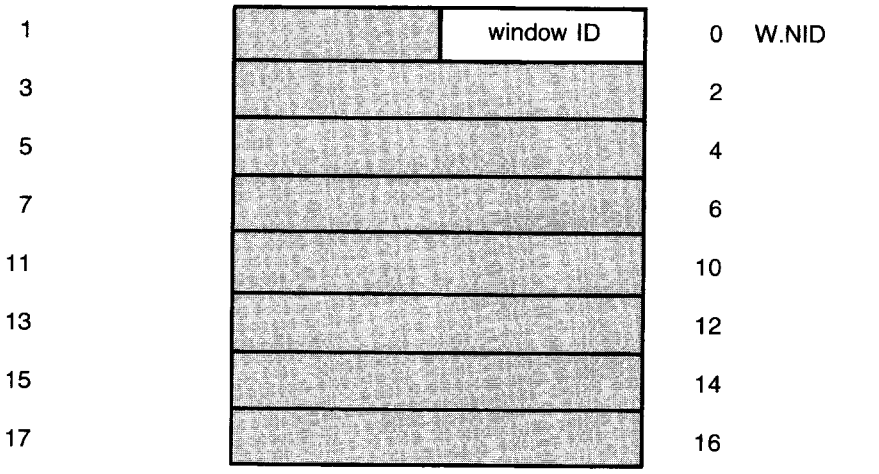
where:

adr = Address of an 8–word area defining the window to be eliminated. Information is also returned to this area by the ELAW$ directive. Two supplementary directives are available to define (WDBDF$) or define and fill (WDBBK$) such an area, called the window definition block or WDB. The WDBDF$ and WDBBK$ directives are described in Appendix C. The descriptions below show the offsets automatically defined by these directives.

Macro Expansion

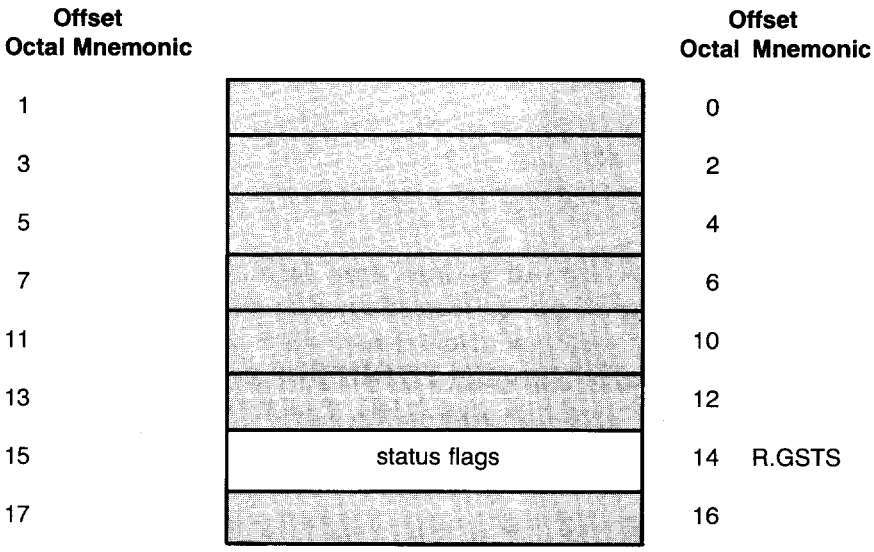
```
ELAW$   WDBADR
.BYTE   119.,2          ;DIC=119., DPB SIZE=2 WORDS
.WORD   WDBADR          ;ADDRESS OF WDB
```

Data Passed in WDB



adr+W.NID The ID of the window to be eliminated (returned at the same location in the WDB by the CRAW$ directive).

Data Returned in WDB



adr+W.NSTS Two bits in this word give status information:

WDBDF$,
WDBBK$

Mnemonic	Bit	Meaning
WS.ELW	13 = 1	The window was successfully eliminated.
	= 0	The window was not eliminated.
WS.UNM	14 = 1	The window eliminated was mapped to a resident library and has been unmapped.
	= 0	The window eliminated was not mapped; no unmapping was done.

Local Symbol Definitions

E.LABA Window definition block address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.NVW Bad window ID. Either outside the range 1–7 or matches no currently created window for this job.

5.9 EXIT$ — Task Exit

The EXIT$ directive terminates execution of the calling job. Control passes to the job keyboard monitor* at the keyboard monitor entry point (P.NEW, Section 2.5). Any devices still open are reset; that is, closed without performing the usual cleanup operations.

Macro Call

EXIT$S [*err*]

where:

err = Error routine address.

Macro Expansion

```
EXIT$S   ERR
MOV      (PC)+,-(SP)        ;PUSH DPB ONTO THE STACK
,BYTE    51.,1              ;DIC = 51., DPB SIZE = 1 WORD
EMT      377                ;TRAP TO THE EMULATOR
JSR      PC,ERR             ;CALL ERROR ROUTINE
```

NOTE

The $S form is recommended for this directive because (1) only a one-word DPB is generated and (2) no BCC instruction is generated for the $S form (or the $C form), since control never returns in-line unless an error occurs. Thus, the $S form takes less space than the $ form/DIR$ combination. And, because the one-word DPB requires only one MOV instruction, the $S form takes less space and no more execution time than the $C form.

Local Symbol Definitions

None

DSW Return Codes

IE.SDP DIC or DPB size is invalid. This occurs only if your program changes the first word of the DPB at execution time.

---

*The job keyboard monitor is the default keyboard monitor unless the job has run the SWITCH utility or has issued an .RTS directive (Section 3.19) to create a job keyboard monitor.

5.10 EXST$ — Exit with Status

Like EXIT$, the EXST$ directive terminates execution of the calling job. In addition, EXST$ may print a message at the job's terminal. The message varies according to a status value passed as a parameter in the call:

Status Value	Symbol	Message Printed
0	EX$WAR	%TASK EXIT STATUS: Warning
1	EX$SUC	(no message printed)
2	EX$ERR	%TASK EXIT STATUS: Error
4	EX$SEV	?TASK EXIT STATUS: Severe error

NOTE

If the job is detached when the EXST$ directive is executed, the job hibernates because of the attempt to print on the job's terminal.

Control passes to the job keyboard monitor* at the keyboard monitor entry point (P.NEW, Section 2.4). Any devices still open are reset; that is, closed without performing the usual cleanup operations. Any attached resident libraries are detached, and any address windows created are eliminated.

Some older run-time systems based on RSX (for example, RMS11.RTS for V6C) do not emulate this directive and return IE.SDP error code. To guard against this, code an EXIT$ directive following the EXST$ directive.

Macro Call

EXST$ *sts*

where:

sts = Status value defining the message to be printed on the job's terminal, as described above.

Macro Expansion

```
EXST$     STATS
.BYTE     29.,2       ;DIC=29., DPB SIZE = 2 WORDS
.WORD     STATS       ;EXIT STATUS WORD
```

*The job keyboard monitor is the default keyboard monitor unless the job has run the SWITCH utility or has issued an .RTS directive (Section 3.19) to create a job keyboard monitor.

Local Symbol Definitions

E.XSTS	Exit status word (2)
EX$WAR	Warning status code
EX$SUC	Success status code
EX$ERR	Error status code
EX$SEV	Severe error status code

DSW Return Codes

No errors are possible with EXST$.

5.11 EXTK$ — Extend Task

The EXTK$ directive lets you increase the amount of memory allocated to the user job image. For compatibility with RSX–11M, the argument specified in the call indicates a positive or negative increment of 32–word blocks. The RSTS/E monitor, however, actually allocates space for the user job image in 1K–word units. So the RSX emulator keeps track of the user job image size as a number of 32–word blocks. When an EXTK$ directive is executed, the emulator checks to see if the extension or reduction needs or frees one or more 1K–word blocks of memory. If so, the emulator directs the monitor to make the new allocation. If not, the current allocation remains.

NOTE

You should not use the .CORE directive (Section 3.6) before an EXTK$ directive in the same program. The .CORE directive is executed directly by the monitor, without the intervention (or knowledge) of the RSX emulator. If a .CORE is executed before an EXTK$, the emulator has incorrect information on the current size of the user job image when it executed the EXTK$, and it bases the new allocation on the size before the .CORE was executed.

You cannot extend the size beyond the smallest of:

1. The job's private maximum size. (A private job maximum size can be set by the system manager.)
2. Maximum size allowed by the RSX run-time system. If RSX emulation was not installed as part of the monitor, this limit is 28K words. If RSX emulation was installed as part of the monitor, this limit is (32K–32) words.
3. The system-wide maximum user job image size. (Set by the system manager; see SWAP MAX, *RSTS/E System Generation Manual.*)

You also cannot extend the size of the image into virtual address space used by an address window created for use with resident libraries (see CRAW$ directive, Section 5.6).

If there is not enough contiguous memory for the extension to be made where the job currently is in memory, the user job image is swapped out to disk and then returned to memory at the new size.

If no increment is given in the call, the emulator allocates the amount of space the job had initially, based on information built in at link time (with TKB); that is, before any EXTK$ was executed.

Macro Call

EXTK$ [*inc*]

where:

inc = The number of 32–word blocks by which the task size is to be extended (positive value for *inc*) or reduced (negative value for *inc*).

Macro Expansion

```
EXTK$       40
.BYTE       89.,3       ;DIC = 89., DPB SIZE = 3 WORDS
.WORD       40          ;EXTEND 40(8) BLOCKS (1K WORDS)
.WORD       0           ;RESERVED WORD
```

Local Symbol Definitions

E.XTIN Extend increment (2)

DSW Return Codes

IE.SUC Successful completion.

IE.UPN The requested increment would have caused the user job image size to exceed that allowed (see discussion, above). The user job image size stays at the current size.

5.12 GLUN$ — Get LUN Information

The GLUN$ directive fills a 6–word buffer with information about a physical device unit to which a logical unit (LUN) is assigned.

Macro Call

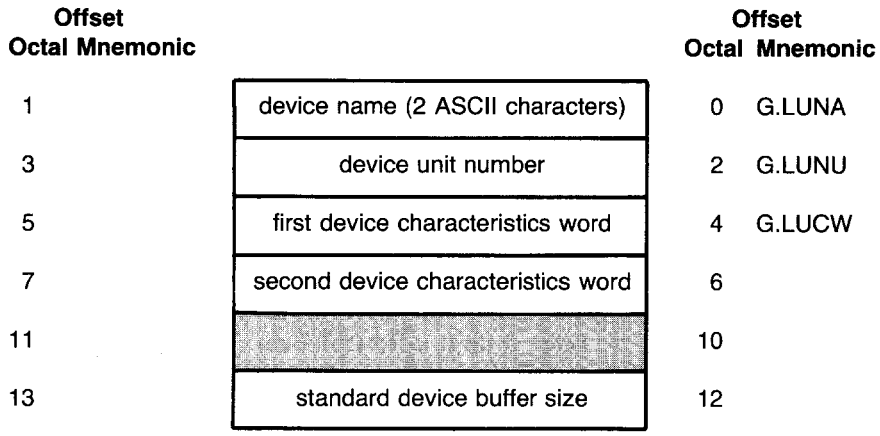
GLUN$ *lun,buf*

where:

lun = Logical unit number.

buf = Address of 6–word buffer to receive the information.

Buffer Format



buf+0 Name of the assigned device; two ASCII characters. This must be a valid physical device name. Exceptions are the logical device names OV and LB, which default to SY if they are not defined, and CL, which defaults to TT.

buf+2 Unit number of the assigned device. If this word is –1, it indicates there is no unit number. If this word is 0 or a positive number, it is a real unit number.

buf+4 First device characteristics word:

Bit 0 = 1, record-oriented device
Bit 1 = 1, carriage-control device
Bit 2 = 1, terminal device
Bit 3 = 1, directory device
Bit 4 = 1, single-directory device
Bit 5 = 1, sequential device
Bits 6–15, reserved

Settings for bits 0–5 are (in octal):

Terminal = 03
Disk = 10
Card Reader = 01
Line printer = 02
Magtape = 40
Paper tape reader/punch = 01

buf+6 If this word is 0, it indicates the job is nonprivileged. If this word is 10 (octal), it indicates the job is privileged.

buf+12 The standard device buffer size to use for input or output on this device.

Macro Expansion

```
GLUN$       7,LUNBUF
.BYTE       5,3         ;DIC = 5, DPB SIZE = 3 WORDS
.WORD       7           ;LOGICAL UNIT NUMBER 7
.WORD       LUNBUF      ;ADDRESS of 6-WORD BUFFER
```

Local Symbol Definitions

G.LULU Logical unit number (2)

G.LUBA Buffer address (2)

The following offsets are assigned relative to the start of the LUN information buffer:

G.LUNA Device name (2)

G.LUNU Device unit number (1)

G.LUFB Flags byte (not used in RSTS/E) (1)

G.LUCW Four device characteristics words (8)

DSW Return Codes

IS.SUC Successful completion

IE.ULN Unassigned LUN

IE.ILU Invalid logical unit number (must be in range $1–14_{10}$)

IE.SDP DIC or DPB size is invalid

5.13 GMCR$ — Get MCR (CCL) Command Line

The GMCR$ directive transfers an 80–byte CCL command line to the job. When a program file is installed in RSTS/E to be invoked with a CCL command, it must contain code to analyze what the user typed to invoke the file. The GMCR$ directive returns the command line typed by the user in an 80–byte buffer in the DPB.

This macro has no $S form. Because the DPB receives the CCL command line, the space must be allocated in read/write memory.

Macro Call

GMCR$

Macro Expansion

```
GMCR$
.BYTE     127.,41.    ;DIC = 127., DPB SIZE = 41. WORDS
.BLKW     40.         ;80-CHARACTER CCL COMMAND BUFFER
```

Local Symbol Definitions

G.MCRB CCL command buffer (80)

DSW Return Codes

+n	Successful completion. The value n is the number of data bytes transferred (excluding the termination character).
IE.AST	Task was not run as a CCL command, or the GMCR$ directive has already been executed.
IE.SDP	DIC or DPB size is invalid.

5.14 GPRT$ — Get Partition (Job) Parameters

The GPRT$ directive fills a 3–word buffer with "partition" parameters. For RSTS/E, a partition is considered to correspond to the job area.

Macro Call

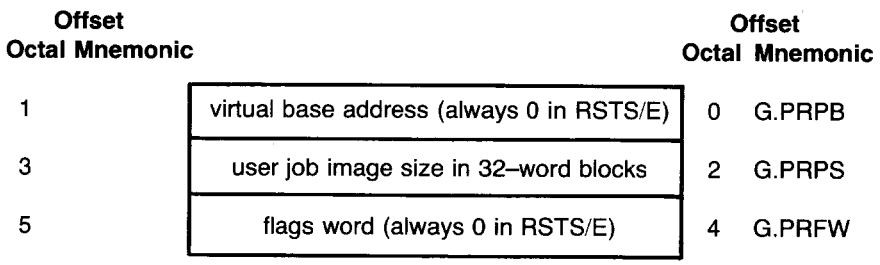
GPRT$ [*name*],*buf*

where:

name = Partition name. Omit to get actual job image size in *buf*+2.

buf = Address of 3–word buffer where information is to be returned.

Buffer Format



name = Partition name. Omit to get actual job image size in buf+2.

buf+2 "Partition size" (user job image size in RSTS/E), expressed as a multiple of 32 words. If you give a partition name, the system SWAP MAX is returned, in 32-word units.

buf+4 "Partition flags word." Always returned as 0, since all jobs are system-controlled in RSTS/E. (RSX–11M returns 1 for a user-controlled partition.)

Macro Expansion

```
GPRT$     ,DATBUF
,BYTE     65.,4       ;DIC=65.,  DPB SIZE = 4
,WORD     0,0         ;GENERATE 0 WORDS IF NO NAME
,WORD     DATBUF      ;ADDRESS OF 3-WORD BUFFER
```

Local Symbol Definitions

G.PRPN Partition name (4)

G.PRBA Buffer address (2)

The following offsets are assigned relative to the start of the buffer:

G.PRPB Partition virtual base address (2)

G.PRPS Partition size (2)

G.PRFW Partition flags word (2)

DSW Return Codes

Successful completion is indicated by carry clear, and the starting address of the partition is returned in the DSW (always 0 in RSTS/E). Unsuccessful completion is indicated by carry set and the following code in the DSW:

ID.SDP DIC or DPB size is invalid.

5.15 GTIM$ — Get Time Parameters

The GTIM$ directive fills an 8–word buffer with the current day and time of day. All time values are stored as one-word binary integers. The value ranges (in decimal) are shown below.

Macro Call

GTIM$ *buf*

where:

buf = Address of 8–word buffer.

Buffer Format

Offset Octal	Mnemonic		Offset Octal	Mnemonic
1		year (since 1900); such as 78_{10} = 1978	0	G.TIYR
3		month ($1–12_{10}$); such as 1 = January	2	G.TIMO
5		day of month ($1–31_{10}$)	4	G.TIDA
7		hour ($0–23_{10}$); such as 0 = midnight	6	G.TIHR
11		minute ($0–59_{10}$)	10	G.TIMI
13		second ($0–59_{10}$)	12	G.TISC
15		tick of second*	14	G.TICT
17		ticks per second*	16	G.TICP

Macro Expansion

```
GTIM$     DATBUF
.BYTE     61.,2       ;DIC = 61., DPB SIZE = 2 WORDS
.WORD     DATBUF      ;ADDRESS OF 8-WORD BUFFER
```

*A tick is either 1/60th or 1/50th of a second, depending on the clock in use and/or the line frequency. (Systems running with the KW11P clock at crystal speeds, rather than at line frequency, have a "tick" of 1/50th of a second. If the system is operating off a 60 Hz power line, a "tick" is 1/60th of a second.)

Local Symbol Definitions

G.TIBA Buffer address (2)

The following offsets are assigned relative to the start of the buffer:

G.TIYR Year (2)

G.TIMO Month (2)

G.TIDA Day (2)

G.TIHR Hour (2)

G.TIMI Minute (2)

G.TISC Second (2)

G.TICT Clock tick of second (2)

G.TICP Clock ticks per second (2)

DSW Return Codes

IS.SUC Successful completion.

IE.SDP DIC or DPB size is invalid.

5.16 GTSK$ — Get Task (Job) Parameters

The GTSK$ directive fills a 16–word buffer with parameters for the "task"; as with "partition" in GPRT$, "task" in the RSTS/E environment is considered to be equivalent to the job.

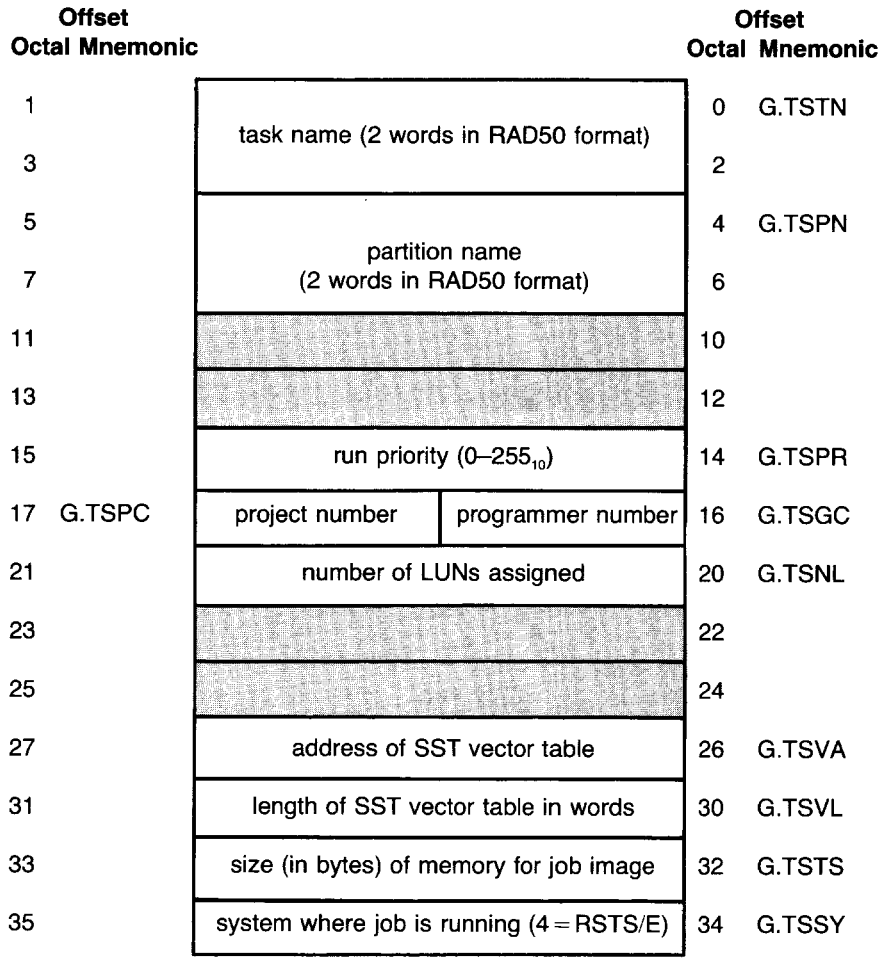
Macro Call

GTSK$ *buf*

where:

buf = Address of the 16–word buffer to receive the parameters.

Buffer Format



buf+0 Task name. The name supplied for the TASK=name option at the time the task was built (linked) with TKB; otherwise, the file name of the file being run, in RAD50 format.

buf+4 Partition name specified for the PAR=name option at the time the task was built (linked) with TKB. If no such name was specified, the characters "GEN " (GEN followed by three blanks), to indicate the general partition, the default system-controlled partition in RSX–11M.

buf+14 Run priority; may range from 0 to 255 (decimal). This is the RSTS/E job priority plus 128 (decimal).

buf+16 Programmer number of the user currently running the program.

buf+17 Project number of the user currently running the program.

buf+20 Number of logical unit numbers (LUNs) currently assigned.

buf+26 Address of SST vector table. If *buf*+30 = 0, this word is meaningless.

buf+30 Length of SST vector table, in words. If there is no SST vector table, this word is set to 0.

buf+32 Job size, in bytes.

buf+34 System in which the program/job/task is running:

0 = RSX–11D
1 = RSX–11M
2 = RSX–11S
3 = IAS
4 = RSTS/E
5 = VAX/VMS
6 = RSX–11M–PLUS
7 = RT–11

This value is always 4 on RSTS/E systems.

Macro Expansion

```
GTSK$     DATBUF
.BYTE     63.,2      ;DIC=63., DPB SIZE = 2 WORDS
.WORD     DATBUF     ;ADDRESS OF 16-WORD BUFFER
```

Local Symbol Definitions

G.TSBA Buffer Address

The following offsets are assigned relative to the buffer:

G.TSTN Task name (4)

G.TSPN Partition name (4)

G.TSPR Priority (2)

Local Symbol Definitions (Cont.)

G.TSGC Programmer number (1)

G.TSPC Project number (1)

G.TSNL Number of logical units (2)

G.TSVA SST vector address (2)

G.TSVL Length of SST vector table (2)

G.TSTS Size of space for user job image (2)

G.TSSY System being run under (2)

DSW Return Codes

IS.SUC Successful completion

IE.SDP DIC or DPB size is invalid

5.17 MAP$ — Map Address Window

The MAP$ directive maps a previously created address window to an attached resident library. In other words, MAP$ relates the virtual address range defined by a CRAW$ directive (Section 5.6) to actual locations in memory within a resident library that has been attached to the job by an ATRG$ directive (Section 5.8).

If the window specified is already mapped, it is unmapped from its previous actual memory locations and remapped to the new area. A job may map a maximum of seven address windows at any given time.

Macro Call

MAP$ *adr*

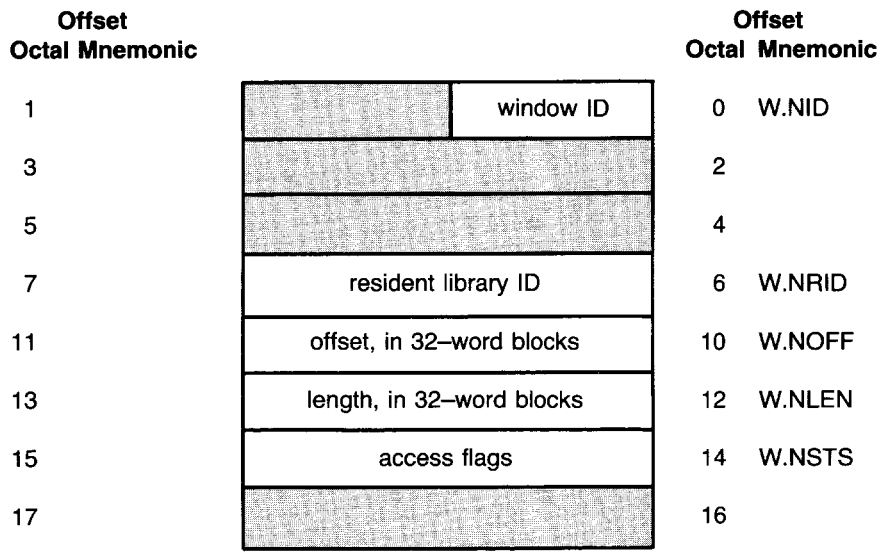
where:

adr = Address of an 8–word area defining the window to be mapped. Information is also returned to this area by the MAP$ directive. Two supplementary directives are available to define (WDBDF$) or define and fill (WDBBK$) such an area, called the window definition block or WDB. The WDBDF$ and WDBBK$ directives are described in Appendix C. The descriptions below show the offsets automatically defined by these directives.

Macro Expansion

```
MAP$      WDBADR
.BYTE     121.,2          ;DIC=121., DPB SIZE=2 WORDS
```

Data Passed in WDB



adr+W.NID — The ID of the window to be mapped (returned at the same location in the WDB by the CRAW$ directive).

adr+W.NRID — The ID of the resident library to which the window is to be mapped (returned at offset R.GID in the RDB by the ATRG$ directive that attached the job to the resident library).

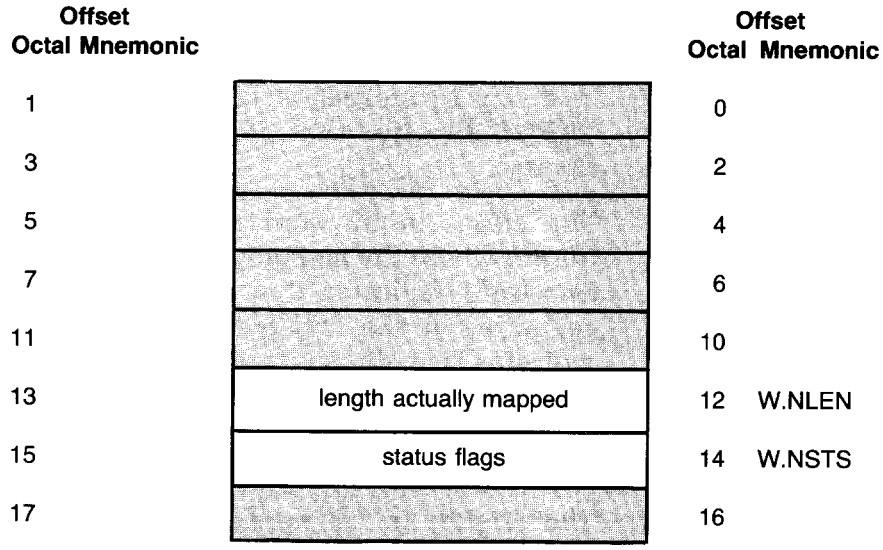
adr+W.NOFF — The offset, in 32–word blocks, from the start of the library where the mapping is to begin. A value of zero for this word indicates that the window is to be mapped beginning at the first byte of the library. A value of 1 indicates that the window is to be mapped beginning at the 33rd word of the library (starting address +64), and so forth. The offset cannot indicate a starting address past the end of a library.

adr+W.NLEN — The length, in 32–word blocks, to be mapped to the library. This value cannot be greater than the size of the window defined in the CRAW$ directive with which the window was created. In addition, this value, combined with the offset value at *adr*+W.NOFF, cannot indicate an address beyond the end of the library.

