RT–11 System Subroutine Library Manual

Order Number AA–PD6MA–TC

August 1991

This manual contains current reference data about the system subroutine library (SSL), a collection of routines callable from high-level languages (FORTRAN and C).

Revision/Update Information: This information was previously published, along with reference data about the system macro library, as part of the *RT–11 Programmer's Reference Manual,* AA–H378D–TC.

Operating System: RT–11 Version 5.6

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, August 1991

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.

Any 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 or equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013.

© Digital Equipment Corporation 1991
All rights reserved. Printed in U.S.A.

The Reader's Comments form at the end of this document requests your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: CTS–300, DDCMP, DECnet, DECUS, DECwriter, DIBOL, MASSBUS, MicroPDP–11, Micro/RSX, PDP, Professional, Q-bus, RSTS, RSX, RT–11, RTEM–11, UNIBUS, VMS, VT, and the DIGITAL logo.

S1436

This document was prepared using VAX DOCUMENT, Version 1.2.

Contents

Preface ix

Summary of Changes xiii

Chapter 1 Using the System Subroutine Library

1.1 Overview 1–1
1.1.1 SYSLIB Functional Organization 1–2
1.1.2 Applicability 1–2
1.2 System Conventions 1–2
1.2.1 Naming Conventions 1–3
1.2.2 Subroutines and Functions 1–3
1.2.3 Channel Numbers 1–3
1.2.4 Completion Routines 1–4
1.2.5 Completion Routine Restrictions 1–4
1.2.6 Device Blocks 1–5
1.2.7 INTEGER*4 Support Functions 1–5
1.2.8 User Service Routine (USR) Requirements 1–6
1.2.9 Subroutines Requiring Additional Queue Elements 1–14
1.2.10 System Restrictions 1–14
1.3 Calling SYSLIB Subroutines or Functions 1–15
1.4 FORTRAN/MACRO Interface 1–23
1.4.1 Subroutine Register Usage 1–24
1.4.2 FORTRAN Programs Calling MACRO Subroutines 1–25
1.4.3 MACRO Routines Calling FORTRAN Programs 1–27
1.5 FORTRAN Programs in a Foreground/Background Environment 1–29
1.5.1 Calculating Workspace for a FORTRAN Foreground Program 1–30
1.5.2 Running a FORTRAN Program in a Foreground/Background Environment 1–31
1.6 Linking with FORLIB 1–33
1.7 SYSLIB Services Not Provided by Programmed Requests 1–33
1.7.1 Time Conversion and Date Access 1–33
1.7.2 Program Suspension 1–34
1.7.3 Two-Word Integer Support (INTEGER*4) 1–34
1.7.4 Radix–50 Conversion 1–35
1.7.5 Character String Operations 1–35
1.7.6 Control of Global Regions 1–36
1.8 Character String Functions 1–36
1.8.1 Allocating Character String Variables 1–37

1.8.2 Passing Strings to Subprograms 1–38
1.8.3 Using Quoted-String Literals 1–39

Chapter 2 System Subroutine Description and Examples

ABTIO/IABTIO 2–2
AJFLT/IAJFLT 2–3
CALL$F 2–5
CHAIN 2–6
CHCPY/ICHCPY 2–8
CLOSEC/ICLOSE 2–10
CLOSZ/ICLOSZ 2–12
CMAP/ICMAP 2–14
CMKT/ICMKT 2–16
CNTXS/ICNTXS 2–17
CONCAT 2–18
CRAW/ICRAW 2–20
CRRG/ICRRG 2–24
CSI/ICSI 2–25
CSTAT/ICSTAT 2–29
CVTTIM 2–31
DATE/DATE4Y 2–32
DELET/IDELET 2–34
DEVICE/IDEVICE 2–36
DJFLT 2–37
DSTAT/IDSTAT 2–38
ELAW/IELAW 2–40
ELRG/IELRG 2–41
ENTER/IENTER 2–42
FPROT/IFPROT 2–44
FREER/IFREER 2–45
GCMAP/IGCMAP 2–46
GETR/IGETR 2–48
GFDAT/IGFDAT 2–52
GFINF/IGFINF 2–53
GFSTA/IGFSTA 2–55
GICLOS/GIOPEN/GIREAD/GIWRIT (GIDIS) 2–57
GMCX/IGMCX 2–62
GTDIR/IGTDIR 2–63
GTDUS/IGTDUS 2–69
GTIM 2–74
GTJB/IGTJB 2–75
GTLIN/IGTLIN 2–77
HERR/IHERR 2–79
IADDR 2–81

IDATE 2–82
IDCOMP 2–84
IFWILD 2–86
IGTENT 2–89
IJCVT 2–91
INDEX 2–92
INSERT 2–93
IPEEK 2–94
IPEEKB 2–96
IRAD50 2–97
ISPY 2–98
ISWILD 2–99
ITLOCK 2–101
ITTINR 2–102
ITTOUR 2–104
IWEEKD 2–105
JADD 2–106
JAFIX 2–107
JCMP 2–108
JDFIX 2–110
JDIV 2–111
JICVT 2–113
JJCVT 2–114
JMOV 2–115
JMUL 2–116
JREAD/JREADC/JREADF/JREADW 2–118
JSUB 2–125
JTIME 2–127
JWRITE/JWRITC/JWRITF/JWRITW 2–128
KPEEK 2–137
KPOKE 2–138
LEN 2–140
LOCK/UNLOCK 2–141
LOOKUP 2–144
MAP 2–147
MRKT 2–148
MSDS 2–150
MTATCH 2–151
MTDTCH 2–152
MTGET 2–153
MTIN 2–154
MTOUT 2–155
MTPRNT 2–156
MTRCTO 2–157
MTSET 2–158

MTSTAT 2–159
MWAIT 2–160
POKE/IPOKE 2–161
POKEB/IPOKEB 2–162
PRINT 2–163
PROTE/IPROTE 2–164
PURGE 2–165
PUT/IPUT 2–166
R50ASC 2–167
RAD50 2–168
RAN/RANDU 2–169
RCHAIN 2–170
RCTRLO 2–171
*RCVD/*RCVDC/*RCVDF/*RCVDW 2–172
*READ/*READC/*READF/*READW 2–179
RENAM/IRENAM 2–191
REOPN/IREOPN 2–193
REPEAT 2–195
RESUME 2–197
SAVES/ISAVES 2–198
SCCA/ISCCA/MSCCA 2–199
SCHED/ISCHED 2–201
SCOMP/ISCOMP 2–203
SCOPY 2–204
*SDAT/*SDATC/*SDATF/*SDATW 2–205
SDTTM/ISDTTM 2–214
SERR/ISERR 2–216
SETCMD 2–218
SFDAT/ISFDAT 2–219
SFINF/ISFINF 2–220
SFSTA/ISFSTA 2–223
SLEEP/ISLEEP 2–226
SPCPS/ISPCPS 2–227
*SPFN/*SPFNC/*SPFNF/*SPFNW 2–229
STRPAD 2–239
SUBSTR 2–240
SUSPND 2–241
$SYTRP 2–242
TIMASC 2–244
TIME 2–245
TIMER/ITIMER 2–246
TRANSL 2–248
TRIM 2–250
TWAIT/ITWAIT 2–251
UNMAP/IUNMAP 2–252

UNPRO/IUNPRO 2–253
UNTIL/IUNTIL 2–254
VERIFY/IVERIFY 2–256
WAIT/IWAIT 2–257
*WRITE/*WRITC/*WRITF/*WRITW 2–258

Index

Figures

1–1 A FORTRAN Program in Memory 1–10
1–2 Subroutine Argument Block 1–24
1–3 Argument Block for Program FINITA 1–27

Tables

1 Functions and Subroutines Deleted from SYSLIB xiii
2 Routines Added or Changed for I-D Space xiv
3 Other Changes to SYSLIB Subroutines xiv
1–1 PSECT Ordering for FORTRAN Programs (Low to High Memory) 1–9
1–2 PSECT Ordering for PDP–11 RTL Programs 1–11
1–3 SYSLIB Subroutines and Functions 1–16
1–4 Return Value Conventions for Function Subroutines 1–25
1–5 SYSLIB Conversion Calls 1–34
1–6 Character String Functions 1–36
2–1 Device Support (SF.AWR/SF.ARD) 2–229
2–2 Device Support (SF.MWE/SF.MWR) 2–230

Preface

This manual contains reference data about RT–11 system subroutines, a collection of call routines contained in system library SYSLIB.OBJ. As a FORTRAN programmer, you access RT–11 Monitor services through these call routines to the system subroutine library. Using SYSLIB subroutines, you can write almost all application programs in FORTRAN without having to write code in any assembly language. This library is also accessible from PDP–11C.

Reference data about the system macro library, previously contained in the *RT–11 Programmer's Reference Manual* is now contained in a separate manual, *RT–11 System Macro Library Manual*. See *Associated Documents*.

Intended Audience

This information is provided for use by advanced RT–11 users, including FORTRAN IV, FORTRAN–77, and MACRO–11 assembly language programmers and C language programmers.

Document Structure

Chapter 1 — Using the System Subroutine Library

Describes implementation and effective use of subroutines contained in SYSLIB.OBJ; provides examples that demonstrate subroutine flexibility and value in working programs.

Chapter 2 — System Subroutine Description and Examples

Presents all SYSLIB functions and subroutines in alphabetical order; provides a detailed description of each one. Gives examples of each call in a FORTRAN program.

Associated Documents

The RT–11 Documentation Set consists of the following associated documents:

Basic Books

- *Introduction to RT–11*
- *Guide to RT–11 Documentation*
- *RT–11 Commands Manual*
- *PDP–11 Keypad Editor User's Guide*
- *PDP–11 Keypad Editor Reference Card*

- *RT–11 Quick Reference Manual*
- *RT–11 Master Index*
- *RT–11 System Message Manual*
- *RT–11 System Release Notes*

Installation Specific Books

- *RT–11 Automatic Installation Guide*
- *RT–11 Installation Guide*
- *RT–11 System Generation Guide*

Programmer Oriented Books

- *RT–11 IND Control Files Manual*
- *RT–11 System Utilities Manual*
- *RT–11 System Macro Library Manual*
- *RT–11 System Subroutine Library Manual*
- *RT–11 System Internals Manual*
- *RT–11 Device Handlers Manual*
- *RT–11 Volume and File Formats Manual*
- *DBG–11 Symbolic Debugger User's Guide*

Conventions

The following conventions are used in this manual:

Convention	Meaning
Black print	In code examples, black print indicates output lines or prompting characters the system displays.
Braces ({ })	In command examples, braces enclose mutually exclusive options. You can choose only one of the options contained in braces.
Brackets [ ]	In command examples, square brackets enclose optional parameters, qualifiers or values. For example: **i = ISFDAT (chan,dblk[,idate][,iold])**
UPPERCASE characters	In command examples, uppercase characters are command elements that should be entered exactly as given.
lowercase characters	In command syntax examples, lowercase characters are command elements for which you specify a value. For example: **CALL POKEB (iaddr,ivalue)**
Command Syntax	Functions and subroutines have different formats: A function has the format: **i = POKEB (iaddr,ivalue)** A subroutine has the format: **CALL POKEB (iaddr,ivalue)** Alternative commands are shown as: **i = POKEB (iaddr,ivalue)** **CALL POKEB (iaddr,ivalue)**
RET	RET in examples and in the example installation represents the RETURN key.
CTRL/*x*	CTRL/*x* indicates a control key sequence. While pressing the key labeled Ctrl, press another key. For example: CTRL/C

Summary of Changes

This section summarizes additions, deletions and changes to the system subroutine library (SYSLIB). Refer to Chapter 2 for detailed description of RT–11 system subroutines.

Changes Between SYSLIB and FORTRAN OTS (FORLIB and F77OTS)

The following SYSLIB changes affect the relationship between SYSLIB and the FORTRAN Object Time Systems (OTS).

- Functions and subroutines DATE, IDATE, RAN, and RANDU, previously in the distributed FORTRAN subroutine libraries, are now located in SYSLIB.OBJ.
- The following functions and subroutines (see Table 1), specific to FORTRAN programming, have been deleted from the distributed RT–11 system subroutine library, SYSLIB.OBJ, because they do not work without a resident FORTRAN OTS. These functions and subroutines have been added to the FORTRAN IV distributed FORLIB and the FORTRAN–77 distributed F77OTS.

Table 1: Functions and Subroutines Deleted from SYSLIB

GETSTR	IFREEC	INTSET
IASIGN	IGETC	IQSET
ICDFN	IGETSP	PUTSTR
IFETCH	ILUN	SECNDS

NOTE

Because IQSET is no longer in SYSLIB.OBJ, FORTRAN programmers who need to add queue elements for certain other SYSLIB functions, should refer to the FORTRAN IV distributed FORLIB and the FORTRAN–77 distributed F77OTS.

SYSLIB Subroutines

Changes for I-D Space

Table 2 lists subroutine changes resulting from the addition of Supervisor Mode, I-D space. One of the major features of V5.6 SYSLIB support added mapping routines that begin with letter *M*.

The *mapping* version of a routine is called in the same manner as an unmapped version, but has an added argument that specifies the type of mapping required. In

these cases, mapping is shown as an optional parameter in the generic command string. For example, ISDATW/MSDATW is shown as:

Form:

```
i = ISDATW (buff,wcnt)

i = MSDATW (buff,wcnt[,bmode])
```

Table 2: Routines Added or Changed for I-D Space

ICMAP	IUNMAP	JREADW	MREAD	MREADC	MSPFNC
ICRAW	ISCCA	JWRITE	MREADW	MRVCDW	MTATCH
ICRRG	JREAD	JWRITW	MRKT	MSDS	MWRITC
IELRG	JREADC	MAP	MSDAT	MSPFN	MWRITE
IGCMAP	JREADW	MRCVD	MSDAT	MSDATC	MWRITW
IGCMX	JWRITC	MRCVDC	MSPFNW	MSDATW	

SYSLIB Subroutine Changes

Table 3 lists subroutines that have been changed in V5.6.

Table 3: Other Changes to SYSLIB Subroutines

DEVICE	IFPROT	ISPY
GIWRIT	IGTDUS	IUNTIL
GTLIN	ISFDAT	LOOKUP
ICSTAT	ISPFN	SCCA

Chapter 1
Using the System Subroutine Library

1.1 Overview

The system subroutine library is a collection of FORTRAN- and C-callable routines contained in the SYSLIB.OBJ system library which also contains overlay handlers, utility functions, a character string manipulation package, and two-word integer support routines. High-level language programmers use these subroutines to take full advantage of the latest RT–11 system features. The linker also uses this library to resolve undefined globals.

If you are not familiar with the *PDP–11 FORTRAN Language Reference Manual* and the *RT–11/RSTS/E FORTRAN IV User's Guide*, and the *Guide to PDP–11 C,* you should refer to these manuals before using the material described in this chapter. C language programmers should also refer to the *PDP–11 C Run-Time Library Reference Manual* for information about functions and macros.

The system subroutine library provides the following capabilities:

- Complete RT–11 I/O facilities, including synchronous, asynchronous, and event-driven modes of operation. FORTRAN subroutines can be activated upon completion of an input/output operation.
- Timed scheduling of completion routines, standard in the multijob and mapped monitors, is a SYSGEN option for the SB monitor.
- Facilities for communication between foreground and background jobs.
- FORTRAN language interrupt service routines for user devices.
- Complete timer support facilities, including
  - — Timed suspension of execution in multijob or mapped environments
  - — Conversion of different time formats, and time-of-day information
  - — Timer support for facilities using either 50- or 60-cycle clocks
- Facilities for creating, attaching, detaching, and eliminating global regions in extended memory.
- All RT–11 auxiliary input/output functions: opening, closing, renaming files; creating or deleting files on any device.
- All monitor-level information functions, such as job partition parameters, device statistics, and input/output channel statistics.
- Support interface and multiterminal environment.

- Access to the RT–11 command string interpreter (CSI).
- Access to limited instruction-data (I-D) space and Supervisor mode support.
- Mapping routines, similar to unmapped routines, having added argument(s) that specify mapping required.
- Character string manipulation package supporting variable-length character strings.
- INTEGER*4 support routines that allow two-word integer computations.

In reference to variables, unless otherwise specified, *INTEGER* means INTEGER*2, (16-bit integer) and *REAL* means REAL*4 (single-precision floating point). Integer and real arguments to subprograms are indicated in this section as follows:

i = INTEGER*2 arguments
j = INTEGER*4 arguments
a = REAL*4 arguments
d = REAL*8 arguments

1.1.1 SYSLIB Functional Organization

RT–11 system subroutines and functions are presented in alphabetical sequence of their generic name. For example, because READC, IREADC, and MREADC can be called as subroutines or as functions, they are presented together under *READC to facilitate easy lookup. An I-prefixed name denotes use as a function; an M-prefixed name denotes the function or subroutine has extra arguments that specify mapping.

Functionally related calls to enable or disable functions are presented together to facilitate easy lookup. For example, LOCK and UNLOCK are presented together as LOCK/UNLOCK.

1.1.2 Applicability

In general, SYSLIB routines were written for use with RT–11 V2 or later and FORTRAN IV V1B or later versions for RT–11 or FORTRAN may lead to unpredictable results. SYSLIB now supports virtually all monitor requests. Do not use a SYSLIB routine on an older monitor that does not support the request implemented by the routine.

1.2 System Conventions

This section describes system conventions that must be followed for proper operation of calls to the system subroutine library. (For applicable restrictions, see Section 1.2.10.)

1.2.1 Naming Conventions

In FORTRAN, subroutine names starting with I-through-N (inclusive) are, by default, integer returns; names starting with A-through-H and O-through-Z are real returns.

SYSLIB names are the same as those used in SYSMAC except that, when names in SYSMAC start with letters other than I-through-N, the letter I is appended to the beginning. Also the following conventions apply:

- Names that start with the letter I ordinarily are functions that return a 16-bit value.
- Names that start with the letter J return a 16-bit value, but operate on a 32-bit value.
- Names that start with the letter K (such as KPEEK) are functional extensions of names beginning with I (such as IPEEK), functionally the same, except that the K-version has an optional argument.
- Names that start with the letter M (such as MRCVD) indicate that mapping or multimode mapping has been added to a function (such as IRCVD).

  Names that start with the letter M might also identify multiterminal equivalents of generic functions or subroutines; for example, MTOUT and MTPRINT are multiterminal equivalents of ITTOUR and PRINT.

1.2.2 Subroutines and Functions

If a SYSLIB routine returns a value, it is more useful as a function than as a subroutine. If the routine does not return a value, it should only be used as a subroutine. In instances where they can function as well in either role, SYSLIB descriptions are presented for both forms. Generally, subroutines whose names start with letters other than I-through-N either do not return a useful value or return a floating-point value.

1.2.3 Channel Numbers

A channel number is a logical identifier for a file used to communicate with RT–11. When you open a file on a particular device, you assign a channel number to that file. When you refer to an open file, just refer to the appropriate channel number.

The FORTRAN system has 16(decimal) channels available. The call IGETC assigns a channel to your program and notifies the FORTRAN I/O system, which also uses these channels, that the channel is in use. When there is no longer need for a channel, the program should close the channel with a CLOSEC, ICLOSE, or a PURGE SYSLIB call. The channel should also be closed and returned to the FORTRAN I/O system with a IFREEC call.