A value of 0 for this word defaults to the size of the window or the space remaining in the library, whichever is smaller.

adr+W.NSTS — Set bit 1 = 1 (WS.WRT) for read/write access. A separate setting for access in MAP$ and in ATRG$ allows you to attach to a library read/write and map a portion of the library read-only. You cannot, however, attach to a library read-only and then map to the library read/write.

Data Returned in WDB



adr+W.NLEN Length, in 32–word blocks, actually mapped by the call.

adr+W.NSTS Bit 14 = 1 (WS.UNM) if the window specified was unmapped before the new map.

Local Symbol Definitions

M.APBA Window definition block address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.ALG The offset and length specified are inconsistent; either the mapping attempted to go beyond the end of the library, or the length is greater than the created window.

IE.PRI The mapping did not succeed because user privileges did not allow the access desired.

IE.NVW Either the resident library ID or the address window ID is incorrect. (The job is not currently attached to the specified resident library, or no address window has been created with the specified window ID.)

5.18 QIO$ and QIOW$ — Queue I/O Request (and Wait)

The QIO$ and QIOW$ directives do non-file-structured input/output. In the RSTS/E environment, QIO$ and QIOW$ do the same thing. Input/output in RSTS/E is "synchronous." That is, you cannot request I/O, do other processing, and get an interrupt when the I/O completes. In RSTS/E, control returns to the program only when the I/O is complete.

As "non-file-structured" implies, the QIO$ and QIOW$ directives are primarily useful for input/output on a terminal, card reader, paper tape reader/punch, or line printer.

Macro Call

$$\left\{\begin{matrix}\text{QIO\$}\\ \\ \text{QIOW\$}\end{matrix}\right\} \quad fnc,lun,[efn],[pri],[isb],[ast][,par]$$

where:

fnc = I/O function code:

IO.ATT	Attach. In RSTS/E, opens the device specified by the logical unit number (lun).
IO.DET	Detach. In RSTS/E, closes the device specified by the logical unit number (lun).
IO.RLB IO.RVB	Read. IO.RLB and IO.RVB do the same thing in RSTS/E. The actions performed depend on the device, as described below.
IO.RAL	For terminals only, performs a read in ODT mode. (See .TTDDT, Section 3.27.)
IO.WAL	For terminals only, performs a write in binary mode. (See RECORD 4096% for terminal output, *RSTS/E Programming Manual.*)
IO.WLB IO.WVB	Write. IO.WLB and IO.WVB do the same thing in RSTS/E. The actions performed depend on the device, as described below.

lun = Logical unit number; defines the device for which the I/O operation is to be performed. Must have been previously defined with an ALUN$ directive.

efn = Event flag. (Ignored in RSTS/E.)

pri = Priority. (Ignored in RSTS/E.)

isb = I/O status block. Address of a 2–word buffer to which information is returned about the outcome of the I/O operation. The contents vary according to the function code and device, as described below.

ast = Asynchronous trap address. (Ignored in RSTS/E.)

par = Parameter list. Of the form <p1,p2[,,,,p3]> for RSTS/E users. Form and meaning varies according to function code and device, as described below.

Format and Description by Function Code

{QIO$ / QIOW$} IO.ATT,*lun*[,,*isb*]

Opens the device specified by the logical unit number (*lun*). Status information is returned to the 2–word buffer at address *isb*.

{QIO$ / QIOW$} IO.DET,*lun*[,,*isb*]

Closes the device specified by the logical unit number (*lun*). Status information is returned to the 2–word buffer at address *isb*.

{QIO$ / QIOW$} {IO.RLB / IO.RVB} ,*lun*,,[*isb*],,,<*stadd*,*size*[,,,*block*]>

For devices termed "record-oriented" in the RSX–11M environment (see GLUN$, Section 5.12), IO.RLB and IO.RVB return a record (a line on a terminal, card on a card reader, record on a paper tape reader). The data is read into the buffer whose starting address is defined by *stadd* and whose length in bytes is defined by *size*. The amount of data read never exceeds the buffer size. If the buffer size is smaller than the "record," the next IO.RLB or IO.RVB for that *lun* reads another buffer-full. The block parameter is omitted for these devices.

For disk, IO.RLB and IO.RVB fill the buffer whose starting address is defined by *stadd* and whose length in bytes is defined by size. The read begins with the device cluster number specified by the block parameter. If *block* is omitted or 0, sequential reads are performed and the next read begins with the next device cluster. The buffer size should be a multiple of the device cluster size for the disk (see Appendix B).

NOTE

Only privileged jobs can do non-file-structured disk I/O.

For all devices, the *isb* parameter is the address of a 2–word buffer to which status information on the read is returned. If *isb* is omitted, no such status is returned.

$$\left\{\begin{matrix}\text{QIO\$}\\ \text{QIOW\$}\end{matrix}\right\}\left\{\begin{matrix}\text{IO.WLB}\\ \text{IO.WVB}\end{matrix}\right\}\ \textit{,lun,,[isb],,,<stadd,size[,vfc]>}$$

For devices termed "record-oriented" in the environment (see GLUN$, Section 5.12), IO.WLB and IO.WVB write the contents of the buffer to the device specified by the logical unit number (*lun*). The starting address for the buffer is defined by *stadd*, and the length in bytes is defined by *size*. For devices which are "record-oriented" and "carriage-control" devices, the *vfc* parameter specifies an action to be performed after the buffer contents are written. (The *vfc* parameter is a vertical format control character for terminals and line printers.) Values for *vfc* and actions taken are listed in Table 5–1. The *vfc* parameter is omitted for devices that are not carriage-controlled.

For disk, IO.WLB and IO.WVB write the contents of the buffer to disk, beginning at the device cluster number specified by the *block* parameter. If *block* is omitted, sequential writes are performed. The next write (for sequential writes) will begin with the next device cluster. The buffer size for disk writes must be a multiple of the device cluster size.

NOTE

No job can write to disk in non-file-structured mode while any other user is accessing the disk.

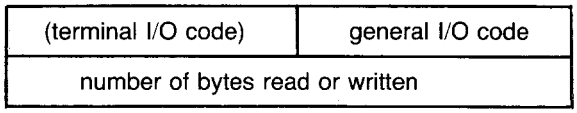
For all devices, the *isb* parameter defines the starting address of a 2–word buffer to which status information on the write is to be returned. If you omit *isb*, no such status information is returned.

Table 5–1: Vertical Format Control Characters

Octal Value	Character	Meaning
040	blank	SINGLE SPACE — Output a line feed, print the contents of the buffer, and output a carriage return. Normally, printing immediately follows the previously printed line.
060	0	DOUBLE SPACE — Output two line feeds, print the contents of the buffer, and output a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
061	1	PAGE EJECT — Output eight line feeds (or, for a line printer or an LA180, output a form feed), print the contents of the buffer, and output a carriage return.
053	+	OVERPRINT — Print the contents of the buffer and output a carriage return. Normally overprints the previous line.
044	$	PROMPTING OUTPUT — Output a line feed and print the contents of the buffer. This mode of output is intended for use with a terminal on which a prompt message is output and input is then read on the same line.
000	null	INTERNAL VERTICAL FORMAT — Print the buffer contents without adding vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed for each I/O request.

Contents of I/O Status Buffer

The general format of the information returned to the 2–word status buffer defined by *isb* is:



The first word contains information on the success or failure of the I/O operation requested. As with $DSW, you can use mnemonic codes to test this word. The Task Builder (TKB) resolves the mnemonics automatically.

For all devices and operations except terminal reads, you test the low-order byte of the first word to determine the I/O status code. Successful terminal reads provide additional information on what character terminated the line typed at the terminal. For terminal reads, you test the low-order byte first to see if the read was successful (IS.SUC). If it was, you can then test the entire first word to find out if the line was terminated with a carriage return (IS.CR) or an ESC or ALT character (IS.ESC).

The second word of the block contains the number of bytes read or written, if the read or write was successful. No information is returned in this word for IO.ATT and IO.DET operations.

I/O codes are listed below. Remember that IS.CR and IS.ESC are full-word values; the others are tested by the low-order byte only.

NOTE

The RSX emulator does the requested I/O operation using the non-file-structured OPNFQ subfunction of CALFIP, the CLSFQ subfunction of CALFIP, .READ, and .WRITE. It does not clear the FIRQB or XRB when it returns control to the user program. The codes given in square brackets below [ ] are the RSTS/E errors that are returned to the emulator. You can do further checking in the case of IE.BAD, for example, by checking byte 0 of the FIRQB.

Return Codes

IS.SUC Successful completion of I/O operation. For terminal reads, you can determine the terminating character of the line by testing the full first word of the I/O status buffer for:

- IS.CR The terminal line was terminated with a carriage return/line feed combination. (The terminating characters are not included in the character count in the second word. They do appear in the input buffer.)
- IS.ESC The terminal line was terminated with an ESC (called ALTMODE on some terminals). (The terminating character is not included in the character count in the second word. It does appear in the input buffer.)

IE.BAD This code is returned by the RSX emulator when an error occurs on the I/O operation that did not fit into any of the error categories diagnosed below. To determine the RSTS/E error, examine byte 0 of the FIRQB.

IE.DAA An open (IO.ATT) was attempted for a device that this job has already opened. The device must be closed (IO.DET) before it can be opened again. [NOTCLS]

IE.DNR Some hard device I/O error occurred (a read for an off-line device, a write for a physically write-locked device, and so forth). [HNGDEV]

IE.EOF An end-of-file mark, record, or control character (CTRL/Z for terminals) was recognized on a read operation. [EOF]

IE.EOT No more room is available on the device for a write operation (end of paper tape, end of magtape, not enough disk space available, and so forth). [NOROOM]

IE.PRI Privilege violation. A nonprivileged job attempted to read or write to disk while the disk was in use. [PRVIOL]

IE.NLN A close (IO.DET) was attempted for a device that was not open. [NOTOPN]

IE.RSU Shareable resource in use. The device is not available; it is assigned to another user. [NOTAVL]

IE.VER Unrecoverable error, such as parity error, bad card column, and so forth. [DATERR]

Macro Expansion

```
QIO$      IO.RVB,7,,,,IOSTAT,,<IOBUFR,80.,>
.BYTE     1,12.       ;DIC=1, DPB SIZE = 12. WORDS
.WORD     IO.RVB      ;FUNCTION CODE FOR READ
.WORD     7           ;LOGICAL UNIT NUMBER 7
.BYTE     0,0         ;EFN,PRI IGNORED IN RSTS/E
.WORD     IOSTAT      ;I/O STATUS BUFFER
.WORD     0           ;AST IGNORED IN RSTS/E
.WORD     IOBUFR      ;BUFFER ADDRESS
.WORD     80.         ;BYTE COUNT = 80.
.WORD     0           ;NO VFC NEEDED FOR READ
.WORD     0           ;THIS PARAM. IGNORED IN RSTS/E
.WORD     0           ;THIS PARAM. IGNORED IN RSTS/E
.WORD     0           ;NO BLOCK FOR THIS INVOCATION
```

The expansion for QIOW$ is the same except that the DIC = 3.

Local Symbol Definitions

Q.IOFN I/O function code (2)

Q.IOLU Logical unit number (2)

Q.IOEF Event flag number (1)

Q.IOPR Priority (1)

Q.IOSB Address of I/O status block (2)

Q.IOAE AST address (2)

Q.IOPL Parameter list (6 words) (12)

DSW Return Codes

IS.SUC Successful completion.

IE.LUN Unassigned logical unit number (*lun*). Use ALUN$.

IE.ILU Invalid logical unit number (*lun*). Must be in the range 0–14.

IE.SDP DIC or DPB size is invalid.

5.19 SCCA$S — Specify Control/C AST

The SCCA$S directive (actually, a macro) lets you specify an address to which control is to be transferred if the user types a CTRL/C while the program is executing. Only the $S form is allowed, because no DPB is ever generated. The SCCA$S directive is unique to the RSTS/E environment; no similar directive exists for the RSX–11M operating system.

The SCCA$S expands to code that takes the address specified and stores it in location 24 (part of the first 1000 bytes of memory used by the run-time system, as described in Section 4.2). The RSX run-time system checks this location when a CTRL/C asynchronous trap occurs, and if it is nonzero, clears the word and transfers control to the location that was specified. If this word is zero, the RSX run-time system falls back to its own processing of CTRL/C traps. Thus, SCCA$S is a "one-time" operation; you must reset the trap as part of the trap-processing routine, if desired.

Your routine can handle the CTRL/C in any way it sees fit. An RTI instruction returns control to the point where it left off at the time of the trap. The stack upon entry to such a routine is:

SP → (PC) at the time trap occurred
(PS) at the time trap occurred
word to which SP pointed before the trap

Macro Call

SCCA$S [*arg*]

where:

arg = Contains the address (usually specified in the form "#addr") to which control passes should the user at the job's terminal type a (CTRL/C) combination. If this address is omitted, word 24 is cleared, indicating that the RSX run-time system is to handle such traps.

Macro Expansion

```
SCCA$S      #CTRAP
MOV         #CTRAP,@#24
```

If you omit the address, the expansion is:

```
SCCA$S
CLR         @#24
```

Local Symbol Definitions

None

DSW Return Codes

None. (SCCA$S does not execute an EMT; control does not pass to the RSX emulator when SCCA$S is executed.)

5.20 SFPA$ — Specify Floating-Point-Processor Exception Address

The SFPA$ directive tells the RSX emulator one of two things:

1. The job wants to handle floating-point-processor exception traps itself. The RSX emulator passes control to the address specified in the directive when such an asynchronous trap occurs.*
2. The job does not want to handle floating-point-processor exception traps. The address is omitted in the directive. In this case, if an FPP exception trap occurs, the RSX emulator aborts the job, displaying the message "?Reserved instruction trap" on the job's terminal (TI:).

If you specify an address in the SFPA$ directive, control will be passed to the address when an FPP trap occurs. The stack will have the following information:

SP → Floating exception address (FEA)
Floating exception code (FEC)
(DSW) at the time the trap occurred
(PC) at the time the trap occurred
(PS) at the time the trap occurred
word pointed to by SP before the trap

Your routine can process the exception in any way it sees fit. To exit from the trap-handling routine, pop the FEA and FEC from the stack and issue an ASTX$ directive (Section 5.4).

Macro Call

SFPA$ [*add*]

where:

add = Address to which control passes when an FPP trap occurs.

Macro Expansion

```
SFPA$    FLTAST
.BYTE    111.,2     ;DIC = 111., DPB SIZE = 2 WORDS
.WORD    FLTAST     ;ADDRESS OF FLOATING POINT AST
```

* The FPP is the floating-point processor available on all PDP–11 processors that can run RSTS/E, except the PDP–11/35 and 40. The FPP executes concurrently with the PDP–11 central processor. Thus, an error in the FPP unit occurs "asynchronously" with the execution of the PDP–11 CPU.

Local Symbol Definitions

S.FPAE AST entry address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.SDP DIC or DPB size is invalid.

5.21 SPND$S — Suspend

The SPND$S directive instructs the system to suspend the execution of the issuing program. A program can suspend only itself, not another program. The program is restarted when the user types anything terminated by a RETURN or other delimiter.*

Because this directive requires only a 1–word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as the DIR$ macro.

Macro Call

SPND$S [*err*]

where:

err = Error routine address

Macro Expansion

```
SPND$S  ERR
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   45.,1           ;SPND$S MACRO DIC, DPB SIZE=1 WORD
EMT     377             ;TRAP TO THE EMULATOR
BCC     .+6             ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR          ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions

None

DSW Return Codes

IS.SPD Successful completion (task was suspended).

IE.ADP Part of the DPB is out of the issuing task's address space.

IE.SDP DIC or DPB size is invalid.

*This directive is implemented on RSTS/E systems using the .READ directive to the job's terminal (channel 0). Thus, if you are using special terminal modes (binary input, echo control, etc.), the SPND$S directive will clear these modes.

5.22 SVDB$ — Specify SST Vector Table for Debugging Aid

The SVDB$ directive, like the SVTK$ directive, defines a table of addresses to which control is to pass when certain synchronous system traps (SSTs) occur. When an SST occurs, the RSX emulator first checks the table defined with SVDB$ to see if an address has been specified for that trap. If so, control is passed to that address. If not, the emulator then checks the table defined with SVTK$ (if any).

In other words, the SVDB$ sets up an SST vector table whose entries, if nonzero, override (but do not destroy) any SST vector table entries defined with SVTK$. The SVDB$ directive is handy within a debugging routine that uses BPT instructions, for example, to set breakpoints. With SVDB$, the debugging routine can divert breakpoint traps to itself, regardless of SVTK$ directives in other modules of the job.

For example, the Octal Debugging Tool (ODT) object module uses the SVDB$ directive to divert traps to itself, regardless of what may have been requested with SVTK$ in other modules.* This makes it possible to debug without changing your source code; you can link with ODT, use ODT to insert breakpoints, test and debug, and when finished, relink without ODT. It is not necessary to change source code to debug, only relink. The SVDB$ makes this possible.

NOTE

To avoid interfering with ODT, you should not use this directive in a user program.

Macro Call

SVDB$ [*adr*,*len*]

where:

adr = Address of SST vector table. If *adr* and *len* are both omitted, any SST vector table previously defined with an SVDB$ directive will be deassigned; that is, the emulator will not check for SVDB$ addresses. (A previously executed SVTK$ will remain in effect.)

len = Length of SST vector table, in words. As shown below, up to eight addresses can be specified. If you wish to use only the third, you could save space by using a *len* of 3, rather than a *len* of eight and filling the last 5 words of the buffer with zeros. Specifying a length greater than 8 has the same effect as a length of 8.

---

* The ODT object module (the file ODT.OBJ) can be linked to user modules with the RSTS/E Task Builder (TKB) using the /DA switch, as described in the *RSTS/E Task Builder Reference Manual*. ODT.OBJ is different from ODT.BAC, which is a BASIC–PLUS utility in RSTS/E. The commands and syntax for ODT.OBJ are described in the *IAS/RSX–11 ODT Reference Manual*.

The vector table format is shown below. A nonzero value is interpreted as an address to which control is to pass when that particular trap occurs. A zero value for a word indicates you have no interest in handling that trap. Should such a trap occur, the emulator will check the vector table assigned by SVTK$, if any. If there is no SVTK$ vector table, the emulator will abort the job and display an error message at the job's terminal.

Buffer Format

Octal Offset		Octal Offset
1	odd address or nonexistent memory error	0
3	memory protect violation	2
5	T-bit trap or execution of a BPT instr.	4
7	execution of an IOT instruction	6
11	execution of a reserved instruction	10
13	execution of a non-RSX, non-RSTS EMT instr.	12
15	execution of a TRAP instruction	14
17	PDP–11/40 flt. pt. exception	16

Macro Expansion

```
SVDB$     SSTTBL,3
.BYTE     103.,3      ;DIC=103. ,DPB SIZE = 3 WORDS
.WORD     SSTTBL      ;ADDRESS OF SST TABLE
.WORD     3           ;SST TABLE LENGTH = 3 WORDS
```

Local Symbol Definitions

S.VDTA Table address (2)

S.VDTL Table length (2)

DSW Return Codes

IS.SUC Successful completion.

IE.SDP DIC or DPB size is invalid.

5.23 SVTK$ — Specify SST Vector Table for Task

The SVTK$ directive defines a table of addresses to which control is to pass when certain synchronous system traps (SSTs) occur. When an SST occurs, the RSX emulator will first check to see if a similar table has been defined for this job with an SVDB$ directive (Section 5.22). If so, any nonzero entries in that table will override those defined with an SVTK$. If not, the emulator uses the trap addresses defined with SVTK$.

Macro Call

SVTK$ [*adr,len*]

where:

adr = Address of SST vector table. If you omit both *adr* and *len*, any SST vector table previously defined with an SVTK$ directive will be deassigned. (A previously executed SVDB$ will remain in effect.)

len = Length of SST vector table, in words. As shown below, up to eight addresses can be specified. If you wish to use only the first five, you could save space by using a *len* of 5, rather than a *len* of 8 and filling the last 3 words of the buffer with zeros. Specifying a length greater than 8 has the same effect as a length of 8.

The vector table format follows. A nonzero value is interpreted as an address to which control is to pass when that particular trap occurs. A zero value indicates you have no interest in handling that trap.

The contents of the program status register and the program counter (PS and PC) at the time of the trap have been pushed on the stack when the trap-handling routine is entered. Certain other values may have been pushed on the stack. The values depend on the trap. If the trap was a memory protect violation, the stack contains:

(SP)→ Instruction backup register (SR1)*
Virtual PC of the faulting instruction (SR2)*
Memory protect status register (SR0)*
(PC) at time of trap
(PS) at time of trap

If the trap was a TRAP instruction or EMT other than 377, the stack contains:

(SP)→ Instruction operand (low-order byte) times 2, non-sign extended
(PC) at time of trap
(PS) at time of trap

*RSTS/E RSX emulation returns 0 for SR0 and SR1; SR2 contains the PC at the time of the trap.

All items except the (PS) and (PC) must be popped from the stack before the trap-handling routine exits. You can then return control to the point where execution left off with an RTI or RTT instruction.

Buffer Format

Octal Offset		Octal Offset
1	odd address or nonexistent memory error	0
3	memory protect violation	2
5	T-bit trap or execution of a BPT instruction	4
7	execution of an IOT instruction	6
11	execution of a reserved instruction	10
13	execution of a non-RSX, non-RSTS EMT instruction	12
15	execution of a TRAP instruction	14
17	PDP–11/40 floating point exception	16

Macro Expansion

```
SVTK$     SSTTBL,4
.BYTE     105.,3      ;DIC = 105., DPB LENGTH = 3 WORDS
.WORD     SSTTBL      ;ADDRESS OF SST TABLE
.WORD     4           ;SET TABLE LENGTH = 4 WORDS
```

Local Symbol Definitions

S.VTTA Table address (2)

S.VTTL Table length (2)

DSW Return Codes

IS.SUC Successful completion.

IE.SDP DIC or DPB size is invalid.

5.24 UMAP$ — Unmap an Address Window

The UMAP$ unmaps a specified address window from a resident library. The unmapping does not eliminate the window (created with a CRAW$ directive), nor does it release the APRs used by the window. The virtual address window can now be remapped to new actual memory locations, with the MAP$ directive, if desired.

Macro Call

UMAP$ *adr*

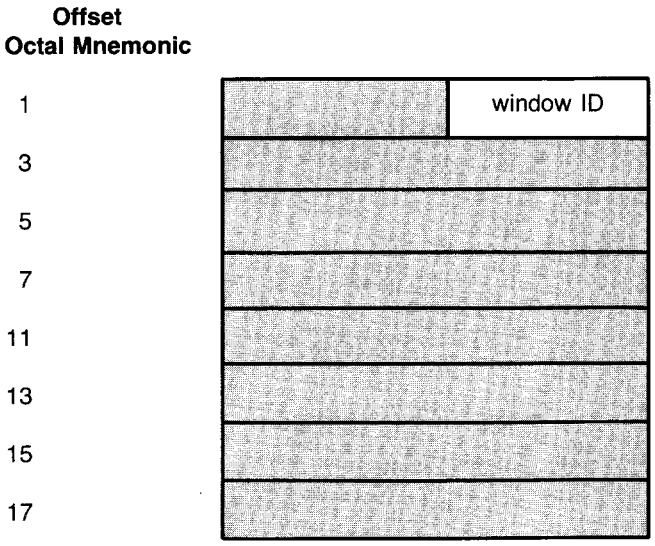
where:

adr = Address of an 8–word area defining the window to be unmapped. Information is also returned to this area by the UMAP$ directive. Two supplementary directives are available to define (WDBDF$) or define and fill (WDBBK$) such an area, called the window definition block, or WDB. The WDBDF$ and WDBBK$ directives are described in Appendix C. The descriptions below show the offsets automatically defined by these directives.

Macro Expansion

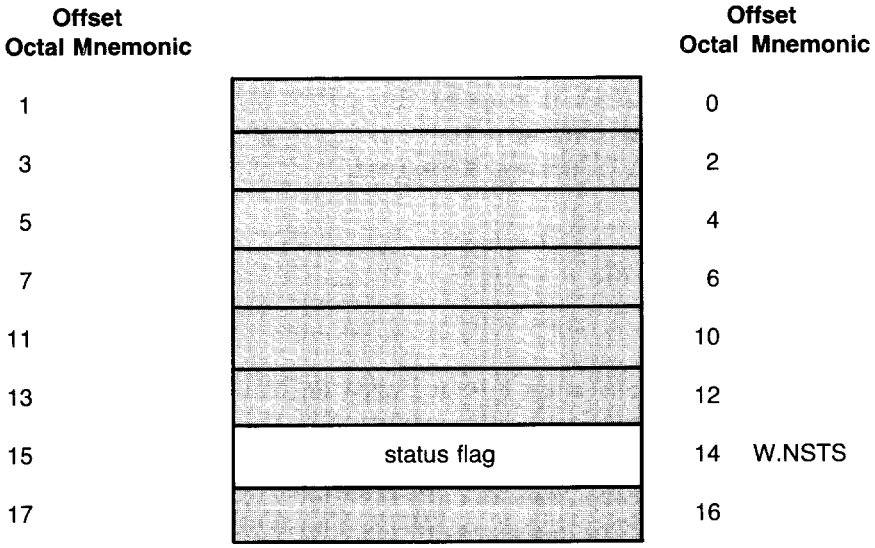
```
UMAP$     WDBADR
,BYTE     123.,2      ;DIC=123.,, DPB SIZE=2 WORDS
,WORD     WDBADR
```

Data Passed in WDB



adr+W.NID The ID of the window to be unmapped (returned at the same location in the WDB by the CRAW$ directive).

Data Returned in WDB



adr+W.NSTS Bit 14 = 1 (WS.UNM) if the window specified was successfully unmapped.

Local Symbol Definitions

U.MABA Window definition block address (2)

DSW Return Codes

IS.SUC Successful completion.

IE.ITS The window ID specified is either invalid (outside the range 1–7) or not currently mapped.

5.25 WSIG$ — Wait for Significant Event Flag

The WSIG$ directive is included for compatibility with RSX–11M. In RSX–11M, some recoverable error conditions require waiting for a "significant event" to happen, at which time a retry of the operation that failed may succeed. On a RSTS/E system, any events that may be termed significant within the RSTS/E monitor are transparent to user jobs. Hence, the WSIG$ directive in RSTS/E merely causes the calling job to sleep for one second. The job can then retry whatever operation failed previously.

Macro Call

WSIG$

Macro Expansion

```
WSIG$
.BYTE     49.,1        ;DIC=49., DPB SIZE=1 WORD
```

Local Symbol Definitions

None

DSW Return Codes

IS.SUC Successful completion.

IE.SDP DIC or DPB size is invalid.

5.26 WTSE$ — Wait for Single Event Flag

The WTSE$ directive is included for compatibility with RSX–11M, in which some programs are written with a QIO$ followed by a WTSE$, to cause the program to wait until the I/O is complete. This has the same effect as a QIOW$ and, since all I/O in RSTS/E is synchronous, QIO$ functions the same as QIOW$. Thus, the WTSE$ directive simply returns immediately; it is a "no-op" in RSTS/E.

Macro Call

WTSE$ *efn*

where:

efn = Event flag number (ignored in RSTS/E).

Macro Expansion

```
WTSE$     0
.BYTE     41.,2      ;DIC=41., DPB SIZE = 2 WORDS
.WORD     0          ;FLAG IGNORED IN RSTS/E
```

Local Symbol Definition

W.TSEF Event flag number (2)

DSW Return Codes

IS.SUC Successful completion.

IE.SDP DIC or DPB size is invalid.

Chapter 6
RT11 Run-Time System Environment

6.1 Introduction

The RT11 run-time system emulates the "Single-Job" monitor of the RT–11 operating system on RSTS/E systems. You can use many of the "programmed requests" available to MACRO programmers on RT–11 systems. In addition, the RT11 run-time system provides some directives, suited to the RSTS/E environment, that are not available for the RT–11 operating system.

The size limit for programs running under the RT11 run-time system is 27K words, slightly less than the 28K words allowed by the RT–11 operating system. The run-time system itself takes a 4K–word high segment. It also uses 1K words in the user job image area as a "scratch pad." As noted in Chapter 2, run-time systems are usually mapped read-only, so that they can be shared by more than one user on a RSTS/E time-sharing system. The RT–11 operating system, however, allows a user to access certain areas within the resident monitor. To allow a similar capability under RSTS/E, then, these areas must be located in the user job image area, which is mapped read/write. This area is described in more detail in Section 6.7.

If you use the RT11 directives described in Chapter 7, you must assemble your program with the MACRO assembler and link the modules with the LINK linker. The *RSTS/E RT11 Utilities Manual* describes how to run MACRO and LINK to assemble and link your program.

6.1.1 Advantage: Transportable Code

The RT11 directives are useful if you are coding a program to be run under both the RT–11 and RSTS/E operating systems. Most of the RT–11 "single-job" programmed requests are emulated. Of those which are not, most are processed as "no-ops" on RSTS/E systems, including the foreground/background (FB) and extended memory (XM) calls. That is, most will not cause errors on RSTS/E systems. Exceptions are listed in Section 7.1.

Those RT–11 requests that are emulated are fitted to the RSTS/E environment. For example, under the RT–11 operating system, there are three types of read: .READ, .READW, and .READC. The .READ initiates a read operation and transfers control back to the user program; the read may or may not be complete. The .READW, or read-and-wait, transfers control back to the user program only after the read operation is completed. The .READC initiates a read operation, transfers control to the user program in-line, and, when the read is complete, transfers control to a user-specified "completion routine."

Under RSTS/E, all input/output is synchronous; that is, control is returned to the user program only when the I/O is complete. So, .READ and .READW are implemented the same under RSTS/E—as a "read and wait." The .READC will transfer control to the completion routine when the read operation is finished.

The significance of the differences depends on what you want to do. If you are transporting an existing program from RT–11 to RSTS/E, you can generally be confident that it will work as it should under RSTS/E.

If you are developing a program for RSTS/E only, you need not be concerned with how the directives work on RT–11. You can choose .READ or .READW — both will work the same. You could also choose .READC if, for example, you wanted to code one completion routine to be used after different read operations.

If you are developing a program on RSTS/E to be run under RT–11, or both RSTS/E and RT–11, you need to know how the directives work under RT–11 and under RSTS/E. You would choose between .READ, .READW, and .READC according to what they do on both systems.

In general, Chapters 6 and 7 describe what RT11 directives are available, and what they do, under RSTS/E. Some comparison is made here to RT–11, but you must refer to the *RT–11 Programmer's Reference Manual* for a complete description of how these directives work in the RT–11 environment.

You can also transport assembled (binary) programs from RT–11 systems to RSTS/E systems, keeping in mind the limitations of the emulated RT11 environment. You must change the run-time system name that RSTS/E maintains in the file directory entry for each runnable file. The easiest way to do this is with the RTS (run-time system) switch in PIP:

```
PIP *.*/RTS:RT11=MT0:*.*
```

This command will transfer all files on magnetic tape unit 0 to your account on the public disk structure as files to be run under the RT11 run-time system.

For binary files that have already been transferred to your system, type:

```
PIP file.typ/RTS:RT11
```

6.1.2 General Services

Besides transportable code, using the RT11 emulator provides:

1. Simple input/output. The I/O operations under the RT11 run-time system are simple to code and include file-structured input/output operations.
2. Fast assembly and linking. The MACRO assembler and LINK linker execute somewhat faster than their RSX emulator counterparts MAC and TKB.

6.2 System Macro Library

The RT11 emulator directives that you code into your program are macro calls; they must be expanded into executable code at assembly time. The macro expansions for the RT11 emulator directives are contained in the system macro library — $SYSMAC.SML. You can name this library file in the input-file list when you assemble your program. For example:

```
MACRO OBJ,OBJ=$SYSMAC.SML/M,SRC1,SRC2
```

However, the MACRO assembler searches the library automatically to resolve undefined symbols, so this is not really necessary.

In your code, though, you must use the MACRO–11 .MCALL directive to define the directives you use as external macros needed to assemble the source program. The .MCALL must appear before the first directive is called. For example:

```
.MCALL .READ,.DATE,,,,,
   .
   .
   .
.DATE
```

Note that some RT11 emulator directives have the same form as general monitor directives — .DATE, for example. If you want to use the RT11 .DATE, specify it in an .MCALL directive. Then, regardless of whether or not you assemble with the prefix file COMMON.MAC, the .DATE will be expanded from the system library as a call to the RT11 run-time system.

If you want the general monitor .DATE, do not specify .DATE in an .MCALL directive; use the "special prefix" emulator trap instruction — EMT 377 — immediately preceding the .DATE call, and assemble with COMMON.MAC.

In fact, you must precede all general monitor directives that you use with an EMT 377. RT11 uses EMTs in the range 0–100, the same range used by the general monitor directives. To bypass the general monitor calls, the RT11 run-time system defines a "special prefix" EMT for the monitor, as described in Section 2.5, so that all EMTs except those preceded by an EMT 377 are processed by the RT11 run-time system.

Note, however, that you should not use the general monitor calls .CHAIN, .EXIT, .CORE, .RTS, .FSS, .RUN, .CCL, or .RSX from a program running under control of the RT11 run-time system. Results for using these directives under RT11 emulation are unpredictable.

6.3 Directive Processing

It is easier to understand how you specify RT11 directives and their arguments if you first understand how the directives are expanded at assembly time, and how they are processed by the RT11 run-time system at execution time.

Generally, the expansions end with an EMT (Emulator Trap) instruction, passing control to the RSTS/E monitor. The monitor then passes control to the RT11 run-time system for processing.* The RT11 run-time system examines the low byte of the EMT instruction to determine either (1) the action to be performed or (2) the place to look for information that defines the action to be performed.

For example, if the call translates to code ending in an EMT in the range from 340–357, the RT11 run-time system determines the action to be performed from the EMT and looks in R0 and/or on the stack for other arguments used in the call.

If, on the other hand, the call translates to an EMT 374 instruction, the RT11 run-time system looks in R0 for a one-byte function code and, if necessary, a one-byte argument. For an EMT 375 instruction, the RT11 run-time system looks in R0 for the address of an argument block. It will then examine the argument block to find the function code defining the action and whatever other arguments it expects, depending on the function.

Table 6–1 summarizes the EMTs processed by the RT11 run-time system. Unless you want to bypass the directive expansions and load R0 and other arguments yourself, the import of Table 6–1 is:

1. All the directives use register R0. You must preserve the contents of R0, if necessary, by saving and restoring the contents before and after issuing a directive. Some directives return information to R0; otherwise, the contents of R0 are unpredictable upon completion of a call. All other registers are preserved as you left them.

2. Some of the directives require that you allocate space in your program for arguments. The expansions for such directives will fill the argument block you specify in the call with the values you specify as other arguments in the call, as described in Section 6.4.

---

* The RT11 run-time system has the PF.EMT bit set in the P.FLAG word in the pseudo-vector region (Section 2.5). Thus, the RT11 run-time system can use all possible values in the low byte of the EMT instruction, just as the RT–11 operating system does. The low-byte of the P.FLAG word is set to 377, which, in combination with the setting of the PF.EMT bit, defines the special prefix EMT 377 described in Section 6.2.

If an error occurs when a directive is processed, control returns to the user program with the carry bit set in the program status word (PSW) and an error code in byte 52_8 of the low 1000_8 bytes of memory.

Table 6–1: EMT Instructions Recognized by the RT11 Run-Time System

EMT Low-Byte	Meaning
EMT 377	Reserved; this is the "special prefix" EMT described in Section 6.2. You must code this EMT immediately before any of the general monitor calls described in Chapter 3.
EMT 376	Used internally by the RT11 run-time system. Never execute this EMT from a user program; the program will abort with an error. The error message displayed is "?M –OVLY ERROR AT USER PC nnnnnn."
EMT 375	Indicates a call that has several arguments: R0 contains the address of the first byte of an argument block. (R0) = [address of argument block]
EMT 374	Indicates a call with one argument. R0 contains a function code in the high-order byte and the argument in the low-order byte. (R0) = [function code | argument]
EMTs 372–373	Ignored by the RT11 run-time system. Do not use these EMTs; they are reserved for future use.
EMTs 360–371	EMTs specific to the RT11 run-time system under RSTS/E; not available under the RT–11 operating system.
EMTs 340–357	Indicates a directive that expects arguments on the stack, in R0, or both.
EMTs 0–337	Indicates a directive used by Version 1.0 of the RT–11 operating system. These EMTs are also recognized by the RT11 run-time system.

6.4 Call Forms

The call formats in Chapter 7 show RT11 directives in two basic formats: those with an "area" argument and those without. The area argument indicates that the call uses an argument block. You must allocate space for the argument block, as noted in Section 6.3.

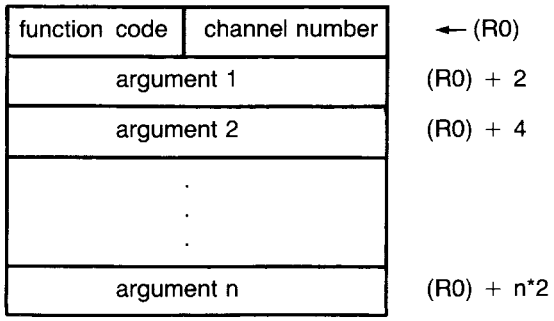
6.4.1 Format for Calls Using Argument Blocks

Directives that show an area argument use an argument block, as described in Section 6.3. The format for these directives is shown in Chapter 7 as:

```
.RTCALL  area, arg1, arg2, ...., argn
```

The area argument is the address of the first byte of the area you have set aside for the argument block. The remaining arguments (arg1, arg2, and so forth) are the values needed for the call. They will be placed in the argument block at execution time.

In general, the argument block is formatted as follows:



R0 points to location x. The high byte (location x+1) contains the function code defining the action to be performed. The low byte defines a channel number, if one is needed for the call. Remaining arguments, if any, are stored in subsequent words.

Consider the hypothetical directive .HYPOT, using four arguments including the area argument:

```
.HYPOT area, arg1, arg2, arg3
```

If all four arguments are specified in the call, it is expanded at assembly time to code that:

1. Moves the area argument (an address) to R0.
2. Moves a function code to the address specified by (R0)+1, and, if a channel number is used, moves that channel number to (R0). (The notation (x) means "the contents of x" — thus the channel number is moved to the address specified by the contents of R0.)
3. Moves remaining arguments to following words in the argument block.
4. Executes an EMT 375.

Thus, all of the arguments for this form should be appropriate as operands in MOV instructions. For example:

```
AREA:   .WORD   0,0,0
            .
            .
            .
        .HYPOT  #AREA,#4,NUMNUM,#300
```

Under certain conditions, you can leave out arguments in directives using an argument block. If you leave out area, for example, you must load R0 with the address of the argument block yourself, before executing the directive. For example:

```
MOV         #AREA,R0
.HYPOT      ,#4,NUMNUM,#300
```

If you leave out any of the other arguments, the expansion will not load any new values into those positions in the argument block; those words are left untouched. For example:

```
AREA:   .WORD   0,0,0
            .
            .
            .
        .HYPOT  #AREA,,NUMNUM,#300
```

The execution of .HYPOT will use the values 0, the contents of location NUMNUM, and 300 as arguments. Suppose that the following directive is executed next:

```
.HYPOT    #AREA,#3,,#200
```

The argument block will contain the values 3, the contents of location NUMNUM, and 200, and these values will be used for the directive's execution.

6.4.2 Format for Calls Not Using Argument Blocks

For those calls that do not use an argument block, the format in Chapter 7 is given as:

.RTCALL arg1, arg2, ..., argn

These directives expand to code that stores the arguments, if any, in R0 and/or the stack. (If the stack is used, it is "cleaned" by the RT11 emulator; that is, any values pushed on the stack are popped before control returns to the user program.)

Again, the arguments should be appropriate as operands in MOV instructions. Certain arguments for this form can be omitted. These are noted in Chapter 7 as enclosed in brackets. For example, if the macro call format were that shown below, you could omit the last argument.

.HYPOT arg1, arg2[, arg3]

6.5 Channel Number and Device Block Arguments

Many RT11 directives use arguments that define a "channel number" and "device block." The following general comments apply to these arguments.

6.5.1 Channel Number Arguments

Like the general monitor calls, RT11 directives which handle input/output use a "channel number" to refer to a device. Allowable RT11 channel numbers range from 0 through 15_{10} or 17_8. When you open a file or device using

the RT11 directives .LOOKUP or .ENTER, you give a channel number in this range. The RT11 emulator relates the number specified in the call to the first free RSTS/E channel number, with the following considerations:

1. The job's terminal (RSTS/E channel 0) is never "free" for the emulator to relate to an RT11 channel number. In RT11, you do I/O to the job's terminal with specific directives, such as .PRINT and .GTLINE. Thus, since RSTS/E channel numbers range from $0–15_{10}$ but RSTS/E channel 0 is always "busy" as far as the RT11 run-time system is concerned, you can open up to 15_{10} channels using any 15_{10} of the 16_{10} possible RT11 channel numbers (ranging from 0 to 15_{10}).

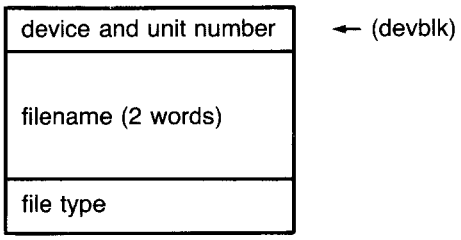
2. The RT11 emulator always attempts to allocate RT11 channel number 15_{10} to RSTS/E channel 15_{10}. Channel 15_{10} is the channel on which the monitor opens the user job image file when a .RUN (Section 3.20) is executed. The run-time system loads the file and begins its execution; if the program is overlaid, the run-time system keeps channel 15_{10} open. If the program is not overlaid, the run-time system closes channel 15_{10} after the file is loaded.

   So, if your program is not overlaid and you open RT11 channel 15_{10}, the call will succeed (unless you already have channel 15_{10} open). If your program is overlaid, you cannot open channel 15_{10}; any attempt to do so will result in an error indicating the channel is in use.

Other than channel 15_{10}, there is no correspondence between the RT11 channel number and the RSTS/E channel number. They will probably be different.

6.5.2 Device Block Arguments

Several RT11 directives use a device and file specification. The argument in these cases is an address pointing to the first word of a 4–word "device block" that you allocate in your program. The format of the device block is 12 RAD50 characters:



You can use any valid RSTS/E device and unit number (0–7) for the first word of the device block. You can also use logical names up to three characters long. If you need device numbers that have more than one digit (for example, KB27 or TT12), you must use the following algorithm:

1. Form the RAD50 representation of the zeroth unit of the device you want:

```
BASEKB=^RKB0
```

2. Add the decimal value of the unit number wanted. For example, to get the proper representation of KB36:

```
KBN36=BASEKB+36.
```

3. Use this value as the first word of the device block. For keyboards, there is no file name and type, so the device block could be constructed as:

```
DEVBLK:  .WORD  KBN36
         .WORD  0,0,0
```

Note that whatever you construct will also have some meaning as RAD50 characters. If you have assigned a logical with the same RAD50 bit pattern as some device/unit-number that you construct, the logical will take precedence. The above example, for instance, forms the RAD50 characters KCZ. If you assigned the logical name KCZ to the line printer, any call referring to DEVBLK in the above example would use the line printer.

6.6 Low 1000 Bytes for RT11 Run-Time System

The RT11 run-time system uses the following locations in the low 1000_8 bytes of virtual address space:

Locations (Octal)	Contents
30,31	This word contains the contents of FIRQB + FQNENT at the time the .RUN was executed. (See .RUN, Section 3.20, and the description of P.RUN, Section 2.5.4.)
32,33	This word contains the contents of XRB + 0 when the run-time system was entered at P.RUN (Section 2.5.4).
40,41	This word contains the starting address of the job.
42,43	This word contains the initial value of the stack pointer. If this value is not set by the user program in an .ASECT or through a LINK option, the LINK linker defaults this word to 1000.
44,45	This word contains the Job Status Word (JSW). Used as a flag word for the run-time system. Certain bits are maintained by the RT11 run-time system exclusively, while others can be set or cleared by the user program. The bits you can set are marked with an asterisk in the list below. Do not use currently unassigned bits for your own purposes; future releases of the run-time system may use them.

Bit	Meaning
*15	If set with .ASECT, core common is preserved when the user program is executed. (You cannot set the bit at run time; core common will already have been cleared.)
*14	Lowercase bit. Disables automatic conversion of lowercase to uppercase when set.
*13	Reenter bit. When set, indicates that the program can be restarted from the terminal with the REENTER command (Section 6.8).

Locations (Octal)	Bit	Meaning
	*12	Special mode TT bit. When set, indicates that the job is in a special keyboard input mode. See .TTYIN/.TTINR directive descriptions, Section 7.41.
	9	Overlay bit. Set by LINK if the job uses the linker overlay structure.
	8	CHAIN bit. If set, virtual addresses 500–776 are loaded from the memory image file when the job is started. (These words are normally used to pass parameters across .CHAINs.) The bit is set by the run-time system if and only if the job was entered with the RT11.CHAIN directive.
	*6	Inhibit TT wait bit. When set, the job can accept input from the job's terminal in "ODT mode" with .TTYIN/.TTINR; that is, one character at a time, as the characters are typed, rather than one character at a time once a whole line has been typed.
46,47		This word points to the first word of the RT11 read/write "scratch pad" area in the user job image area.
50,51		This word contains the high memory address. The run-time system maintains the highest virtual address the user program can use in this word. It is initially set to the address of the last word of the user program. It can be changed with the .SETTOP directive (Section 7.36) to any address up to the start of the "scratch pad" area. That is, the maximum value this word can be is the contents of the word at location 46,47 minus 2.
52		This byte is the EMT error code. If the directive results in an error, the code number of the error is always returned in byte 52 and the carry bit is set. Always refer to location 52 as a byte, not a word.
54,55		This word contains the beginning address of the "scratch pad" area.

6.7 "Scratch Pad" Area in User Job Image

As mentioned in Section 6.1, the RT11 run-time system uses some of your high address space as a "scratch pad." Some of the information in this area parallels the information in the RT–11 operating system, as described in the *RT–11 Programmer's Reference Manual*. Other information is unique to RSTS/E. You can, for example, set a word in this area with a project-programmer number (PPN) to be used when opening a file. (RT–11 does not allow a PPN in its file specification; RSTS/E does.)

In any case, the RT11 emulator adds the scratch pad area to the end (high virtual addresses) of the user job image at execution time. If no SIZE keyboard monitor command is used (see the *RSTS/E System User's Guide*), the emulator determines the highest address of the program from the .SAV file, an item calculated by the LINK linker, and rounds up to the next multiple of 4000 (1K words) to determine the starting address of the scratch pad. If the SIZE keyboard monitor command is used, the emulator places the scratch pad in the top 1K words of the address space requested with SIZE.

Thus, the maximum size of MACRO programs running under the RT11 run-time system is 27K words. Any program larger than this will cause a "Maximum memory exceeded" error when the file is run.

You can determine the start of the scratch pad area from word 54 in the low 1000 bytes of memory. The offsets into the scratch pad area described below may be of use to you. The first six words are of special significance. The following general comments apply to the first six words. The directives .LOOKUP, .ENTER, .RENAME, .REOPEN, .DELETE, and .SETFQB examine the first six words, use any values there as described, and clear the first six words before returning control to the user program. The directives .READ and .WRITE examine, use, and clear only the first two words. The .CSIGEN directive clears the first six words before performing any of the .LOOKUP and .ENTER calls resulting from its examination of the command string.

Octal Offset	Mnemonic Offset	Contents
0	PPN	If nonzero, used as a project-programmer number (PPN) for all of the calls named above except .READ and .WRITE. (Note that for .SETFQB, this value is loaded into FIRQB+FQPPN and overrides any project-programmer number resulting from examination of the string by the corresponding .CSISPC.) For .READ and .WRITE, this word forms the "modifier word" (at XRB+XRMOD) in the corresponding RSTS/E .READ or .WRITE directive.
2	PROTEC	If nonzero, this word is the protection code to be used for all of the calls named above except .READ and .WRITE. (Note that for .SETFQB, this value is loaded into FIRQB+FQPROT and overrides any protection code resulting from examination of the string by the corresponding .CSISPC.) For .READ and .WRITE, this word forms the most significant bits of the block number in the corresponding RSTS/E .READ or .WRITE (at XRB+XRBLKM). This value supplements the block number argument from the RT11 .READ or .WRITE call.
4	MODE	For RT11 calls .LOOKUP, .ENTER, and .REOPEN, this word forms the "mode" word at FIRQB+FQMODE in the corresponding RSTS/E CALFIP call. For .SETFQB, this word is loaded into FIRQB+FQMODE and overrides any /MODE qualifier encountered by .CSISPC.
6	CLUSTR	For RT11 directives that create files, such as .ENTER, this word forms the cluster size (at FIRQB+FQCLUS) for the corresponding RSTS/E CALFIP call. For .SETFQB, this word is loaded into FIRQB+FQCLUS, and overrides any /CLUSTERSIZE qualifier encountered by .CSISPC.
10	POSITN	For RT11 directives that create files, such as .ENTER, this word forms the device cluster number for file placement (at FIRQB+FQNENT) in the corresponding RSTS/E CALFIP call. For .SETFQB, this word is loaded into FIRQB+FQNENT and overrides any /POSITION qualifier encountered by .CSISPC.
12	CRMSBS	The most significant bits of the file size for RT11 calls that preallocate space for a file on its creation (.ENTER); used at FIRQB+FQSIZM in the corresponding CALFIP call. For .SETFQB, this byte is loaded into FIRBQ+FQSIZM.

Octal Offset	Mnemonic Offset	Contents
14	LOGTBL	The RT11 run-time system keeps the user logical area here, instead of in locations $734–776_8$ of the low 1000_8 bytes. See the discussion of USRPPN, USRPRT, and USRLOG in Section 2.4 for a description of the format of this 16_{10}–word area.
66	NOCTLC	This word is used by the RT11 run-time system to stop user-typed CTRL/Cs from interrupting program execution during certain crucial sequences (.SRESET, .HRESET, .LOOKUP, .ENTER, .REOPEN, .CLOSE, and .SAVESTATUS). When set to 177777, user-typed CTRL/Cs will interrupt execution. A zero or positive value inhibits CTRL/C interrupts. While this word is zero or positive, any user-typed CTRL/C will increment the contents of this word by one. When the emulator reenables CTRL/Cs, it processes any pending CTRL/C by either (1) passing control to a user routine specified with .SETCC (Section 7.34) or (2) if no .SETCC is in effect, passing control to the job keyboard monitor. If the job keyboard monitor is RT11, the keyboard monitor commands CONTINUE or CCONTINUE will resume execution of the program at the point where it left off. Your program can clear this word to inhibit CTRL/Cs, process them later in whatever manner you see fit, and reenable CTRL/C interrupts by setting this word to 177777. Remember, however, that the RT11 emulator will clear the word and then reenable interrupts when .SRESET, .HRESET, .LOOKUP, .ENTER, .REOPEN, .CLOSE, or .SAVESTATUS is executed.
276		Emulator version number—the same as the RT–11 operating system version number that the current RT11 run-time system emulates.
277		Emulator release number—the same as the RT–11 operating system release number that the current RT11 run-time system emulates.
300		Configuration word. Set for RSTS/E environment, as follows: Bit 6 = 1 indicates that FP11 floating-point hardware exists. Otherwise, bit 9 is always set to 1, and all other bits are set to 0. (If your program tests these bits on an RT–11 system, the code will also work on RSTS/E.)
370		Extension configuration word. Under RSTS/E, bits 0–3 are always 0, bit 8 is always 1 (indicating the EIS option is present), and bits 9, 14, and 15 are always 0. (The conditions they indicate on RT–11 systems are not determinable by the RT11 run-time system under RSTS/E.)
372		SYSGEN options word. Always 0 on RSTS/E systems.
374		Always 0 on RSTS/E systems.

Chapter 7
RT11 Emulator Directives

7.1 Introduction

Tables 7–1 and 7–2 list the directives that are and are not processed by the RT11 run-time system on RSTS/E systems.

Some of the RT–11 operating system's single-monitor calls are not emulated on RSTS/E systems. The following are simply ignored: .CMKT, .HERR, .LOCK, .QSET, .MRKT, .RELEASE, .SERR, and .UNLOCK. The .CDFN call always returns an error 0 on RSTS/E systems. The following single-monitor calls are not expanded to include EMTs, and their results on RSTS/E systems are unpredictable: .INTEN, .MFPS, .MTPS, and .SYNCH.

The RT–11 foreground/background (FB) and extended monitor (XM) calls are not supported; they are essentially "no-ops" on RSTS/E systems. They do not return errors.

In the remaining sections of this chapter, the directives are described in detail, in alphabetical order. Note that for all calls, you need only specify the first six characters. For example, the abbreviation .SAVES will work as well as .SAVESTATUS.

All the examples in this chapter show Version 2 expansions. Version 1 expansions require different arguments in the calls. See RT–11 system documentation if you need to use Version 1 expansions.

Table 7–1: RT–11 Calls Not Functional on RSTS/E

Ignored	Return Error 0	Unpredictable
.CMKT	.CDFN	.INTEN
.HERR		.MFPS
.LOCK		.MTPS
.QSET		.SYNCH
.MRKT		
.RELEASE		
.SERR		
.UNLOCK		
All FB calls		
All XM calls		

Table 7–2: RT11 Run-Time System Directives

Mnemonic	EMT	Function Code	Description
..V1.., ..V2..	—		Allows you to assemble directives as they would be expanded under Version 1 or Version 2 of RT11.
Input/Output and File Operations			
.ENTER	375	2	Creates a file and opens it on a channel.
.LOOKUP	375	1	Opens an already existing file on a channel. .LOOKUP can also be used for non-file-structured I/O operations.
.READ/.READW/.READC	375	10	Transfers data from a file or device on an open channel to a memory buffer. I/O is synchronous on RSTS/E systems, so for all three reads, control returns to the user program only when the transfer is complete. .READ and .READW are identical on RSTS/E systems. .READC transfers control to a user-specified completion routine when the transfer is complete.

(continued on next page)

Table 7–2: RT11 Run-Time System Directives (Cont.)

Mnemonic	EMT	Function Code	Description
Input/Output and File Operations			
.WRITE/.WRITW/.WRITC	375	11	Transfers data from a memory buffer to a file or device on an open channel. As with the .READ/W/C directives, .WRITE and .WRITEW operate identically on RSTS/E; .WRITC transfers control to a completion routine after the transfer is complete.
.CLOSE	374	6	Terminates I/O operations on a channel, performing cleanup operations (positioning tape, and so forth) and releasing the channel for other use.
.PURGE	374	3	Releases a channel without performing the normal cleanup operations. In particular, a file opened with .ENTER and not yet closed is not made a permanent file.
.HRESET	357	—	Closes all open channels, without performing normal cleanup operations. In particular, files opened with .ENTER and not yet closed are not made permanent.
.SRESET	352	—	Functions the same as .HRESET on RSTS/E.
.SAVESTATUS	375	5	Closes a file, keeping the information needed to .REOPEN the file later.
.REOPEN	375	6	Opens a file closed with .SAVESTATUS. Using the .SAVESTATUS/.REOPEN combination is easier than using standard opens and closes; the device, file name and type, and project-programmer number are saved automatically with .SAVESTATUS. You only need to keep track of the address in which you stored the file information to reopen it.
.DELETE	375	0	Deletes a disk or DECtape file.
.RENAME	375	4	Changes a file's name on disk or DECtape.

(continued on next page)

Table 7–2: RT11 Run-Time System Directives (Cont.)

Mnemonic	EMT	Function Code	Description
Input/Output and File Operations (Cont.)			
.PRINT	351	—	Displays an ASCII string on the job's terminal.
.GTLINE	345	—	Accepts a line of input from the job's terminal.
.TTYIN/.TTINR	340	—	Transfers one character from the job's terminal to R0.
.TTYOUT/.TTOUTR	341	—	Transfers one character from R0 to the job's terminal.
Special Functions Related to I/O			
.SPFUN	375	32	Performs special functions for disk, terminal, magnetic tape, and flexible diskette.
.CSIGEN	344	—	Analyzes an ASCII string (in memory or from the job's terminal) for a standard RT11 command, and opens files accordingly.
.CSISPC	345	—	Analyzes an ASCII string (in memory or from the job's terminal) for a standard RT11 command, and sets up device blocks for later opens.
.DOFSS (RSTS/E Only)	365		Analyzes an ASCII string in memory for a RSTS/E file name and returns information as for .FSS. Must be used instead of .FSS (general monitor directive) for programs running under RT11 run-time system.
.DSTATUS	342	—	Returns information about a device to an area in the user program.
.WAIT	374	0	Checks to see if channel is open.
.SETFQB (RSTS/E Only)	360		Sets the FIRQB (Section 2.4) from information returned by .CSISPC (Section 7.7).
.CLRFQB (RSTS/E Only)	370		Clears the FIRQB (Section 2.4).
.CLRXRB (RSTS/E Only)	371		Clears the XRB (Section 2.4).
.FETCH	343	—	Checks to see if device is available on the system.

(continued on next page)

Table 7–2: RT11 Run-Time System Directives (Cont.)

Mnemonic	EMT	Function Code	Description
Special Functions Related to I/O			
.RCTRLO	355	—	Restarts programmed output to job's terminal that was stopped by user-typed CTRL/C or CTRL/O.
.SCCA	374	35	Causes a user-typed CTRL/Z to be passed on to the user program.
.SETCC (RSTS/E Only)	362		Defines an entry point for CTRL/C traps; allows the user program to handle its own CTRL/Cs.
.ERRPRT (RSTS/E Only)	364		Prints text corresponding to RSTS/E error code on the job's terminal.
System/Job Information			
.DATE	374	12	Returns the current date.
.DATTIM (RSTS/E Only)	361		Converts date or time from RSTS/E internal format to ASCII string.
.GTIM	375	21	Returns the time of day.
.GTJB	375	20	Returns the current job number.
.GVAL	375	34	Returns the contents of a word in the scratch pad area.
.SETTOP	354	—	Sets location 50 (indicating top of user job image area) to the address specified.
Execution Control			
.CHAIN	374	10	Transfers control to another program (file) to be run under the RT11 run-time system.
.GETCOR (RSTS/E Only)	366		Expands size of user job image. Must be used when .CHAIN is used to transfer control to a program larger than the current program. Must be used instead of the .CORE (general monitor directive) for programs running under RT11 run-time system.
.DOCCL (RSTS/E Only)	367		Examines a string for valid RSTS/E CCL command. If valid, transfers control to new program. Must be used instead of the .CCL (general monitor directive) for programs running under RT11 run-time system.

(continued on next page)

Table 7–2: RT11 Run-Time System Directives (Cont.)

Mnemonic	EMT	Function Code	Description
Execution Control (Cont.)			
DORUN (RSTS/E Only)	363		Transfers control to another program (file) that runs under a non–RT11 run-time system. Must be used instead of .RUN (general monitor directive) for programs running under RT11 run-time system.
.TWAIT	375	4	Timed wait — suspends job execution for specified number of clock ticks.
.TRPSET	375	3	Sets address to be entered for reserved instruction and odd address errors; normally trapped to kernel mode addresses 4 and 10.
.EXIT	350	—	Terminates execution of the calling job. Note that you cannot use the RSTS/E .EXIT from a program running under the RT11 run-time system.
.SFPA	375	30	Sets address to be entered for floating point errors.

7.2 .CHAIN — Pass Control to Another Program Under RT11

The .CHAIN directive transfers control to another program without altering the run-time system. The .CHAIN leaves I/O channels from the old program still open for the new program. Thus, you can "get around" the 27K word limit for programs to be run under the RT11 run-time system. If the program you are chaining to takes more space than the current program, use the .GETCOR directive (Section 7.18) to expand the space allocated for the user job image area before executing the .CHAIN.

Virtual addresses $500–507_8$ are expected to contain the device and file name of the program to chain to, in the RT11 "device block" format, as described in Section 6.5.2. The area from locations $510–777_8$ can be used to pass information between the chained programs. Make sure, though, that the program you are chaining to does not inadvertently destroy the data in locations $510–777_8$ by pushing data on the stack. The linker normally defaults the initial stack setting to 1000_8.

An executing program can tell whether it was chained to or run by some other means by examining bit 8 of the Job Status Word (JSW), described in Section 6.6. If bit 8 = 1, the program was invoked with .CHAIN, and locations $500–777_8$ are preserved from the program that issued the .CHAIN. If bit 8 = 0, the program was not entered with an RT11 .CHAIN.

The .CHAIN directive works only for RT11–to–RT11 transfers. If you want to chain to a program to be run under some run-time system other than RT11, use the .DORUN directive. The description of .DORUN also describes what happens for RT11 programs receiving control from non–RT11 programs.

Macro Call