The ICDFN call can activate up to 255(decimal) channels. ICDFN sets aside memory in the job area to accommodate status information for the extra channels. Use the ICDFN call during the initialization phase of your program. You can use all channels numbered higher than 15(decimal). The FORTRAN I/O system uses channels 0 through 15(decimal).

You must allocate channels in the main program routine or its subprograms. Do not allocate channels in routines that are activated as the result of I/O completion events or ISCHED or ITIMER calls.

1.2.4 Completion Routines

A completion routine is a subprogram that executes asynchronously with a main program and is scheduled to run as soon as possible after the completion of an associated event, such as an I/O transfer or the passing of a specified time interval. All completion routines of the current job have higher priority than other parts of the job. When a completion routine is initiated (because of its associated event), it interrupts execution of the job and continues to execute until it relinquishes control.

Completion routines can be written in FORTRAN or assembly language, depending on the function called. Assembly language completion routines exit with a RETURN instruction. FORTRAN completion routines exit by the execution of a RETURN or END statement in the subroutine. Names of all completion routines external to the routine being coded and passed to scheduling calls must be specified in an EXTERNAL statement in the FORTRAN program unit issuing the call:

A completion routine written in FORTRAN can have a maximum of two arguments:

Form:

SUBROUTINE crtn [(iarg1,iarg2)]

where:

crtn	is the name of the completion routine
iarg1	is the equivalent to R0 on entry to an assembly language completion routine
iarg2	is equivalent to R1 on entry to an assembly language completion routine

For information on the meaning of R1 and R0 contents, see the *RT–11 System Macro Library Manual*:

If an error occurs in a completion routine or in a subroutine at completion level, the error handler traces back through to the original interruption of the main program. Thus, the traceback is shown as though the completion routine were called from the main program.

1.2.5 Completion Routine Restrictions

Certain restrictions apply to completion routines that are activated by the following calls:

INTSET	IREADF	ISPFNC	IWRITF
IRCVDC	ISCHED	ISPFNF	MRKT
IRCVDF	ISDATC	ITIMER	
IREADC	ISDATF	IWRITC	

When using these calls the following restrictions apply:

- No channels can be allocated by calls to IGETC or freed by calls to IFREEC from a completion routine. Channels to be used by completion routines should be allocated and placed in a COMMON block for use by the routine.

  Even if the completion routine itself does not issue any programmed requests, but does perform I/O to a logical unit number through the OTS, that logical unit number must be opened from the main level. To accomplish this, either issue the first I/O access or an OPEN statement from main level. A completion routine may not call CLOSE to close a logical unit.

- FORTRAN subroutines are reusable but not reentrant. That is, a given subroutine can be used many times as a completion routine or as a routine in the main program, but a subroutine executing as main program code does not work properly if it is interrupted and then called again at the completion level. This restriction applies to all subroutines that can be invoked at the completion level while they are active in the main program.

- FORTRAN completion routines can be called only by SYSLIB functions that end in F. Conversely, MACRO completion routines cannot be called by SYSLIB functions that end in F. (SYSLIB function names ending in the letter F interface to the FORTRAN run-time system.)

1.2.6 Device Blocks

A device block is a four-word block of Radix–50 information that specifies a physical device and a file name. In FORTRAN, you can use one of three methods to set up this block as follows:

- Use the DIMENSION and DATA statements. For example,

```
Dimension IFILE(4)
Data IFILE /3rSY ,3rFIL,3rE  ,3rXYZ/
```

- Translate the available ASCII file description string into Radix–50 format, using the SYSLIB calls IRAD50, R50ASC, and RAD50. For example,

```
Real*8 FSPEC
Call IRAD50(12, 'SY FILE  XYZ', FSPEC)
```

- Use SYSLIB call ICSI to call the Command String Interpreter (CSI) to accept and parse standard RT–11 command strings.

1.2.7 INTEGER*4 Support Functions

For a description of INTEGER*4 functions for use by the MACRO programmer, see Section 1.7.3.

When you use the DATA statement to initialize INTEGER*4 variables, you must specify both the low- and high-order parts. For example, the code that follows initializes only the first word:

```
Integer*4 J
Data J /3/
```

The following example shows the correct way to initialize an INTEGER*4 variable to a constant, such as 3:

```
Integer*2 M(2)
Data M /3, 0/   !low order, high order
```

If you are initializing an INTEGER*4 variable to a negative value such as -4, the high-order (second word) part must be the continuation of the two's complement of the low-order part. For example,

```
Integer*4 L
Integer*2 L2(2)
Equivalence (L, L2)
Data L2 /-4, -1/ !initialize L to -4
```

1.2.8 User Service Routine (USR) Requirements

The RT–11 User Service Routine (USR) is always resident in all mapped monitors; therefore, this discussion applies to unmapped monitors only. User-written routines that interface to the FORTRAN Object Time System (OTS) must account for the location of the USR. PDP–11 C user-written routines have similar requirements. USR swapping requirements for FORTRAN and C are discussed in this section.

The USR occupies 2K words. When your program calls a SYSLIB routine that requests a USR function (such as IENTER or LOOKUP) or when the USR is invoked by the FORTRAN OTS, the USR is swapped into memory if it is nonresident. The FORTRAN OTS is designed so that the USR can swap over it.

Because letting USR swap over certain kinds of data and code causes unpredictable results, you must restrict interrupt service routines and completion routines to locations outside the USR swapping area. Identify the limits of this swapping area by examining the link map and, if necessary, change the order of object modules and libraries as specified to linker.

The following subroutines require the USR:

CLOSEC,ICLOSE
GETSTR (only if first I/O operation on logical unit)
GTLIN
ICDFN (single job only)
ICSI
IDELET
IDSTAT
IENTER
IFETCH
IQSET
IRENAM
ITLOCK (only if USR is not in use by another job)
LOCK (only if USR is in a swapping state)
LOOKUP
PUTSTR (only if first I/O operation on logical unit)

Controlling USR Swapping

You can control USR swapping by using the KMON commands SET USR NOSWAP and SET USR SWAP:

- SET USR NOSWAP prevents swapping and freezes the USR in memory.
- SET USR SWAP reverses this, allowing the USR to swap under program control.

Another alternative is to compile your FORTRAN main program with the /NOSWAP option if you are sure that there is space just below the foreground partition or RMON to make the USR permanent for the duration of your program. Use this option if your program does not need the 2K words of memory that the USR occupies. If the /NOSWAP option is not specified, the USR swaps over locations 1000–11000, the 2K words of your program above the base address, and the part of a FORTRAN program least likely to violate the USR restrictions.

To prevent USR swapping for part of the program execution time and to allow the USR to swap out at other times, use the LOCK, UNLOCK, and ITLOCK calls:

- LOCK call locks the USR into main memory and attaches it to the requesting job.
- The UNLOCK call lets the USR swap again and be used by another job.
- The LOCK and UNLOCK calls are used in a foreground program to prevent interference from the background during initialization and completion phases and to minimize the number of swaps.
- If ITLOCK determines another job is already using the USR, it returns an error code that lets the program try for a lock, but continue with other action if it fails.

Keeping the USR Resident

For a FORTRAN main program, you can keep the USR resident by using the FORTRAN/NOSWAP command (or the /U compiler option) at compile time. This forces the USR to remain resident while the program is executing. You cannot use this option if your FORTRAN programs require the extra 2K words of memory.

Allowing the USR to Swap

As with a MACRO program, the only reason to permit the USR to swap with a FORTRAN program is to gain access to an additional 2K words of memory. The USR normally swaps over the FORTRAN OTS (Object Time System). However, problems occur when the FORTRAN OTS and the program together are less than 2K words long. In this case, the USR swaps over the program's impure data area, with unpredictable results. (Since this error is frequently made by inexperienced programmers, setting the USR to NOSWAP and retrying a program is the first thing you should do when debugging a FORTRAN program that does not execute properly.) And USR swapping does not depend on your program's high limit—that is, if the USR is allowed to swap, it most definitely will swap. So, do not permit USR swapping unless your program really needs the extra memory. To enable swapping for a FORTRAN program, make sure the SET USR SWAP command is in effect, and eliminate the /NOSWAP or the /U option at compile time.

PSECT Ordering for FORTRAN

To change the position of code or data to avoid the USR swapping area, or to move the USR itself, consider the use of program section (PSECT) ordering. PSECTs contain code and data identified by names as segments of the object program. Attributes associated with each PSECT direct the Linker to combine several separately compiled FORTRAN program units, assembly language modules, and library routines into an executable program.

The order in which program sections are allocated in the executable program is the same order in which they are presented to the Linker. Applications sensitive to this ordering typically separate those sections containing read-only information (such as executable code and pure data) from impure sections containing variables.

The main program unit of a FORTRAN program (normally the first object module in sequence presented to LINK) declares PSECT ordering as shown in Table 1–1.

The USR can swap over pure code, but must not be loaded over constants or impure data that can be used as arguments to the USR. The ordering shown in Table 1–1 collects all pure sections before collecting impure data in memory.

It is important to understand where and how the USR swaps so you can design your FORTRAN program correctly. For a FORTRAN program, the FORTRAN OTS places a value in $UFLOA *location 46* to set up the USR swapping function. When a FORTRAN program is running, the USR will automatically start swapping at the base of OTS$I. $UFLOA of the *System Communication Area* contains the address where the USR will swap. If the value of $UFLOA is zero, the USR will swap at its default location, below RMON and handlers.

The FORTRAN compiler examines the sections of your program and sorts them based on two major attributes: read-only versus read-write, and pure code versus data. Generally, program instructions are read-only, and program data is read-write. If you use assembly language routines, use the same PSECT as the FORTRAN compiler. That is, place pure code and read-only data in section USER$I, and impure data in USER$D. The compiler forces PSECT into the order shown in Table 1–1. PSECT attributes shown in this table are abbreviated as follows:

RW, RO—Read/Write, Read Only
I, D—Instructions, Data
REL—Relocatable
CON, OVR—Concatenated, Overlaid
LCL, GBL—Local with overlay segment, Global across segments
SAV—Unconditionally place PSECT in root segment.

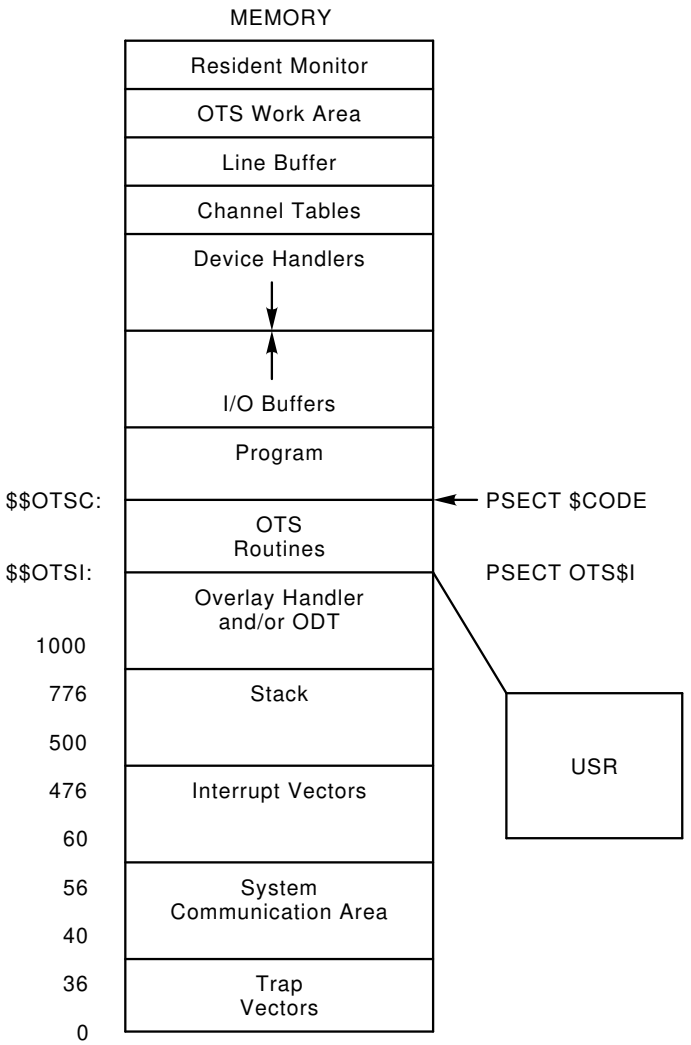
See the *RT–11/RSTS/E FORTRAN IV User's Guide* and *RT–11/FORTRAN 77 User's Guide* for more information on program sections. See also *RT–11 System Internals Manual* and *RT–11 System Utilities Manual* for information on USR swapping and PSECT ordering.

Table 1–1: PSECT Ordering for FORTRAN Programs (Low to High Memory)

Section Name	Attributes
	FORTRAN IV
OTS$I	RW, I, LCL, REL, CON
OTS$P	RW, D, GBL, REL, OVR
SYS$I	RW, I, LCL, REL, CON
USER$I	RW, I, LCL, REL, CON
$CODE	RW, I, LCL, REL, CON
OTS$O	RW, I, LCL, REL, CON
SYS$O	RW, I, LCL, REL, CON
$DATAP	RW, D, LCL, REL, CON
OTS$D	RW, D, LCL, REL, CON
OTS$S	RW, D, LCL, REL, CON
SYS$S	RW, D, LCL, REL, CON
$DATA	RW, D, LCL, REL, CON
USER$D	RW, D, LCL, REL, CON
.$$$$.	RW, D, GBL, REL, OVR
Other COMMON Blocks	RW, D, GBL, REL, OVR
	FORTRAN 77
$CODE1	RW, I, LCL, REL, CON
$PDATA	RW, D, LCL, REL, CON
$IPDATA	RW, D, LCL, REL, CON
$VARS	RW, D, LCL, REL, CON
$TEMPS	RW, D, LCL, REL, CON
$SAVE	RW, D, GBL, REL, CON

This ordering collects all pure sections before impure data in memory. The USR can safely swap over sections OTSI, OTSP, SYS$I, USER$I, and $CODE. Figure 1–1 shows the arrangement of components when a FORTRAN program is loaded into memory. The global symbol $$OTSI marks the start of the pure code area. The global symbol $$OTSC marks its end and the beginning of the impure data area. FORTRAN puts the value of $$OTSI into location 46, and the USR swaps into memory starting at that address, thus overlaying the first 2K words of your program.

Figure 1–1: A FORTRAN Program in Memory



As with a MACRO program, your FORTRAN program should not have certain instructions or data in the area where the USR will swap. As a general rule, the following items should not be in the USR swap area:

- Routines that request USR functions (such as IENTER and LOOKUP)
- Data structures for USR requests
- Interrupt service routines
- Completion routines
- Data areas for interrupt service routines and completion routines

The FORTRAN system itself must also be concerned with USR swapping and its inherent restrictions. For example, the PSECT OTS$O contains the FORTRAN

OTS routines to open files. This PSECT follows $CODE in the PSECT ordering. If the start of OTS$O is within 2K words of $$OTSI, the essential information for the file operation is stored on the job stack before the USR swaps over the code in OTS$O.

The best way to make sure that the USR swaps into a safe place in your FORTRAN program is to examine the link map to determine if the USR will swap over restricted sections. That is, see if the first 2K words above $$OTSI can be overlaid safely. If not, relink the program and change the order of object modules and libraries you specify to the linker. One problem is caused by using SYSLIB routines that place important USR data in the lower 2K words of the job image. An example is the IFETCH routine, which uses a device block in the program. The USR swaps over the device block just before it is used, causing an error. To avoid a situation like this, do not set up device names as constants for a SYSLIB call. Instead, use DATA-initialized variables. This ensures that the information will be stored high enough in the job image to avoid being overlaid by the USR.

PSECT Ordering for PDP–11 C

Table 1–2 lists PSECTs used by the PDP–11 Run Time Library and outlines their use. Under RT–11 if the USR is not resident, the PDP–11 C RTL will attempt to set the USR to swap at the location of the root C$STDI and C$OTSI PSECTs. See *PDP–11 C Guide to PDP–11 C* for more information on PSECT ordering.

Table 1–2: PSECT Ordering for PDP–11 RTL Programs

Section Name	Use
C$CCT0 C$CCT2	Character collating table. Used for locale-specific routines to determine the collating sequence of each character set.
C$CMT0 C$CMT2	Character mapping table. Used for locale-specific routines to determine the results of character mapping functions for each character set.
C$CTT0 C$CTT2	Character testing table. Used for local-specific routines to determine the results of character testing functions for each character set.
C$END0 C$END1 C$END2 C$END3	The C$ENDx PSECTs are used for end-of-task processing. The addresses of functions to be called by the PDP–11 C RTL at task-exit time are place in the PSECT C$END1. For instance, the address of the routine that ensures all files are closed is placed in C$END1. This is separate from the atexit system function. The PSECTs C$END0, C$END1, and C$END3 are reserved for use by the PDP–11 C RTL. The addresses of routines to be called at task exit can be placed in the PSECT C$END2. Modules that define this PSECT may not reside in a resident library.
C$INI0 C$INI1 C$INI2 C$INI3	Similar to the C$ENDx PSECTs, the C$INIx PSECTs are used to provide the addresses of routines to be called at task startup. The PSECTs C$INIO, C$INI1, and C$INI3 are reserved for use by the PDP–11 C RTL. The PSECT C$INI2 is available to place the addresses of routines to be called at task startup. Modules that define this PSECT may not reside in a resident library.

Table 1–2 (Cont.): PSECT Ordering for PDP–11 RTL Programs

Section Name	Use
C$INIR	Code for initialization routines.
C$MFT0 C$MFT2	Monetary formatting table. Used for locale-specific routines to determine the results of monetary formatting functions for each character set.
C$NFT0 C$NFT2	Numeric formatting table. Used for locale-specific routines to determine the results of numeric formatting functions for each character set.
C$OTSC	Constant data for PDP–11 C Object Time System routines.
C$OTSD	Read data for the PDP–11 C OTS routines.
C$OTSH	RT–11 only. Used to determine size of C$OTSI and C$STDI PSECTs.
C$OTSI	Instructions for PDP–11 C OTS routines. These routines handle most of the math and conversion functions.
C$OTSJ	RT–11 only. Used to determine size of C$OTSI and C$STDI PSECTs.
C$OTSR	Constant data for PDP–11 C OTS routines.
C$OTSW	Writable storage for PDP–11 C OTS routines. Modules that contain this PSECT may not reside in a resident library.
C$STDC	Constant data for the Standard Library routines.
C$STDD	Read data for the Standard Library routines.
C$STDI	Instructions for the Standard Library routines.
C$STDR	Constant data for the Standard Library routines.
C$TIM0 C$TIM2	Time formatting table. Used for locale-specific routines to determine the results of time formatting functions for each character set.
$PIOXT	I/O Transfer Vector. This is used to allow PDP–11 C to easily access several low-level I/O systems. $PIOXT contains two addresses for each low-level I/O action used by PDP–11 C. One address is for support for native I/O for that action. The other is for support for either RMS or FCS I/O for that action. Modules that define this PSECT may not reside in a resident library.
$PRLUN	Bit mask used for reserving LUNs. The first word indicates the number of words that follow. These make up a mask. Modules that define this PSECT may not reside in a resident library.
$$C	The PDP–11 C OTS work area. This is read/write data space used by the RTL. Modules that define this PSECT may not reside in a resident library.
$$CAST	OTS work area PSECT containing structure required by asctime function.
$$CCLK	OTS work area PSECT containing storage required for correct use of the clock function.