```
.CHAIN
```

Request Format

R0 =

10	0

Errors

Since .CHAIN is intended to pass control to another program, no errors are returned to the calling program. If the new program cannot be chained to, the .CHAIN is abandoned, and the keyboard monitor is entered. (Control passes to the P.NEW entry point of the RT11 run-time system, which is the "keyboard monitor" entry point for RT11.)

Example

The following code requests an expansion to 20K words and chains to a file named MYFILE.SAV in the user's account:

```
FILE:   .RAD50    /MYFILESAV/     ;Define text passed in chain
           .
           .
           .
        MOV       #20.,R0         ;Indicate 20.K now needed
        .GETCOR
        CLR       500             ;Default for the device
        MOV       FILE,502        ;Move
        MOV       FILE+2,504      ;  the
        MOV       FILE+4,506      ;    file name
        .CHAIN                    ;Invoke the new program
```

7.3 .CLOSE — Close a Channel

Use the .CLOSE directive to terminate input/output on a particular channel. The .CLOSE directive (1) performs whatever action is appropriate to the device (such as updating a disk directory to indicate a new file, repositioning a magnetic tape, and so forth), (2) frees the RT11 channel number for use with another device, and (3) frees the RSTS/E channel from its association with the RT11 channel number (effectively making another channel available).

A .CLOSE on a file opened with .ENTER makes the file permanent; the file is tentative until that time. Note that, unlike the RT–11 operating system, RSTS/E does not deallocate unused space. Thus, if you preallocated space when you opened the file with .ENTER, the file contains all this space after the .CLOSE.

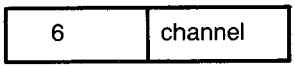
Macro Call

.CLOSE *channel*

where *channel* is a channel number in the range $0–17_8$. See Section 6.5.1 for rules on channel numbers.

R0 Format

(R0) =



Errors

No errors are possible with .CLOSE. A .CLOSE for a channel that is not currently open is simply ignored.

Example

The following closes channel 5:

```
.CLOSE #5
```

7.4 .CLRFQB — Clear the FIRQB

This directive clears the FIRQB area in the low 1000_8 bytes of virtual memory (see Section 2.4). No data is passed or returned for this directive.

Macro Call

```
.CLRFQB
```

Note that .CLRFQB, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example below.

Errors

No errors are possible with the .CLRFQB directive.

Example

```
.CLRFQB = EMT+370
          .
          .
          .
        .CLRFQB
```

7.5 .CLRXRB — Clear the XRB

This directive clears the XRB area in the low 1000_8 bytes of virtual memory (see Section 2.4). No data is passed or returned with .CLRXRB.

Macro Call

.CLRXRB

Note that .CLRXRB, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example below.

Errors

No errors are possible with the .CLRXRB directive.

Example

```
.CLRXRB = EMT+371
          .
          .
          .
        .CLRXRB
```

7.6 .CSIGEN — Examine String for RT Command, Open Files

The .CSIGEN directive examines a string (either in memory or from the job's keyboard) to process a standard RT11 command string. If the string is to be accepted from the job's keyboard, the .CSIGEN first prints an asterisk (*) prompt on the keyboard and then waits for a line to be typed.

If the string is in the correct form, any files or devices open on RT11 channels 0–10 are closed and then either reopened or left inactive, depending on whether or not a file is given for that "slot":

Any valid RSTS/E file specification can appear in the string, including file specifications with project-programmer numbers and logical names. However, the file specifications cannot contain wildcards. Files on the left side of the equal sign (if any) are opened with .ENTER, and the files on the right side of the equal sign (if any) are opened with .LOOKUP. For example, consider the string:

```
NOW.OBJ,NOW.LST,NOW.CRF=THEN.MAC
```

The file NOW.OBJ would be opened with .ENTER on channel 0; NOW.LST, on channel 1; NOW.CRF, on channel 2. The file THEN.MAC would be opened with .LOOKUP on channel 3. Two more examples are:

```
NOW.OBJ,,NOW.CRF
```

The file NOW.OBJ would be opened with .ENTER on channel 0 and NOW.CRF on channel 2. Channels 1 and 3–10 would be inactive.

```
=LP:
```

The line printer would be opened with .LOOKUP on channel 3.

RSTS/E Options

The file specifications can include any of the standard RSTS/E options processed by the monitor itself (see .FSS, Section 3.10). These include: /CLUSTERSIZE, /RONLY, /MODE, /PROTECTION, /FILESIZE or /SIZE, and /POSITION. Any of these options encountered in a string examined by .CSIGEN will be handled properly.

RT11 Options

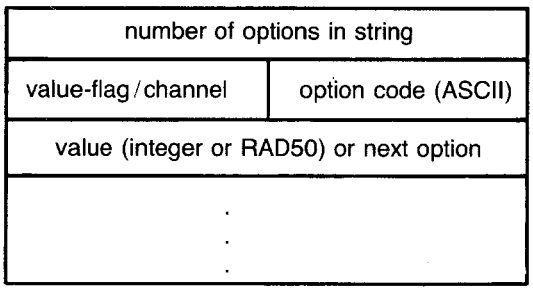
The .CSIGEN directive under RSTS/E also does RT11–type analysis of options. One or more options can appear after any of the file specifications. The general form of an option is:

/a:val1:val2:...:valn

The slash indicates an option; it must be followed by a one-character option code. (The character does not have to be printable.) The slash/character pair can be followed by one or more values. A colon indicates that a value follows. If given, the value must be followed by:

1. An octal number in the range 0 through 177777.
2. A decimal integer in the range –32768. through 32767. (the terminating period must appear).
3. From one to three alphanumeric characters, the first of which must be alphabetic.

The .CSIGEN directive simply examines any options in the string to ensure that they are in the correct form, reformats them, and pushes them on the stack for the user program to examine. The information pushed on the stack is ordered by channel; if an option is given for the file specification for channel 0 (assuming there is a file specification in the "first slot"), it is pushed on the stack first, followed by option information for the file specification for channel 1 (if any), and so forth. The last word pushed on the stack is an integer value indicating the total number of options in the string. An option having two values is formatted as two options.



Bit 15 of a value-flag/channel byte is set to 1 if the option had an associated value, to 0 if it did not. Thus, if bit 15 is set, the next word is the value associated with the option. (The option code itself is in the low byte of this word.) If bit 15 is 0, the next word is the next option (if any). Bits 8–14 of a value-flag/channel byte indicate the RT11 channel number with which the file specification having the option is associated.

The example given at the end of this section includes option switches and shows more clearly how options and values are pushed on the stack.

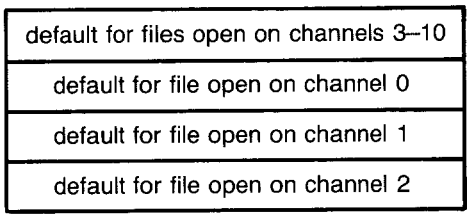
.CSIGEN

Macro Call

.CSIGEN *devspc,defext,*[*cstrng*][,*linbuf*]

devspc Ignored on RSTS/E systems.

defext The address of a four-word area containing default file types, in RAD50 format. The format of the area is:



These values are then used when no corresponding file type is given in the examined string. For example, for typical use in the RT11 environment, the first word would contain MAC in RAD50 format; the second, OBJ; the third, LST; and the fourth, CRF.

cstrng The address of the ASCIZ string to be examined or a #0 if input is to come from the job's terminal. If this argument is not specified, input is automatically taken from the job's terminal.

linbuf The address where the original string (if accepted from the job's terminal) is to be stored. This area should be 81_{10} bytes long. The string will be terminated with a zero byte, rather than a carriage-return/line-feed combination.

If this argument is omitted, the original string is not kept.

Errors

If an error occurs when input is accepted from the job's terminal, the user gets another try. An error message is printed on the terminal, along with another * prompt, and the directive waits for another line to be typed.

If the string being examined is in memory and an error occurs, the carry bit is set, and one of the following error codes is returned in byte 52. The options and option-count are pushed from the stack.

Code	Explanation
0	Illegal command (bad separators, illegal file name, command too long, and so forth).
1	A device specification was not recognizable or is not available on the system.
2	Unused.
3	An attempt to open a file with .ENTER failed because of a full directory.
4	An input file (to the right of the equal sign) was not found in a .LOOKUP.

Example

```
DEFEXT: .RAD50     /MACOP1OP2OP3/
            .
            .
            .
        .CSIGEN    #0,#DEFEXT,#0
```

Assume that the line typed at the job's terminal is:

```
OUTFIL/M:32,,,ODDFIL=[2,200]HISFIL,DT0:HERFIL/G:NRD:20
```

When control is returned to the user program following the .CSIGEN, the file OUTFIL.OP1 on the public structure has been opened with .ENTER on channel 0, and the file ODDFIL.OP3 on channel 3. The file HISFIL.MAC on the public structure, under account [2,200], has been opened with .LOOKUP. (If the calling job were not privileged, an error message could have been displayed on the terminal, and another line accepted.) The file HERFIL.MAC on DECtape unit 0 has also been opened with .LOOKUP. The stack contains:

	Word	Octal Value	Explanation
(SP) →	1	3	Three options were found in the string. (The G option has two values.)
	2	103107	Last option has a value, and the option is associated with channel 6. The option code is G.
	3	055124	Value of option is NRD in RAD50 format.
	4	103107	Next-to-last option has a value, and the option is associated with channel 6. The option code is G.
	5	000020	The value of the option is 20 (octal).
	6	100115	The first option has a value, and the option is associated with channel 0. The option code is M.
	7	000036	The value of the option is 36_8 (32_{10}).

7.7 .CSISPC — Examine String for RT Command, Create Devblk

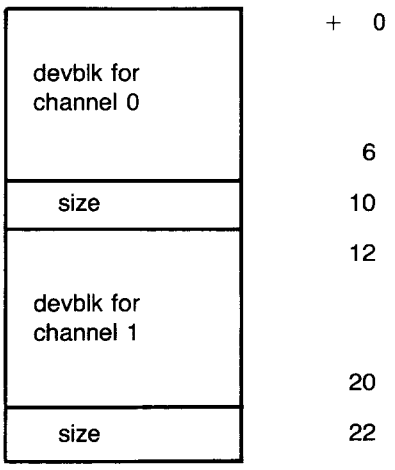
Like .CSIGEN, the .CSISPC directive parses a string (either in memory or from the job's terminal) to see if it is a valid RT11–type command string. If the string is to be accepted from the job's terminal, .CSISPC first displays an asterisk (*) prompt on the screen and then waits for a line to be typed. Instead of closing and opening files, however, .CSISPC simply returns a 39_{10}–word block of information about the files named in the string. The user program can later use this information to open the files, if desired. See the discussion of .CSIGEN (Section 7.6) for command string format and option processing.

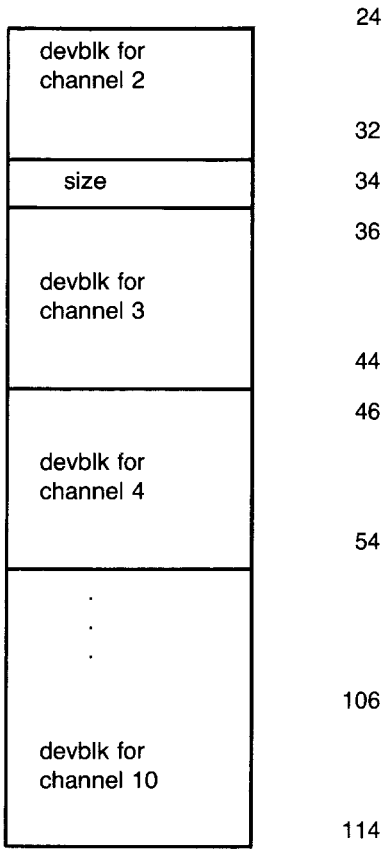
Note that .CSISPC, like .CSIGEN, stores RSTS/E–specific file information (project-programmer number, and so on) within the scratch pad area. Furthermore, you can issue a .SAVESTATUS or .SETFQB referencing the device-block information returned by .CSISPC (although .SAVESTATUS will work only for input files to the right of the equal sign — opened with .LOOKUP). If you use .CSISPC in this way, be sure that you reference addresses in the "outspec" area returned by .CSISPC. If you move the information in the outspec area and then refer to the new area in .SAVESTATUS or .SETFQB, the RSTS/E–specific information, such as project-programmer number, will not be used. (The emulator keeps a pointer in the impure area relating the project-programmer number and other RSTS/E–specific information to the area returned by .CSISPC.)

Macro Call

.CSISPC *outspec,defext,cstrng*[,*linbuf*]

outspec The address of a 39_{10}–word area to contain the file descriptors produced by .CSISPC. This area can overlay the space allocated to *cstrng*, if desired. The format of the information returned is:





Information for the files for channels 0, 1, and 2 is stored in 5–word blocks. The first four words are RAD50 information in the standard RT11 "device block" format (see Section 6.5.2). The last word is the file size indicated by a /FILESIZE or /SIZE option, if any was encountered for these files in the string. (Since files on the left side of the equal sign are opened with .ENTER, they can have a size for preallocation.) Information for the files for channels 3–10 is stored in 4–word blocks, in the standard RT11 "device block" format. If any files are omitted in the string, the corresponding 4– or 5–word block is filled with zeros.

defext The address of a four-word block containing the RAD50 default file types. (See the discussion of .CSIGEN for the format of this area.) These defaults are returned to the *outspec* area if no types are given in the examined string.

cstrng The address of the ASCIZ string to be examined or a #0 if input is to come from the job's terminal. If this argument is not specified, input is automatically taken for the job's terminal. The string must follow the rules for an RT11 command string, as described for .CSIGEN (Section 7.6).

linbuf The address where the original string (if accepted from the job's terminal) is to be stored. This area should be at least 81_{10} bytes long. The string is terminated with a zero byte instead of a carriage-return/line-feed combination.

If this argument is omitted, the original string is not kept.

Errors

If an error occurs when input is accepted from the job's terminal, the user gets another try. An error message is printed on the terminal, along with another * prompt, and the directive waits for another line to be typed.

If the string being examined is in memory and an error occurs, the carry bit is set, and one of the following error codes is returned in byte 52. The options and option-count are pushed from the stack.

Code	Meaning
0	Illegal command line (bad separators, illegal file name, command too long, and so forth).
1	A device specification was not recognizable or is not available on the system.

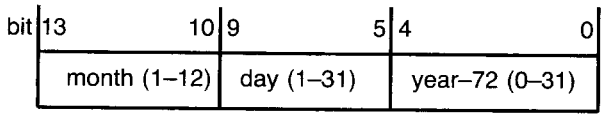
Example

The following code accepts a line of input from the terminal, checks it for RT11 command format, and stores the resulting device block information in a 39_{10}–word area called OUTSPC:

```
DEFEXT:    .WORD      0,0,0,0        ;NO DEFAULT FILE TYPES
OUTSPC:    .BLKW      39.            ;DEVICE BLOCK AREA
               .
               .
               .
           .CSISPC    #OUTSPC,#DEFEXT,#0
```

7.8 .DATE — Return Current Date to R0

The .DATE directive returns the current month, day, and year to R0 in the following format:



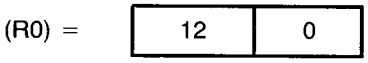
The year value in bits 4–0 is the current year minus 72.

Note that this directive gets its date information from the RSTS/E clock. Unlike the RT–11 operating system, you do not set the values returned with the DATE keyboard monitor command.

Macro Call

.DATE

R0 Format



Errors

No errors are returned.

Example

The subroutine shown below can be assembled separately and linked to a user program:

```
.TITLE DATE.MAC
;
;CALLING SEQUENCE:
;
;       JSR         PC,DATE
;
;INPUT:  NONE
;
;OUTPUT: R0 = DAY (1-31)
;        R1 = MONTH (1-12)
;        R2 = YEAR - 72
;
```

.DATE

```
        .MCALL      .DATE
DATE::
        .DATE                       ;GET THE SYSTEM DATE
        MOV         R0,R2           ;COPY THE DATE
        BIC         #^C37,R2
        ASR         R0              ;PUT THE MONTH ON A BYTE BOUNDARY
        ASR         R0              ;
        MOV         R0,R1           ;COPY THE DATE
        SWAB        R1              ;PUT THE MONTH IN THE LOW BYTE
        BIC         #^C37,R1        ;ISOLATE THE MONTH
        ASR         R0              ;SHIFT THE DAY TO THE BYTE BOUNDARY
        ASR         R0              ;
        ASR         R0              ;
        BIC         #^C37,R0        ;ISOLATE THE DAY
        CLC                         ;INDICATE NO ERROR
        RTS         PC

        .END
```

7.9 .DATTIM — Return Date or Time

The .DATTIM directive is available only to RT11 emulator users under RSTS/E; it is not available under the RT–11 operating system. .DATTIM converts a date or time from the RSTS/E system internal format (as returned by .DATE, Section 3.7) to an ASCII string suitable for printing. The format of the string is the same as that returned by the BASIC–PLUS DATE$ and TIME$ functions. That is, a date will be in the form "27–Jun–81" or, if the system uses the numeric date option, in the form "81.06.27". Similarly, a time will be in the form "02:30 PM" or "14:30".

The .DATTIM directive assumes that R0 contains the address of the area to which the string is to be returned. It also expects to find the date or time in RSTS/E system internal format in the XRB. If a date is desired, it should be in XRB+0. If a time is desired, the word at XRB+0 should be cleared and the time stored in XRB+2. Note that these are the locations returned by the RSTS/E .DATE directive, Section 3.7.

You can request a specific format for date or time rather than accepting the format selected at system generation. If you set bit 15 of XRB+0 (date) or XRB+2 (time), you request a specific format rather than the default selected at system generation. Bit 15 = 0 indicates the default. If bit 15 = 1, then bit 14 = 0 implies numeric format; bit 14 = 1 indicates alphanumeric format.

Upon return from the call, R0 contains the address of the first byte after the time or date string.

Macro Call

```
.DATTIM
```

Note that .DATTIM, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Errors

No errors are possible with .DATTIM.

Example

The following program prints the current date and time at the job's terminal.

.DATTIM
(RSTS/E Only)

```
            EXAMPLE:

            .MCALL    .EXIT,.PRINT
            .PRIV = EMT+377           ;PREFIX EMT TO RSTS/E (FOR .DATE)
            .DATTIM   = EMT+361       ;DATE/TIME EMT TO RT11

            .CSECT

START:      .PRIV,    .DATE           ;GET DATE INFORMATION FROM RSTS/E
            MOV       XRB+2,-(SP)     ;SAVE THE TIME
            MOV       DATE$,R0        ;SET ADDRESS FOR RETURNED STRING
            .DATTIM                   ;GET THE DATE STRING
            CLRB      (R0)+           ;CLEAR THE LAST BYTE FOR ASCIZ STRING
            .PRINT    #DATMSG         ;PRINT THE DATE MESSAGE
            CLR       XRB             ;CLEAR THE DATE FROM XRB
            MOV       (SP)+,XRB+2     ;SET THE TIME IN XRB
            MOV       TIME$,R0        ;SET THE ADDRESS FOR THE TIME STRING
            .DATTIM                   ;GET THE TIME STRING
            CLRB      (R0)+           ;MAKE THE STRING ASCIZ
            .PRINT    #TIMMSG         ;PRINT THE TIME MESSAGE
            .EXIT                     ;END THE PROGRAM


DATMSG:     .ASCII    /THE DATE IS /
DATE$:      .BLKW     12
TIMMSG:     .ASCII    /THE TIME IS /
TIME$:      .BLKW     12

 .END       START
```

7.10 .DELETE — Delete File from Disk or DECtape

The .DELETE directive deletes a file from a disk or DECtape, deleting it from the device directory and releasing the space for other use. The file description given in the call includes a device, file name, and type. Set the PPN offset (Section 6.7) to include a project-programmer number for the file, if desired. Otherwise, the project-programmer number of the calling job is used.

Note that on RSTS/E systems, unlike the RT–11 operating system, it is possible to delete files which are open; no error is returned.

Macro Call

.DELETE *area,channel,devblk,seqnum*

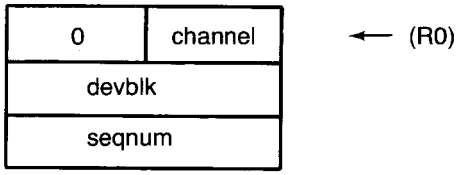
area Address of a three-word argument block.

channel Ignored on RSTS/E systems.

devblk The address of a four-word device block containing the device and file specification. See Section 6.5.7 for the format of this area. See also the PPN offset description in Section 6.7 for specifying a project-programmer number.

seqnum Ignored on RSTS/E systems.

Argument Block Format



Errors

Code	Meaning
1	The file named cannot be found.
2	The device specified is not a disk or DECtape.

Example

The following code deletes the file name FILNAM.TYP in the user's account on disk unit 1:

```
AREA:       .BLKW       3
DEVBLK:     .RAD50      /DK1FILNAMTYP/
                .
                .
                .
            .DELETE     #AREA,#1,#DEVBLK
```

7.11 .DOCCL — Do a RSTS/E .CCL

This directive allows a program running under control of the RT11 run-time system to examine a string to see if it is a valid RSTS/E Concise Command Language (CCL) command. It *must* be used by programs running under the RT11 run-time system, rather than the general monitor .CCL command. If the string is a valid CCL command, the RT11 run-time system restores the low 1000 bytes of memory to what is expected by the monitor and issues a general monitor .CCL command. (See Section 3.3 for a description of .CCL.)

The starting address of the string to be examined is passed in R0. The string itself must be ASCIZ (terminated with a zero byte).

Macro Call

.DOCCL

Note that .DOCCL, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Errors

The errors returned for .DOCCL are the same, and in the same format, as for .CCL, Section 3.3.

Example

The following code executes the CCL command PIP:

```
.DOCCL = EMT+367
STG:    .ASCIZ    /PIP/
          .
          .
          .
        MOV       #STG,R0
        .DOCCL
```

7.12 .DOFSS — Do a RSTS/E .FSS

The .DOFSS directive examines a string to see if it is a valid RSTS/E file specification. The RT11 emulator restores the low 1000_8 bytes of memory to what the monitor expects and executes a .FSS directive; you *must* use .DOFSS rather than .FSS for a program running under the RT11 run-time system.

The .DOFSS directive assumes that R0 contains the starting address of the string and that the string itself is ASCIZ (terminated with a zero byte). See the discussion of .FSS (Section 3.10) for an explanation of how the string is processed. The data returned on completion of .DOFSS is the same as for .FSS.

Note that .DOFSS, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Macro Call

.DOFSS

Errors

The errors for .DOFSS are the same, and in the same format, as for .FSS (Section 3.10).

Example

The following code checks the string FILE.NAM to see if it is a valid RSTS/E file specification:

```
.DOFSS = EMT+365
STG:    .ASCIZ     /FILE.NAM/
           .
           .
           .
        MOV        #STG,R0
        .DOFSS
```

7.13 .DORUN — Chain to Non–RT11 RTS Program

Use the .DORUN directive to transfer control to programs that run under a non–RT11 run-time system. The .DORUN directive must be used from programs running under the RT11 run-time system, rather than the .RUN directive, since RT11 keeps user logical information in a different area in low-core than the monitor expects. The .DORUN directive causes the emulator to restore the user logical information to its normal place in the low 1000_8 bytes of memory and translates an RT11 file specification to the proper form in the FIRQB for a .RUN.

The .DORUN assumes that R0 contains the address of a five-word file definition. The first four words contain a normal RT11 "device block" specification (Section 6.5.2); the last word contains a parameter to be passed to the run-time system at FIRQB+FQNENT. If the address passed in R0 points to a device-block within an "outspec" area returned by a previous .CSISPC directive, all of the RSTS/E–specific information from the file specification (project-programmer number, and so forth) will be loaded into the proper area in the FIRQB. Values in the first six words of the scratch pad area will take precedence over the values found in the string examined by .CSISPC.

Macro Call

```
.DORUN
```

Note that .DORUN, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Errors

Errors are returned as for a .RUN directive (Section 3.20).

Example

The following program uses .CSISPC to scan a user-typed file specification, ensures that the program is entered with a zero parameter passed to the run-time system, and then attempts to execute the file with .DORUN. If an error occurs for either the .CSISPC or the .DORUN, an error message is printed at the job's terminal.

```
.SETFQB = EMT+360 ;DEFINE EMT FOR SETFQB
.DORUN  = EMT+363 ;DEFINE EMT FOR .DORUN

        .CSISPC   OUTSPC,DEFEXT,0 ;READ, SCAN COMMAND STRING
        BCS       ERROR           ;COMMAND STRING IN ERROR
        MOV       OUTSPC,R0       ;GET DEVBLK FOR FIRST FILE
        CLR       4*2(R0)         ;PASS 0 AT FIRQB+FQNENT
        .DORUN                    ;TRY TO RUN THE FILE
ERROR:  .PRINT    #ERRMES         ;IF CONTROL RETURNS, ERROR OCCURRED
        .EXIT

ERRMES: .ASCIZ    /?CAN'T RUN THAT FILE/
```

7.14 .DSTATUS — Return Device Status

The .DSTATUS directive returns information about a device to a four-word area defined in the user program.

Macro Call

.DSTATUS *statblk,devnam*

statblk Address of a four-word area to contain the returned status information.

devnam Address of one word containing a device and unit number, in RAD50 format. The word can also be a three-character user logical name in RAD50 format.

Information Returned to Statblk

status word (see below)
handler size (= 0 for RSTS/E)
load address (=160000 for RSTS/E)
device size (= 0 for RSTS/E)

Status Word Format

flags	handler index

The low byte contains the RT11 handler index for the device:

Octal Value	Device
0	disk
1	DT (DECtape)
3	LP (line printer)
4	KB (keyboard)
7	PR (paper tape reader)
10	PP (paper tape punch)
11	MT (magnetic tape)
14	CR (card reader)

The high byte describes the device characteristics. Word values are shown below; note that you can combine the device-characteristics flags. A value of 140000 for the entire word would indicate a file-structured, read-only disk.

Device Characteristics (Octal Word Value)	Meaning
100000	Device is file-structured
040000	Device is read-only
020000	Device is write-only

Errors

Code	Meaning
0	The device named in the *devnam* argument is not known on the system.

Example

The following code checks for status information for MT0:

```
STAT:       .BLKW       4
DEVNAM:     .RAD50      /MT0/
                .
                .
                .
            .DSTATUS    #STAT,#DEVNAM
```

7.15 .ENTER — Open File for Output

In RSTS/E terms, the .ENTER directive opens a file for output; that is, it creates a new file and associates the file with a channel number. You then use the channel number to refer to the file for other I/O operations.

A RSTS/E "tentative open" is done; that is, the file is not permanent until a .CLOSE is executed with the same channel number as specified in the .ENTER. Then, the tentative file is made permanent, and any already existing file of the same name on the same device is deleted. A .PURGE, .HRESET, or .SRESET destroys the tentative file; any already existing file of the same name on the same device is left intact.

You can preallocate space for disk files by setting an argument in the call. Note that you can also set a project-programmer number, protection code, mode, and other RSTS/E parameters applicable to opening a file for output by setting words in the scratch pad area (see Section 6.7).

Macro Call

.ENTER *area,channel,devblk,len,seqnum*

area Address of a four-word argument block.

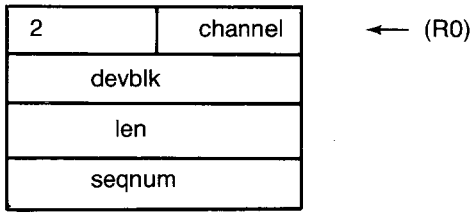
channel A channel number in the range $0–17_8$. See Section 6.5.2 for rules on channel numbers.

devblk The address of a four-word device-block containing the device and file specifications. See Section 6.5.2 for the format of this area.

len File size to be preallocated for the file, in 512–byte blocks (for disk files only). Values may range from 1–32767. For large disk files on RSTS/E, you must set the MSB byte in the scratch pad area (Section 6.6). A zero or negative value is ignored on RSTS/E.

seqnum This argument is ignored on RSTS/E systems.

Argument Block Format



Errors

Code	Meaning
0	Channel is already in use.
1	The file cannot be preallocated to the specified size, or the device is full.

Example

The following code opens the file NEWFIL.SAV on MT0 for output:

```
AREA:    .BLKW    4
DEVBLK:  .RAD50   /MT0NEWFILSAV/
            .
            .
            .
         .ENTER   #AREA,#1,#DEVBLK,#0,#0
```

7.16 .ERRPRT — Print RSTS/E Error Message

The .ERRPRT directive prints the RSTS/E error message text corresponding to an error code on the user terminal. The call assumes that the error code is in R0. The text is printed without a carriage-return/line-feed.

Macro Call

.ERRPRT

Note that .ERRPRT, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Errors

No errors are possible with the .ERRPRT directive.

Example

```
        .MCALL      .EXIT,.PRINT,.REGDEF
        .REGDEF

.ERRPRT = EMT+364

.CSECT

START:  MOV         256.,R1     ;THE NUMBER OF ERRORS TO PRINT
        CLR         R0          ;START WITH ERROR ZERO
10$:    .ERRPRT                 ;PRINT ERROR TEXT
        MOV         R0,-(SP)    ;SAVE R0
        .PRINT      #CRLF       ;PRINT A CR/LF
        MOV         (SP)+,R0    ;RESTORE R0
        INC         R0          ;INCREMENT R0 FOR NEXT MESSAGE
        SOB         R1,10$      ;LOOP UNTIL ALL 256 ARE PRINTED
        .EXIT

CRLF:   .BYTE       0
.END    START
```

7.17 .EXIT — Program Exit

The .EXIT directive terminates execution of the calling job. Control passes to the job keyboard monitor* at the keyboard monitor entry point (P.NEW, Section 2.5).

If R0 = 0 when the .EXIT is executed, any channels still open will be released, without the usual cleanup operations. In particular, any temporary files (opened with .ENTER and not yet closed) will be deleted.

Note that the RT11 .EXIT *must* be used from programs running under the RT11 run-time system, rather than the general monitor .EXIT or .RTS directives.

Macro Call

.EXIT

Errors

No errors are possible with the .EXIT directive.

Example

The following code exits such that files opened with .ENTER and not yet closed will be deleted:

```
SAVOLD:  MOV      #0,R0
         .EXIT
```

* The job keyboard monitor will be the default keyboard monitor unless the job has run the SWITCH utility or has issued a general monitor .RTS directive (Section 3.19) to specify a job keyboard monitor.

7.18 .FETCH — Check for Device Available

On RSTS/E systems, the .FETCH directive checks to see if a specified device is available on the system.

Macro Call

.FETCH *addr,devnam*

addr Ignored on RSTS/E.

devnam The address of a one-word, RAD50 device name. The format of this word is the same as that described for the first word of a device block (Section 6.5.1).

Errors

Code	Meaning
0	The device specified does not exist on the system.

Example

The following code checks to see if LP0 is available on the system:

```
DEVNAM:  .RAD50    /LP0/
             .
             .
             .
         .FETCH    #0,#DEVNAM
```

7.19 .GETCOR — Change Job Image Size

This directive changes the amount of space currently allocated for the user job image. You *must* use .GETCOR rather than the general monitor .CORE directive; with .GETCOR, the RT11 emulator checks to ensure that the requested expansion will not destroy the scratch pad area, moves the scratch pad area if necessary, and then expands the job image size using the .CORE directive (Section 3.6). Thus, the same rules concerning size changes for .CORE also apply to .GETCOR.

The .GETCOR assumes that the new job image size, in K words, is in R0. If the required expansion is not possible, the carry bit will be set on return from the call. Otherwise, the size change succeeded, and word 54 now points to the new location for the start of the scratch pad area.

Note that you must use .GETCOR to expand the job image size before you use .CHAIN to transfer control to a program that requires more memory than the current program.

Macro Call

.GETCOR

Note that .GETCOR, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Errors

If the requested change cannot be made, the carry bit is set on return from the call.

Example

The following code uses .GETCOR to expand the user job image area to 24 K words before doing a .CHAIN:

```
.GETCOR = EMT+366
          .
          .
          .
        MOV         #24.,R0
        .GETCOR
        BCS         ERRTN
        .CHAIN
```

7.20 .GTIM — Return Time-of-Day

The .GTIM directive returns the current time of day to a two-word area defined by the user program. The current time is given in terms of clock ticks past midnight. There are 60 clock ticks per second for a 60–Hz line frequency clock and 50 clock ticks per second for a 50–Hz line frequency clock or a crystal-controlled clock.

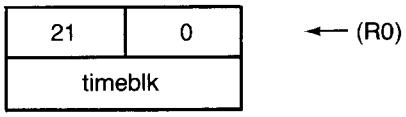
Macro Call

.GTIM *area,timblk*

area The address of a two-word argument block.

timblk The address of a two-word block where the time is to be returned. The first word will contain the most significant bits of the current time, and the second word the least significant bits.

Argument Block Format



Errors

No errors are possible with the .GTIM directive.

Example

The following code returns the current time to a two-word area named TIME:

```
AREA:    .BLKW    2
TIME:    .BLKW    2
          .
          .
          .
         .GTIM
```

7.21 .GTLIN — Get Line from Job's Terminal

The .GTLIN directive accepts a line of input from either the job's terminal or core common and stores it in a buffer for later examination by the user program. You can also specify an ASCIZ string to be displayed at the terminal before the line is accepted.

.GTLIN reads the line from core common instead of the job's terminal if the program is invoked with a parameter word of 8192_{10}. (For example, the parameter word has this value when a program is run by a CCL command.) A second .GTLIN causes the program to exit.

By default, .GTLIN returns a string in uppercase with all spaces removed. To obtain lowercase letters, set bit 14 in the Job Status Word before you issue the .GTLIN.

Macro Call

.GTLIN *linbuf*[,*prompt*]

linbuf The address of an area to receive the input line. Up to 81_{10} bytes can be accepted from a terminal, so the buffer should be at least that long. The line is stored in memory and terminated with a zero byte rather than with a carriage-return/line-feed combination.

prompt The address of a string to be printed on the job's terminal before input is accepted. The string itself must be terminated with a null (0) byte or an octal 200 byte. If the string is terminated with a null byte, the display will conclude with a carriage-return/line feed. If terminated with a 200 byte, no carriage-return/line feed is displayed.

Errors

No errors are possible with the .GTLIN request.

Example

The following example prompts the job's terminal and accepts a line of input:

```
PROMPT:  .ASCIZ      /YES OR NO?/
LINBUF:  .BLKB       81.
            .
            .
            .
         .GTLIN      #LINBUF,#PROMPT
```

7.22 .GTJB — Return Job High Limit

The .GTJB directive on RT–11 systems returns job parameters to an eight-word area in memory. On RSTS/E systems, the only relevant parameter is the job's high memory limit. All other words are set to zero.

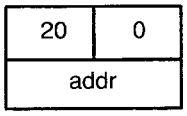
Macro Call

.GTJB *area,addr*

area The address of a two-word argument block.

addr The address of an eight-word block to which the job parameters are returned. On RSTS/E systems, only the second word is relevant: it is set to the high memory limit of the user job image. All other words are set to zero.

Argument Block Format



Errors

No errors are possible with the .GTJB directive.

Example

The following code stores job information in an area called JOBINF:

```
AREA:       .BLKW     2
JOBINF:     .BLKW     8.
                .
                .
                .
            .GTJB     #AREA,#JOBINF
```

7.23 .GVAL — Get Value from Scratch Pad

The .GVAL directive returns the contents of a word in the scratch pad area to R0. You specify an offset from the beginning of the scratch pad.

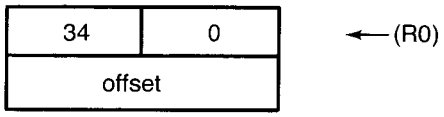
Macro Call

.GVAL *area,offset*

area The address of a two-word argument block.

offset The offset, in bytes, from the beginning of the scratch pad area of the word desired. The offset value must be even. An odd offset will cause a trap to kernel mode vector at 4.

Argument Block Format



Errors

Code	Meaning
0	The offset given is beyond the limits of the scratch pad area.

Example

The following example returns the contents of the PROTEC offset in the scratch pad area to R0:

```
AREA:       .BLKW       2
              .
              .
              .
            .GVAL       #AREA,#2
```

7.24 .HRESET — Hardware Reset

The .HRESET (hardware reset) directive closes all open channels for the program without performing any of the normal cleanup operations (except to disassociate the RT11 channel numbers from their RSTS/E counterparts). For example, no trailer labels are written to magnetic tape, no form feed is given for the line printer, and so forth. Temporary files (created with .ENTER but not yet closed) are deleted.

Macro Call

```
.HRESET
```

Errors

No errors are possible with the .HRESET directive.

Example

```
.HRESET
```

7.25 .LOOKUP — Open File for Input

The .LOOKUP directive opens an already existing file and associates it with a channel number; in RSTS/E terms, an "open for input." The channel used is then busy until one of the following directives is executed:

.CLOSE
.SAVESTATUS
.SRESET
.HRESET
.PURGE
.CSIGEN (if the channel number is in the range $0–10_8$)

Note that you can set a project-programmer number applicable to a RSTS/E open for input by setting the appropriate words in the scratch pad area before executing the .LOOKUP. See Section 6.7.

Macro Call

.LOOKUP *area,channel,devblk,seqnum*

area The address of a three-word argument block.

channel A channel number in the range $0–17_8$. See Section 6.5.1 for rules on channel numbers.

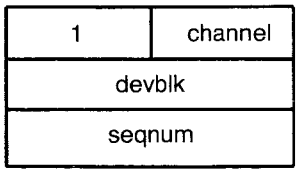
devblk The address of a four-word device block containing the device and file specifications. See Section 6.5.2 for general rules on the format of device blocks.

The file name portion of the device block is ignored for non-file-structured devices, such as a paper tape. If you want to do non-file-structured input/output on a file-structured device, clear the first word of the file name portion of the device block. The device will be positioned at absolute block 0 of the device.

seqnum This argument is ignored on RSTS/E systems.

.LOOKUP

Argument Block Format



(R0)

Errors

Code	Meaning
0	Channel already open.
1	The file requested (in the device block) was not found.

Example

```
ERRBYT=52
ARGBLK: .BLKW       5                ;LEAVE SPACE FOR ARGUMENT BLOCK
DEVBLK: .RAD50      /DT3/            ;DEFINE DEVICE,
        .RAD50      /DATA  001/       ;FILENAME AND TYPE
            .
            .
            .
        .LOOKUP     #ARGBLK,#7,#DEVBLK   ;OPEN ON CHANNEL 7
        BCC         LDONE            ;FILE WAS FOUND
        TSTB        @#ERRBYT         ;ERROR--TEST FURTHER BELOW
        BNE         NFD              ;FILE NOT FOUND
        .PRINT      #CAMSG           ;PRINT 'CHANNEL ACTIVE'
        .EXIT
NFD:    .PRINT      #NFMSG           ;PRINT 'FILE NOT FOUND'
        .EXIT
CAMSG:  .ASCIZ      /CHANNEL ACTIVE/
NFMSG:  .ASCIZ      /FILE NOT FOUND/
```

7.26 .PRINT — Display String on Job's Terminal

The .PRINT directive displays a specified ASCII string on the job's terminal. The line can be terminated with or without a carriage-return/line-feed combination. Note that on RSTS/E systems, if the terminal is detached when a .PRINT is issued, the job will "hibernate." That is, execution of the job will be suspended until the terminal is reattached to the job.

Macro Call

.PRINT *strgadd*

strgadd The starting address of the string to be printed. The string itself must be terminated with either a null (0) byte or a byte value of 200_8. If the string is terminated with a null byte, the display ends with a carriage-return/line-feed. If the string is terminated with a 200_8 byte, no carriage-return/line-feed is given.

Errors

No errors are possible with the .PRINT directive.

Example

```
S1:     .ASCIZ    /THIS STRING WILL HAVE CR-LF FOLLOWING/
S2:     .ASCII    /THIS STRING WILL HAVE NO CR-LF FOLLOWING/
        .BYTE     200
        .EVEN
          .
          .
          .
        .PRINT    #S1
        .PRINT    #S2
```

7.27 .PURGE — Release Channel

The .PURGE directive can be used to release a channel when it is not desirable to close the file normally. Any temporary file created by a .ENTER is discarded. A .PURGE on a file that was opened with .LOOKUP will cause no action other than to free the channel; any data written to the file will still be there. If the specified channel is not open, the .PURGE acts simply as a no-op; no error is returned.

Macro Call

.PURGE *channel*

channel A channel number in the range 0–17_8. See Section 6.5.1 for rules on channel numbers.

R0 Format

(R0) =

3	channel

Errors

No errors are possible with .PURGE.

Example

```
.TITLE    .PURGE.MAC
;THIS EXAMPLE VERIFIES THAT CHANNELS 0 - 7 ARE FREE.
          .MCALL     .PURGE,.EXIT
START:
          CLR        R1        ;START WITH CHANNEL 0
1$        .PURGE     R1        ;PURGE A CHANNEL
          INC        R1        ;BUMP TO NEXT CHANNEL
          CMP        R1,#8.    ;IS IT AT CHANNEL 8 YET/
          BLO        1$

          .EXIT
          .END       START
```

7.28 .RCTRLO — Reverse CTRL/O

The .RCTRLO directive restarts any programmed output to the user's terminal after such output has been stopped by the user's typing a CTRL/O or CTRL/C.

Terminal service on RSTS/E maintains a "discard all program output" indicator for each terminal on the system. When this indicator is set, the driver will ignore any output requests for that terminal. When the indicator is clear, output to the terminal proceeds normally. The .RCTRLO directive clears this indicator.

The driver sets the indicator when the user types a CTRL/C combination and reverses it when the user types a CTRL/O.

Thus, when it is vital that a message get through to the terminal, use a .RCTRLO directive before the .PRINT or other request for output to make sure that the message is displayed regardless of any user-typed CTRL/O or CTRL/C.

Macro Call

.RCTRLO

Errors

No errors are possible with the .RCTRLO directive.

Example

```
EOW:        .ASCIZ     /THE END OF THE WORLD IS NIGH/
               .
               .
               .
            .RCTRLO
            .PRINT     #EOW
```

7.29 .READ/.READW/.READC — Read Data

On RSTS/E systems, .READ, .READW, and .READC transfer data from a file or device previously opened on a channel to a user buffer. Once the transfer is complete, control returns to the user program in-line, for .READ and .READW, or to a user-specified completion routine, for .READC. You specify the maximum number of words to be transferred; the actual number of words transferred is returned in R0.

Note that you can set a "modifier word" in the scratch pad area (Section 6.7) for devices that use this information. (See .READ, Section 3.17, for possible values of the modifier word.)

Macro Call

$$\left\{\begin{matrix}\text{.READ}\\ \text{.READW}\end{matrix}\right\} \textit{area,channel,bufadd,wrdcnt,blknum}$$

or

.READC *area,channel,bufadd,wrdcnt,cmprtn,blknum*

area Address of a five-word argument block.

channel A channel number in the range $0–17_8$, previously defined in an open (.LOOKUP).

bufadd Address of a buffer to contain the data to be read.

wrdcnt The maximum number of words to be read. The actual number of words read may be less than *wrdcnt*; it will never be more. The actual number of words transferred is returned in R0.

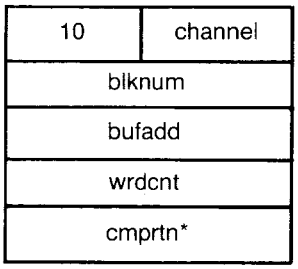
cmprtn (For .READC only.) The address of a completion routine to which control is to be transferred when the read is complete.

blknum For files (opened with file-structured .LOOKUP), the block number relative to the start of the file where the read is to begin. (To access large files on RSTS/E systems, you must set the word at offset 2 in the scratch pad area, as described in Section 6.7.)

For devices (opened with non-file-structured .LOOKUP), the absolute block number or device cluster number where the read is to begin.

For successive reads, your program must increment *blknum*.

Argument Block Format



* (The last word = 0 for .READW, or 1 for .READ.)

Errors

Code	Meaning
0	Attempt to read past end-of-file.
1	Hard error occurred on channel.
2	Channel is not open.

Example

The following code reads data from the device open on channel 7 to a 512–byte user buffer:

```
AREA:       .BLKW       5
BUF:        .BLKW       512.
                .
                .
                .
            .READ       #AREA,#7,#BUF,#512.,#0
```

7.30 .RENAME — Rename a File

The .RENAME directive changes the name of a file on disk or DECtape and gives that file the current date in its directory entry. If a file with the new name already exists on the specified device, it is deleted.

Macro Call

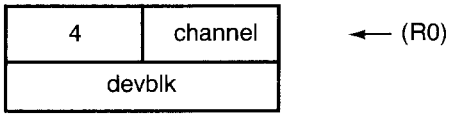
.RENAME *area,channel,namblk*

area Address of a two-word argument block.

channel Ignored on RSTS/E systems.

namblk The address of an eight-word area containing two four-word device and file descriptor blocks. (See Section 6.5.2 for the format of each of these four-word blocks.) The first four-word block is the old file name and the device where it is stored. The second four-word block is the new file name; the device descriptor must be specified, and must be the same as the device descriptor in the first four-word block. If a project-programmer number is specified in the first six words of the scratch pad, it will be used for both the old and the new file. If a protection code is given in the first six words of the scratch pad, it will be used for the new file. Other values in the first six words of the scratch pad are ignored.

Argument Block Format



Errors

Code	Meaning
1	No file with the "old" name was found on the device.
2	The device specified is not a disk or DECtape.

Example

```
AREA:     .BLKW     2
NAMBLK:   .RAD50    /DT3/
          .RAD50    /OLDFIL/
          .RAD50    /MAC/
          .RAD50    /DT3/
          .RAD50    /NEWFIL/
          .RAD50    /MAC/
                .
                .
                .
          .RENAME   #AREA,#0,#NAMBLK
```

7.31 .REOPEN — Reopen File Closed with .SAVESTATUS

The .REOPEN directive reopens a file that was closed with a .SAVESTATUS directive (Section 7.32). You can use a different channel number than was used in the original .LOOKUP and .SAVESTATUS, as long as the new channel is not already open.

The .SAVESTATUS and .REOPEN directives are easier to use than .CLOSE and .LOOKUP, if your program needs to open more than 15_{10} files during its execution. The .SAVESTATUS directive retains the information necessary to reopen a file, including the project-programmer number. Thus, you do not need to reset the PPN offset in the scratch pad area.

Macro Call

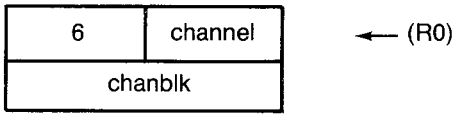
.REOPEN *area,channel,chanblk*

area Address of a two-word argument block.

channel A free channel number in the range $0–17_8$. See Section 6.5.1 for rules on channel numbers.

chanblk The address of the five-word block where the channel status information was stored (with .SAVESTATUS, Section 7.32).

Argument Block Format



Errors

Code	Meaning
0	The specified channel is in use. The .REOPEN has not been done.

Example

The following code reopens a file closed with .SAVESTATUS using the device block area named SAVBLK:

```
AREA:     .BLKW     2
SAVBLK:   .BLKW     5
             .
             .
             .
          .REOPEN   #AREA,#6,#SAVBLK
```

7.32 .SAVESTATUS — Save Status of File for Later .REOPEN

The .SAVESTATUS directive closes a disk or DECtape file, saving all the information needed to reopen the file later in execution of the program (see .REOPEN, Section 7.31). The information saved is stored in a five-word area in the user program and includes the project-programmer number for the file. Thus, the .SAVESTATUS/.REOPEN combination is easier to use on RSTS/E systems than .CLOSE/.LOOKUP in this case; you do not have to set the PPN offset in the scratch pad area.

The .SAVESTATUS directive can be used only if the file was opened with .LOOKUP. If the file was opened with .ENTER, a .SAVESTATUS on that channel will return an error.

Macro Call

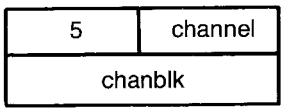
.SAVESTATUS *area,channel,chanblk*

area Address of a two-word argument block.

channel A channel number in the range 0–17_8, previously used in a .LOOKUP directive for disk or DECtape.

chanblk Address of a five-word area to store the current information for the file. On RSTS/E systems, the first four words of this area are set to the device, file name, and type, in the standard RT11 device block format (Section 6.5.2). The last word of this area is set to the project-programmer number (PPN) of the file.

Argument Block Format



Errors

Code	Meaning
0	The channel specified is not currently open; that is, a previous .LOOKUP on the channel was never done.
1	The file was opened with .ENTER, or else the file is not on a disk or DECtape device. The .SAVESTATUS is illegal.

Example

The following code saves the status of the file open on channel 12_{10} for a later reopen:

```
AREA:     .BLKW     2
SAVBLK:   .BLKW     5
             .
             .
             .
          .SAVEST   #AREA,#12.,,#SAVBLK
```

7.33 .SCCA — Pass CTRL/Z to User Program

The .SCCA directive allows a user program to recognize and process a CTRL/Z combination typed on the job's terminal. When called with any non-zero value for the *addr* argument, subsequent CTRL/Zs typed at the job's terminal are translated to 003_8 and passed like any other character to the program when it issues a .TTYIN or .GTLIN. (Normally, a CTRL/Z causes the RT11 emulator to exit the running program when a .READ or .TTYIN is done.)

Note that the operation of .SCCA is quite different under RSTS/E than under the RT–11 operating system, where it allows a user program to trap CTRL/C combinations. (The .SETCC directive must be used for this function under the RT11 run-time system under RSTS/E.) The intent is to allow you to code an .SCCA to trap the delimiter equivalent in function on the two systems. On RSTS/E systems, CTRL/Z is the "type-ahead terminator." You can, for example, type RUN $MACRO, type the command line for the assembly, type a CTRL/Z, and then type RUN $LINK. You do not have to wait for the assembly to finish before typing the command to link. On RT–11 systems, this capability is handled with a CTRL/C. Hence, the .SCCA directive processes the character appropriate to the system on both RSTS/E and RT–11 systems.

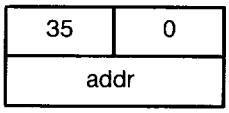
Macro Call

.SCCA *area,addr*

area The address of a two-word argument block.

addr Any non-zero value for this argument causes CTRL/Zs to be passed to the user program, as described above. A zero value returns CTRL/Z processing to the emulator.

Argument Block Format



Errors

No errors are possible with the .SCCA directive.

Example

The following code causes CTRL/Zs to be passed to the program:

```
AREA:       .BLKW       2
                .
                .
                .
            .SCCA       #AREA,#1
```

7.34 .SETCC — Process CTRL/C

The .SETCC directive defines a routine to be entered when the user types one CTRL/C on the job's terminal. (Two fast CTRL/Cs will still abort the running program.) The address of the routine to be entered is assumed to be in R0 when the .SETCC is executed.

The .SETCC is a "one-shot" directive; you must reissue the .SETCC from the trap routine if you wish it to remain in effect. You must also reenable CTRL/O processing with .RCTRLO (Section 7.28); exit from the routine with an RTI instruction.

See Section 6.7, the discussion of offset 66_8 in the scratch pad area, for further information on CTRL/C processing under the RT11 run-time system.

Macro Call

.SETCC

Note that .SETCC, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Errors

No errors are possible with the .SETCC directive.

Example