Table 1–2 (Cont.): PSECT Ordering for PDP–11 RTL Programs

Section Name	Use
$$CEXI	OTS work area PSECT containing storage required to register the addresses of the functions to be called during the executions of the atexit( ) routine.
$$CGEN	OTS work area PSECT containing storage required to support the getenv ( ) function.
$$CLOC	OTS work area PSECT containing storage required to support the locale functions.
$$CMLL	OTS work area PSECT containing storage required to support memory allocation functions.
$$CSIG	OTS work area PSECT containing storage required to support the signal functions.
$$CSIO	OTS work area PSECT containing storage required to support standard I/O operations.
$$CTIM	OTS work area PSECT containing storage required struct tm.

USR Lockout and Timing—All Monitors

If while one job is using the USR, another job requests it, the requesting job will be blocked until the other job releases the USR. The requesting job may be locked out for seconds or minutes at a time. Interrupt service and completion routines can run, but mainline code cannot. You can minimize or eliminate these resulting timing problems by observing the following:

- Do not use devices with slow directory operations, such as magtapes.
- Write real-time operations as completion and interrupt service routines in your foreground job so that a locked-out mainline program does not impede real-time operations.
- Separate USR and real-time operations.
- Use the ITLOCK call and avoid SYSLIB calls that request the USR while the USR is owned by another job.

A real-time foreground job has the following typical structure:

- An initialization phase that opens all required channels and begins a real-time operation
- A real-time phase that performs interrupt service and I/O operations
- A completion phase that halts real-time activity and then closes the channels.

Maintaining this structure in the foreground enables the background task to do USR operations during the real-time phase without locking out the foreground. This action simplifies USR swapping because the USR can swap over interrupt routines and I/O buffers as long as they are inactive.

1.2.9 Subroutines Requiring Additional Queue Elements

All subroutines in the following list require added queue elements for their proper operation. Subroutines prefixed with asterisks can be called as they are or may be prefixed with a letter I or M if they are called as functions or have added arguments for mapping. For example, *RCVD can have the form RCVD, IRCVD or MRCVD. These subroutines are as follows:

*RCVD, *RCVDC, *RCVDF, *RCVDW
*READ, *READC, *READF, *READW
IWAIT
SCHED/ISCHED
*SDAT, *SDATC, *SDATF, *SDATW
SLEEP/ISLEEP
*SPFN, *SPFNC, *SPFNF, *SPFNW
TIMER/ITIMER
TWAIT/ITWAIT
UNTIL/IUNTIL
*WRITC, *WRITE, *WRITF, *WRITW
MRKT
MWAIT

One queue element per job is automatically allocated. Issuing more than one request from the list requires extra queue elements. Additional queue elements can be allocated by a call to the IQSET function.

NOTE

IQSET is no longer contained in SYSLIB.OBJ. If you need to add queue elements for certain SYSLIB functions, refer to FORTRAN IV distributed FORLIB and to FORTRAN–77 distributed F77OTS.

1.2.10 System Restrictions

Consider the following system restrictions when coding a FORTRAN program that uses SYSLIB.

- Programs using IPEEK, IPOKE, IPEEKB, IPOKEB, or ISPY to access system-specific addresses, such as FORTRAN, monitor, or hardware addresses, are not guaranteed to run under future releases or on configurations other than those on which they were written. When using these functions, document their use so you can check your references against the current documentation. Also, these routines may act differently under the mapped monitor. IPEEK and IPOKE are not equivalent to programmed requests .PEEK and .POKE. Although IPEEK and IPOKE are equivalent to KPEEK and KPOKE, Digital recommends using KPEEK and KPOKE because they function better in a virtual environment.
- Various functions in SYSLIB return values that are of type *integer*, *real*, or *double precision*. To specify an implicit statement that changes the defaults for external function types, you must:

— Explicitly declare the type of those SYSLIB functions that return integer or real results.

— Be sure that the arguments to the SYSLIB routines are the correct type for the routine. Double-precision functions must always be declared to be type DOUBLE PRECISION (or REAL*8). Failure to observe this restriction leads to unpredictable results.

- Names of all completion routines external to the routine being coded and which are passed to scheduling calls (such as ISCHED, ITIMER, and IREADC) must be specified in an EXTERNAL statement in the FORTRAN program issuing the call.
- Certain arguments to SYSLIB calls must be located so that the USR is prohibited from swapping over them at execution time. This kind of swapping can occur when the OTS$I section (which contains the all-pure code and data for the module) is less than 2K words in length. Avoid swapping in this uncommon situation either by typing the SET USR NOSWAP command to make the USR resident before starting the job, or by compiling the mainline routine with a /NOSWAP option. You can also use the linker /BOUNDARY option to make OTS$O start at word boundary 11000(octal). (This problem generally occurs only with small FORTRAN programs.)

  In FORTRAN IV, FORTRAN 77 and C Language, program sections (PSECTs) are used to collect code and data into appropriate areas of memory. If USR is needed, but not resident, it will swap over a FORTRAN program, starting at the symbol OTS$I for 2K words of memory.
- Unless explicitly stated, null arguments should not be used in calls to SYSLIB routines.

1.3 Calling SYSLIB Subroutines or Functions

SYSLIB function subprograms and subroutines are called in the same manner as user-written subroutines. In general, if SYSLIB routines return a value, they are more useful as functions than as subroutines. If they do not return a value, they should be used only as subroutines. When functions or subroutines serve equally well in either role they are called routines. Call them in whichever way they are most useful. Subroutines whose names start with letters other than I-through-N either do not return a useful value or return a floating-point value.

PDP–11 C supports SYSLIB routines described in this document. The interface used to call routines is the FORTRAN subroutine linkage.

Table 1–3 lists SYSLIB functions/subroutines and briefly describes each within several types of categories:

- File Oriented Operations
- Data Transfer Operations
- Channel Oriented Operations

- Device and File Specifications
- Timer Support Operations
- RT–11 Services
- INTEGER*4 Support Functions
- Character String Functions
- Radix–50 Conversion Operations
- Multiterminal Operations
- Graphics (GIDCAL) Call Routines

SYSLIB entries are listed in the column that identifies their optimum use as functions, subroutines or equally well as either, depending on whether or not it is useful to return a value. Note the convention of prefixing subroutine calls with an I when called as functions; and prefixing both with an M when functions or subroutines have an added argument that specifies mapping. The *Map* column has two entries:

No SLB—Subroutine/Function cannot be used in Supervisor library.
No I-D—Subroutine/Function cannot be used in separated I-D space.

The *Restrictions* column lists restrictions to use of a function or subroutine; for example, ICNTXS can be used only in multijob environments.

SYSLIB subroutines IFREER and IGETR support mapping programmed requests, and FORTRAN virtual arrays can access extended memory.

Table 1–3: SYSLIB Subroutines and Functions

Subroutine	Function	Type	Map	Restrictions
ABTIO	IABTIO	Chan	–	–
	AJFLT	I*4	–	–
CALL$F	–	–	–	Macro
CHAIN	–	RT–11	–	–
CHCPY	ICHCPY	Chan	–	Multijob
CLOSEC	ICLOSE	File	–	–
CLOSZ	ICLOSZ	File	–	–
CMAP	ICMAP	Mapping	–	Full Mapping
CMKT	ICMKT	Timer	–	Timer
CONCAT		String	–	–
CNTXS	ICNTXS	RT–11	–	Multijob
CRAW	ICRAW	Mapping	–	Mapping

Table 1–3 (Cont.): SYSLIB Subroutines and Functions

Subroutine	Function	Type	Map	Restrictions
CRRG	ICRRG	Mapping	–	Mapping
CSI	ICSI	Dev/File Spec	No SLB	–
CSTAT	ICSTAT	Chan	–	–
CVTTIM		Timer	–	–
DATE		Timer	No SLB	–
DATE4Y		Timer	No SLB	–
DELET	IDELET	File	–	–
DEVICE	IDEVIC	RT–11	–	–
	DJFLT	I*4	–	–
DSTAT	IDSTAT	RT–11	–	–
ELAW	IELAW	Mapping	–	Mapping
ELRG	IELRG	Mapping	–	Mapping
ENTER	IENTER	File	–	–
FPROT	IFPROT	File	–	–
FREER	IFREER	Mapping	No SLB	Mapping
GCMAP	IGCMAP	Mapping	–	Full Mapping
GETR	IGETR	Mapping	–	Mapping
GFDAT	IGFDAT	File	–	–
GFINF	IGFINF	File	–	–
GFSTA	IGFSTA	File	–	–
GICLOS		Graphics	No SLB	Pro
GIOPEN		Graphics	No SLB	Pro
GIREAD		Graphics	No SLB	Pro
GIWRIT		Graphics	No SLB	Pro
GMCX	IGMCX	Mapping	–	Mapping
GTDIR	IGTDIR	Dev/File Spec	No SLB	–
GTDUS	IGTDUS	Dev/File Spec	–	–
GTIM		Timer	–	–
GTJB	IGTJB	RT–11	–	–
GTLIN	IGTLIN	Data Transfer	–	–
HERR	IHERR	RT–11	–	–

Table 1–3 (Cont.): SYSLIB Subroutines and Functions

Subroutine	Function	Type	Map	Restrictions
	IADDR	RT–11	–	–
IAJFLT	IAJFLT	I*4	–	–
	IDATE	Timer	–	–
	IDCOMP	Timer	–	–
IDJFLT	IDJFLT	I*4	–	–
	IFWILD	String	No SLB	–
	IGTENT	Dev/File Spec	No SLB	–
	IJCVT	I*4	–	–
INDEX	INDEX	String	–	–
INSERT		String	–	–
	IPEEK	RT–11	–	–
	IPEEKB	RT–11	–	–
IRAD50	IRAD50	RAD50	No SLB	–
	ISPY	RT–11	–	–
	ISWILD	String	No SLB	–
	ITLOCK	RT–11	–	–
	ITTINR	Data Transfer	–	–
	ITTOUR	Data Transfer	–	–
	IWEEKD	Timer	No SLB	–
JADD	JADD	I*4	–	–
JAFIX	JAFIX	I*4	–	–
	JCMP	I*4	–	–
	JDFIX	I*4	–	–
JDIV	JDIV	I*4	–	–
JICVT	JICVT	I*4	–	–
JJCVT		Timer,I*4	–	–
JMOV	JMOV	I*4	–	–
JMUL	JMUL	I*4	–	–
JREAD	JREAD	Data Transfer	No SLB	–
JREADC	JREADC	Data Transfer	No SLB	–

Table 1–3 (Cont.): SYSLIB Subroutines and Functions

Subroutine	Function	Type	Map	Restrictions
JREADF	JREADF	Data Transfer	No I-D, No SLB	–
JREADW	JREADW	Data Transfer	No SLB	–
JSUB	JSUB	I*4	–	–
JTIME		Timer	–	–
JWRITC	JWRITC	Data Transfer	No SLB	–
JWRITE	JWRITE	Data Transfer	No SLB	–
JWRITF	JWRITF	Data Transfer	No I-D, No SLB	–
JWRITW	JWRITW	Data Transfer	No SLB	–
	KPEEK	RT–11	–	–
KPOKE	KPOKE	RT–11	–	–
	LEN	String	–	–
LOCK		RT–11	–	–
LOOKUP		File	–	–
MAP	MAP	Mapping	–	Mapping
MGETR	MGETR	Mapping	No SLB	Mapping
MRCVD	MRCVD	Data Transfer	–	Full mapping, multijob
MRCVDC	MRCVDC	Data Transfer	–	Full mapping, multijob
MRCVDW	MRCVDW	Data Transfer	–	Full mapping, multijob
MREAD	MREAD	Data Transfer	–	Full mapping
MREADC	MREADC	Data Transfer	–	Full mapping
MREADW	MREADW	Data Transfer	–	Full mapping
MRKT	MRKT	Timer	–	Timer.
MSCCA	MSCCA	RT–11	–	Full mapping
MSDAT	MSDAT	Data Transfer	–	Full mapping, multijob
MSDATC	MSDATC	Data Transfer	–	Full mapping, multijob
MSDATW	MSDATW	Data Transfer	–	Full mapping, multijob
MSDS	MSDS	Mapping	–	Full mapping
MSPFN	MSPFN	Data Transfer	–	Full mapping
MSPFNC	MSPFNC	Data Transfer	–	Full mapping
MSPFNW	MSPFNW	Data Transfer	–	Full mapping

Table 1–3 (Cont.): SYSLIB Subroutines and Functions

Subroutine	Function	Type	Map	Restrictions
MTATCH	MTATCH	Multiterm	–	Multiterm
MTDTCH	MTDTCH	Multiterm	–	Multiterm
MTGET	MTGET	Multiterm	–	Multiterm
MTIN	MTIN	Multiterm	–	Multiterm
MTOUT	MTOUT	Multiterm	–	Multiterm
MTPRNT	MTPRNT	Multiterm	–	Multiterm
MTRCTO	MTRCTO	Multiterm	–	Multiterm
MTSET	MTSET	Multiterm	–	Multiterm
MTSTAT	MTSTAT	Multiterm	–	Multiterm
MWAIT		Chan	–	Multijob
MWRITC	MWRITC	Data Transfer	–	Full mapping
MWRITE	MWRITE	Data Transfer	–	Full mapping
MWRITW	MWRITW	Data Transfer	–	Full mapping
POKE	IPOKE	RT–11	–	–
POKEB	IPOKEB	RT–11	–	–
PRINT		Data Transfer	–	–
PROTE	IPROTE	RT–11	–	–
PURGE		Chan	–	–
PUT	IPUT	RT–11	–	–
R50ASC		RAD50	No SLB	–
	RAD50	RAD50	No SLB	–
RANDU	RAN	Math	–	–
RCHAIN		RT–11	–	–
RCTRLO		RT–11	–	–
RCVD	IRCVD	Data Transfer	–	Multijob
RCVDC	IRCVDC	Data Transfer	–	Multijob
RCVDF	IRCVDF	Data Transfer	No I-D, No SLB	–
RCVDW	IRCVDW	Data Transfer	–	Multijob
READ	IREAD	Data Transfer	–	–
READC	IREADC	Data Transfer	–	–

Table 1–3 (Cont.): SYSLIB Subroutines and Functions

Subroutine	Function	Type	Map	Restrictions
READF	IREADF	Data Transfer	No I-D, No SLB	–
READW	IREADW	Data Transfer	–	–
RENAM	IRENAM	File	–	–
REOPN	IREOPN	Chan	–	Multijob
REPEAT		String	–	–
RESUME		RT–11	–	–
SAVES	ISAVES	Chan	–	Multijob
SCCA	ISCCA	RT–11	–	–
SCHED	ISCHED	Timer	No SLB	Timer
SCOMP	ISCOMP	String	–	–
SCOPY		String	–	–
SDAT	ISDAT	Data Transfer	–	Multijob
SDATC	ISDATC	Data Transfer	–	Multijob
SDATF	ISDATF	Data Transfer	No I-D, No SLB	Multijob
SDATW	ISDATW	Data Transfer	–	Multijob
SDTTM	ISDTTM	Timer	–	–
SERR	ISERR	RT–11	–	–
SETCMD		RT–11	–	–
SFDAT	ISFDAT	File	–	–
SFINF	ISFINF	File	No SLB	–
SFSTA	ISFSTA	File	No SLB	–
SLEEP	ISLEEP	Timer	–	Timer
SPCPS	ISPCPS	RT–11	–	SPCPS support
SPFN	ISPFN	Data Transfer	–	–
SPFNC	ISPFNC	Data Transfer	–	–
SPFNF	ISPFNF	Data Transfer	No I-D, No SLB	–
SPFNW	ISPFNW	Data Transfer	–	–
STRPAD		String	–	–
SUBSTR		String	–	–

Table 1–3 (Cont.): SYSLIB Subroutines and Functions

Subroutine	Function	Type	Map	Restrictions
SUSPND		RT–11	–	–
$SYTRP		rt–11	No I-D, No SLB	Macro
TIMASC		Timer	–	–
TIME		Timer	–	–
TIMER	ITIMER	Timer	–	Timer.
TRANSL		String	–	–
TRIM		String	–	–
TWAIT	ITWAIT	Timer	–	Timer.
UNLOCK		RT–11	–	–
UNMAP	IUNMAP	Mapping	–	Mapping
UNPRO	IUNPRO	RT–11	–	–
UNTIL	IUNTIL	Timer	–	Timer.
VERIFY	IVERIF	String	No SLB	–
WAIT	IWAIT	Chan	–	–
WRITC	IWRITC	Data Transfer	–	–
WRITE	IWRITE	Data Transfer	–	–
WRITF	IWRITF	Data Transfer	No I-D, No SLB	–
WRITW	IWRITW	Data Transfer	–	–

SYSLIB descriptions in Chapter 2 present these routines as calls or functions or both, as applicable. Some subroutines and functions have an added argument that specifies mapping. In these cases, the function or subroutine has an M prefix. In the following example, *RCVD* can be called as a function or subroutine, including the M-prefix version having a mapping argument:

```
CALL RCVD (buff,wcnt)
i = IRCVD (buff,wcnt)
CALL MRCVD (buff,wcnt[,BMODE=strg])
i = MRCVD (buff,wcnt[,BMODE=strg])
```

Function Subprograms

A function may return an error code value or other information useful to the purpose of the calling routine. Function subprograms receive control by means of a function reference as follows:

i = function name ([arguments])

Subroutines

Subroutines are invoked by a CALL statement as follows:

CALL subroutine name [(arguments)]

Routines

Routine is the term that describes subroutines called as function subprograms, if a return value is desired, or called as subroutines, if no return value is desired.

Some subroutines have two acceptable formats. For example, you can call the CLOSEC subroutine or specify the ICLOSE function, since the latter returns an integer error code return useful in showing either a normal return or error condition.

Quoted-string literals are useful as arguments of calls to routines in SYSLIB, especially the character string routines. These literals are allowed in subroutine and function calls (see Section 1.8.3).

1.4 FORTRAN/MACRO Interface

FORTRAN calling routines and subroutines follow a well-defined set of)conventions by which MACRO programmers adhering to these conventions can write FORTRAN-callable routines such as those in SYSLIB:

- Transfer of control
- Transfer of information
- Memory usage
- Register usage

Control is transferred to a subroutine by the following assembly language syntax:

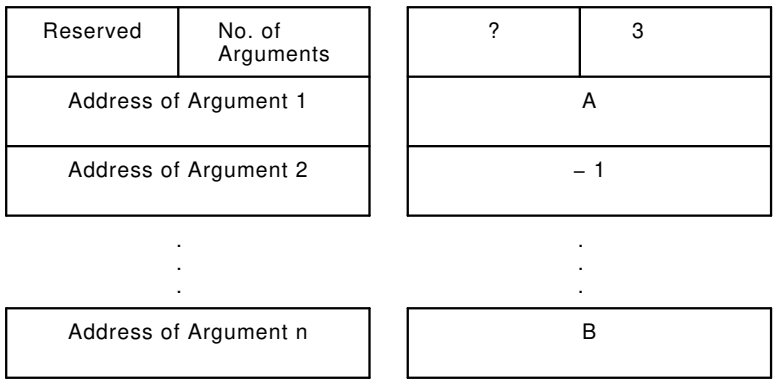
CALL SUBR

When control passes to the subroutine SUBR, R5 points to an argument block like that shown in the left-hand block in Figure 1–2. Null arguments in CALL statements must be entered as comma pairs (, ,). For example, CALL SUBR (A, ,B) . As shown in the right-hand block of Figure 1–2, the value -1 is stored in the argument block as the address of a null argument.