```
 .SETCC = EMT+362
MESAGE:  .ASCIZ       /YOU TYPED A CTRL/C!/
            .
            .
            .
         MOV          #CTLC,R0
         .SETCC
            .
            .
            .
CTLC:    .RCTRLO
         .PRINT       #MESAGE
         MOV          #CTLC,R0
         .SETCC
         RTI
```

7.35 .SETFQB — Set Up FIRQB

The .SETFQB directive can be used after a .CSISPC (Section 7.6); the two calls combined return information to the FIRQB and XRB as if the file specification string had been examined by .FSS.

The call assumes that R0 contains the address of the first word of a device block returned by .CSISPC. As noted in the discussion of .CSISPC, the device-block information must not have been moved; thus, the address in R0 *must* be the address of the first word of a device block within the "outspec" area returned by the last .CSISPC.

Note that any information in the first six words of the scratch pad area will override any RSTS/E–specific information returned by the .CSISPC. In addition, the .SETFQB clears the first six words of the scratch pad after using the information.

Macro Call

.SETFQB

Note that .SETFQB, like all RSTS/E–specific directives under RT11, is not expanded at assembly time. (It is not defined in the SYSMAC.SML file as a directive.) You can code a definition of the directive in a data section of your program or use .MACRO to define it. See the example.

Errors

No errors are possible with the .SETFQB directive.

Example

The following code returns information to the FIRQB and XRB about the file following the equal sign in an RT command:

```
.SETFQB = EMT+360
          .
          .
          .
        .CSISPC     #OUT,#EXT,#CMD
        MOV         #OUT+<3*5*2>,R0
        .SETFQB
```

7.36 .SETTOP — Expand to Start of Scratch Pad

On RSTS/E systems, the .SETTOP directive sets the contents of the word at location 50_8 in the low 1000_8 bytes of memory to the value specified in .SETTOP. The value itself cannot equal or exceed the starting address of the scratch pad area (see Section 6.7). If such a value is specified, .SETTOP will simply set the contents of location 50_8 to the address of the first word below the starting address of the scratch pad. The value actually set in word 50_8 is also returned to R0 on completion of the call.

Macro Call

.SETTOP *adr-val*

adr-val A value specifying the address to which location 50_8 is to be changed.

Errors

No errors are possible with the .SETTOP directive.

Example

The following code sets the contents of location 50_8 to the address of the word below the start of the scratch pad:

```
.SETTOP #-2
```

7.37 .SFPA — Set Floating-Point Error Address

The .SFPA directive indicates that you wish to handle floating-point errors yourself. You specify the address of a routine to be entered if such errors occur. The .SFPA works for errors on both the FIS (floating-point instruction set on the PDP–11/40) and the FPP or FPU hardware floating-point units (the PDP–11/45 and 11/70 asynchronous unit and its equivalents). In the case of FPP or FPU hardware, two words have been pushed on the stack: (1) the Floating-Point Exception Code (FEC) at the top of the stack and (2) the Floating Error Address (FEA) one word down. Your routine can process these in any way it sees fit and should pop these two words from the stack before issuing an RTI instruction to return control to the program where it left off when the errors occurred.

The .SFPA directive is a "one-shot" directive; it works only for one floating-point error trap. You must reexecute the .SFPA from your error-processing routine to ensure that successive errors are handled in the same way. You can disable traps to your error routine by issuing the .SFPA directive with a zero address argument.

NOTE

You should *not* use .SFPA in a routine called by a FORTRAN program, because programs compiled under FORTRAN have their own FPP/FIS exception handling routines and expect to find the FPU status as they have set it.

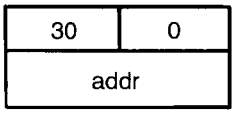
Macro Call

.SFPA *area,addr*

area The address of a two-word argument block.

addr The address of the routine to be entered when a floating-point error occurs.

Argument Block Format



Errors

No errors are possible with the .SFPA directive.

Example

```
AREA:       .BLKW       2
                .
                .
                .
            .SFPA       #AREA,#ERRTN
```

7.38 .SPFUN — Special Functions for I/O

The .SPFUN directive handles special functions for disk, terminal, magnetic tape, and flexible diskette.

Macro Call

.SPFUN *area,channel,func,buf,[wcnt],[blk][,crtn]*

area Address of a six-word argument block.

channel Channel number, $0–17_8$, defining the device on which the special function is to be performed. Must have been previously opened with .LOOKUP.

func Special function code. To request magnetic tape special functions emulating RT11's .SPFUN, set this argument to one of the following values:

373 Rewind to load point
377 Write end-of-file
376 Forward 1 record
375 Backspace 1 record
372 Offline rewind

Any other value for this argument is used as the function code (at XRB + XRLEN) in a RSTS/E .SPEC directive (Section 3.23).

buf Must be set to zero if the func argument is 373, 377, 376, 375, or 372. Otherwise, the value of this argument is used as the word at XRB + XRLOC in a RSTS/E .SPEC directive.

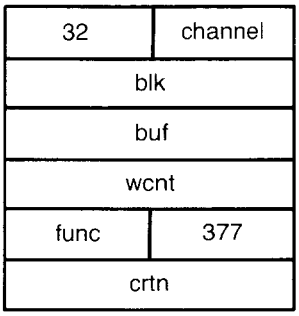
wcnt Ignored (and can be omitted) if the func argument is 373, 377, 376, 375, or 372. Otherwise, the value of this argument is used as the word at XRB + XRBC in a RSTS/E .SPEC directive.

blk Ignored (and can be omitted) if the func argument is 373, 377, 376, 375, or 372. Otherwise, the value of this word is used as the word at XRB + XRBLK in a RSTS/E .SPEC directive.

crtn Address of a completion routine to which control is to be transferred when the operation is complete. If this argument is omitted, control returns to the instruction following the .SPFUN when the operation is completed.

(Note that, if you are using .SPFUN to do a RSTS/E .SPEC, the RT11 emulator fills in the RSTS/E channel number and device handler values in the XRB for you.)

Argument Block Format



Errors

Code	Meaning
0	Tried to read past end-of-file.
1	Hard error occurred on channel.
2	Channel is not open.

Example

The following code rewinds a magtape (open on channel 4):

```
AREA:       .BLKW       6
              .
              .
              .
            .SPFUN      #AREA,#4,#373,#0
```

7.39 .SRESET — Software Reset

On RSTS/E systems, the .SRESET directive does the same thing as a hardware reset (.HRESET). It closes all open channels for the program without performing any of the normal cleanup operations (except to disassociate the RT11 channel numbers from their RSTS/E counterparts). For example, no trailer labels are written to magnetic tape, no form feed is given for the line printer, and so forth. Tentative files (created with .ENTER but not yet closed) are deleted.

Macro Call

```
.SRESET
```

Errors

No errors are possible with the .SRESET directive.

Example

```
.SRESET
```

7.40 .TRPSET — Intercept Traps to 4 and 10

The .TRPSET directive allows the user program to trap to a user routine on errors that normally cause the program to abort execution. Control will pass to the address specified in the call when the job tries to execute a reserved instruction (normally trapped to kernel mode address 10) or issues an instruction with an odd address (normally trapped to kernel mode address 4).

The carry bit indicates which error occurred. When control is passed to the error-handling routine, the carry bit is 0 if the error was an odd-address error. If the carry bit is 1, the error was a reserved-instruction error. An RTI instruction will return control to the user program from the trap routine.

The .TRPSET directive is a "one-shot" directive; that is, once executed, it affects only the next error causing such a trap. You must reissue the .TRPSET from your error-handling routine if you wish to handle such errors consistently.

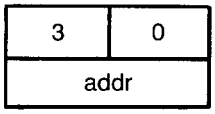
Macro Call

.TRPSET *area,addr*

area The address of a two-word argument block.

addr The address to which control is to be passed when a reserved-instruction or odd-address error occurs.

Argument Block Format



Errors

No errors are possible with the .TRPSET directive.

Example

```
AREA:       .BLKW       2
                .
                .
                .
            .TRPSET     #AREA,#TRPRTN
                .
                .
                .
TRPRTN:     BCC         ODDADR
            (code for reserved instruction)
                .
ODDADR:     (code for odd address)
```

7.41 .TTYIN/.TTINR — One-Character Read From Terminal

On RSTS/E systems, .TTYIN and .TTINR both work the same. They accept input from the job's terminal and transfer one character to the low byte of R0.

Bits 6 and 12 of the job status word (JSW) affect how the transfer is done.

If bit 12 of the JSW = 0, the .TTYIN/.TTINR waits until a whole line has been typed at the job's terminal and then transfers one character to the low byte of R0. Succeeding .TTYIN/.TTINR requests will then transfer remaining characters, including carriage return and line feed. In addition, the usual monitor processing for terminal input occurs: all characters typed are echoed at the terminal, and CTRL/U and DELETE delete a line or a character, respectively.

If bit 12 of the JSW = 1, "ODT–mode" input is done. That is, characters are transferred when they are available, rather than waiting for a whole line to be typed before passing a character to the user program. One .TTYIN/.TTINR transfers one character. Furthermore, characters typed are not echoed at the terminal, and CTRL/U and DELETE are simply passed on to the user program. (CTRL/C and CTRL/O are processed as usual, however. CTRL/S and CTRL/Q are processed, if that option was requested for your terminal with TTYSET.) No ALTMODE conversion is done.

If bit 6 = 1, control returns to the user program with the carry bit set if no character can be returned to the program. If bit 6 = 0, control does not return to the user program until a character can be passed to the program.

If used, bits 6 and 12 in the JSW must be set by the user program; they are cleared automatically when a program terminates.

NOTE

Single character I/O places heavy demand on RSTS/E system resources and can degrade system performance. These calls are provided only for compatibility with RT–11.

Macro Calls

.TTYIN [*addr*]

or

.TTINR

addr The address of a location where the character is to be stored, in addition to being stored in R0. If the address is omitted, the character is stored only in R0.

Errors

Code	Meaning
0	No character or characters are available in the monitor buffer for terminal I/O.

Example

```
.TTYIN
```

7.42 .TTYOUT/.TTOUTR — Transfer One Character to Job's Terminal

On RSTS/E systems, the .TTYOUT and .TTOUTR directives both work the same; they transfer one character from the low byte of R0 to the job's terminal. The intent of these requests on RT–11 systems is to allow programs that do not need to wait for an entire line to be printed on the terminal to do other processing in the meantime. On RSTS/E systems, I/O is synchronous, and using .TTYOUT or .TTOUTR is slower than using .PRINT. Use .PRINT whenever possible.

Note that on RSTS/E systems, if the terminal is detached when a .TTYOUT or .TTOUTR is executed, the job "hibernates." That is, execution of the job is suspended until the terminal is reattached to the job.

Macro Call

.TTYOUT [*addr*]

or

.TTOUTR

addr The address of a memory location containing the character to be printed. The character is loaded into R0 first. When control returns from the call, R0 still contains the character. If this address is omitted, the character in the low byte of R0 is printed at the job's terminal.

Errors

No errors are possible with the .TTOUTR directive under RSTS/E.

Example

```
PROMPT:    .ASCII   /*/
             .
             .
             .
             MOV #PROMPT,R0
             .TTYOUT
```

7.43 .TWAIT — Timed Wait

The .TWAIT directive suspends execution of the job for a specified number of clock ticks: a "sleep," in RSTS/E terms.

Macro Call

.TWAIT *area,addr*

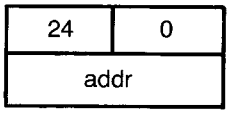
area The address of a two-word argument block.

addr The address of a two-word block containing the number of clock ticks to sleep. The first word contains the high-order bits; the second word contains the low-order bits.

NOTE

Since sleep works in terms of seconds on RSTS/E systems, the RT11 emulator simply divides the number of clock ticks specified by 60 and executes a .SLEEP. Thus, you need not consider whether you have a 60–Hz or 50–Hz clock; simply specify the wait time as (seconds * 60_{10}).

Argument Block Format



Errors

No errors are possible with .TWAIT.

Example

The following code suspends program execution for 5 seconds:

```
AREA:     .BLKW     2
ADDR:     .BLKW     2
              .
              .
              .
          CLW       #ADDR
          MOV       #300.,#ADDR+2
          .TWAIT    #AREA,#ADDR
```

7.44 .WAIT — Check for Channel Open

On RSTS/E systems, the .WAIT directive checks to make sure that the specified channel is open. If not, the call returns with an error.

Macro Call

.WAIT *channel*

channel The channel number to be tested, $0–17_8$.

R0 Format

R0 =

0	channel

Errors

Code	Meaning
0	The specified channel is not open.

Example

The following code checks to see if channel 7 is open and branches if it is not:

```
.WAIT     #7
BCS       #NOTOPN
```

7.45 .WRITE/.WRITW/.WRITC — Write Data

On RSTS/E systems, .WRITE, .WRITW, and .WRITC transfer data from a user buffer to a file or device previously opened on a channel. After the transfer is complete, control is passed to the user program in-line, for .WRITE and .WRITW, or to a user-specified completion routine, for .WRITC. The number of words actually written is returned in R0 when the call completes.

Macro Call

$$\left\{\begin{matrix}\text{.WRITE}\\ \text{.WRITW}\end{matrix}\right\} \quad \textit{area,channel,bufadd,wrdcnt,blknum}$$

or

.WRITC *area, channel,bufadd,wrdcnt,cmprtn,blknum*

area Address of a five-word argument block.

channel A channel number in the range 0–17_8, previously defined in an open (.ENTER or .LOOKUP).

bufadd Address of the buffer containing the data to be written.

wrdcnt The number of words to be written.

cmprtn (For .WRITC only.) The address of a completion routine to which control is transferred when the write is complete. (For .WRITE and .WRITW, control is returned to the instruction following the directive when the write is complete.)

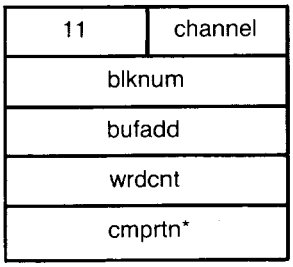
blknum For files (opened with file-structured .ENTER or .LOOKUP), the block number relative to the start of the file where the write is to begin. (To access large files on RSTS/E systems, you must set the word at offset 2 in the scratch pad area, as described in Section 6.7.)

For devices (opened with non-file-structured .ENTER or .LOOKUP), the *blknum* argument is used as the absolute block number or device cluster number where the write is to begin.

To do successive writes, your program must increment *blknum*.

.WRITE
.WRITW
.WRITC

Argument Block Format



* The last word is set to 1 for .WRITE or to 0 for .WRITW.

Errors

Code	Meaning
0	No more room is available on the device. For example, the end-of-tape marker has been encountered, no more available disk space, and so forth.
1	Some hard I/O device error occurred. For example, the specified device is off-line, is physically write-locked, and so forth.
2	The channel specified is not open.

Example

The following code writes a 256_{10}–word block to the device open on channel 12:

```
AREA:     .BLKW     5
BUF:      .BLKB     512.
            .
            .
            .
          .WRITE    #AREA,#12.,,#BUF,#256.,,#0
```

7.46 ..V1../..V2.. — Use Version 1/Version 2 Expansion

If you have a source program that was coded for Version 1 or Version 2 of the RT–11 operating system and you want to assemble on RSTS/E using the expansions that were used for these versions, you must use either the ..V1.. directive (for Version 1 expansions) or the ..V2.. directive (for Version 2 expansions).

You cannot use both the ..V1.. and ..V2.. directives. You can use directives from more than one version by specifying the ..V1.. directive and simply using the directives from later versions. If the same directive exists in two or three versions, though, you get the expansion from the earliest version.

These directives generate code that defines and sets a variable called ...V1 to either 1 or 2, which the MACRO assembler uses to determine the expansions to use. So, if the ..V1.. or ..V2.. directive is used, it must appear before any other RT11 directives in your program.

Note that all the examples in this chapter show Version 2 expansions. Version 1 expansions require different arguments in the calls. See RT–11 system documentation if you need to use Version 1 expansions.

Macro Call

```
..V1..
```

or

```
..V2..
```

Appendixes

Appendix A
Full List of Errors

ERR.STB Mnemonic	Decimal Value	Octal Value	Full Error Text
BADDIR	1	1	?Bad directory for device
BADNAM	2	2	?Illegal file name
INUSE	3	3	?Account or device in use
NOROOM	4	4	?No room for user on device
NOSUCH	5	5	?Can't find file or account
NODEVC	6	6	?Not a valid device
NOTCLS	7	7	?I/O channel already open
NOTAVL	8	10	?Device not available
NOTOPN	9	11	?I/O channel not open
PRVIOL	10	12	?Protection violation
EOF	11	13	?End of file on device
ABORT	12	14	?Fatal system I/O failure
DATERR	13	15	?Data error on device
HNGDEV	14	16	?Device hung or write locked
HNGTTY	15	17	?Keyboard wait exhausted
FIEXST	16	20	?Name or account now exists
DTOOOF	17	21	?Too many open files on unit
BADFUO	18	22	?Illegal SYS ( ) usage
INTLCK	19	23	?Disk block is interlocked
WRGPAK	20	24	?Pack ids don't match
NOTMNT	21	25	?Disk pack is not mounted
PAKLCK	22	26	?Disk pack is locked out
BADCLU	23	27	?Illegal cluster size
PRIVAT	24	30	?Disk pack is private
INTPAK	25	31	?Disk pack needs 'REBUILDing'
BADPAK	26	32	?Fatal disk pack mount error
DETKEY	27	33	?I/O to detached keyboard
CTRLCE	28	34	?Programmable ^C trap
SATTBD	29	35	?Corrupted file structure
DEVNFS	30	36	?Device not file-structured
BADCNT	31	37	?Illegal byte count for I/O

NOBUFS	32	40	?No buffer space available
B.4	33	41	?Odd address trap
B.10	34	42	?Reserved instruction trap
B.250	35	43	?Memory management violation
B.STAK	36	44	?SP stack overflow
B.SWAP	37	45	?Disk error during swap
B.PRTY	38	46	?Memory parity failure
MAGSEL	39	47	?Magtape select error
MAGRLE	40	50	?Magtape record length error
NRRTS	41	51	?Non-res run-time system
VCSERR	42	52	?Virtual buffer too large
VCAERR	43	53	?Virtual array not on disk
SIZERR	44	54	?Matrix or array too big
VCOERR	45	55	?Virtual array not yet open
BSERR	46	56	?Illegal I/O channel
LINERR	47	57	?Line too long
FLTERR	48	60	%Floating point error
EXPERR	49	61	%Argument too large in exp
FMTERR	50	62	%Data format error
FIXERR	51	63	%Integer error
BDNERR	52	64	%Illegal number
LOGERR	53	65	%Illegal argument in log
SQRERR	54	66	%Imaginary square roots
SUBERR	55	67	?Subscript out of range
MINVER	56	70	?Can't invert matrix
ODD	57	71	?Out of data
ONBAD	58	72	?ON statement out of range
NEDERR	59	73	?Not enough data in record
IOLERR	60	74	?Integer overflow, for loop
DIVBY0	61	75	%Division by 0
NORTS	62	76	?No run-time system
FIELDE	63	77	?Field overflows buffer
NORACS	64	100	?Not a random-access device
NOTMTA	65	101	?Illegal MAGTAPE() usage
ERRERR	66	102	?Missing special feature
BADSWT	67	103	?Illegal switch usage
	68	104	Unused error message
	69	105	Unused error message
	70	106	Unused error message
STMERR	71	107	?Statement not found
EXITTM	72	110	?RETURN without GOSUB
EXITNR	73	111	?FNEND without function call
UNDFNI	74	112	?Undefined function called
COSERR	75	113	?Illegal symbol
TLOPNV	76	114	?Illegal verb
TLNZSP	77	115	?Illegal expression
TLNOIT	78	116	?Illegal mode mixing
TLIFFE	79	117	?Illegal IF statement
TLCONI	80	120	?Illegal conditional clause
TLNOTF	81	121	?Illegal function name
TLQDUM	82	122	?Illegal dummy variable
TLMFND	83	123	?Illegal FN redefinition
TLRNNM	84	124	?Illegal line number(s)
MODERR	85	125	?Modifier error
	86	126	Unused error message
OUTCAS	87	127	?Expression too complicated

FUNERR	88	130	?Arguments don't match
TLTMAF	89	131	?Too many arguments
TLINCD	90	132	%Inconsistent function usage
CPNSDF	91	133	?Illegal DEF nesting
CPUPFR	92	134	?FOR without NEXT
CPUFNX	93	135	?NEXT without FOR
CPUPDF	94	136	?DEF without FNEND
CPUPED	95	137	?FNEND without DEF
TLJNKY	96	140	?Literal string needed
TLNOFN	97	141	?Too few arguments
SASYNE	98	142	?Syntax error
SAFNOS	99	143	?String is needed
SASNOI	100	144	?Number is needed
TLURTP	101	145	?Data type error
TLXDIM	102	146	?1 or 2 dimensions only
FUCORE	103	147	?Program lost-sorry
RESERR	104	150	?RESUME and no error
DIMED2	105	151	?Redimensioned array
TLIDIM	106	152	?Inconsistent subscript use
NOGOTO	107	153	?ON statement needs GOTO
EOSERR	108	154	?End of statement not seen
TLCNTD	109	155	?What?
TLPRNM	110	156	?Bad line number pair
EDBMCE	111	157	?Not enough available memory
EDEXON	112	160	?Execute only file
NRNERR	113	161	?Please use the RUN command
EDCONE	114	162	?Can't continue
EDARSV	115	163	?File exists–RENAME/REPLACE
PRERRS	116	164	?PRINT–USING format error
BADSWT	117	165	?Illegal switch usage
PRNER1	118	166	?Bad number in PRINT–USING
NONOIM	119	167	?Illegal in immediate mode
PRNER2	120	170	?PRINT–USING buffer overflow
BADERR	121	171	?Illegal statement
DISERR	122	172	?Illegal field variable
STPERR	123	173	Stop
DIMERR	124	174	?Matrix dimension error
NOMATH	125	175	?Wrong math package
XCDCOR	126	176	?Maximum memory exceeded
SCAERR	127	177	%SCALE factor interlock

Appendix B
Device Information

This appendix summarizes MODE and RECORD values and other useful information for:

- Disks
- Flexible diskettes
- Magnetic tape
- Line printers
- Terminals
- Pseudo keyboards

For your convenience, values are given in both decimal and octal, and FIRQB and XRB offsets are listed. All decimal values have a decimal point; values without a decimal point are in octal.

This appendix is a "quick reference." See the *RSTS/E Programming Manual* if you need more detail on any topic, except where otherwise noted.

B.1 Disks

This section summarizes disk MODE values and lists the device cluster size and total size for each disk that RSTS/E supports.

B.1.1 MODE Values

Tables B–1 and B–2 summarize disk MODE values for file-structured and non-file-structured access.

Table B–1: MODE Values for File-Structured Disk Access (FIRQB+FQMODE)

Decimal	Octal	Function
0.	0	Normal read/write
1.	1	Update
2.	2	Append
4.	4	Guarded update (4 + 1)
8.	10	Special extend (RSTS/E updates file's size and retrieval pointers during extend operations)
16.	20	Create contiguous file
32.	40	Create tentative file
64.	100	Create contiguous file conditionally
128.	200	No supersede
256.	400	Random data caching
512.	1000	Create file and place at beginning of directory (with 2000)
1024.	2000	Create file and place at end of directory
2048.	4000	Sequential data caching (with 400)
4096.	10000	Read normally regardless (privileged)
8192.	20000	Open file read-only
16384.	40000	Write UFD (privileged)

Table B–2: MODE Values for Non-File-Structured Disk Access (FIRQB+FQMODE)

Decimal	Octal	Function
0.	0	Access device clusters.
128.	200	Access disk blocks.
512.	1000	Read beyond last writable portion of disk; suppress error logging. (RSTS/E uses this mode in its online DSKINT program; it is not recommended for general use.)

B.1.2 Disk Device Sizes

Table B–3 lists the device cluster size and device size (in 512–byte blocks) for each disk that RSTS/E supports. All values are in decimal.

Table B–3: Disk Device Sizes

Disk	Device Cluster Size	Device Size
RX50	1	800
RF11	1	1024 times number of platters
RS03	1	1024
RS04	1	2048
RK05	1	4800
RK05F	1	4800 per unit; 2 units for each drive
RL01	1	10220
RL02	1	20460
RD51	1	21600
RK06	1	27104
RK07	1	53768
RC25	1	50902 per unit; 2 units per spindle
RP02	2	40000
RP03	2	80000
RM02	4	131648
RM03	4	131648
RP04	4	171796
RP05	4	171796
RA80	4	237208
RM80	4	242575
RP06	8	340664
RA60	8	400175
RM05	8	500352
RA81	16	888012

B.2 Flexible Diskettes

Tables B–4 and B–5 summarize MODE and RECORD values for flexible diskettes.

Table B–4: Flexible Diskette MODE Values (FIRQB+FQMODE)

Decimal	Octal	Function
0.	0	Block mode
16384.	40000	Sector mode

Table B–5: Flexible Diskette RECORD Values (XRB+XRBLK)

Decimal	Octal	Function
8192.	20000	Access logical record 0
16384.	40000	Write Deleted Data Mark
32767.+1.	100000	Perform this I/O operation in block mode

B.3 Magnetic Tape

This section summarizes MODE and CLUSTERSIZE values for magnetic tape.

B.3.1 File-Structured Processing

Tables B–6 and B–7 summarize MODE and CLUSTERSIZE values for file-structured magnetic tape.

Table B–6: MODE Values for File-Structured Magnetic Tape (FIRQB + FQMODE)

Decimal	Octal	Function
0.	0	Read file label at current tape position
2.	2	Do not rewind tape when searching for file
16.	20	Write over existing file
32.	40	Rewind tape before searching for file
64.	100	Rewind on CLOSE
128.	200	Open for append
512.	1000	Write new file label without searching
16384.	40000	Search for DOS–formatted file label
24576.	60000	Search for ANSI–formatted file label

Table B–7: CLUSTERSIZE Values for ANSI Magnetic Tape Files

Label Field Name	CLUSTERSIZE (FIRQB+FQCLUS)		Label Result
	Decimal	Octal	
Record format	0.	0	U = undefined.*
	16384.	40000	F = fixed-length.
	32767.+1.	100000	D = variable length.
	–16384.	140000	S = spanned.**
Record length (in bytes)	Betw 0. and 4095.	Betw 0 and 7777	For U, always 0. For F, fixed record length. For D, maximum record length. For S, not used.**
System Dependent (File characteristics)	0.	0	M = carriage control embedded.
	4096.	10000	A = FORTRAN carriage control.
	8192.	20000	(space) = implied carriage control. When printed, line feed precedes and carriage return follows each record.

* RSTS/E undefined record formats cannot be processed directly by other operating systems.

** RSTS/E does not support ANSI format S records.

B.3.2 Non-File-Structured Processing

MODE values in non-file-structured magnetic tape processing have the form:

MODE(FIRQB+FQMODE) = D + P + S

where:

D (density) = 12.(14) for 800 BPI
256.(400) for 1600 BPI, phase-encoded

P (parity) = 0.(0) for odd
1.(1) for even

S (stay) = 0.(0) to clear MODE value after CLOSE
8192.(20000) to retain MODE value after CLOSE

Note that DIGITAL recommends the use of odd parity. When you use even parity, you cannot write binary data. In addition, many operating systems and tape drives do not support even parity.

For information on magnetic tape special functions, the magnetic tape status word, and the file characteristics word, see the description of .SPEC for magnetic tape (Section 3.23.3).

B.4 Line Printers

This section summarizes line printer MODE and RECORD values.

Table B–8: Line Printer MODE Values (FIRQB+FQMODE)

Decimal	Octal	Function
1. – 127.	1 – 177	Sets form length to number of lines per page for software formatting (512., 1000) and automatic page skip (2048., 4000). (This is the QUE system program's LPFORM option.)
128.	200	Changes the character 0 (zero) to O (letter O).
256.	400	Truncates lines that are longer than unit was configured for instead of printing the rest of the line on the next physical line on the page.
512.	1000	Enables software formatting. Forms control characters are $\geq 200_8$.
1024.	2000	Translates lowercase characters to uppercase characters. Applies only to uppercase and lowercase line printers.
2048.	4000	Skips six lines (that is, skips over perforation line) at the bottom of each form.
4096.	10000	Moves paper to top of hardware form.
8192.	20000	Suppresses form feed on CLOSE.

Table B–9: Line Printer RECORD Values (XRB+XRMOD)

Decimal	Octal	Function
2.	2	Print over perforation (disables MODE 4000 for this output step).
4.	4	Do not return control to program until output is complete or an error occurs.
8.	10	Clear pending output buffers before buffering characters for the request.
8192.	20000	Return control to program if output stall is to occur. (See .WRITE directive for more information.)

B.5 Terminals

This section summarizes MODE and RECORD values for terminals. It also includes information about echo control mode and VT100 ANSI–compatible escape sequences.

B.5.1 Terminal MODE and RECORD Values

Tables B–10 through B–12 summarize terminal MODE values and RECORD values for terminal input and output.

Table B–10: Terminal MODE Values (FIRQB+FQMODE)

Decimal	Octal	Function
1.	1	Enable binary input from a terminal
2.	2	Reserved for TECO
4.	4	Suppress automatic CR/LF at right margin
8.	10	Enable echo control (turns off other modes and automatically enables MODE 4)
16.	20	Guard program against CTRL/C interruption and dial-up line hibernation
32.	40	Enable incoming XON/XOFF processing
64.	100	Reserved
128.	200	Enable use of RUBOUT as a delimiter on video terminals
256.	400	Set escape sequence mode

Table B–11: RECORD Values for Terminal Input (XRB + XRMOD)

Decimal and Octal Values	Function
8192. 20000	Perform conditional input (execute input request without waiting for data to be available).
32767. + 1. + K 100000 + K	Perform multiterminal service input from assigned keyboard number K.
32767. + 1. + 16384. + S 140000 + S	Perform multiterminal service input from any assigned keyboard: S = 0 — Wait until input is available from any terminal. The error ?Data error on device may occur due to a race condition with CTRL/C. 1. < S < 255. 1 < S < 377 — Wait up to S seconds for input from any terminal and then return ?Data error on device if no input is available. S = 8192. S = 20000 — Request input immediately; return ?Data error on device if no input is pending.

Table B–12: RECORD Values for Terminal Output (XRB + XRMOD)

Decimal	Octal	Function
256.	400	Declare echo control field (use with MODE 10).
4096.	10000	Output binary data to terminal.
8192.	20000	Return control to program if output stall is to occur. (See .WRITE directive for more information.)
32767. + 1. + K	100000 + K	Perform multiterminal service output to assigned keyboard K.

B.5.2 Echo Control Mode

In echo control mode, the system strips the parity bit from all characters. All characters returned to your program have ASCII values in the range 1 to 177 (octal). The system does not pass synchronization or editing characters to your program. Delimiters are passed to your program but are never echoed. Table B–13 summarizes the echo control mode character set.

Table B–13: Echo Control Mode Character Set

Type of Character	ASCII Code (Octal)	Code Returned to User	Comments
Ignored	0		Used as filler for timing.
Delimiters	Private	?	Private delimiter.
	3	3	^C (CTRL/C).
	4	4	^D (CTRL/D).
	12	12	Line feed.
	14	14	Form feed.
	15	15,12	Carriage return (with line feed appended).
	32	32	^Z (CTRL/Z); generates error 11_{10}.
	33	33	If "NO ESC SEQ" is in effect and escape character is received, 33 is returned to user and is treated as a delimiter.
			If "ESC SEQ" is in effect, escape character triggers an escape sequence. The escape sequence is returned to user and the whole sequence is considered the delimiter.
	175	33 or 175	If "NO ESC" is in effect, 175 is translated to escape (33).
			If "ESC" is in effect, 175 is data.
	176	33 or 176	If "NO ESC" is in effect, 176 is translated to escape (33).
			If "ESC" is in effect, 176 is data.

Table B–13: Echo Control Mode Character Set (Cont.)

Type of Character	ASCII Code (Octal)	Code Returned to User	Comments
Editing	177	–	RUBOUT (DEL character); on video terminals, generates a backspace followed by the paint character and another backspace; on hard-copy terminals, echoes deleted characters between backslashes (\\).
	25	–	^U (CTRL/U); repeatedly simulates RUBOUT until no characters remain in field.
Data	40–137	40–137	Normal 64–character graphic set.
	140–176	100–136	If "NO LC INPUT," lowercase letters are translated to uppercase.
	140–176	140–176	If "LC INPUT," lowercase letters are returned to user.
Synchronization	21	–	XON (CTRL/Q). Resume suspended output (if the STALL characteristic is set).
	23	–	XOFF (CTRL/S). Suspend output (if the STALL characteristic is set).
Other	1,2,5,6, 7,10,11, 13,16–20, 22,24, 26–31, 34–37,	–	Echoed as BEL (code 7) but otherwise ignored.
	21,23	–	If the terminal is set "NO STALL," synchronization characters are also echoed as BEL (code 7) and ignored.

Declaring a Field in Echo Control Mode

Use .WRITE to declare a field. The .WRITE must include the value 256.(400) at XRB+XRMOD and the value N at XRB+XRBC. N, which must be between 1 and the size of the buffer, describes how many bytes in the buffer represent the field declaration:

N = 1 The byte contains field size and overflow handling information. The field size must be in the range 1. to 127.(1 – 177). Adding 128.(200) to the field size specifies keypunch overflow handling instead of normal overflow handling.

N = 2 The first byte contains field size and overflow handling as described for N = 1. The second byte contains the ASCII value of the paint character. If this byte is 0 or N = 1, a space is the paint character.

N > 2 The first (N–2) bytes contain a prompt to display on the terminal before the field. Byte (N–1) is the field size declaration as described for N = 1. The last byte is the paint character as described for N = 2.

B.5.3 Escape Sequences

Table B–14 summarizes the VT100 ANSI–compatible escape sequences that move the cursor, erase all or part of the screen, and control line size and VT100 character attributes (bold, underscore, blink, and reverse video). The table uses the symbols Pl, Pc, and Pn, where:

Pl Means line number.

Pc Means column number.

Pn Is a decimal parameter expressed as a string of ASCII digits. The parameter's meaning for each escape sequence is explained in the table. Separate multiple parameters with a semicolon (;). If you omit a parameter or specify 0, the terminal uses the default parameter value for that escape sequence.

Be sure to include the left square bracket ([) in the escape sequence prefix where shown in the table. Escape sequences cannot contain embedded spaces. Refer to the *VT100 User Guide* for a complete description of VT100 escape sequences.

Table B–14: VT100 ANSI–compatible Escape Sequences for Screen Control

Escape Sequence	Description
Cursor Movement	
ESC[PnA	Moves the cursor up n lines without affecting the column position. The parameter Pn specifies the number of lines. The default value is one line.
ESC[PnB	Moves the cursor down n lines without affecting the column position. The parameter Pn specifies the number of lines. The default value is one line.
ESC[PnC	Moves the cursor forward (right) n columns without affecting the line position. The parameter Pn specifies the number of columns. The default value is one column.
ESC[PnD	Moves the cursor backward (left) n columns without affecting the line position. The parameter Pn specifies the number of columns. The default value is one column.

(continued on next page)

Table B–14: VT100 ANSI–compatible Escape Sequences for Screen Control (Cont.)

Escape Sequence	Description
Cursor Movement	
ESC[Pl;PcH	Direct cursor address – moves the cursor to the specified line and column position. If you do not specify a line or column position, the cursor moves to the home position, which is the top left corner of the screen.
ESCD	Index – moves the cursor to the current column position on the next line.
ESCM	Reverse index – moves the cursor to the current column position on the preceding line.
ESCE	Moves the cursor to the first column position on the next line.
Erasing	
ESC[K or ESC[0K	Erases from the current cursor position to the end of the line.
ESC[1K	Erases from the beginning of the current line to the cursor.
ESC[2K	Erases the entire line containing the cursor.
ESC[J or ESC[0J	Erases from the current cursor position to the end of the screen.
ESC[1J	Erases from the beginning of the screen to the current cursor position.
ESC[2J	Erases the entire screen.
Line Size (Double Height and Double Width)	
ESC#3	Changes the current line to the top half of a double-height double-width line.
ESC#4	Changes the current line to the bottom half of a double-height double-width line.
ESC#6	Changes the current line to a double-width single-height line.
To display double-height characters, use the ESC#3 and ESC#4 sequences as a pair on adjacent lines and send the same characters to both lines. The use of double-width characters reduces the number of characters per line by half.	
Character Attributes (require Advanced Video Option)	
ESC[Pn;Pn;Pn;...;m	Turns bold, underscore, blink, and reverse video attributes on and off. Pn can have the following values: 0 or none All attributes off 1 Bold on 4 Underscore on 5 Blink on 7 Reverse video on The terminal executes the parameters in order and ignores any other parameter values. Unlike line size commands, which affect only the current line, the character attributes affect the entire screen. Remember to turn them off before ending your program.

B.6 Pseudo Keyboards

This section summarizes MODE and RECORD values for pseudo keyboards. It also lists errors your program can receive on a pseudo keyboard output request.

Table B–15: Pseudo Keyboard MODE Values (FIRQB+FQMODE)

Decimal	Octal	Function
0.	0	System kills controlled job when the pseudo keyboard is closed.
1.	1	System detaches controlled job when the pseudo keyboard is closed.

Table B–16: RECORD Option Bit Values for Pseudo Keyboard Output (XRB+XRMOD)

Bit	Value		Result
	Decimal	Octal	
0	1.	1	If set, the system does not check job status before sending data to the pseudo keyboard.
1	2.	2	If set, the system tests whether pseudo keyboard is waiting for a system command (CTRL/C state) or is waiting for program input (KB wait state).
2	4.	4	If set, the system does not send data to the pseudo keyboard but instead returns control to the controlling job.
3	8.	10	If set, and there are no small buffers for keyboard input, the system waits until small buffers are available. However, your program receives the NOROOM error if the output buffer chain is full.
4	16.	20	If set, the system kills the job currently running at the pseudo keyboard.

Table B–17: Possible Errors on Pseudo Keyboard Output Request

Error	Meaning
INUSE	Job at pseudo keyboard is not ready for input
NOROOM	No buffer space is available
NOSUCH	No controlled job exists at pseudo keyboard
CTRLCE	Job at pseudo keyboard is not in CTRL/C state

Appendix C
Supplementary RSX Directives for Resident Libraries

The RSX emulator directives that deal with resident libraries (ATRG$, DTRG$, CRAW$, ELAW$, MAP$, and UMAP$) use 8–word areas to pass and receive data to and from the emulator. These areas, called the resident library definition block (RDB) and window definition block (WDB), can be defined or defined and filled using supplementary directives included in the RSXMAC.SML file.

Note that the expansions for these directives are the same as in the RSX–11M environment, where their use is more extensive. Only the arguments relevant to RSTS/E are described here.

C.1 RDB Directives

Two directives are available for use with resident library definition blocks (RDBs): RDBDF$ and RDBBK$.

The RDBDF$ directive simply assigns literal values to the offsets and status bit mnemonics shown in Chapter 5 for the RDB areas for the ATRG$ and DTRG$ directives. You can use these mnemonics to reference offsets and bit values in an RDB you have allocated space for in your program.

The RDBBK$ directive defines these offsets and, in addition, generates code to allocate space for the RDB and fills it with values you specify in the call.

The form for the relevant arguments in the ATRG$ call is:

RDBBK$,,*libnam*,,<RS.WRT> (for read/write access)

or

RDBBK$,,*libnam*,,<RS.RED> (for read-only access)

where *libnam* is the name of the resident library to be attached.

For example, the RDBBK$ call below expands to the instructions that follow.

```
RDBBK$   ,,DATLIB,,<RS.WRT>
```

Expansion

```
.WORD   0
.WORD   0
.RAD50  /DATLIB/
.WORD   0
.WORD   0
.WORD   RS.WRT
.WORD   0
```

(RS.WRT is assigned to a literal value of 2, so bit 1 is set in the seventh word of the RDB, requesting read/write access.)

In addition, all the offsets necessary to reference the data returned in the RDB by ATRG$ or DTRG$ are generated. For example, you can use the mnemonic RS.UNM to test bit 14 of the word at offset R.GSTS in the RDB after execution of a DTRG$ directive. RS.UNM is assigned to a literal value of 40000_8 by the RDBBK$ directive.

C.2 WDB Directives

Two directives are available for use with window definition blocks (WDBs): WDBDF$ and WDBBK$.

The WDBDF$ directive simply assigns literal values to the offsets and status bit mnemonics shown in Chapter 5 for the WDB areas for the CRAW$, ELAW$, MAP$, and UMAP$ directives. You can use these mnemonics to reference offsets and bit values in a WDB you have allocated space for in your program.

The WDBBK$ directive defines these offsets and, in addition, generates code to allocate space for the WDB and fills it with values you specify in the call.

The form for the relevant arguments in a CRAW$ call is:

WDBBK$ *apr,siz,rid,off,len,<bit1![bit2!bit3]>*

apr = The base apr.

siz = The size of the window.

rid = The resident library ID.

off = The offset into the library, in 32–word blocks.

len = The length to be mapped.

bit... = Mnemonic values for bit settings, separated by exclamation points. Relevant mnemonics for CRAW$ are:

WS.MAP = Window is to be mapped.
WS.WRT = Map with read/write access.
WS.RED = Map with read-only access.

For example, the WDBBK$ call below expands to the instructions that follow.