The lower byte of the first word of the argument block contains the number of arguments that are passed to the subroutine. The rest of the argument block contains the addresses of those arguments. The argument block is *n+1* words long for *n* arguments.

The program counter is the linkage register. The subroutine obtains its arguments through R5. In FORTRAN, the calling program saves the registers, and the subroutine leaves the contents of the stack pointer intact before returning to the

Figure 1–2: Subroutine Argument Block



calling program. The RETURN statement of the subroutine is placed by the assembly language instruction *RETURN*.

The name of the subroutine must be declared *global* with the .GLOBL directive in the calling program or with the double colon (::) construction in the called program.

NOTE

Be sure that the called program does not modify the argument block passed by the calling program to a subprogram.

1.4.1 Subroutine Register Usage

A subroutine called by a FORTRAN program does not have to preserve any registers. However, each push onto the stack must be matched by a pop off the stack before exiting from the routine.

User-written assembly language programs must preserve all pertinent registers before calling FORTRAN subroutines or SYSLIB routines, then restore registers after the subroutine returns. The CALL$F routine is provided to perform this register save and restore. See CALL$F description in Chapter 2.

Function subroutines return a single result in a register. Table 1–4 shows the register assignments for returning the different variable types.

NOTE

Floating-point results are returned in the *general purpose registers* and not in the *Floating Point Unit (FPU) registers*. Assembly language subprograms that use the FP11 Floating Point Unit may be required to save and restore the FPU status.

Table 1–4: Return Value Conventions for Function Subroutines

Type	Result Placed In	
INTEGER*2	R0	
LOGICAL*1		
INTEGER*4	R0	low-order result
LOGICAL*4	R1	high-order result
REAL(*4)	R0	high-order result (including sign and exponent)
	R1	low-order result
DOUBLE PRECISION or REAL*8	R0	highest-order result (including sign and exponent)
	R1	next higher order
	R2	next higher order
	R3	lowest-order result
COMPLEX	R0	high-order result
	R1	low-order result
	R2	high-order imaginary result
	R3	low-order imaginary result

1.4.2 FORTRAN Programs Calling MACRO Subroutines

FORTRAN programs can call MACRO subroutines, but several rules must be followed. In the following example, the program FINITA is a MACRO subroutine that can be called from a FORTRAN program:

```
        .TITLE  FINITA
;+
;       Put INIT into the LARRY elements starting at IARRAY
;       IERR = FINITA (IARRAY, INIT, LARRAY)
;       Default INIT to 0
;
;       IERR =   0 success
;               -1 invalid LARRY (negative)
;               -2 missing argument
;-
        .GLOBL  $SYSLB          ;SYSLIB version and value for $NOARG
        .GLOBL  $NXADR, $NXVAL  ;routines to get args
                                ;IN: R0 is default, R4 count
                                ;    R5 current arg list pointer
                                ;OUT:R0 is addr / value
                                ;    R4 decremented count
```

```
                                ;    R5 incremented pointer
                                ;    Carry set if omitted arg
FINITA::
        MOV     (R5)+,R4        ;Put arg count in R4,
                                ; point R5 to first arg
        CALL    $NXADR          ;Get addr of array
        BCS     20$             ;Error, missing arg
        MOV     R0,R1           ;save first arg addr
        CLR     R0              ;use 0 for omitted value
        CALL    $NXVAL          ;Get value to init with
        MOV     R0,R2           ;save it
        CALL    $NXVAL          ;Get words in array
        BCS     20$             ;Error, missing arg
        TST     R0              ;Is the length gt 0
        BLE     30$             ;no, can't init
10$:
        MOV     R2,(R1)+        ;init array
        SOB     R0,10$          ;according to count
        RETURN                  ;done (R0 = 0)
20$:
        MOV     #-2,R0          ;indicate missing args
        RETURN
30$:
        MOV     #-1,R0          ;indicate invalid count
        RETURN
        .END
```

Call the preceding routine as follows:

Macro Call:

CALL FINITA (IAR,IVAL,N)

where:

FINITA	is the name of the subroutine
IAR	is the name of the array to initialize
IVAL	is the initialized value of the array
N	is the number of elements to initialize

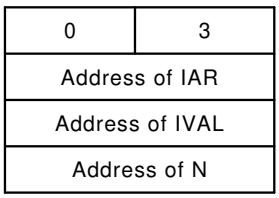
This program illustrates the rules that must be observed when calling a MACRO program. The name of the subroutine is made global by using the .GLOBL directive.

Register 5 (R5) is used to pass the arguments. For the program FINITA, the argument block would appear as shown in Figure 1–3.

Registers R0 through R4 can be freely used because the calling program saves them. When arguments have been retrieved, you can also use R5.

On completion, the subroutine returns to the calling program through a RETURN. If your MACRO program pushes data on the stack, make sure that all data is popped off the stack before the RETURN is executed.

Figure 1–3: Argument Block for Program FINITA



The following FORTRAN program named FINITB calls the subroutine FINITA.

```
        Program FINITB
C
C       FORTRAN program to call MACRO subroutine
C
        Integer*2 Array
        Dimension Array(10)
        Data Array /-1,-2,-3,-4,-5,-6,-7,-8,-9,-10/
C
        N = 7                   !init first seven elements
        Do 10 I = 1, 10         !use 10 init values
            Call FINITA (ARRAY, I, N)
            Write (5, 100) (ARRAY(J), J = 1, 10)
10          Continue
100     Format (' ', 10I4)
        End
```

Compile and link both programs, then run the program by typing:

```
.RUN FINITB RET
```

The initialized array will be output to the terminal as follows:

```
 1   1   1   1   1   1   1   1  -8  -9  -10
 2   2   2   2   2   2   2   2  -8  -9  -10
 3   3   3   3   3   3   3   3  -8  -9  -10
 .
 .
 .
 9   9   9   9   9   9   9   9  -8  -9  -10
10  10  10  10  10  10  10  10  -8  -9  -10
```

1.4.3 MACRO Routines Calling FORTRAN Programs

If you want to call FORTRAN subroutines from a MACRO program, create a dummy main program. For example,

```
        Program FORINT          !setup FTN environment
C
C       MAIN program to call MACRO subroutines which in turn
C       call FORTRAN subroutines.
C
        Call FMAC2F             !call first MACRO routine
        Call FMACSF             !call second MACRO routine
        End
```

where:

FMAC2F is the name of a MACRO program that can call FORTRAN or MACRO routines.

Creating a dummy program causes the FORTRAN main program to perform the initialization necessary for FORTRAN subroutines.

In the following example, MACRO program FMAC2F calls a FORTRAN subroutine named FMAXMN:

```
        .TITLE  FMAC2F          ;MACRO calling FORTRAN
        .GLOBL  FMAXMN          ;FORTRAN program to call
FMAC2F::                        ;entry point
        MOV     #ARGBLK,R5      ;point to argument block
        CALL    FMAXMN          ;call routine
        RETURN                  ;done

ARGBLK: .WORD   2               ;two arguments
        .WORD   I               ;address of first
        .WORD   J               ;address of second

I:      .WORD   28.             ;value of first argument
J:      .WORD   76.             ;value of second argument
        .END
```

First, set up the argument block either on the stack or in a separate area in your MACRO program. Then point R5 to the top of the argument block prior to calling the FORTRAN subroutine with a CALL FMAXMN. In the FMAC2F program shown previously, the argument block is set up in an area of your program.

In the following example, a program named FMACSF performs the same operation as the FMAC2F program, except that it places the argument blockf on the stack:

```
        .TITLE  FMACSF          ;MACRO calling FORTRAN
        .GLOBL  FMAXMN          ;FORTRAN program to call
FMACSF::                        ;entry point
        MOV     #J,-(SP)        ;build argument block on stack
        MOV     #I,-(SP)        ; ...
        MOV     #2,-(SP)        ; ...
        MOV     SP,R5           ;point to it
        CALL    FMAXMN          ;call routine
        ADD     #3*2,SP         ;pop argument block from stack
        RETURN                  ;done

I:      .WORD   28.             ;value of first argument
J:      .WORD   76.             ;value of second argument
        .END
```

If you set up the argument block on the stack, you must remove the arguments from the stack prior to the execution of the RETURN. Before calling the FORTRAN subroutine, you must save all pertinent registers. You do not know which registers the FORTRAN subroutine is using. The stack pointer remains unchanged across the call.

You must define the name of the FORTRAN subroutine that the MACRO program calls as a global. In the FORTRAN subroutine, execute normal FORTRAN statements and return to the MACRO program with a RETURN statement.

The following program is the FORTRAN subroutine FMAXMIN:

```
        Subroutine FMAXMN (IN1, IN2)
C
C       FORTRAN subroutine called by MACRO subroutines
C
        Integer*2 Big, Small
        If (IN1 .gt. IN2) Then
            BIG = IN1
            SMALL = IN2
        Else
            BIG = IN2
            SMALL = IN1
        End If
        Type 10, BIG
        Type 20, SMALL
10      Format (' The bigger number is ', I10)
20      Format (' The smaller number is ', I10)
        Return
        End
```

After assembling and linking the programs, type:

```
.RUN FORINT RET
```

The program executes as follows:

```
The bigger number is 76
The smaller number is 28

The bigger number is 76
The smaller number is 28
STOP --
```

1.5 FORTRAN Programs in a Foreground/Background Environment

FORTRAN programs can be run in a foreground/background environment, which enables the efficient use of CPU execution time. (See Chapter 5 of *Introduction to RT–11* for a description of running programs in an FB environment.)

Before running your foreground program, use the LOAD command to load the device handlers required by the foreground job. These device handlers are placed in memory between RMON and the USR and KMON, causing USR and KMON to move down in memory.

Next, use the FRUN command to load your foreground program in memory between the device handlers and the USR, which causes the USR and KMON to move further down in memory. You must allocate sufficient workspace when running a FORTRAN program in the foreground. Allocate workspace by using the /BUFFER:n option of the FRUN command. Also ensure that any FORTRAN program you run in the foreground has adequate stack space. You can use one of the options supported by the linker (See the *RT–11 System Utilities Manual*).

The background area must be at least 4K words long to accommodate the USR and KMON. Until you run a background job with the RUN command, KMON is the background job.

When the USR is required (in unmapped monitors), you must set up a 2K-word area in each job for the swapping to occur correctly; that is, there must be space for at least 2K words in the background area and 2K words in the foreground area. For an explanation of USR swapping, see Section 1.2.8.

1.5.1 Calculating Workspace for a FORTRAN Foreground Program

Additional workspace must be allocated in memory when running a FORTRAN program in the foreground of a foreground/background environment. For a foreground job, the space is allocated by the /BUFFER:n option of the FRUN command. (A background job uses whatever space is available between its high limit and the system's low limit.) When you allocate additional workspace in memory to run a FORTRAN IV program in the foreground, calculate the space required by using the following formula:

```
n = [[504+(35*N)+(R-136)+A*512]/2]+[10*qcount]+[6*num]+[25*INTSET]+[64+R/2]
```

where:

`A`	Specifies the maximum number of files open at one time. Each file opened as double buffered should be counted as two files.
`N`	Specifies the maximum number of simultaneously open channels (logical unit numbers). This value is specified when the compiler is built and can be overridden with the /UNITS option during main program compilation.
`R`	Specifies the maximum formatted sequential record length. This value is specified when the compiler is built and can be overridden with the /RECORD option during main program compilation; the default value is 136.
`qcount`	Specifies queue elements.
`num`	Specifies the number of channels.
`INTSET`	Specifies the SYSLIB INTSET function.

Include the following optional elements in the formula if you want to use the indicated system subroutine library (SYSLIB) functions:

`[10*qcount]` Specifies space for queue elements, which the IQSET function requires.

`[6*num]` Specifies space for the number of channels, which the ICDFN function requires.

`[25*INTSET]` Specifies space for the number of INTSET calls issued in the program, which the INTSET function requires.

`[64+R/2]` Specifies space for completion routines and a second record buffer. Any functions, including INTSET, that invoke completion routines must include 64_{10} words plus the number of words needed to allocate the second record buffer (default is 68 decimal words).

The length of the record buffer is controlled by the /RECORD option to the FORTRAN compiler. If the /RECORD option is not used, the allocation in the formula must be 136_{10} bytes, or the length that was set at FORTRAN installation time.

Note that the numbers in the formula presented above are all decimal quantities for ease in computation, using a calculator. Remember, however, that in entering a decimal number in the /BUFFER:n option of FRUN, you must include the decimal point in the numeric value of *n*.

1.5.2 Running a FORTRAN Program in a Foreground/Background Environment

This section outlines the procedure for running two FORTRAN programs, one in the background and one in the foreground.

The background program named FBACK is as follows:

```
        Program FBACK   !background demo program
        Parameter JSW = "44             !JSW address
        Parameter TTSPC = "010000       !TT special mode bit
        Call IPOKE (JSW, TTSPC .or. IPEEK (JSW))
100     Continue
        Call Print ('Hello from the background')
        Call ITTOUR (ITTINR())          !echo input char
        Go To 100                       !loop until killed
        End
```

This program prints the message "Hello from the background" and will print the message each time you input a character at the terminal.

The foreground program named FFORE is as follows:

```
        Program FFORE   !foreground demo program
        Parameter JSW = "44             !JSW address
        Parameter TTSPC = "010000       !TT special mode bit
        Call IPOKE (JSW, TTSPC .or. IPEEK (JSW))
100     Continue
        Call Print ('Hello from the foreground')
        Call ITTOUR (ITTINR())          !echo input char
        Go To 100                       !loop until killed
        End
```

After compiling both programs, link them. Link the foreground program using the LINK command with the /FOREGROUND option. This option produces a relocatable load module with a .REL file type. For example,

```
.LINK/FOREGROUND FFORE RET
```

Then you can assign the device that will be used for the output of the foreground program. You must also load into memory the peripheral device handlers needed by the foreground program.

The command FRUN loads and starts execution of a .REL program as the foreground job. At this point, typing the command:

```
.FRUN FFORE RET
```

causes the following error message to display, indicating that additional workspace allocation is required and that the /BUFFER option must be used. (Refer to the previous section for the formula to calculate the additional space needed.)

```
?Err 62 FORTRAN start fail
```

The command should be typed as follows:

```
.FRUN FFORE/BUFFER:2000 RET
```

Execution of this command results in the following output at the terminal:

```
F>
Hello from the foreground
B>
.
```

The system first identifies the message as foreground output, then the foreground job executes and outputs its message. The background monitor next prints the characters B> and a period (.), indicating that control has returned to monitor command mode. Command input remains directed to the background job.

For example, when you type:

```
.RUN FBACK RET
```

the background job will display the following message:

```
Hello from the background
```

Each time you type a character to the terminal, say an "L", the message will be repeated, as follows:

```
LHello from the background
```

Use the CTRL/F command to direct terminal input to the foreground job. The system prints F> to remind you that you are now directing input to the foreground job. When you type a character, such as "Y", the foreground job message will be displayed.

```
F>
YHello from the foreground
```

Type a CTRL/B to return to the background job or a CTRL/C to return to monitor command mode. If you are returning to a background environment, you should unload the foreground job and any handlers to reclaim memory space for background use. To stop these two example programs, enter CTRL/C to each one.

1.6 Linking with FORLIB

Normally, default system library file SYSLIB.OBJ also includes the overlay handlers and the appropriate FORTRAN run-time system routines.

To add FORLIB.OBJ modules to the default library SYSLIB.OBJ, use the following command:

```
.LIBRARY/INSERT/REMOVE SYSLIB FORLIB RET
Global? $ERRS RET
Global? $ERRTB RET
Global? $OVRH RET
Global? RET
```

1.7 SYSLIB Services Not Provided by Programmed Requests

SYSLIB provides many services that are not handled by single programmed requests, for example:

- Time conversion and date access
- Program suspension
- Two-word integer support (INTEGER*4)
- Radix–50 conversion
- Character string manipulation
- Control of global regions in extended memory

1.7.1 Time Conversion and Date Access

Use the following calls to perform time conversions:

CVTTIM Converts a two-word internal format time to hours, minutes, seconds, and ticks.

JTIME Converts a time given in hours, minutes, seconds, and ticks into the internal two-word time format.

Use the following calls to print out the time:

TIMASC Converts the time in internal two-word format into an eight-character ASCII string.

TIME Returns the current time of day as an eight-character ASCII string.

Access the current system date by issuing the DATE/IDATE call:

where:

DATE Returns date as a string value.

DATE4Y Returns the date as a string value with a four-digit year value.

IDATE Returns date as an integer value.

1.7.2 Program Suspension

You can suspend execution of a program with ITWAIT, ISLEEP, and IUNTIL calls, where:

ITWAIT Suspends program execution for a specified number of ticks.

ISLEEP Suspends running program for a specified number of hours, minutes, seconds and ticks.

IUNTIL Suspends job execution until a specific time of day, in hours, minutes, seconds, and ticks.

1.7.3 Two-Word Integer Support (INTEGER*4)

This support is primarily for FORTRAN IV. It also can be used from MACRO, but FORTRAN–77 has INTEGER*4 functionality built in. You can make calls to SYSLIB to manipulate a 32-bit integer that uses two words of storage. The first word contains the low-order part of the value and the second word contains the sign and the high-order part of the value. The range of numbers that is represented is -2^{31} to $2^{31}-1$. This format differs from the two-word internal time format that stores the high-order part of the value in the first word and the low-order part in the second word. Table 1–5 shows the calls you use to convert from one format to another.

Table 1–5: SYSLIB Conversion Calls

From	To	Call
INTEGER*4	REAL*4	AJFLT/IAJFLT
INTEGER*4	REAL*8	DJFLT/IDJFLT
INTEGER*4	INTEGER*2	IJCVT
REAL*4	INTEGER*2	JAFIX
REAL*8	INTEGER*4	JDFIX
INTEGER*2	INTEGER*4	JICVT

INTEGER*2 are 16-bit integers; INTEGER*4 are 32-bit integers; REAL*4 are 2-word, single-precision floating-point numbers; REAL*8 are 4-word, double-precision floating-point numbers.

Calls are also available for you to perform arithmetic operations on INTEGER*4 values, move a value to a variable, and convert a two-word internal time format to and from an INTEGER*4 value.

1.7.4 Radix–50 Conversion

You can convert ASCII characters to or from Radix–50, using RAD50, IRAD50, and R50ASC,

where:

IRAD50 Converts a specified number of characters of Radix–50 and returns the number of characters converted as a function result.

RAD50 Encodes RT–11 file descriptors in Radix–50 notation.

R50ASC Converts a specified number of Radix–50 characters to ASCII.

1.7.5 Character String Operations

SYSLIB provides character string functions that perform string operations such as:

- Concatenating strings
- Comparing strings
- Copying strings
- Replacing strings
- Computing the number of characters in a string

The following example is a program that demonstrates calling a SYSLIB character-string subroutine from a macro program:

```
        .TITLE  FEGT2;2         ;calling SYSLIB from MACRO
        .GLOBL  CONCAT  CALL$F  ;SYSLIB routines
        .MCALL  .PRINT  .EXIT   ;macros
START::
        MOV     #STRCON,R2      ;Point to final buffer
        MOV     #ARGBLK,R5      ;Point to argument block
        MOV     #CONCAT,R0      ;Point to routine to call
        CALL    CALL$F          ;Call it, saving registers R1-R4
        .PRINT  R2              ;Print resulting string
        .EXIT                   ;and away

ARGBLK: .WORD   3               ;3 arguments
        .WORD   STRNG1          ;first input string address
        .WORD   STRNG2          ;second input string address
        .WORD   STRCON          ;output string buffer address

STRNG1: .ASCIZ  "Research and"

STRNG2: .ASCIZ  " Development"

STRCON: .BLKB   31

        .END    START
```

Running this program concatenates string 1 and string 2, producing the following terminal output:

```
Research and Development
```

For detailed description of character string functions, see Section 1.8.

1.7.6 Control of Global Regions

Global regions are areas in extended memory which can be used independently from the program that created them; that is, they are shareable among programs. Use the IGETR/MGETR and IFREER system subroutines to create, attach to, detach from, and eliminate global regions in extended memory. All facilities for global region control are available, using the MGETR and IFREER subroutines:

- MGETR creates or attaches to a global region. (IGETR obsolete; retain for compatibility.)
- IFREER detaches from or eliminates a global region.

For a complete description of global region support, see the *RT–11 System Internals Manual*.

1.8 Character String Functions

SYSLIB character string functions and routines provide variable-length string support for RT–11 FORTRAN and for MACRO programs. SYSLIB calls that perform character string operations are listed in Table 1–6.

Table 1–6: Character String Functions

Call	Operation
CONCAT	Concatenates variable-length strings
INDEX	Returns the position of one string in another
INSERT	Inserts one string into another
LEN	Returns the length of a string
REPEAT	Repeats a character string
SCOMP	Compares two strings
SCOPY	Copies a character string
STRPAD	Pads a string with blanks on the right
SUBSTR	Copies a substring from a string
TRANSL	Performs character modification
TRIM	Removes trailing blanks
VERIFY	Verifies the presence of characters in a string

String Storage

Strings are stored in BYTE or LOGICAL*1 arrays that you define and dimension. These arrays store strings in ASCII format as one character per array element, plus a zero element to indicate the current end of the string.

ASCII Code

The ASCII code used in this string package is the same as that employed by FORTRAN for A-type FORMAT items, ENCODE/DECODE strings, and object-time format strings. Whenever quoted strings are used as arguments in the CALL statement, ASCIZ strings are generated for these routines by the FORTRAN compiler. Note that a null string (a string containing no characters) can be represented in FORTRAN by a variable or constant of any type that contains the value zero or by a LOGICAL variable or constant with the .FALSE. value.

String Length

The length of a string can vary at execution time from zero characters to one less than the size of the array that stores the string. The maximum size of any string is 32767 characters.

Strings can contain any of the seven-bit ASCII characters except null(0), since the null character is used to mark the end of the string. The inclusion of a terminating zero byte constitutes an *ASCIZ format*, the format set up by a MACRO assembler directive .ASCIZ. This directive automatically sets up strings with a terminating zero byte. Bit 7 of each character must be cleared; therefore, valid characters have a decimal representation range from 1 to 127, inclusive.

In many routines, it is difficult to predict the length of the string produced. To prevent a string from overflowing the array that contains it, you can specify an optional integer argument to the subroutine. This argument, called *len* limits the length of an output string to the value specified for *len plus one* (for the null terminator), so that the array receiving the result must be at least len-plus-one elements in size.

NOTE

If the string is larger than the array, and you do not specify a correct *len* argument, other data may be destroyed and cause unpredictable results.

When *len* is specified, you can also include optional argument *err*. *Err* is a logical variable that should be initialized by the FORTRAN program to .FALSE. If a string function is given the arguments *len* and *err*, and *len* is actually used to limit the length of the string result, then *err* is set to the .TRUE. value. If *len* is not used to truncate the string, *err* is unchanged; that is, it retains a .FALSE. value.

The argument *len* can appear alone; however, *len* must appear if *err* is specified.

Several routines use the concept of character position in which each character in a string is assigned a *position number*, where the first character in the string is at position one.

1.8.1 Allocating Character String Variables

A one-dimensional BYTE array can contain a single string whose length can vary from zero characters to one fewer than the dimensioned length of the array. In the following example, array A is used as a string variable that can contain a string of 44 or fewer characters.

```
Byte A(45)              !allocate 1 string
```

Similarly, a two-dimensional BYTE array can be used to contain a one-dimensional array of strings, each of which can have a length up to one less than the first dimension of the BYTE array. There can be as many strings as the number specified for the second dimension of the BYTE array. The program in the following example creates string array W that has ten string elements, each of which can contain up to 20 characters. String I in array W is referenced in subroutine or function calls as W(1,I).

```
Byte W(21,10)   !Allocate an array of 10 strings
```

In the following example, the program allocates a two-dimensional string array.

```
Byte T(14,5,7)          !Allocate a 5 by 7 array of
                        !13 character strings
```

Each string in array T may vary in length to a maximum of 13 characters. String I,J of the array can be referenced as T(1,I,J). Note that T is the same as T(1,1,1). This dimensioning process can create string arrays of up to six dimensions (represented by BYTE arrays of up to seven dimensions).

1.8.2 Passing Strings to Subprograms

BYTE arrays that contain strings can be placed in a COMMON block and referenced by any or all routines with a similar common declaration. However, when you place a BYTE array in a common block, make sure that one of the following is true:

- Array is even in length
- Odd-length arrays are paired to result in an overall even length.
- Strings are together as the last elements in the COMMON block; otherwise, all succeeding variables in the COMMON block may be assigned odd addresses.

A BYTE array has an odd length only if the product of its dimensions is odd. For example,

```
Byte B(10,7)    !10*7 = 70, even length
Byte H(21)              !21, odd length
```

These might be handled as shown in the following example:

```
Common A1, A2, A3(10), H        !odd size at end
```

or

```
Byte HPAD               !odd length
Common  A1, A2, H, HPAD, A3(10) !Pair H and HPAD for even
```

These restrictions apply only to BYTE variables and arrays.

A single string can be passed by using its array name as an argument. In the following example, the program passes string R to subroutine SUBR.

```
Byte R(21)      !20 char string variable
Call    Subr (R)
```

If the calling program has declared a multidimensional array, and only one string of that array is to be passed to a subroutine, then the subroutine call should specify the first element of the string to be passed (this requires that the first dimension of the array equals the maximum length of each string).

For example,

```
        Byte NAMES(81,20)       !20 names max 80 chars each
        Do 10 NAMNUM=1, 20      !get 20 lines of input
            Call GTLIN (NAMES(1, NAMNUM)) !from terminal/command file
10      Continue
```

If the maximum length of a string argument is unknown in a subroutine or function, or if the routine is used to handle many different lengths, the dummy argument in the routine should be declared as a BYTE array with a dimension of one, such as BYTE ARG(1). In this case, the string routines correctly determine the length of ARG whenever it is used, but it is not possible to determine the maximum size of any string that can be stored in ARG. If a multidimensional array of strings is passed to a routine, it must be declared in the called program with the same dimensions that were specified in the calling program.

NOTE

The length argument specified in many of the character string functions refers to the maximum length of the string, excluding the necessary null byte terminator. The length of the BYTE array to receive the string must be at least one greater than the length argument.

1.8.3 Using Quoted-String Literals

You can use quoted strings as input arguments to any of the string routines invoked as functions or with the CALL statement. In the following example, the program compares the string in the array NAME to the constant string SMYTHE, R and sets the value of the integer variable accordingly.

```
        Call SCOMP (NAME, 'SMYTHE, R', M)
```

Chapter 2
System Subroutine Description and Examples

This chapter presents all SYSLIB functions and subroutines in alphabetical order by generic name. For example, because READ, IREAD, and MREAD can be called either as subroutines or functions, presenting them together under *READC will simplify lookup. An I-prefixed name indicates its use as a function; an M-prefixed name indicates that mapping is specified.

Each description briefly defines the subroutine or function; gives its argument list, and defines each parameter and argument contained in the argument list. Function results and errors are listed for each, as appropriate. Descriptions include specific examples for each function or subroutine or refer you to examples elsewhere in the chapter.

ABTIO/IABTIO

ABTIO/IABTIO aborts I/O on a specified channel.

Form:

```
CALL ABTIO (chan)
 i = IABTIO (chan)
```

where:

chan is the channel number for which to abort I/O

Errors:

Value	Meaning
i = 0	Success.

Error message *TRAP $MSARG* will display if *chan* argument is missing.

Example:

```
       Program FABTIO  !demo ABTIO
C
C       Pump out 9 buffers to LP using non-wait mode I/O.
C       Abort the I/O.  This should cause the printout to
C       be truncated.
C
C       NOTE: using LS7 as a trick to bypass SPOOLING, since
C       spooling is normally applied to LP, LP0, LS and LS0.
C
        Integer*2 DBLK(4)
        Data DBLK /3rLS7, 3rTES, 3rTXX, 3rTMP/ !LS7 to sneak around SP
        Integer*2 BUFFER(256,9), CHARS, CRLF
C
        CHARS = '11'                    !begin at 1
        CRLF = '015'o + ('012'o * '400'o) ! CR / LF pair in word
        ICHAN = IGETC ()
        Call IQSET (20)                 !get more queue elements
        Call ENTER (ICHAN, DBLK, 0)
        Do 300, J = 1, 9
            Do 200, I = 1, 256
                BUFFER(I,J) = CHARS
                If (IMOD (I, 60 * 2) .eq. 0) BUFFER(I,J) = CRLF
 200            Continue
            CHARS = CHARS + '400'o + 1  !then 2 ... 9
            Call WRITE (256, BUFFER(1,J), J - 1, ICHAN)
300         Continue
C
C       Comment out the call to ABTIO and observe the difference
C
        Call ABTIO (ICHAN)              !stop it in midstream
        Call CLOSEC (ICHAN)
        End
```

AJFLT/IAJFLT

AJFLT/IAJFLT converts an INTEGER*4 value to a REAL*4 value and returns that result as the function value.

Form:

ares = AJFLT (jsrc)
i = IAJFLT (jsrc,ares)

where:

jsrc is the INTEGER*4 variable to be converted

ares is the REAL*4 variable or array element to receive the converted value

Function Results:

i = -1 Normal return; result is negative.

= 0 Normal return; result is 0.

= 1 Normal return; result is positive.

Errors:

Value	Meaning
i = -2	Significant digits were lost during the conversion.

Unpredictable results will occur if the *jsrc* argument is omitted.

Example:

```
       Program FIAJFL !FORTRAN IV
        Integer*4 JVAL,    J5
        Integer*2 IVAL(2), I5(2)
        Equivalence (IVAL(1), JVAL), (I5(1), J5)
        Real*4 RESULT
C
        IVAL(1) = 123   !initial value
        IVAL(2) = 1     !really 65536+123 (65659)
        I5(1) = 5       !constant value
        I5(2) = 0       ! ...
100     Continue
            IERR = IAJFLT (JVAL, RESULT)
            Type 101, RESULT
101         Format (' ', '!FIAJFL-I-Results', f16.0)
            IERR = JMUL (JVAL, J5, JVAL)
            If (IERR .eq. -2) Go To 200
            Go To 100
200     Continue
        Type 102
102     Format (' ', '!FIAJFL-I-Overflow')
        End
```

The following example converts the INTEGER*4 value in JVAL to single precision (REAL*4), multiplies it by 3.5, and stores the result in VALUE:

```
Real*4 VALUE, AJFLT, THREE5
Data THREE5 / 3.5/
Integer*4 JVAL
JVAL = 123456789
VALUE = AJFLT (JVAL) * THREE5
```

CALL$F

CALL$F can be called only from a MACRO–11 program.

The CALL$F routine saves the contents of general registers R1 through R4 across a call to another routine that might destroy the contents of those registers. CALL$F saves the contents of R1 through R4 on the stack, calls the other routine, and then restores the saved register contents.

Form:

```
MOV     #rtn,R0
MOV     #arg,R5
CALL    CALL$F
```

where:

rtn is the address of the routine you want to call. The current contents of registers 1 through 4 are preserved during execution of the called routine

arg is the starting address of the argument list for the routine you want to call

Errors:
None. Any errors are returned by the routine called by CALL$F.

Example:

```
        .GLOBL  GTLIN, CALL$F   ;routines from SYSLIB

         .PSECT  CODE,I          ;program code fragment

         MOV     #GTLIN,R0       ;routine to call (ultimately)
         MOV     #PARMS,R5       ;argument list to use
         CALL    CALL$F          ;call GTLIN, save R1-R4

         .PSECT  DATA,D          ;program data fragment

PARMS:   .WORD   2               ;number of arguments
         .WORD   BUF             ;response buffer
         .WORD   PROMPT          ;prompt string

BUF:     .BLKB   81.             ;buffer
PROMPT:  .ASCII  "Enter username>" ;prompt
         .BYTE   200             ;without terminating cr/lf
```

CHAIN

The CHAIN subroutine lets a background program transfer control directly to another background program and pass specified information to it. CHAIN cannot be called from a completion or interrupt routine. The FORTRAN impure area is not preserved across a chain. Therefore, when chaining from one program to another, the information must be reset in the program being chained to. When chaining to any other program, you should explicitly close the opened logical units with calls to the CLOSE routine. Any routines specified in a FORTRAN USEREX library call are not executed if a CHAIN is accomplished. (See Appendix B in the *RT–11/RSTS/E FORTRAN IV User's Guide.*

Form:

CALL CHAIN (dblk,var,wcnt)

where:

dblk is the address of a four-word Radix–50 descriptor of the file specification for the program to be run (See device block discussion in Chapter 1) for the format of the file specification

var is the first variable (which must start on a word boundary) in a sequence of variables with increasing memory addresses to be passed between programs in the chain parameter area (absolute locations 510 to 777). A single array or a COMMON block (or portion of a COMMON block) is a suitable sequence of variables

wcnt is a word count specifying the number of words (beginning at var) to be passed to the called program. The argument wcnt may not exceed 60. If no words are passed, then a word count of 0 must be supplied.

If the size of the chain parameter area is insufficient, it can be increased by specifying the /B (or /BOTTOM) option to LINK for both the program executing the CHAIN call and the program receiving control.

The data passed can be accessed through a call to the RCHAIN routine. For more information on chaining to other programs, see the .CHAIN programmed request.

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

Example:

The following example transfers control from the main program to FRCHAI.SAV on BIN and passes it variables (See also RCHAIN for FRCHAI.FOR):

```
        Program FCHAIN  !demonstrate CHAIN routine
C
C       Chain to  BIN:FRCHAI.SAV passing 100,200,301
C
        Integer*2 DBLK(4)
        Data DBLK /3rBIN, 3rFRC, 3rHAI, 3rSAV/
        COMMON /CHAIND/ I, J, K         !force order
        Data I /100/, J /200/, K / 301/ !initialize
C
        Call CHAIN (DBLK, I, 3)
        End
```

CHCPY/ICHCPY

Multijob Only

CHCPY/ICHCPY opens a channel for input, logically connecting it to a file that is currently open by another job for either input or output. CHCPY/ICHCPY is used in a multijob situation to gain shared access to a file already opened by another job. It substitutes for an OPEN (or LOOKUP or ENTER) in your program, instead obtaining the parameters of the file from the other job. An ICHCPY must be done before the first read or write on *ochan*.

Form:

```
CALL CHCPY (chan,ochan[,jobblk])
 i = ICHCPY (chan,ochan[,jobblk])
```

where:

chan	is the channel the job will use to read the data. You must obtain this channel through an IGETC call, or you can use channel 16 or higher if you have done an ICDFN call
ochan	is the channel number of the other job that is to be copied
jobblk	is a pointer to a three-word ASCII job name

Notes

- If the other job's channel was opened with an IENTER function or a .ENTER programmed request to create a file, your channel indicates a file that extends to the highest block that the creator of the file had written at the time the CHCPY was executed.
- A channel that is open on a sequential-access device should not be copied, because requests can become intermixed.
- Your program can write on a copied channel to a file that is being created by the other job, just as your program could if it were the creator. When your channel is closed, however, no directory update takes place.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Specified job does not exist. Or specified channel (*ochan*) open.
= 2	Channel (*chan*) is already open.

Error message *TRAP $MSARG* will display if *chan* or *ochan* argument is missing.

Example:

```
       Program FCHCPB !BG program for CHCPY
        Parameter SUCCS = '001'o
        Parameter FATAL = '010'o
        Integer*2 BUFFER(513)
        Integer*2 FCHCPF(3)
        Byte CHCPF(6)
        Data CHCPF /'F', 'C', 'H', 'C', 'P', 'F'/
        Equivalence (FCHCPF(1), CHCPF(1))
C
        BUFFER(513) = 0         !null just in case
        ICHAN = 14
        JCHAN = 15
        IERR = ICHCPY (ICHAN, JCHAN, FCHCPF)
        If (IERR .ne. 0) Go To 100
        IERR = IREADW (512, BUFFER, 0, ICHAN)
        If (IERR .ne. 512) Go To 200
        Call PRINT (BUFFER)
        Call Exit (SUCCS)
C
100     Type *, '?FCHCPB-F-ICHCPY failed with code ', IERR
        Call Exit (FATAL)
200     Type *, '?FCHCPB-F-IREADW failed with code ', IERR
        Call Exit (FATAL)
        End

        Program FCHCPF !FG program for ICHCPY
        Integer*2 DBLK(4)
        Data DBLK /3rSRC, 3rFCH, 3rCPB, 3rFOR/
C
        ICHAN = 15
        IERR = LOOKUP (ICHAN, DBLK)
        If (IERR .lt. 0) Go To 100
        Call SUSPND     !sleep (forever)
C
100     Type *, '?FCHCPF-F-LOOKUP failed with code ', IERR
        End
```

CLOSEC/ICLOSE

The CLOSEC subroutine terminates activity on the specified channel and frees it for use in another operation.

Form:

```
CALL CLOSEC (chan[,i])
 CALL ICLOSE (chan[,i])
 i = ICLOSE(chan)
```

where:

chan	is the channel number to be closed. This argument must be located so that the USR cannot swap over it
i	is the error returned if a protection violation occurs

Notes

Under certain conditions, a handler for the associated device and USR must be available when issuing a .CLOSE for a channel opened with a .ENTER or .LOOKUP:

- .CLOSE requires a handler and USR, if it is:
  - — A special directory device (magtape).
  - — An RT–11 standard directory device, and the file was opened with a .ENTER.
- All other RT–11 operations do not require either handler or USR.

A CLOSEC or PURGE must eventually be issued for any channel opened for input or output. A CLOSEC call specifying a channel that is not open is ignored.

A CLOSEC performed on a file that was opened via an IENTER causes the device directory to be updated to make that file permanent. If the device associated with the specified channel already contains a file with the same name and type, the old copy is deleted when the new file is made permanent. If the file name is protected, then a protection error is generated and two files will exist with the same name. A CLOSEC on a file opened via LOOKUP does not require any directory operations.