```
WDBBK$   7,128.,0,0,0,<WS.MAP!WS.RED>
```

Expansion

```
.BYTE   0,7
.WORD   0
.WORD   128.
.WORD   0
.WORD   0
.WORD   0
.WORD   WS.MAP!WS.RED
```

This WDB, when used in a CRAW$ directive, would create a window 4K words long (128_{10} 32–word blocks) in APR 7. The call also specifies an offset and map length of zero and read-only mapping. Note that to use this WDB, you have to supply the resident library ID, which you get from the RDB returned by an ATRG$ call. To fill in the library ID, move the word at offset R.GID in the RDB to offset W.NRID in the WDB.

The WDBBK$ directive also defines the offsets and bit settings referred to in the discussion of ELAW$, MAP$, and UMAP$.

Index

A

Abbreviation point (CCL), 3–81
ABRT$, 5–4
Account
  creating, 3–304
  deleting, 3–270
  reading data for, 3–312
  resetting data for, 3–312
Account number
  , 3–98
  !, 3–98
  $, 3–98
  %, 3–98
  @, 3–98, 3–218
  in file specification, 3–98
  wildcard, 3–98
Accounting information dump, 3–235
Active page register, 2–1
Addressing, 2–1, 2–2
ALUN$, 5–5
AME, 4–2
ANALYS program, 3–306
ANSI magnetic tape
  CLUSTERSIZE values for, B–6
Application Migration Executive, 4–2
APR, 2–1, 2–2, 2–4, 2–19, 2–20
  kernel mode, 2–3, 3–129
  used to map windows, 3–138
  user mode, 2–3
Argument block, 6–6
Assembler, 1–3
ASSFQ, 3–11 to 3–13
Assigning a device, 3–11
AST, 2–15, 5–46
ASTX$, 5–7
Asynchronous trap, 2–15, 5–7, 5–46
Asynchronous trap addresses, 2–23
ATRFQ, 3–133 to 3–136
ATRG$, 5–9
Attach to job, 3–240
Attach to resident library, 2–7
Attribute, 3–238, 3–281

B

B.10, 2–21
B.250, 2–21
B.4, 2–21
B.PRTY, 2–25
B.STAK, 2–25
B.SWAP, 2–25
Backspacing magnetic tape, 3–197
BASIC–PLUS run–time system, 1–2, 2–4, 2–20
BASIC–PLUS SYS calls, 3–229 to 3–356
Binary file
  creating, 3–17
  opening, 3–17
Binary mode, 3–186
.BLKB0, 3–6
.BLKW0, 3–6
BLOCK option, 3–161, 3–360
Block–random devices, 3–158

Block–sequential devices, 3–158
BPT instruction, 2–15
Broadcast to terminal, 3–183
.BSECT, 3–5
Buffer size
  for reads, 3–158, 3–159
  for writes, 3–358
Buffering, disabling full line, 3–209
Byte–oriented devices, 3–158

C

Caching
  disabling disk, 3–252
  enabling disk, 3–252
CALFIP, 3–10 to 3–80
  summary of subfunctions, 3–10
CALL, 3–6
CALLR, 3–6
CALLRX, 3–6
CALLX, 3–6
Card reader
  reads for, 3–158
Carry bit, 7–59
.CCL, 3–81 to 3–85
CCL
  adding CCL command, 3–250
  deleting CCL command, 3–250
  getting command line, 5–28
  RT11 .DOCCL, 7–24
  string passed in CORCMN, 2–12
CCL command, 3–81
  /DETACH switch, 3–83
  /SIZE switch, 3–82
.CCL directive, 2–13
  /DETACH switch, 2–28
  /SIZE switch, 2–29
CCLLST, 3–346
.CHAIN, 3–86, 3–285
  RT11, 7–7 to 7–8
Channel
  checking for open, 7–64
  closing, 3–14, 7–40
  closing under RT11, 7–9, 7–58
  releasing, 7–44
  reset, 3–77
Channel number, 3–8
  under RT11, 6–7
.CLEAR, 3–87 to 3–88
  general description, 2–9
.CLOSE, 7–9
CLOSE
  with "negative channel number" (reset), 3–77
Close flag, 3–263
Closing a channel, 3–14
.CLRFQB, 7–10
CLRFQB routine, 3–8
CLRFQX routine, 3–8
.CLRXRB, 7–11
CLRXRB routine, 3–8
CLSFQ, 3–14 to 3–16
CLUSTERSIZE option, 3–26, 3–34, 3–67
  for ANSI magnetic tape, B–6
/CLUSTERSIZE switch, 3–96, 7–12
CLUSTR offset, 6–11
Command, CCL, 3–81
COMMON option, TKB, 2–7
COMMON.MAC, 2–8, 3–3 to 3–6, 3–229, 6–3
  how to assemble with, 3–3
  macros provided with, 3–3
  mnemonics assigned to FIRQB, 2–11
  mnemonics for pseudo–vectors, 2–15
Concise command language. *See CCL*
Context, job, 2–10, 2–27
Control character,vertical format, 5–42t
Conversion, date and time, 3–258
CORCMN
  definition, 2–12
  format, 2–13f
  within low 1000 bytes, 2–8f
.CORE, 2–20, 3–89 to 3–91
  can cause swapping, 2–9
Core common
  definition, 2–12
CRAFQ, 3–137 to 3–142
CRAW$, 5–12
CRBFQ, 3–17 to 3–22
Create
  binary file, 3–17
  file, 3–23
  logged–in job to enter keyboard monitor, 3–286
  logged–in job to run program, 3–284
  logged–out job, 3–283
  temporary file, 3–32
CREFQ, 3–23 to 3–31
CRMSBS offset, 6–11
CRTFQ, 3–32 to 3–37
.CSIGEN, 7–12 to 7–15
.CSISPC, 7–16 to 7–18
CSRTBL, 3–346
CTRL/C, 2–15, 2–24
  asynchronous trap for, 5–46
  processing, 7–52
  restarting output stopped by, 3–213
  stopping action of, 6–12

CTRL/O
  canceling, 3–183
  restarting output stopped by, 3–213
  reversing, 7–45
CTRL/O effect, 2–24
CTRL/Z, 7–51

D

DALFQ, 3–38 to 3–39
Data space, 2–4
Dataset, hanging up, 3–282
.DATE, 3–92 to 3–93
  RT11, 7–19 to 7–20
Date
  changing system, 3–261
  conversion, 3–258
  current, 5–31
  return to R0, 7–19
  returning under RT11, 7–21
.DATTIM, 7–21, 7–22
DCL run–time system, 1–2, 2–26
DDB, 3–275
DDCTBL, 3–348
DDNFS, 3–28, 3–29, 3–69, 3–70, 3–111, 3–112
DDRLO, 3–28, 3–29, 3–69, 3–70, 3–111, 3–112
DDWLO, 3–28, 3–29, 3–69, 3–70, 3–111, 3–112
DEAFQ, 3–40 to 3–41
Deassign
  all devices, 3–38
  device, 3–40
Declare receiver, 3–117
DECnet/E, 3–116
DECtape
  deleting file from, 3–50
  get directory for, 3–42
  reads for, 3–158, 3–159
  rename file, 3–74
  writing data to, 3–358
Default keyboard monitor, 2–25, 3–95, 3–167, 3–168, 7–33
  See also Primary run–time system
  declaring, 3–319
Default runnable file type, 2–20
DEFORG, 3–4
.DELETE, 7–23
Deleting
  account, 3–270
  file, 3–50
Delimiters
  for line, 3–158
Delimiters (Cont.)
  multiple private, 3–185 to 3–194
  private, 3–185, 3–186
Density, setting magnetic tape, 3–197
Detach
  from resident library, 3–143
/DETACH switch, in CCL command, 3–83
Detached job quota, 3–248
Detaching job, 3–263
DEVCNT, 3–344
Device
  assigning, 3–11, 3–214, 3–236
  block–random, 3–159, 3–358
  block–sequential, 3–158, 3–358
  byte–oriented, 3–158, 3–358
  checking availability, 7–34
  deassigning, 3–40, 3–214, 3–222, 3–262
  deassigning all, 3–38, 3–226, 3–260
  disk sizes, B–3
  handler index, 3–113
  logical name, 3–97, 3–109
  open (non–file–structured), 3–65
  physical name, 3–97
  reading, 3–158
  reassigning, 3–214, 3–236
  record–oriented, 5–41
  return status, 7–28
  snagging assign, 3–217
  snagging reassign, 3–217
  writing data to, 3–357
  zeroing, 3–355
Device block, 6–7, 6–8
  creating, 7–16
Device data block, 3–275
Device handler index, 3–30, 3–71
Device–type flags, 3–29, 3–70, 3–111
DEVNAM, 3–346
DEVOKB, 3–346
DEVPTR, 3–344
DEVSYN, 3–346
DIC, 4–3
DIGITAL Command Language. *See DCL*
DIR$, 4–5
  general form, 4–6
  other features, 4–7
Directive
  expansions for RSX, 4–3
  expansions for RT11, 6–4
  ID Code (DIC), 4–3
  monitor, 1–4
  Parameter Block (DPB), 4–3
  RSX, 1–4
  RT11, 1–4
  Status Word, 4–4

Directives
  emulator, 1–4
Directory
  User File. *See UFD*
  get information, 3–42
  lookup, 3–43
  lookup by file name, 3–294
  lookup on index, 3–265
  magnetic tape lookup, 3–268
  privileged and nonprivileged access, 3–42
  wildcard lookup, 3–296
DIRFQ, 3–42 to 3–49
Disable terminal, 3–254
Disappearing RSX run–time system, 1–2, 2–6, 2–7, 3–137, 3–164, 4–2
Disk
  changing logical name of, 3–328
  changing quota, 3–254
  deleting file from, 3–50
  device cluster sizes, B–3
  device sizes, B–3
  directory lookup by file name, 3–294
  dirty bit, 3–300
  disabling caching, 3–252
  enabling caching, 3–252
  file lookup by name, 3–55
  get directory for, 3–42
  locking blocks, 3–180
  MODE values for, B–2
  mounting, 3–298
  reads for, 3–159
  rename file, 3–74
  swapping to, 3–88, 3–176
  wildcard directory lookup, 3–296
  writing data to, 3–358
Disk quota, 3–248
DLNFQ, 3–50 to 3–52
DMC11/DMR11
  number of receive buffers, 3–66, 3–67
  reads for, 3–158
  receive buffer size, 3–66, 3–67
  writing data to, 3–358
.DOCCL, 7–24
.DOFSS, 7–25
.DORUN, 7–26
DPB
  for $ form, 4–5
  for $C form, 4–8
  for $S form, 4–9
  general form, 4–3
$DPB$$, 4–8
.DSECT, 3–4
.DSTATUS, 7–28, 7–29
$DSW, 4–4
DSW return code, 4–4
DTRFQ, 3–143 to 3–145
DTRG$, 5–17
Dump, snap shot, 3–272, 3–306

E

Echo
  disabling, 3–183, 3–202
  enabling, 3–183, 3–202
  stopping, 3–212
Echo control mode
  character set, B–10t
  declaring a field, B–11
ELAFQ, 3–146 to 3–148
ELAW$, 5–19
EMT, 2–3f
  377, 4–4
  instruction, 2–17, 2–22, 6–4, 6–5
  special prefix, 1–4, 2–18, 2–22, 6–3
EMT logging, 3–117, 3–125
Emulator, 1–2
  directives, 1–4
  installing RSX in monitor, 3–164
  RSX directives, 5–1 to 5–58
  RT11 directives, 7–1 to 7–67
  trap, 2–17
End–of–file (EOF), magnetic tape, 3–197
.ENTER, 7–30
Entry points, 2–25
.EQUATE, 3–5
ERCTL, 3–346
.ERLOG, 3–94
ERR.STB, 3–7
ERRFQ, 3–53 to 3–54
Error log, 2–19
Error message
  printing text, 7–32
  returning text, 3–53, 3–273
  RSTS/E set, A–1 to A–3
Error mnemonics, 3–7, A–1 to A–3
Errors
  logging, 3–94
  on pseudo keyboard output request, B–14
.ERRPRT, 7–32
ESC SEQ mode, 3–186
Escape sequences, 3–185, 3–186
  VT100 ANSI–compatible, B–12t
.EXIT, 3–95
  RT11, 7–33
Exit with status, 5–22
EXIT$, 5–21
Expand
  memory allocation, 3–89

Expand (Cont.)
  memory size, 2–9
EXST$, 5–22
Extend memory allocation, 3–89
Extended–memory calls (RT11), 6–1
EXTK$, 3–90, 5–24

F

FCB, 3–275
FCBLST, 3–346
FEA, 2–23, 5–7, 5–48, 7–55
FEC, 2–23, 5–7, 5–48, 7–55
.FETCH, 7–34
File
  adding system, 3–334
  associating with run–time system, 3–301
  changing statistics, 3–246
  creating, 3–23
  creating binary, 3–17
  creating temporary, 3–32
  deleting, 3–50
  deleting from DECtape, 7–23
  deleting from disk, 7–23
  listing system, 3–334, 3–338
  lookup by name (disk), 3–55
  open existing, 3–65, 7–41
  open for input, 3–65, 7–41
  open for output, 7–30
  opening binary, 3–17
  opening file–structured, 3–23
  opening temporary, 3–32
  placement and modification, 3–279
  processor, 3–229
  read attributes, 3–238
  read under RT11, 7–46
  reading, 3–158
  removing system, 3–334, 3–336
  renaming, 3–74, 7–48
  reopen, 7–49
  returning attribute, 3–281
  saving status of, 7–50
  specification, 3–96, 7–25
  spooling, 3–330
  tentative, 3–77
  wildcard lookup, 3–55
  write attributes, 3–238
  writing data to, 3–357
File characteristics word, magnetic tape, 3–199
File control block, 3–275
File name
  in file specification, 3–99
File name string scan, 2–13, 3–9, 3–96
  under RT11, 7–25
File processor, 3–80
File request queue block, 2–11
File specification, 3–96
  order of elements in, 3–99
File structure, RSTS/E, 3–271
File type
  default runnable, 2–20
  in file specification, 3–99
File–structured
  open, 3–23
FILESIZE option, 3–25
/FILESIZE switch, 3–96, 7–12
FIP, 3–229
FIP calls, 3–80
FIRQB
  clearing under RT11, 7–10
  data returned to, 3–8
  definition, 2–11
  general format, 2–11f
  mnemonics assigned to, 2–11
  on P.RUN entry, 2–29
  presetting to 0, 3–7
  routine to clear, 3–8
  setting up under RT11, 7–53
  size of, 2–11
  translating string to, 3–96
  within low 1000 bytes, 2–8f
Fixed monitor locations, 3–130t
FJBIG
  bit in KEY, 2–9
Flag
  close, 3–263
  device–type, 3–29, 3–70, 3–111
Flag word 1 (.FSS), 3–103
Flag word 2 (.FSS), 3–101
Flexible diskette
  changing density, 3–200
  MODE values for, B–4
  obtaining density, 3–200
  reads for, 3–159
  RECORD option for, B–4
  special functions for, 3–200
  writing data to, 3–358
FLGFRC, 3–28, 3–29, 3–69, 3–70, 3–111, 3–112
FLGKB, 3–28, 3–29, 3–69, 3–70, 3–111, 3–112
FLGMOD, 3–28, 3–29, 3–69, 3–70, 3–111, 3–112
FLGPOS, 3–28, 3–29, 3–70, 3–111, 3–112
FLGRND, 3–28, 3–29, 3–69, 3–70, 3–111

Floating–point
errors, 7–55
processor, 2–15
processor exception address, 5–48
processor traps, 5–7
unit, 2–10, 2–23, 3–177, 7–55
Force to keyboard, 3–183
Foreground/background, 6–1
FREES, 3–346
.FSS, 2–13, 3–96 to 3–108
as I/O support, 3–9

G

General monitor directives, 3–1 to 3–363
not used under RT11, 3–2
summary, 3–1, 3–2t
Get monitor tables
Part I, 3–343
Part II, 3–345
Part III, 3–347
.GETCOR, 7–35
GLOBAL, 3–6
Global symbols, 3–6
GLUN$, 5–26
GMCR$, 5–28
GPRT$, 5–29
.GTIM, 7–36
GTIM$, 5–31
.GTJB, 7–38
.GTLIN, 7–37
GTSK$, 3–90, 5–33
.GVAL, 7–39

H

Handler index, device, 3–30, 3–71, 3–113, 7–28
Hanging up a dataset, 3–282
High segment, 2–4, 2–5f, 2–6
detailed discussion, 2–15 to 2–31
Horizontal position, 3–156
.HRESET, 7–40

I

I and D space, 2–4
I/O
general directives for, 3–9
non–file–structured, 5–1, 5–39
page, 2–3f
special functions for, 3–180
special functions under RT11, 7–4, 7–56
status buffer, 5–42
I/O (Cont.)
synchronous, 5–39
under RSX run–time system, 5–39
.IDENT, 3–3
INCLUDE, 3–4
Infinite–wait read, 2–26, 3–161
INIT, 2–4
Installation options, 2–6
Instruction space, 2–4

J

JBSTAT, 3–344
JBWAIT, 3–344
JFBIG, 3–87, 3–88, 3–176
JFFPP, 2–10, 3–87, 3–88, 3–176
JFLOCK, 3–87, 3–88
bit in KEY, 2–9
JFNOPR, 3–87, 3–88, 3–176
bit in KEY, 2–9
JFPRIV, 2–10, 3–87, 3–88, 3–176
JFSPRI, 2–10, 3–87, 3–88, 3–176
JFSYS, 2–10, 2–31, 3–87, 3–88, 3–176
JMPX, 3–6
Job
aborting, 5–4
attaching to, 3–240
context information, 2–10, 3–177
creating logged–in, 3–284, 3–286
creating logged–out, 3–283
definition, 2–5f
detaching, 3–263
general discussion, 1–5
getting high limit, 7–38
indication of log–in, 2–9
killing, 3–248, 3–254
new on system, 2–26
parameters, 5–29, 5–33
preallocate memory for, 2–19
private maximum size, 5–24
reattaching to, 3–242
returning statistics on, 3–204
returning status information on, 3–340
suspending with .SLEEP, 3–178
suspending with .TWAIT, 7–63
suspending with SPND$S, 5–50
suspending with UU.STL, 3–333
swap console with, 3–244
swapping to disk, 2–9, 3–88, 3–176
timing information on, 3–206
Job area, 1–2, 2–5f
Job context information, 2–10, 2–27, 3–167
Job image
changing size, 3–89, 7–35

Job image (Cont.)
maximum size, 2–20
minimum size, 2–20
Job keyboard monitor, 2–25, 3–95, 3–167, 3–168, 5–21, 7–33
JOB MAX, 3–292
Job size
with disappearing RSX run–time system, 2–6
Job space, 2–4
JOBCNT, 3–346
JOBTBL, 3–344
JSBTBL, 3–344
JSR PC
substitute for, 3–6

K

Kernel mode APRs, 2–3, 3–129
Kernel mode vector
10, 2–21
244, 2–24
250, 2–21
34, 2–23
4, 2–21
KEY, 2–25, 2–27, 2–31
clearing bits in, 3–87
definition, 2–9
setting bits in JFLOCK, 3–176
within low 1000 bytes, 2–8f
Keyboard
disabling echo, 3–202
enabling echo, 3–202
forcing output to, 3–183
getting line from, 7–37
ODT–mode input, 3–209
reads for, 3–158
setting private delimiters for, 3–185
special functions for, 3–183
writing data to, 3–358
Keyboard monitor, 1–2, 2–4, 2–15, 2–25, 3–213
declare default, 3–319
default, 2–25, 3–95, 5–21, 7–33
entry to, 2–25, 2–26
job, 2–25, 3–95, 3–167, 5–21, 7–33
prompts, 2–26
Keyboard monitor wait, 2–26, 3–161
Keyword, 2–9, 2–25, 2–27
clearing bits in, 3–87
setting bits in, 3–176
when refreshed, 2–9
Killing a job, 3–248, 3–254

L

Large spooler, 3–330
flag bits, 3–331
LB:, 3–215
LBR, 1–3
LIBR, 1–3
LIBR option, TKB, 2–7
Librarians, 1–3
Library
cluster, 1–3
macro, 1–3
object, 1–3
resident, 1–3, 2–6
save image, 1–4
universal, 1–3
Line
getting from terminal, 7–37
width, 3–156
Line delimiters, 3–158
Line printer
MODE values for, B–7
"no stall" option, 3–360
RECORD values for, B–7
writing data to, 3–358
LINK, 1–3, 1–4, 2–7, 3–7
Linker, 1–3, 2–7
Loader, 1–2
Loading
resident library, 3–323
run–time system, 3–317
Local data message
receive, 3–125
send, 3–120
Locking disk blocks, 3–180
Log, error, 2–19, 3–94
Log–in, indication of, 2–9
Logged–in job
create, 3–284, 3–286
entry to keyboard monitor, 2–26
Logged–out job
create, 3–283
entry to keyboard monitor, 2–26
Logical names, 3–97, 3–109
adding system–wide, 3–326
assigning, 3–215
deassigning, 3–222
deassigning all, 3–226
for PPNs, 2–14
removing system–wide, 3–327
user's private, 2–14
Logical unit
getting information on, 5–26
Logical unit number, assigning, 5–5

LOGIN, 1–5
Logins, 3–290
  disabling further, 3–302
  enabling, 3–353
  set number allowed, 3–292
LOGNAM, 3–346
Logout, 3–247
  shutup, 3–264
LOGOUT utility, 3–32
.LOGS, 3–109 to 3–115
  as I/O support, 3–9
LOGTBL offset, 6–12
LOKFQ, 3–55 to 3–64
  as I/O support, 3–9
  disk directory lookup, 3–56
  disk wildcard directory lookup, 3–60
.LOOKUP, 7–41
Lookup
  directory, 3–43, 3–265
  special magtape, 3–45
Low segment, 2–4, 2–5f
  first 1000 bytes, 2–7, 2–8f
  first 1000 bytes for RSX, 4–9, 4–10f
  first 1000 bytes for RT11, 6–9

M

MAC assembler, 1–3, 4–1
  using general directives with, 3–1
MACRO assembler, 1–3, 6–1
  using general directives with, 3–1
Macro library, 1–3
MAGLBL, 3–348
Magnetic tape
  backspacing, 3–197
  CLUSTERSIZE values for, B–6
  file characteristics word, 3–199
  get directory for, 3–42
  MODE values for, B–5, B–6
  reads for, 3–158
  setting density, 3–197, B–6
  setting parity, 3–197, B–6
  skipping record, 3–197
  special directory lookup, 3–45, 3–268
  special functions for, 3–195
  status word, 3–198
  writing data to, 3–358
  writing end–of–file, 3–197
MAKSIL, 1–4
MAP$, 5–36
MAPFQ, 3–149 to 3–153
Mapping, 2–2
  address windows, 5–36
  to resident library, 3–137, 3–149
Masks
  private delimiter, 3–190t
MAXCNT, 3–344
.MCALL, 4–3
MEMLST, 3–344
Memory
  changing allocation, 3–89
  changing size, 3–310
  expansion, 2–9
  increasing allocation, 5–24
  management, 2–1
  mapping, 2–2, 2–3f
  page, 2–1, 2–2
  poking, 3–307
  private maximum, 3–177
MEMSIZ, 3–346
.MESAG, 3–116 to 3–126
Message
  EMT logger, 3–125
  receive local data, 3–125
  send local data, 3–120
  send/receive, 3–116
MFDPTR, 3–348
Mnemonics, 3–7
Mode
  ODT (one–character) input, 3–209
  tape, 3–208
MODE offset, 6–11
MODE option, 3–25, 3–34, 3–67
  for disk, B–2
  for flexible diskette, B–4
  for line printer, B–7
  for magnetic tape, B–5, B–6
  for pseudo keyboard, B–14
  for terminal, B–8
/MODE switch, 3–96, 7–12
Monitor
  fixed locations for .PEEK, 3–130t
  general RSTS/E directives, 1–4, 3–1 to 3–363
  RSX emulation in, 4–2
  tables, 3–343, 3–345, 3–347
Multiple private delimiters, 3–185 to 3–194
Multiterminal service, 3–360

N

.NAME, 3–92, 3–127 to 3–128
Name program, 3–127
NOCTLC offset, 6–12
Non–file–structured I/O, 5–1
  in RSX run–time system, 4–2
NRRTS, 2–25

O

Object library, 1–3
ODT, 2–16, 2–20, 3–209, 5–51
ODT mode, 3–183, 3–209, 7–60
Offline, taking magnetic tape, 3–197
ONLCLN, 3–300
Open
  binary file, 3–17
  existing file, 3–65
  file, 3–23
  file or device, 3–65
  file–structured, 3–65
  non–file–structured, 3–65
OPEN FOR INPUT, 3–65, 3–67
OPEN FOR OUTPUT, 3–23, 3–25, 3–34
OPNFQ, 3–65 to 3–73
ORG, 3–4
  requirement for, 3–5

P

P.2CC, 2–16f
  definition, 2–24
P.BAD, 2–16f
  definition for asynchronous traps, 2–24
  definition for synchronous traps, 2–21
P.BPT, 2–16f
  definition, 2–22
P.CC, 2–16f
  definition, 2–24
P.CRAS, 2–16f
  definition, 2–25
P.DEXT, 2–16f, 3–314
  definition, 2–20
P.EMT, 2–16f
  definition, 2–22
P.FIS
  definition, 2–21
P.FLAG, 2–16f, 2–18, 2–22, 6–4
  combinations, 2–20
  definition, 2–17
P.FPP, 2–16f
  definition, 2–23
P.IOT, 2–16f
  definition, 2–22
P.MSIZ, 2–16f, 2–20, 3–89, 3–90, 3–314
P.NEW, 2–16f, 3–168, 3–169, 3–171, 5–21
  definition, 2–25
P.OFF, 2–16f
  definition, 2–17
P.RUN, 2–16f, 2–31
  .CCL passes control to, 3–81
  channel 15 open, 3–9
P.RUN (Cont.)
  definition, 2–28
  entry at, 2–9
  passing control to, 3–172
P.SIZE, 2–16f, 2–20, 3–89, 3–314
P.STRT, 2–16f
  definition, 2–25
P.TRAP, 2–16f
  definition, 2–23
Page, 2–1, 2–2
Page address register, 2–1, 2–2
Page descriptor register, 2–1
Paper tape punch, 3–358
Paper tape reader
  reads for, 3–158
PAR, 2–1, 2–2
Parameter word, 2–31, 3–284, 3–285
Parity, setting magnetic tape, 3–197
Password, changing, 3–254
PAT, 1–3
Patch utility, object module, 1–3
PC, 2–4
PDR, 2–1, 2–19
.PEEK, 3–129 to 3–131
  fixed monitor locations, 3–130
Permanent privilege, 2–10
PF.1US
  bit within P.FLAG, 2–17f
  definition, 2–19
PF.CSZ
  bit within P.FLAG, 2–17f
  definition, 2–19
PF.EMT, 2–22, 6–4
  bit within P.FLAG, 2–17f
  definition, 2–17, 2–18
PF.KBM
  bit within P.FLAG, 2–17f
  definition, 2–19
PF.NER, 2–19
  bit within P.FLAG, 2–17f
PF.REM, 2–19
  bit within P.FLAG, 2–17f
PF.RW, 2–19
  bit within P.FLAG, 2–17f
Physical addressing, 2–1, 2–2
Physical device name, 3–109
.PLAS, 1–3, 3–132 to 3–155
  subfunction summary, 3–132
Poking memory, 3–307
/POSITION switch, 3–96, 7–12
POSITN offset, 6–11
.POSTN, 3–156 to 3–157
PPN offset, 6–11
Preallocate memory, 2–19

Prefix EMT, 1–4, 2–18, 2–22
Primary run–time system, 1–1
.PRINT, 7–43
Priority, 2–10
  changing, 3–310
  run, 3–177
Private delimiters
  and binary mode, 3–186
  characteristics, 3–186
  definition, 3–185
  in keypad applications, 3–185
  masks, 3–190t
  system processing, 3–186
Private memory maximum, 2–9, 3–177
Privilege
  permanent, 2–10, 3–88
  temporary, 2–10, 2–31, 3–88, 3–177
Program
  maximum size under RSX, 5–24
  maximum size under RT11, 6–10
  running, 3–172
  suspending, 3–178, 5–50, 7–63
Program counter register, 2–4
Program name, 3–127
  returned by .DATE, 3–92
Program status word, 2–4
Project–programmer number
  assignable, 3–98
  entering user logical for, 3–215
  in file specification, 3–98
  logical name for, 2–14
  wildcard lookup, 3–308
Prompts, keyboard monitor, 2–26
PROTEC offset, 6–11
Protection code
  default, 2–13
  in file specification, 3–99
/PROTECTION switch, 3–97, 7–12
.PSECT
  requirement for, 3–5
Pseudo keyboard
  errors on output request, B–14
  MODE values for, B–14
  reads for, 3–158
  RECORD values for, B–14
  special functions for, 3–202
  state change, 3–178
  writing data to, 3–358
Pseudo vectors, 2–5
  detailed discussion, 2–15 to 2–31
  format with high segment, 2–16
PSW, 2–4
.PURGE, 7–44

Q

QIO$, 5–39
QIOW$, 5–39
QMAN message receiver, 3–330, 3–332
QUEMAN message receiver, 3–332
Quota
  changing, 3–254
  detached job, 3–248
  disk, 3–248

R

R0, 6–4
.RCTRLO, 7–45
RDB, C–1
RDBBK$, C–1
RDBDF$, C–1
.READ, 3–158 to 3–163
  RT11, 7–46
Read
  device, 3–158
  file, 3–158
  ODT–mode, 3–209, 7–60
Read–only run–time system, 2–19
Read/write run–time system, 2–19
.READC, 7–46
.READW, 7–46
Reattach to job, 3–242
Receive, 3–123
  local data message, 3–125
Receiver ID Block, 3–117, 3–118, 3–123, 3–126
  See also RIB
Record Management Services, 4–2
RECORD option, 3–161
  for flexible diskette, B–4
  for line printer, B–7
  for pseudo keyboard, B–14
  for terminal, B–9
  "no stall", 3–360
Reentrant code, 1–1
Remove receiver, 3–119
.RENAME, 7–48
Rename file, 3–74, 7–48
RENFQ, 3–74 to 3–76
.REOPEN, 7–49
REORDR, 3–31, 3–36, 3–72
RESCOM option, TKB, 2–7
Reset channel, 3–77
Resident library, 1–3, 2–6
  accessing, 3–132
  accessing in RSX, 5–3
  adding, 3–320

Resident library (Cont.)
attaching to, 3–133, 5–9
creating window to, 3–137, 5–12
definition block, C–1
detaching from, 3–143, 5–17
eliminating window to, 3–146
loading, 3–323
mapping windows to, 5–36
maximum number of, 3–133, 5–9
removing, 3–322
space taken by, 2–7
special RSX directives for, C–1
unloading, 3–324
unmapping window from, 5–55
RESLIB option, TKB, 2–7
RETURN macro, 3–6
Rewind tape, 3–197
RIB, 3–117, 3–118, 3–123, 3–126
RMS, 4–2
/RONLY switch, 7–12
RSTFQ, 3–77 to 3–79
...RSX, 4–2
.RSX, 3–164 to 3–166
RSX directives, 1–4
$C form, 4–8, 4–9
expansions, 4–3 to 4–5
for resident libraries, C–1
$ form, 4–6, 4–7
$S form, 4–9
summary of, 5–1
RSX emulation in the monitor, 4–2
RSX run–time system, 2–20
disappearing, 1–2, 2–6, 2–7, 3–137, 3–164
emulator in monitor, 3–164
environment, 4–1 to 4–10
RSXMAC.SML, 4–3, 4–4
RT11 call formats, 6–5
RT11 directives, 1–4
expansions, 6–4
not processed on RSTS/E, 7–1
summary of, 7–2
RT11 linker, 2–7
RT11 run–time system, 1–2, 2–20
environment, 6–1 to 6–12
illegal general monitor calls, 3–2
low 1000 bytes for, 6–9
scratch pad area, 6–10
use of special prefix EMT, 2–18
.RTS, 3–167 to 3–171, 3–289
RTSLST, 3–346
.RUN, 2–20, 2–31, 3–172 to 3–175, 3–285, 3–289
"hard" errors, 3–174
"soft" errors, 3–174, 3–175
Run priority, 2–10, 3–177
Run–burst, changing, 3–310
Run–time system, 2–5f
adding, 3–314
associating file with, 3–301
capability defined, 2–17
choosing, 1–2, 1–3
default definitions, 2–17
exit processing, 3–167
general discussion, 1–1 to 1–5
loading, 3–317
modifying, 2–10
name returned, 3–92
passing control to, 3–167
primary, 1–1
removing, 3–316
similarity to resident libraries, 2–7
space taken by, 2–4
top address, 1–4
unloading, 3–318
when removed, 2–19
writing or modifying, 1–4
Running a program, 3–172

S

SATCTL, 3–344
SATCTM, 3–344
SATEND, 3–348
Save image library, 1–4
SAVE/RESTORE, 3–306
.SAVESTATUS, 7–50
.SCCA, 7–51
SCCA$S, 5–46
Scratch pad, 6–10, 7–54
getting value from, 7–39
Send local data message, 3–120
.SET, 3–176
general description, 2–9
Set terminal characteristics, 3–349
.SETCC, 7–52
.SETFQB, 7–53
.SETTOP, 7–54
.SFPA, 7–55
SFPA$, 5–48
Sharable code, 2–7
Shut down system, 3–264
Shutup logout, 3–264
Significant event, 5–57
SILUS, 1–4
Single event, 5–58
Single–job monitor, 6–1
/SIZE switch, 3–96, 7–12
in CCL command, 3–82

.SLEEP, 3–178 to 3–179
  conditional, 3–178, 3–179
Small buffers, 2–4
Small spooler, 3–330
  flag bits, 3–331
Snagging assign, 3–217
SNAP command, UTILTY, 3–306
Snap shot dump, 3–272, 3–306
SNDLST, 3–346
SP, 2–10, 2–21, 2–23, 2–24, 2–27
.SPEC, 3–180 to 3–203
  for disk, 3–180
  for magnetic tape, 3–195
  for pseudo keyboard, 3–202
  for RX01/RX02 flexible diskette, 3–200
  for terminal, 3–183
Special prefix EMT, 1–4, 2–18, 2–22, 6–3
.SPFUN, 7–56
SPND$S, 5–50
Spooling, 3–330
Spooling package
  micro–RSTS, 3–330
  micro–RSTS, flag bits for, 3–331
  standard RSTS/E, 3–330
  standard RSTS/E, flag bits for, 3–331
.SRESET, 7–58
SST, 2–15, 5–51, 5–53
Stack, 2–10
Stack overflow, 2–10, 2–24
Stack pointer, 2–27
Stack pointer register, 2–10, 2–21, 2–23
STALL characteristic, 3–186
Stall system, 3–333
.STAT, 3–204 to 3–205
Statistics
  changing file, 3–246
  returning for job, 3–204
Status byte, 3–277, 3–278
STATUS variable, 2–28, 3–28, 3–36, 3–104
Status word, magnetic tape, 3–198
Status, exit with, 5–22
String
  display at terminal, 7–43
  scan for file name, 7–25
Suspend
  a job (with .SLEEP), 3–178
  a job (with .TWAIT), 7–63
  a job (with SPND$S), 5–50
  all jobs on system (with UU.STL), 3–333
SVDB$, 5–51
SVTK$, 5–53
Swap console function, 3–244
Swap file, 3–336, 3–338
SWAP MAX, 3–310
SWAP.MAX, 5–24
Swapping, 2–9, 3–88, 3–176
SWITCH program, 3–167
Synchronous system traps, 2–15, 2–21, 5–51, 5–53
$SYSMAC.SML, 6–3
SYSTAT, 2–20, 3–127, 3–300
System default run–time system. *See Default keyboard monitor*
System error log, 2–19
System macro library, 4–3
System–wide logical names
  adding, 3–326
  removing, 3–327

T

Tape mode, 3–183, 3–208
Tape. *See Magnetic tape or DECtape*
Task
  exit, 5–21
  exit with status, 5–22
Task Builder, 1–3, 1–4, 2–7, 4–1, 4–4, 4–9
Temporary file, 3–32
Temporary privilege, 2–10, 2–31, 3–177
Tentative file, 3–77
Terminal
  disabling, 3–254
  disabling echo, 3–202
  enabling echo, 3–202
  ESC SEQ mode, 3–186
  forcing output to, 3–183
  getting line from, 7–37
  MODE values for, B–8
  "no stall" option, 3–360
  ODT–mode input, 3–209
  reading low–speed tape on, 3–208
  reads for, 3–158
  RECORD values for, B–9
  setting characteristics, 3–349
  setting private delimiters for, 3–185
  special functions for, 3–183
  STALL characteristic, 3–186
  writing data to, 3–358
.TIME, 3–206 to 3–207
Time
  changing system, 3–261
  conversion, 3–258
  current, 5–31
  returning under RT11, 7–21, 7–36
  slice, 2–3
Timing, returning for job, 3–206
TITLE, 3–3
TKB, 2–7, 3–7, 4–1, 4–4, 4–9

TMPORG, 3–4
Transfer request block, 2–11
Transportable code
  to RSX–11M, 4–1
  to RT–11, 6–1
  to VAX/VMS under AME, 4–2
Trap, 2–15
  asynchronous, 2–23
  asynchronous CTRL/C, 5–46
  emulator, 2–17
  kernel mode address 10, 7–59
  kernel mode address 4, 7–59
  routines in RSX environment, 5–2
  synchronous, 2–21, 5–51, 5–53
TRAP instruction, 2–23
.TRPSET, 7–59
.TTAPE, 3–208
  undoing, 3–211
.TTDDT, 3–209 to 3–210
.TTECH, 3–211
.TTINR, 7–60
.TTNCH, 3–212
  undoing, 3–211
.TTOUTR, 7–62
.TTRST, 3–213
TTYHCT, 3–346
.TTYIN, 7–60
.TTYOUT, 7–62
.TWAIT, 7–63
Type–ahead
  canceling, 3–183

U

UCTTBL, 3–348
UFD, 3–26
  deleting, 3–355
  positioning on disk, 3–304
  preextending, 3–304
.ULOG, 2–13, 3–214 to 3–228
  subfunction summary, 3–214
UMAP$, 5–55
UMPFQ, 3–154 to 3–155
Universal library, 1–3
Unmapping address window, 3–154, 5–55
UNORG, 3–4
Unstall system, 3–333
UNTCLU, 3–344
UNTCNT, 3–344
UNTLVL, 3–348
UNTOWN, 3–344
User File Directory. *See UFD*
User job area, 1–2, 2–5f
User job image, 2–5f
  changing size, 3–89
  definition, 2–4
  expanding, 5–24
  maximum size, 2–20
  minimum size, 2–20
  preallocate memory for, 2–19
User logical, 3–100
  deassigning, 3–214, 3–222
  deassigning all, 3–226
  destroyed, 2–25
  entering, 3–214
User mode APRs, 2–3
User stack area, 2–10
USRLOG, 3–100, 3–216
  definition, 2–14
  format, 2–14f
  .FSS causes check of, 3–97
  within low 1000 bytes, 2–9f
USRPPN, 3–98, 3–100, 3–216
  definition, 2–13
  within low 1000 bytes, 2–9f
USRPRT, 3–100, 3–216
  definition, 2–13
  within low 1000 bytes, 2–9f
USRSP, 2–25
  definition, 2–10
  within low 1000 bytes, 2–8f
USTAT byte, 3–267
UTILTY, 2–7, 2–16, 2–20
  SNAP command, 3–306
UU.ACT, 3–235
UU.ASS, 3–215 to 3–221, 3–236
UU.ATR, 3–238
UU.ATT, 3–240
UU.BCK, 3–246
UU.BYE, 3–247
UU.CCL, 3–250
UU.CHE, 3–252
UU.CHU, 3–254
UU.CNV, 3–258
UU.DAL, 3–226 to 3–228, 3–260
UU.DAT, 3–261
UU.DEA, 3–222 to 3–225, 3–262
UU.DET, 3–263
UU.DIE, 3–264
UU.DIR, 3–265
UU.DLU, 3–270
UU.DMP, 3–272
UU.ERR, 2–21, 2–25, 3–273
UU.FCB, 3–275
UU.FIL, 3–279
UU.HNG, 3–282
UU.JOB, 3–283 to 3–289

UU.LIN, 3–289, 3–290
UU.LOG, 3–292
UU.LOK, 3–294
UU.MNT, 3–298
UU.NAM, 3–301
UU.NLG, 3–302
UU.PAS, 3–304
UU.POK, 3–307
UU.PPN, 3–308
UU.PRI, 3–310
UU.RAD, 3–312
UU.RTS, 3–314 to 3–325
UU.SLN, 3–326
UU.SPL, 3–330
UU.STL, 3–333
UU.SWP, 3–334
UU.SYS, 3–340
UU.TB1, 3–343
UU.TB2, 3–345
UU.TB3, 3–347
UU.TRM, 3–349
UU.YLG, 3–353
UU.ZER, 3–355
.UUO, 3–229 to 3–356
  subfunction summary, 3–230t
UUOFQ, 3–80

V

..V1.., 7–67
..V2.., 7–67
Version 1 RT11, 7–67
Version 2 RT11, 7–67
Vertical format control characters, 5–42t
Virtual addressing, 2–1, 2–2

W

.WAIT, 7–64
WCB, 3–275
WDB, C–2
WDBBK$, C–2
WDBDF$, C–2
Wildcard
  account read and reset, 3–312
  file lookup, 3–55, 3–296
Wildcard (Cont.)
  project–programmer number lookup, 3–308
Window, 2–7
  creating, 5–12
  eliminating, 3–146, 5–19
  into resident library, 3–137
  mapping to, 3–149, 5–36
  maximum number of, 3–138
  unmapping from, 3–154, 5–55
Window control block, 3–275
Window definition block, C–2
Window ID
  returned by CRAFQ, 3–140
  returned by CRAW$, 5–15
  used by ELAFQ, 3–146
  used by ELAW$, 5–19, 5–20
  used by MAPFQ, 3–150
  used by UMAP$, 5–55
  used by UMPFQ, 3–154, 3–155
.WRITC, 7–65
.WRITE, 3–357 to 3–363
  RT11, 7–65
Writing, 3–357
  one character, 7–62
.WRITW, 7–65
WSIG$, 5–57
WTSE$, 5–58

X

XBUF, 2–3f
XRB, 2–11
  clearing under RT11, 7–11
  data returned to, 3–8
  general format, 2–12f
  mnemonics assigned to, 2–12
  on P.NEW entry, 2–27
  on P.RUN entry, 2–28
  presetting to 0, 3–7
  routine to clear, 3–8
  size of, 2–12
  within low 1000 bytes, 2–8f

Z

Zero device, 3–355

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA and Puerto Rico call **800–258–1710**

In Canada call **800–267–6146**

In New Hampshire, Alaska or Hawaii call **603–884–6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: A&SG Business Manager

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
A&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

*Any prepaid order from Puerto Rico must be placed with the Local Digital Subsidiary:
809–754–7575

RSTS/E
System Directives Manual
AA–D748C–TC

Reader's Comments

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

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

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number. _____

_____

_____

_____

_____

_____

_____

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
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Commercial Engineering Publications MK01-2/E06
RSTS/E Documentation
DIGITAL EQUIPMENT CORPORATION
CONTINENTAL BOULEVARD
MERRIMACK, N.H. 03054

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line