When an entered file is closed, its permanent length reflects the highest block of the file written since the file was entered; for example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area allocated at IENTER time, the unused blocks are reclaimed as an empty area on the device.

Use ICLOSZ, rather than CLOSEC or ICLOSE, to set file size at closure. ICLOSZ has no effect on the file size when the file was opened by a LOOKUP. (See CLOSZ /ICLOSZ.)

Errors:

Value	Meaning
i = 0	Normal return.
= -4	A protected file with the same name already exists on a device. The CLOSEC is performed, resulting in two files on the device with the same name.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

The following example creates a file which becomes a 0-block permanent file:

```
        Program FCLOSE  !demo CLOSEC / ICLOSE
C
C       Create and close DK:TEST.TMP w/o I/O.
C       Note that this makes a permanent file of 0 length.
C       Compare with FCLOSZ.
C
        Integer*2 DBLK(4)
        Data DBLK /3rDK , 3rTES, 3rT  , 3rTMP/
        Parameter SUCCS = '001'o, FATAL = '010'o
C
        ICHAN = IGETC()
        If (ICHAN .lt. 0) Go To 100
        IERR = IENTER (ICHAN, DBLK, 100)
        IF (IERR .lt. 0) Go To (110, 120, 130) IABS(IERR)
        CALL CLOSEC (ICHAN, IERR)
        If (IERR .eq. -4) Go To 200
        Call IFREEC(ICHAN)
        Call Exit (SUCCS)
100     Type *, ' ?FCLOSE-F-No channel available'
        Call Exit (FATAL)
110     Type *, ' ?FCLOSE-F-Channel in use'
        Call Exit (FATAL)
120     Type *, ' ?FCLOSE-F-Not enough room'
        Call Exit (FATAL)
130     Type *, ' ?FCLOSE-F-Device in use'
        Call Exit (FATAL)
200     Type *, ' ?FCLOSE-F-Protected file already exists'
        Call Exit (FATAL)
        End
```

CLOSZ/ICLOSZ

CLOSZ/ICLOSZ terminates activity on the specified channel and frees it for use in another operation. ICLOSZ closes any file opened on that channel by an IENTER and sets the file size to a value you specify. Use ICLOSZ, as opposed to CLOSEC or ICLOSE, when you want to set the file size at closure. ICLOSZ has no effect on the file size when the file was opened by a LOOKUP. See also CLOSEC/ICLOSE.

Form:

```
CALL CLOSZ (chan,size)
 i = ICLOSZ (chan,size)
```

where:

chan	is the INTEGER*2 channel number to be closed. If the specified channel is not open, no action is taken
size	is the size of the file in blocks at closure
i	is a returned INTEGER*2 result of the function

If the handler for the device associated with the channel is marked FILST$ (supports the RT–11 file structure) and the file is opened with IENTER, the value you specify for the file size at closure must be equal to or less than the current allocated file size. If the handler is marked SPECL$ (supports the special directory file structure), RT–11 enforces no size constraints when the file is closed. However, the handler may impose constraints.

The handler for the device associated with the channel must be in memory if the channel was opened with the IENTER subroutine or if the handler is marked SPECL$ (supports the special directory structure).

An ICLOSZ performed on a file that was opened with IENTER causes the device directory to be updated to make that file permanent. If the device associated with the specified channel already contains a file with the same name and type, the old copy is deleted when the new file is made permanent. If the file name is protected, then a protection error is generated. An ICLOSZ on a file opened using a LOOKUP does not require the USR for RT–11 directory devices, but does require the USR for special directory devices.

Errors:

Value	Meaning
i = 0	Normal return
= -2	Size too big
= -3	Invalid operation
= -4	A protected file with the same name already exists on the device.
= -257	Required argument missing

Example:

```
       Program FCLOSZ  !demo  ICLOSZ
C
C       Create DK.TEST.TMP with 100 blocks, and
C       make permanent at that size w/o I/O.
C
        Integer*2 DBLK(4)
        Data DBLK /3rDK , 3rTES, 3rT  , 3rTMP/
        Parameter SUCCS = '001'o
        Parameter FATAL = '010'o
C
        ICHAN = IGETC()
        If (ICHAN .lt. 0) Go To 100
        IERR = IENTER (ICHAN, DBLK, 100)
        IF (IERR .lt. 0) Go To (110, 120, 130) IABS(IERR)
        IERR = ICLOSZ (ICHAN, IERR)
        If (IERR .eq. -4) Go To 200
        Call IFREEC(ICHAN)
        Call Exit (SUCCS)
100     Type *, ' ?FCLOSZ-F-No channel available'
        Call Exit (FATAL)
110     Type *, ' ?FCLOSZ-F-Channel in use'
        Call Exit (FATAL)
120     Type *, ' ?FCLOSZ-F-Not enough room'
        Call Exit (FATAL)
130     Type *, ' ?FCLOSZ-F-Device in use'
        Call Exit (FATAL)
200     Type *, ' ?FCLOSZ-F-Protected file already exists'
        Call Exit (FATAL)
        End
```

CMAP/ICMAP

Full Mapping

CMAP/ICMAP, available only under fully mapped monitors, is a routine that controls mapping for Supervisor mode and I-D space. CMAP establishes CMAP status in Supervisor data space, distinct from User data space.

Form:

```
CALL CMAP (ival [,iold][,ierr])
 ierr = ICMAP (ival [,iold])
```

where:

ierr	Error return
ival	New value for CMAP status
iold	Previous CMAP status

Errors:

Value	Meaning
0	Success.
-257	Required argument (*ival*) argument is omitted.

Example:

```
C       CMPDF.FOR FORTRAN equivalent of .CMPDF
        Parameter CMPR0 = '1'o          !unlock PAR0 S-U
        Parameter CMPR1 = '2'o          !unlock PAR1 S-U
        Parameter CMPR2 = '4'o          !unlock PAR2 S-U
        Parameter CMPR3 = '10'o         !unlock PAR3 S-U
        Parameter CMPR4 = '20'o         !unlock PAR4 S-U
        Parameter CMPR5 = '40'o         !unlock PAR5 S-U
        Parameter CMPR6 = '100'o        !unlock PAR6 S-U
        Parameter CMPR7 = '200'o        !unlock PAR7 S-U
        Parameter CMPAR = '377'o        !PAR locking mask
        Parameter CMSXX = '1000'o       !Change Supy I-D
        Parameter CMSII = '1000'o       !Supy I = D
        Parameter CMSID = '1400'o       !Supy i ne D
        Parameter CMS   = '1400'o       !Supy ID mask
        Parameter CMDUS = '4000'o       !Change PAR locking
        Parameter CMXXS = '20000'o      !Change Supy swapping
        Parameter CMNOS = '20000'o      !Don't swap Supy
        Parameter CMJAS = '30000'o      !Swap Supy
        Parameter CMSUP = '30000'o      !Supy swapping mask
        Parameter CMUXX = '100000'o     !Change User I-D
        Parameter CMUII = '100000'o     !User I = D
        Parameter CMUID = '140000'o     !User I ne D
        Parameter CMU   = '140000'o     !User ID mask
```

```
        Program FCMAP
C
C       Set User D PARs to an unlikely value and then
C       turn on User separated I and D, which should set
C       the User D PARs to match the User I PARs.
C       Verify that this happens.
C
C       Perform the same sort of test on MSDS, separating
C       PARs 4 through 7
C
        Include 'SRC:CMPDF'
        Parameter UDPAR0 = '177660'o, UIPAR0 = '177640'o
        Parameter SDPAR0 = '172260'o
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 REQUES        !request code for CMAP
C
        Do 100, I = UDPAR0, UDPAR0+(7*2), 2
100         Call KPOKE (I, -1)  !set User D PARs to unlikely value
C
        REQUES = CMUID+CMSID+CMSUP
C       request separate U I-D spaces; separate S I-D spaces;
C       and turn on Supy
        Call CMAP (REQUES)      !separate I and D
C
        Do 200, I = SDPAR0, SDPAR0+(7*2), 2
200         Call KPOKE (I, -2)  !set Supy D PARs to unlikely value
C
        REQUES = CMPR7+CMPR6+CMPR5+CMPR4
C       and not to lock S and U D PARs 4 through 7
        Call MSDS (REQUES)      !separate PARs 4 through 7
C                               !This should also copy some D PARs U to S
C
        Do 300, I = UDPAR0, UDPAR0+(7*2), 2
            If (KPEEK (I) .ne. KPEEK (I + (UIPAR0 - UDPAR0)))
        1       Go To 1000      !do they now match?
300         Continue
        Do 400, I = SDPAR0, SDPAR0+(3*2), 2
            If (KPEEK (I) .ne. KPEEK (I + (UDPAR0 - SDPAR0)))
        1       Go To 1100      !do they now match?
400         Continue
        Do 500, I = SDPAR0+(4*2), SDPAR0+(7*2), 2
            If (KPEEK (I) .ne. -2)
        1       Go To 1200      !do they now match?
500         Continue
        Type *,'!FCMAP-I-Success'
        Call Exit (SUCCS)
C
1000    Type *,'?FCMAP-F-U(I,D)PAR values are not the same'
        Call Exit (FATAL)
1100    Type *,'?FCMAP-F-(S,U)DPAR values are not the same'
        Call Exit (FATAL)
1200    Type *,'?FCMAP-F-SDPAR values changes'
        Call Exit (FATAL)
        End
```

CMKT/ICMKT

CMKT/ICMKT cancels one or more scheduling requests (made by an ISCHED, ITIMER, or MRKT routine). Support for CMKT in SB requires that you select timer support during SYSGEN.

Form:

```
CALL CMKT (id[,itime])
i = ICMKT (id[,itime])
```

where:

id is the identification integer of the request to be canceled. If *id* is equal to 0, all scheduling requests are canceled

itime is the name of a two-word area in which the monitor returns the amount of time remaining in the canceled request

For further information on canceling scheduling requests, see the .CMKT programmed request in the *RT–11 System Macro Library Manual*.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	The value of *id* was not equal to 0 and no scheduling request with that identification could be found.

Error message *TRAP $MSARG* will display if *id* argument is missing.

Example:
See MRKT.

CNTXS/ICNTXS

Multijob Only

CNTXS/ICNTXS establishes a list of locations to be saved when a transition is made from one running job to another. CNTXS preserves locations that are not preserved automatically by the monitor. Refer to the *RT–11 System Macro Library Manual*.

Form:

```
CALL CNTXS (addr)
 i = ICNTXS (addr)
```

where:

addr is the pointer to the list of addresses to be preserved. The list is terminated with a zero word

i is a returned INTEGER*2 result of the function.

Errors:

Value	Meaning
i = 0	Normal return
= -1	One or more of the conditions specified by *addr* was violated
= -257	Required argument missing.

Example:

```
        Program FCNTXS !demo ICNTXS
C
C        Setup swapping of vectors 400 through 406
C
         Parameter SUCCS = '001'o, FATAL = '010'o
         Integer*2 LIST(5), NOLIST
         Data LIST /'400'o, '402'o, '404'o, '406'o, 0/
         Equivalence (NOLIST, LIST(5))
C
         IERR = ICNTXS (LIST)    !swap 400--406
         If (IERR .ne. 0) Go To 1000
C
C        ...
C
         Call CNTXS (NOLIST)     !stop swapping
         Call PRINT ('!FCNTXS-I-Success')
         Call Exit (SUCCS)
C
1000     Call PRINT ('?FCNTXS-F-Failed')
         Call Exit (FATAL)
         End
```

CONCAT

The CONCAT subroutine concatenates two character strings.

Form:

CALL CONCAT (a,b,out[,len[,err]])

where:

a is the array containing the left string. The string must be terminated with a null byte

b is the array containing the right string. The string must be terminated with a null byte

out is the array into which the concatenated result is placed. This array must be at least one element longer than the maximum length of the resultant string (that is, one greater than the value of *len*, if specified)

len is the integer number of characters representing the maximum length of the output string. The effect of *len* is to truncate the output string to a given length, if necessary

err is the logical error flag set if the output string is truncated to the length specified by *len*.

You must specify *err* as LOGICAL*1 in FORTRAN 77. It can be any logical type in FORTRAN IV and any integer type in PDP–11C.

CONCAT sets the string in *out* to the value of the string in *a*, immediately followed by the string in *b*, followed by a terminating null character.

NOTE

Any combination of string arguments is allowed, so long as *b* and *out* do not specify the same array.

Concatenation stops when a null character is detected in *b* or when the number of characters specified by *len* has been moved.

If either the left or right string is a null string, the other string is copied to *out*. If both are null strings, then *out* is set to a null string. The old contents of *out* are lost when this routine is called.

Errors:
Error conditions are indicated by *err*, if specified. If *err* is given and the output string would have been longer than *len* characters, then *err* is set to .TRUE.; otherwise, *err* is unchanged.

Error message *TRAP $MSARG* will display if argument *b* or *out* is missing.

Example:
The following example concatenates the string in array STR and the string in array IN and stores the resultant string in array OUT. OUT cannot hold a string longer than 29 characters:

```
        Program FCONCA !demo CONCAT
C
C       Show concatination and truncation
C
        Byte IN(22), OUT(30), STR(10)
C
        Call SCopy ('abcdefghijklmnopqrstu', IN)
        Call SCopy ('123456789', STR)
        Call CONCAT (STR, IN, OUT, 29)
        Call Print (OUT)
        End
```

CRAW/ICRAW

Mapping

CRAW/ICRAW are memory mapping routines which create an address window into an existing memory region. It can be used in both User and Supervisor modes, and can access both I and D space. See also .CRAW in the *RT–11 System Macro Library Manual*.

Form:

```
CALL CRAW (iwdb [,ierr])
 i = ICRAW (iwdb)
```

where:

iwdb	Address of Window Descriptor Block
ierr	Error return

Errors:

Value	Meaning
i = 0	Function completed successfully.
= -1	Window alignment error.
= -2	Attempt to define more than 7 windows.
= -3	Invalid region identifier.
= -5	Combination of offset into region and size of window to be mapped is invalid.
= -16	Mode/space not available.
= -257	Required argument missing.

Example:

```
C       RDBDF.FOR -- FORTRAN equivalent of .RDBDF
        Parameter RGID = 0              !region id subscript
        Parameter RGSIZ = 2/2           !region size subscript
        Parameter RGSTS = 4/2           !region status subscript
        Parameter RGLLN = 6/2           !local RDB size
        Parameter RGNAM = 6/2           !global region name words subscript
        Parameter RGBAS = 10/2          !global region base addr subscript
        Parameter RGLGH = 12/2          !global RDB size
        Parameter RSCRR = '100000'o     !Created region ok
        Parameter RSUNM = '40000'o      !1 or more windows eliminated
        Parameter RSNAL = '20000'o      !region newly allocated
        Parameter RSNEW = '10000'o      !new global region
        Parameter RSGBL = '4000'o       !create within a global region
        Parameter RSCGR = '2000'o       !create global if not found
        Parameter RSAGE = '1000'o       !use auto global elimination
        Parameter RSEGR = '400'o        !eliminate global region
        Parameter RSEXI = '200'o        !eliminate on exit or abort
        Parameter RSCAC = '100'o        !bypass cache
        Parameter RSBAS = '40'o         !base address supplied
        Parameter RSNSM = '20'o         !non-system memory
```

```
C       WDBDF.FOR -- FORTRAN equivalent of .WDBDF
        Parameter WNID = 0           !window ID subscript (low byte)
        Parameter WNAPR = 0          !window APR number subscript (high byte)
        Parameter WNBAS = 2/2        !window base address subscript
        Parameter WNSIZ = 4/2        !window size subscript
        Parameter WNRID = 6/2        !region ID subscript
        Parameter WNOFF = 8/2        !window offset subscript
        Parameter WNLEN = 10/2       !window length subscript
        Parameter WNSTS = 12/2       !window status subscript
        Parameter WNLGH = 14/2       !WDB size
        Parameter WSCRW = '100000'o  !window created ok
        Parameter WSUNM = '40000'o   !1 or more windows unmapped
        Parameter WSELW = '20000'o   !1 or more windows eliminated
        Parameter WSDSI = '10000'o   !D-space inactive
        Parameter WSIDD = '4000'o    !I & D spaces different
        Parameter WSRO =  '1000'o    !read-only
        Parameter WSMAP = '400'o     !create and map
        Parameter WSSPA = '14'o      !space field
        Parameter WSD = '10'o        !D-space
        Parameter WSI = 4            !I-space
C                                    !0 is default space
        Parameter WSMOD = 3          !mode field
        Parameter WSU = 0            !user mode
        Parameter WSS = 1            !supervisor mode
        Parameter WSC = 2            !current mode

        Program FPLAS !demo PLAS requests
C
C       This program has two behaviors depending on wether or not
C       the global region TSTREG exists.
C       If it does not exist, it creates it, get a line from the
C       terminal and stores it in the region.
C       If it does exist, it prints the line stored in the region
C       then eliminates the region.
C       In both cases it displays the mapping context of the region.
C
        Include 'SRC:RDBDF'     !RDB definitions
        Include 'SRC:WDBDF'     !WDB definitions
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter BASADR = '160000'o !PAR 7 for gbl region
        Parameter REGSIZ = (81 + 63) / 64 !size in chunks
        Integer*2 REGNAM(0:1)   !global region name
        Data REGNAM /3rTST, 3rREG/
        Integer*2 WDB (0:WNLGH) !WDB block
        Integer*2 RDB (0:RGLGH) !RDB block
        Character*7 ERRCAL      !code for error call
        Integer*2 AREA(0:1)     !must disable subscript checking
        Integer*2 AREA0         !addr of AREA(0)
        Integer PAR7            !subscript for AREA
C
C       Find a way of referencing address 160000
C
        AREA0 = IADDR (AREA(0)) !find addr of AREA(1)
        PAR7 = ('160000'o - AREA0) / 2 !find "element" of AREA
C                               !that is at 160000
C
C       Create (or attach) the global regions
```

```
C
        RDB(RGSIZ) = REGSIZ     !region size
        RDB(RGSTS) = RSGBL + RSCGR !create or attach global
        RDB(RGNAM+0) = REGNAM(0)!region name
        RDB(RGNAM+1) = REGNAM(1)!region name
        IERR = ICRRG (RDB)      !can we find it?
        ERRCAL = 'CRRG'
        If (IERR .ne. 0) Go To 1000 !error
C
C       Create an address window
C
        WDB(WNAPR) = (BASADR / '20000'o) * '400'o
C                               !Put PAR number in high byte
        WDB(WNSIZ) = REGSIZ     !region size
        WDB(WNRID) = RDB(RGID)  !region ID
        WDB(WNOFF) = 0          !offset 0
        WDB(WNLEN) = 0          !full size
        WDB(WNSTS) = 0          !take all defaults
        IERR = ICRAW (WDB)      !create a window
        ERRCAL = 'CRAW'
        If (IERR .ne. 0) Go To 1000 !error
C
C       Map to it (could be done by ICRAW)
C
        IERR = MAP (WDB)        !map into it
        ERRCAL = 'MAP'
        If (IERR .ne. 0) Go To 1000 !error
C
C       display mapping context
C
        IERR = IGMCX (WDB)      !return mapping context
        ERRCAL = 'GMCX'
        If (IERR .ne. 0) Go To 1000 !error
        Type 1, '    Window ID = ', IAND (WDB(WNID), '377'o),
        1       '   Window APR = ', WDB(WNAPR) / '400'o,
        2       '  Window Addr = ', WDB(WNBAS),
        3       '  Window Size = ', WDB(WNSIZ),
        4       '  Window RgID = ', WDB(WNRID),
        5       'Window Offset = ', WDB(WNOFF),
        6       'Window Length = ', WDB(WNLEN),
        7       'Window Status = ', WDB(WNSTS)
1       Format (8(' ', a16, o7/))
C
C       Decide if this is the first or second run
C
        If (IAND (RDB(RGSTS), RSNEW) .eq. 0) Go To 100
C       else first pass
C
C       Collect a line, put it in the global region and exit
C       leaving the region for the next run of this program
C
        Call GTLIN (AREA(PAR7),,'p') !get a string and put in region
        Call Print ('!FPLAS-I-Pass 1 success')
        Call Exit (SUCCS)
C
C       second pass
C
```

```
C       Get the line from the region and display it,
C       then eliminate the region.
C
100     Continue
        Call PRINT (AREA(PAR7)) !print the string in the region
C
C       Unmap the window
C
        IERR = IUNMAP (WDB)
        ERRCAL = 'UNMAP'
        If (IERR .ne. 0) Go To 1000 !error
C
C       Delete the window
C
        IERR = IELAW (WDB)
        ERRCAL = 'ELAW'
        If (IERR .ne. 0) Go To 1000 !error
C
C       Eliminate the region
C
        RDB(RGSTS) = RSEGR      !eliminate region
        IERR = IELRG (RDB)
        ERRCAL = 'ELRG'
        If (IERR .ne. 0) Go To 1000 !error
        Call Print ('!FPLAS-I-Pass 2 success')
        Call Exit (SUCCS)
C
C       Error processing
C
1000    Continue
        Type *, '?FPLAS-F-', ERRCAL, 'Failed with code', IERR
        Call Exit (FATAL)
        End
```

CRRG/ICRRG

Mapping

CRRG/ICRRG (Create Region) is a mapping routine for Supervisor mode, I-D space. The function allocates or attaches to a region in physical memory for use by the requesting job. See .CRRG and .RDBDF macros in the *RT–11 System Macro Library Manual*.

Form:

```
CALL CRRG (irdb [,ierr])
ierr = ICRRG (irdb)
```

where:

ierr	Error return
irdb	Address of Region Descriptor Block

Errors:

Value	Meaning
i = 0	Function completed successfully.
= -7	No region control blocks available.
= -10	Insufficient memory to allocate region of requested size.
= -11	Invalid region size was specified.
= -13	Global region not found.
= -14	Too many global regions.
= -16	Global region privately owned.
= -17	Global region already exists with different base address.
= -257	Required argument missing.

Example: See CRAW.

CSI/ICSI

CSI/ICSI calls the RT–11 Command String Interpreter in special mode to parse a command string and return file descriptors and options to the program. In this mode, the CSI does not perform any handler IFETCH, CLOSEC, IENTER, or LOOKUP. This subroutine requires the USR.

Form:

```
CALL CSI (filspc,deftyp[,cstring][,option],n)
 i = ICSI (filspc,deftyp[,cstring][,option],n)
```

where:

filspc is the 39-word area to receive the file specifications. The format of this area (considered as a 39-element INTEGER*2 array) is:

Word 1–4	output file number specification
Word 5	output file number 1 length
Word 6–9	output file number 2 specification
Word 10	output file number 2 length
Word 11–14	output file number 3 specification
Word 15	output file number 3 length
Word 16–19	input file number 1 specification
Word 20–23	input file number 2 specification
Word 24–27	input file number 3 specification
Word 28–31	input file number 4 specification
Word 32–35	input file number 5 specification
Word 36–39	input file number 6 specification

deftyp is the table of Radix–50 default file types to be assumed when a file is specified without a file type:

deftyp(1)	is the default for all input file types
deftyp(2)	is the default file type for output file number 1
deftyp(3)	is the default file type for output file number 2
deftyp(4)	is the default file type for output file number 3

cstring is the area that contains the ASCIZ command string to be interpreted; the string must end in a zero byte. If the argument is omitted, the system prints the prompt character (*) at the terminal and accepts a command string. If input is from a command file, the next line of that file is used

option is the name of an INTEGER*2 array dimensioned (4,x) where x represents the number of options defined to the program. This argument must be present if the value specified for *n* is non-zero. This array has the following format of the jth option described by the array:

option(1, j)	is the one-character ASCII name of the option
option(2, j)	is set by the routine to 0, if the option did not occur; to 1, if the option occurred without a value; to 2, if the option occurred with a value
option(3, j)	is set to the file number on which the option is specified
option(4, j)	is set to the specified value if option(2, j) is equal to 2

n is the number of options defined in the array option. The *n* is not optional. If the *n* is omitted, specify *n* as 0.

Notes

The array option must be set up to contain the names of the valid options. For example, use the following to set up names for five options:

```
INTEGER*2 SW(4,5)
DATA SW(1,1)/'S'/,SW(1,2)/'M'/,SW(1,3)/'I'/
DATA SW(1,4)/'L'/,SW(1,5)/'E'/
```

Multiple occurrences of the same option are supported by allocating an entry in the option array for each occurrence of the option. Each time the option occurs in the option array, the next unused entry for the named option is used. When there are identical options, they are placed in the *option array* in reverse order. The last occurrence of *option* in the command line is placed in the first matching entry in *option*. You may want to consider putting both upper and lower case versions of the options in the table, because options might be entered either way.

The arguments of ICSI must be positioned so that the USR cannot swap over them. For more information on calling the Command String Interpreter, see the .CSISPC programmed request.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Invalid command line passed by *cstring* in memory; no data was returned.
= 2	Invalid device specification occurred in the command string *cstring* in memory.

= 3 Invalid option specified; specified option exceeded number allowed in the option array.

Error message *TRAP $MSARG* will display if *filpc, deftyp,* or *n* argument is missing.

Example:

The following example causes the program to loop until a valid command is typed at the terminal:

```
        Program FCSI !demo CSI
C
C       Accept a command line, parse it into file specifications
C       and switches.  Display the results of this parsing.
C       Note that input files use a trick to allow selection of
C       a default type dynamically.  The value of the /I switch is
C       used as the default type for all input files.
C       Switches that are accepted are /S/M/I/L/E.
C
        Parameter OPTNUM = 5    !number of options allowed
C                               !option 1st subscript
        Parameter OPTS = 1, OPTM = 2, OPTI = 3
        Parameter OPTL = 4, OPTE = 5
C                               !names for option subscripts
        Parameter OPTNAM = 1, OPTTYP = 2, OPTFIL = 3, OPTVAL = 4
C                               !values for OPTTYP
        Parameter OPTNON = 0, OPTNOV = 1, OPTJAV = 2
C                               !offset for file name parts
        Parameter DEV = 0, NAME = 1, TYPE = 3, SIZE = 4
C                               !subscripts for FILSPC
        Parameter OUT1 = 1, OUT2 = 6, OUT3 = 11
        Parameter IN1 = 16, IN2 = 20, IN3 = 24
        Parameter IN4 = 28, IN5 = 32, IN6 = 36
        Integer*2 FILSPC(39)    !parsed file specifications
        Integer*2 DEFTYP(4)     !default file types
        Data DEFTYP /-1, 3rOBJ, 3rLST, 3rTMP/
        Integer*2 OPTION(4,OPTNUM) !option array
        Data OPTION(OPTNAM,OPTS) /'S'/, OPTION(OPTNAM,OPTM) /'M'/
        Data OPTION(OPTNAM,OPTI) /'I'/, OPTION(OPTNAM,OPTL) /'L'/
        Data OPTION(OPTNAM,OPTE) /'E'/
        Character*3 ASCDEV, ASCTYP
        Character*6 ASCFIL
C
50      Call CSI (FILSPC, DEFTYP, , OPTION, OPTNUM)
C
C       Display output files
C
        Do 100 I = 1, 3
         J = (I - 1) * (OUT2 - OUT1) + OUT1
         If (FILSPC(J+DEV) .ne. 0) Then
           Call R50ASC (3, FILSPC(J+DEV), ASCDEV)
           Call R50ASC (6, FILSPC(J+NAME), ASCFIL)
           Call R50ASC (3, FILSPC(J+TYPE), ASCTYP)
           Type 1, 'OUT', I, ASCDEV, ASCFIL, ASCTYP, FILSPC(J+SIZE)
1          Format (' ', a3, i1, ' ', a3, ':', a6, '.', a3, ' [', i5, ']')
```

```
            End If
100     Continue
C
C       Display input files
C
        Do 200 I = 1, 6
            J = (I - 1) * (IN2 - IN1) + IN1
            If (FILSPC(J+DEV) .ne. 0) Then
                Call R50ASC (3, FILSPC(J+DEV), ASCDEV)
                Call R50ASC (6, FILSPC(J+NAME), ASCFIL)
                If (FILSPC(J+TYPE) .eq. -1) !use /I value for default
     1              FILSPC(J+TYPE) = OPTION(OPTVAL,OPTI)
                Call R50ASC (3, FILSPC(J+TYPE), ASCTYP)
                Type 2, ' IN', I, ASCDEV, ASCFIL, ASCTYP
2               Format (' ', a3, i1, ' ',  a3, ':', a6, '.', a3)
            End If
200     Continue
C
C       Display options
C
        Do 300 I = 1, OPTE
            If (OPTION(OPTTYP,I) .ne. OPTNON) Then
                If (OPTION(OPTTYP,I) .eq. OPTNOV) Then
                    Type 3, OPTION(OPTNAM,I), OPTION(OPTFIL,I)
3                   Format (' ', '/', a1, ' on file ', i2)
                Else
                    Type 4, OPTION(OPTNAM,I), OPTION(OPTVAL,I),
     1                  OPTION(OPTFIL,I)
4                   Format (' ', '/', a1, ':', o6, ' on file ', i2)
                End If
            End If
300     Continue
        Go To 50
        End
```

CSTAT/ICSTAT

CSTAT/ICSTAT obtains information about a channel.

Form:

```
CALL CSTAT (chan,addr[,strng])
 i = ICSTAT (chan,addr[,strng])
```

where:

chan is the channel whose status is desired

addr is a six-word area to receive the status information. The area, as a six-element INTEGER*2 array, has the following format:

Word 1	channel status word
Word 2	starting absolute block number of file on this channel
Word 3	length of file
Word 4	highest block number written since file was opened
Word 5	unit number of device with which this channel is associated
Word 6	Radix–50 of device name with which the channel is associated

strng is the 3-character area to receive the ASCII device name and unit number associated with the specified channel.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Channel specified is not open.

Error message *TRAP $MSARG* will display if *chan* or *addr* argument is missing.

Example:

The following example obtains channel status information about channel I.

```
        Program FCSTAT !demo CSTAT
C
C       This program opens a file on a FORTRAN unit and then
C       uses CSTAT to find out about the associated RT-11 channel.
C
        Parameter CSW = 1, BGNBLK = 2, LENGTH = 3
        Parameter HIH2O = 4, UNIT = 5, R50DEV = 6
        Integer*2 REPLY(6)      !reply area for CSTAT
        Character*3 NAME        !device name
C
        Open (Dispose='SAVE', File='SY:SWAP.SYS', Readonly,
        1 Status='OLD', Unit=1)
```

```
        Open (Dispose='DELETE', File='DK:TEST.TMP',
        1 Status='NEW', Unit=2)
        Do 200, I = 0, 15
            If (ICSTAT (I, REPLY, NAME) .eq. 0)
        1       Type 1, REPLY(BGNBLK), REPLY(LENGTH), NAME, I
1               Format (' ', ' Beginning block=', I6,
        1                   ' File length=' , I6,
        2                   ' Device=' A3,
        3                   ' Channel=' I2)
200         Continue
        End
```

CVTTIM

The CVTTIM subroutine converts a two-word internal format time to hours, minutes, seconds, and ticks.

Form:

CALL CVTTIM (time,hrs,min,sec,tick)

where:

time is the two-word internal format time to be converted. If time is considered as a two-element INTEGER*2 array, then:

time (1) is the high-order time

time (2) is the low-order time

hrs is the integer number of hours

min is the integer number of minutes

sec is the integer number of seconds

tick is the integer number of ticks (1/60 of a second for 60-Hz clocks; 1/50 of a second for 50-Hz clocks)

Errors:
Error message *TRAP $MSARG* will display if any required argument is missing.

Example:

```
        Program FCVTTI  !demo CVTTIM
C
C       Get current time of day and display appropriate greeting
C
        Integer*4 TIME
C
        Call GTIM (TIME)                !Get current time
        Call CVTTIM (TIME, IHRS, IJUNK, IJUNK, IJUNK) !"parse" it
        If ((IHRS .ge. 0) .and. (IHRS .lt. 8))   Type *, 'Good Grief'
        If ((IHRS .ge. 8) .and.(IHRS .lt. 12))   Type *, 'Good Morning'
        If ((IHRS .ge. 12) .and. (IHRS .lt. 17)) Type *, 'Good Afternoon'
        If ((IHRS .ge. 17) .and. (IHRS .lt. 22)) Type *, 'Good Evening'
        If ((IHRS .ge. 22) .and. (IHRS .lt. 24)) Type *, 'Good Night'
        End
```

DATE/DATE4Y

The DATE and DATE4Y subroutines display current (system) date or format a date you specify.

Previously, the DATE subroutine was located in the distributed FORTRAN subroutine libraries, FORLIB and F77OTS.

DATE stores the date as a 9-byte string as **dd-mmm-yy**. DATE4Y stores the date as an 11-byte string as **dd-mmm-yyyy**.

where:

dd	is the 2-digit day of the month (with leading zero if necessary)
mmm	is the 3-character month (all capital letters)
yy	is the last two digits of the year (DATE subroutine)
yyyy	is the 4-digit year (DATE4Y subroutine)
–	is the separating character

Form:

```
CALL DATE (array[,opdate])
CALL DATE4Y (array[,opdate])
```

where:

array is a predefined array for receiving the date string.

- For DATE, the array must contain at least nine bytes. The 9-byte string is set to blanks if the date is invalid.
- For DATE4Y, the array must contain at least eleven bytes. The 11-byte string is set to blanks if the date is invalid.

opdate is an optional RT–11 date word to be formatted. Specifying a 0 value for *opdate* returns the current system date.

The format of the date word is:

Bits	Contents
0-4	Year minus the base (base specified by bits 14,15)
5-9	Day (1–31)
10-13	Month (1–12)
14,15	Age bits. Age bits extend the directory date by 32(decimal) year increments and have the following meaning:

15	14	Meaning When Set
0	0	Base year 1972
0	1	Base year 2004
1	0	Base year 2036
1	1	Base year 2068

DATE4Y is included within DATE. DATE4Y is functionally the same as DATE, except that it stores a 4-character year rather than a 2-character year.

Errors:
Error message *TRAP $MSARG* will display if *array* argument is missing.

Example:
For DATE4Y, see GTDIR/IGTDIR.
For DATE:

```
       Program FDATE   !demo DATE & IWEEK
C
C       Display current date and day of week
C
        Byte DAYSTR(9)
        Integer*4 DAYS(7)
        Data Days /'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Dat' /
C
        Call IDATE (MONTH, IDAY, IYEAR)
        IWD = IWEEKD (MONTH, IDAY, IYEAR)
        Call DATE (DAYSTR)
        Type 100, DAYS(IWD), DAYSTR
100     Format (' ', 'Today''s date: ', 6x, a4, 1x, 9a1)
        End
```

DELET/IDELET

DELET/IDELET deletes a named file from an indicated device. DELET requires the USR. It is not supported for magtape handlers supplied by Digital.

Form:

```
CALL DELET (chan,dblk[,seqnum])
 i = IDELET (chan,dblk[,seqnum])
```

where:

chan is the channel to be used for the delete operation. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call

dblk is the four-word Radix–50 specification (dev:filnam.typ) for the file to be deleted

seqnum is the file number for magtape operations

Notes

The arguments of DELET must be located so that the USR cannot swap over them.

The specified channel is left inactive when the DELET is complete. DELET requires that the handler to be used be resident (via an IFETCH call or a LOAD command from KMON) at the time the DELET is issued. If the handler is not resident, a monitor error occurs.

For further information on deleting files, see the .DELETE programmed request.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Channel specified is already open.
= 2	File specified was not found.
= 3	Device in use.
= 4	The file is protected and cannot be deleted.

Error message *TRAP $MSARG* will display if *chan* or *dblk* argument is missing.

Example:

```
       Program FDELET !Demo DELET
C
C       Delete the file DK:TSTDEL.TMP
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLK(4)
        Data DBLK /3rDK , 3rTST, 3rDEL, 3rTMP/
C
        If (IDELET (IGETC (), DBLK) .ne. 0) Then
            Call Print ('?IDELET-F-Delete failed')
            Call Exit (FATAL)
        Else
            Call Print ('!IDELET-I-File deleted')
            Call Exit (SUCCS)
        End If
        End
```

DEVICE/IDEVICE

DEVICE/IDEVICE sets up a list of addresses to be loaded with specified values when the program is terminated. If a job terminates or is aborted with a CTRL/C from the terminal, this list is used up by the system to set the specified addresses to the corresponding values.

This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates.

Unless *link arg1* is used, only one address list can be active at any given time. If multiple DEVICE calls are issued, only the last one has any effect. The list must not be modified by the program after the DEVICE call has been issued, and the list must not be located in an overlay or an area over which the USR swaps.

The second argument of the call *link* provides support for a linked list of tables. The link argument is optional and causes the first word of the list to be processed as the link word. With linked lists, each call adds the new list to the previous lists, rather than replacing the previous lists.

Form:

```
CALL DEVICE (ilist[,link])
 i = IDEVICE (ilist[,link])
```

where:

ilist is an integer array that contains two-word elements, each composed of a one-word address and a one-word value to be put at that address, terminated by a zero word. On program termination, each value is moved to the corresponding address.

link is an optional parameter of any value that indicates a linked list table is to be used.

If the linked list form is used, the first word of the array is the link list pointer.

For more information on loading values into device registers, see the .DEVICE programmed request.

Errors:
Error message *TRAP $MSARG* will display if the *ilist* argument is missing.

Errors:

Value	Meaning
i = -1	Value specified for *ilist* is above 160000 (octal)

Example:

```
        Integer*2 IDR11(3)       !DEVICE argument list
         Data IDR11 /"167770, 0, 0/ !addr, value, end of list
C
         Call    DEVICE (IDR11)  !setup for job abort
```

DJFLT

The DJFLT function converts an INTEGER*4 value into a REAL*8 (DOUBLE PRECISION) value and returns that result as the function value. See IDJFLT.

Form:

d = DJFLT(jsrc)

where:

jsrc specifies the INTEGER*4 variable to be converted

NOTES

If DJFLT is used, it must be defined in the FORTRAN program, either explicitly (REAL*8 DJFLT) or implicitly (IMPLICIT REAL*8 (D)). Without a definition, DJFLT is assumed to be REAL*4 (single precision).

The function result is the REAL*8 value that is the result of the operation.

Errors:
Unpredictable results will occur if the *jsrc* argument is omitted.

Example:

```
       Program FDJFLT !FORTRAN IV
        Real*8 VALUE, DJFLT, THREE5
        Data THREE5 / 3.5d0/
        Integer*4 JVAL
        Integer*2 IVAL(2)
        Equivalence (IVAL(1), JVAL)
C
        IVAL(1) = 2     !00002
        IVAL(2) = 1     !65536
        VALUE = DJFLT (JVAL)
        VALUE = VALUE * THREE5
        Type 101, VALUE
101     Format (' ', f16.0)
        End
```

DSTAT/IDSTAT

DSTAT/IDSTAT obtains information about a particular device.

Form:

```
CALL DSTAT (devnam,cblk)
 i = IDSTAT (devnam,cblk)
```

where:

devnam is the Radix–50 device name

cblk is the four-word area used to store the status information. The area, as a four-element INTEGER*2 array, has the following format:

Word 1	Device status word (See .DSTAT)
Word 2	Size of handler in bytes
Word 3	Entry point of handler (non-zero implies that the handler is in memory)
Word 4	Size of the device (in 256-word blocks) for block-replaceable devices; zero for sequential-access devices, the smallest-sized volume for variable-sized devices. The last block on the device is the device size -1

Notes

The arguments of IDSTAT must be positioned so that the USR cannot swap over them.

IDSTAT looks for the device specified by *devnam* and, if found, returns four words of status in *cblk*.

Errors:
Error message *TRAP $MSARG* will display if any required argument is missing.

Value	Meaning
i = 0	Normal return.
= 1	Device not found in monitor tables.

Example:

The following example determines whether the line printer handler is in memory. If it is not, the program stops and prints a message to indicate that the handler must be loaded:

```
        Program FDSTAT !demo DSTAT
        Integer*2 DEVNAM
        Data DEVNAM /3rLP /
        Integer*2 REPLY(4)
        Data REPLY /4*0/
C
        Call DSTAT (DEVNAM, REPLY)
        If (REPLY(3) .eq. 0) Then
            Call Print ('!FDSTAT-I-LP is not in memory')
        Else
            Call Print ('!FDSTAT-I-LP is in memory')
        End If
        End
```

ELAW/IELAW

Mapping

ELAW/IELAW (eliminate window) eliminates a virtual address window. An implied unmapping of the window occurs when its definition block is eliminated.

Form:

```
CALL ELAW (iwdb [,ierr])
 i = IELAW (iwdb)
```

where:

iwdb	is the address of Window Definition Block (WDB)
ierr	is address of location to return error information

Errors:

Value	Meaning
i = 0	Normal return.
= -4	Invalid window identifier specified.
= -257	Required argument missing.

Example:
See CRAW.

ELRG/IELRG

Mapping

ELRG/IELRG (eliminate region) eliminates a dynamic region of physical memory and returns the memory to the free list where it can be used by other jobs.

Form:

```
CALL ELRG (irdb [,ierr])
ierr = IELRG (irdb)
```

where:

ierr	Error return
irdb	Address of Region Definition Block (RDB)

Errors:

Value	Meaning
i = 0	Normal return.
= -3	Invalid region identifier specified.
= -12	Deallocation failure.
= -257	Required argument missing.

Example:
See CRAW.

ENTER/IENTER

ENTER/IENTER allocates space on the specified device and creates a tentative directory entry for the named file. If a file of the same name already exists on the specified device, it is not deleted until the tentative entry is made permanent by issuing either CLOSEC/ICLOSE or CLOSZ/ICLOSZ. The file is attached to the channel number specified. This routine requires the USR.

Form:

```
CALL ENTER (chan,dblk,length[,seqnum])
 i = IENTER (chan,dblk,length[,seqnum])
```

where:

chan is the integer specification for the RT–11 channel to be associated with the file. You must obtain this channel through an IGETC call, or you can use channel 16 or higher, if you have done an ICDFN call.

dblk is the four-word Radix–50 descriptor of the file to be operated upon.

length is the integer number of blocks to be allocated for the file. If 0, the larger of either one-half the largest empty segment or the entire second largest empty segment is allocated. If the value specified for length is -1, the entire largest empty segment is allocated (See the .ENTER programmed request).

seqnum is a magtape file sequence number that can have the values listed below. (*Seqnum* is also a file number for cassette.) If this argument is blank, a value of 0 is assumed.

Value	Meaning
-2	Rewind the magtape and space forward until the file name is found, or until logical-end-of-tape is detected. The magtape is now positioned correctly. A new logical-end-of-tape is implied.
-1	Space to the logical-end-of-tape and enter file.
0	Rewind the magtape and space forward until the file name is found or the logical-end-of-tape is detected. If the file name is found, an error is generated. If the file name is not found, then enter file.
n	Position magtape at file sequence number n if n is greater than zero and the file name is not null.

Notes

- ENTER requires that the appropriate device handler be in memory.
- The arguments of ENTER must be positioned so that the USR does not swap over them.

For further information on creating tentative directory entries, see the .ENTER programmed request.

Errors:

Value	Meaning
i = n	Normal return; number of blocks actually allocated (n = 0 for non-file-structured IENTER).
= -1	Channel (*chan*) is already in use.
= -2	In a fixed-length request, no space greater than or equal to length was found.
= -3	Device in use.
= -4	A file by that name already exists and is protected.
= -5	File sequence number not found.
= -6	File sequence number is invalid.
= -7	Invalid unit number on a special directory device.

Error message *TRAP $MSARG* will display if *chan, dblk* or *length* argument is missing.

Example:

The following example allocates a channel for file TEMP.TMP on SY0. If no channel is available, the program prints a message and halts:

```
        Program FENTER ! demo ENTER
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLK(4)
        Data DBLK /3rDK , 3rTEM, 3rT  , 3rTMP/
C
        ICHAN = IGETC ()
C
C       Create temp work file
C
        If (IENTER (ICHAN, DBLK, 20) .ne. 20) Then
            Call Print ('?FENTER-F-ENTER failed')
            Call Exit (FATAL)
        End If
C
C           use temp file
C
C           ...
C
        Call PURGE (ICHAN)
        Call IFREEC (ICHAN)
        Call Print ('!FENTER-I-ENTER ok')
        Call Exit (SUCCS)
        End
```

FPROT/IFPROT

FPROT/IFPROT sets or removes file protection for a file.

Form:

```
CALL FPROT (chan,dblk[,prot])
 i = IFPROT (chan,dblk[,prot])
```

where:

chan	is the channel number to be used for the protect operation. You must obtain this channel through an IGETC call, or you can use the channel 16(decimal) or higher if you have done an ICDFN call
dblk	is the four-word Radix–50 descriptor of the file to be operated on
prot	1 = protect the file 0 = remove protection from the file

Errors:

Value	Meaning
i = 0	Normal return
= 1	Channel is in use
= 2	File not found or not a file-structured device. To identify which condition returned the error code, issue an IDSTAT to determine if a device is file structured.
= 3	Invalid operation.
= 4	Invalid *prot* value.

Error message *TRAP $MSARG* will display if any required argument is missing.

Example:

This example protects the file SY:RT11FB.SYS against deletion:

```
        Program FFPROT !demo FPROT
C
C       protect SY:RT11FB.SYS
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLK(4)
        Data DBLK /3rSY , 3rRT1, 3r1FB, 3rSYS/
C
        If (IFPROT (IGETC (), DBLK, 1) .ne. 0) Then
            Call Print ('?FFPROT-F-FPROT failed')
            Call Exit (FATAL)
        Else
            Call Print ('!FFPROT-I-Protected: SY:RT11FB.SYS')
            Call Exit (SUCCS)
        End If
        End
```

FREER/IFREER

Mapping

FREER/IFREER detaches from a specified global region that you have attached to using the IGETR/MGETR subroutine. FREER can also eliminate that global region when you specify the type argument. FREER does not eliminate a global region that is attached to another job, but does detach the calling job from that global region.

Form:

```
CALL FREER (work[,<type>])
 i = IFREER (work[,<type>])
```

where:

work is a 7-word work area block. Work area specified in FREER must be the same as the IGETR work area. The first five words of the work area block contain information from the region definition block (RDB):

- A unique region identification (R.GID)
- The size of the region (R.GSIZ)
- The region status word (R.GSTS)
- The region name in two RAD50 words (R.NAME and R.NAME+2)

The last two words in the work block area are reserved by RT–11.

<type> is 'e' for *eliminate*. If you do not specify the *type* argument, you detach but do not eliminate the global region.

Errors:

Value	Meaning
i = 0	Normal return.
= -10	Memory too fragmented to return at .ELRG (-.ELRG)
= -11	Global region not found (-.ELRG)
= -18	Any .ELRG error except memory fragment (-10) and region not found (-11)
= -19	First character of <*type*> is invalid; not 'e'
= -20	Required argument *work* is missing

FREER can be called from MACRO–11 programs if the standard FORTRAN calling convention is followed. All register contents are destroyed across the call. FREER calls IGETC and IFREEC, which are FORTRAN-dependent routines. To use FREER in a MACRO-only program, use the IGETC and IFREEC substitutes shown in the example.

Example:
See GETR/MGETR.

GCMAP/IGCMAP

Full Mapping

GCMAP/IGCMAP returns the previous CMAP status.

Form:

```
CALL GCMAP (iold [,ierr])
ierr = IGCMAP (iold)
```

where:

ierr Error return

iold Previous CMAP status

Value	Meaning
i = 0	Normal return.
= -257	Required argument missing.

Example:

```
        Subroutine FGCMAP        !Display mapping status
C
C       Display the mapping status for this job
C
        Include 'SRC:CMPDF'      !get bit definition for GCMAP
        Parameter CNFG3 = '466'o !third configuration word in fixed area
        Parameter CF3SI = '100000'o !Monitor supports extended mapping
        Parameter CF3HI = '040000'o !Hardware   "        "       "
C
        If (IAND (ISPY (CNFG3), IOR (CF3SI, CF3HI)) .eq.
        1       IOR (CF3SI, CF3HI)) Then
            Call GCMAP (ISTAT)  !get status
            If (IAND (ISTAT, CMUID - CMUXX) .eq. CMUID - CMUXX)
        1           Type *, 'Separated I and D in User space'
            If (IAND (ISTAT, CMJAS - CMXXS) .eq. CMJAS - CMXXS) Then
                Type *, 'Supy space enabled'
                If (IAND (ISTAT, CMSID - CMSXX) .eq. CMSID - CMSXX)
        1           Type *, 'Separated I and D in Supy space'
                If (IAND (ISTAT, CMPAR) .eq. 0) Then
                    Type *, 'All User and Supy Data PARS locked'
                Else
                    If (IAND (ISTAT, CMPR0) .eq. CMPR0)
        1               Type *, 'PAR 0 unlocked'
                    If (IAND (ISTAT, CMPR1) .eq. CMPR1)
        1               Type *, 'PAR 1 unlocked'
                    If (IAND (ISTAT, CMPR2) .eq. CMPR2)
        1               Type *, 'PAR 2 unlocked'
                    If (IAND (ISTAT, CMPR3) .eq. CMPR3)
        1               Type *, 'PAR 3 unlocked'
                    If (IAND (ISTAT, CMPR4) .eq. CMPR4)
        1               Type *, 'PAR 4 unlocked'
                    If (IAND (ISTAT, CMPR5) .eq. CMPR5)
        1               Type *, 'PAR 5 unlocked'
```

```
            If (IAND (ISTAT, CMPR6) .eq. CMPR6)
1               Type *, 'PAR 6 unlocked'
            If (IAND (ISTAT, CMPR7) .eq. CMPR7)
1               Type *, 'PAR 7 unlocked'
        End If
    Else
        Type *, 'Supy space disabled'
    End If
Else
    Type *, 'Monitor / hardware do not support extended mapping'
End If
Return
End
```

GETR/IGETR

Mapping

GETR/IGETR attaches to a specified global region. GETR can initialize a global region by reading a portion of a file into the global region or by calling a specified subroutine.

IGETR does not fetch handlers. Any handler required by I/O in GETR must be loaded or fetched by the program.

Form:

```
CALL GETR (arguments)
 i = GETR (arguments)
(work,char,name,addr  [,csize][,offset[,msize]]
[,chan[,blk]][,file[,blk]]  [,sbrtn,-1])
```

where:

work is a 7-word work area block. The first five words of the work area block contain information from the region definition block (RDB):

- A unique region identification (R.GID)
- The size of the region (R.GSIZ)
- The region status word (R.GSTS)
- The region name in two RAD50 words (R.NAME and R.NAME+2)
- The last two words in the work block area are reserved by RT–11.
- The work area specified in GETR must also be the work argument specified in FREER.

char is a character constant specifying the type of ownership of the global region. Only the first letter of the character constant need be specified and that letter must be enclosed in single quotes (’). Specify one of the following:
’private’—Program solely owns global region
’shared’—Global region available to other programs
’age’—Enables automatic global elimination

name is the 2-word name of the global region in six RAD50 characters

addr is a variable specifying the global region’s base address. The base address must begin on a PAR boundary (4K-word multiples beginning at 000000)

csize is the size of the global region you want to create, expressed in words. If you specify *csize* as zero or omit it, the actual global region size is used. Specifying zero for *csize* is invalid unless the global region already exists

offset is the offset from the beginning of the global region, expressed in units. A unit is 64(decimal) bytes. The offset is the number of units you skip before mapping begins. If you specify *offset* as zero or omit it, you begin mapping at the beginning of the global region

msize is the number of words you wanted mapped to the global region. If you specify *msize* as zero or omit it, you map the whole global region

chan is a channel opened on a file from which to read initialization data. When specifying *chan*, the argument value must be from 0 through 255(decimal), and the *blk* argument cannot be -1

file is a pointer to a 4-word data block. The last three words contain a device and file specification for a file containing initialization data to open and read. If the device specification is valid, the first word contains a value greater than 255(decimal)

The value in the *blk* argument must be -1.

sbrtn is the name of a subroutine that initializes the global region. The addr and *msize* arguments are passed to that subroutine

The value in the *blk* argument must be -1.

blk is the number of the first block to use in the file that initializes the global region. Specify a zero value in this argument to load from the beginning of the file

The value must be -1 if *blk* is coupled with the *sbrtn* argument.

The *work, name, addr, csize, offset, msize, chan, file,* and *blk* arguments are INTEGER*2 values. The *sbrtn* argument is EXTERNAL type.

IGETR can be called from MACRO–11 programs, if the standard FORTRAN calling convention is followed. All register contents are destroyed across the call. GETR calls IGETC and IFREEC, which are FORTRAN-dependent routines. To use IGETR in a MACRO-only program, use the following IGETC and IFREEC substitutes:

Errors:

Value	Meaning
i = 0	Normal return (success).
= -1	Invalid addr alignment (detected by IGETR).
= -2	No window definition block for .CRAW.
= -3	Any .CRAW error except no window definition block (-2).
= -4	End-of-file on .READW.

= -5	I/O error on .READW.
= -6	Channel closed when .READW attempted; channel not available from IGETR.
= -7	No region control block for .CRRG.
= -8	Insufficient memory for .CRRG.
= -9	Reserved.
= -10	Memory too fragmented to return at .ELRG.
= -11	Global region not found (and no nonzero size specified).
= -12	No global region control block for .CRRG.
= -13	Reserved.
= -14	Reserved.
= -15	.LOOKUP found channel already open.
= -16	.LOOKUP could not find requested file.
= -17	.LOOKUP found device in use and not shareable.
= -18	Any .ELRG error except memory too fragmented (-10).
= -19	First character of char argument invalid (not 'p', 's', or 'a').
= -20	Required argument missing: *work, char, name, or addr*.

IGETR returns the following errors if the .SERR programmed request is in effect:

Value	Meaning
i = -129	Called USR from completion routine.
= -130	No device handler; this operation needs one.
= -131	Error doing directory I/O.
= -132	.FETCH error. An I/O error occurred while the handler was being used or an attempt was made to load the handler over USR or RMON.
= -133	Error reading an overlay.
= -134	No more room for files in the directory.
= -135	Reserved.
= -136	Invalid channel number. Number is greater than number of channels that exist.
= -137	Invalid EMT, an invalid function code has been decoded.
= -138	Reserved.
= -139	Reserved.
= -140	Invalid directory.

= -141 Unloaded XM handler.

= -142 Reserved.
through -146

GFDAT/IGFDAT

GFDAT/IGFDAT returns the file creation date from a file's directory entry (E.DATE word). GFDAT is not supported for the distributed special directory handlers LP, LS, MM, MS, MT, MU, and SP.

Form:

```
CALL GFDAT (chan,dblk,idate)
 i = IGFDAT (chan,dblk,idate)
```

chan is a BYTE or INTEGER*2 channel number

dblk is a 4-element INTEGER*2 array containing a 4-word device and file specification in Radix–50; the file specification for which you want to return the creation date

idate on return, contains the requested INTEGER*2 creation date of the specified file

Errors:

Value	Meaning
i = 0	Normal return
= -1	Channel was open
= -2	File not found, or not a file-structured device. To determine what condition returned the error code, issue a .DSTAT request to determine if a device is file structured
= -3	Invalid operation (internal error)
= -4	Invalid offset (internal error)
= -257	Required argument missing

Example:

See IDCOMP.

GFINF/IGFINF

GFINF/IGFINF returns the word contents of the directory entry offset you specify from a file's directory entry. GFINF is not supported for the distributed special directory handlers LP, LS, MM, MS, MT, MU, and SP.

Form:

```
CALL GFINF (chan,dblk,offset,ival)
 i = IGFINF (chan,dblk,offset,ival)
```

where:

chan	is a BYTE or INTEGER*2 channel number
dblk	is a 4-element INTEGER*2 array containing a 4-word device and file specification in Radix–50; the file specification for which you want to return directory entry information
offset	is the octal byte offset for the directory entry word you want. The offset must be even. For example, specifying offset 12 returns the contents of E.USED in *ival*
ival	on return, contains the requested INTEGER*2 directory entry word

Function Result:

Value	Meaning
i = 0	Normal return
= -1	Channel was open
= -2	File not found, or not a file-structured device. (If it is necessary to determine what condition returned the error code, issue a .DSTAT request to determine if a device is file structured.)
= -3	Invalid operation (internal error)
= -4	Invalid offset
= -257	Required argument missing

Example:

```
       Program FGFINF !demo GFINF
C
C       Display the file "time",using the contents of offset 10(10)
C       of the directory entry.  This is treated as units of seconds
C       after midnight / 3.
C
        Parameter FATAL = '010'o
        Integer*2 CHAN          !Channel number
        Integer*2 FILSPC(39)    !DBLK(s)
        Integer*2 DEFTYP(4)     !default extensions
        Data DEFTYP /4*0/       !none
        Integer*2 DBLK(4)       !DBLK
        Integer*2 ERROR         !error/success value
```

```
        Integer*2 TIME          !time word value
        Integer*2 HOUR          !hour part
        Integer*2 MINUTE        !minute part
        Integer*2 SECOND        !second part
        Equivalence (FILSPC(16), DBLK(1)) !use 1st input file only
C
        CHAN = IGETC ()
1000        Continue
            Call PRINT (' ')    !clean up display
            ERROR = ICSI (FILSPC, DEFTYP, , ,0) !get filename
            If (ERROR .ne. 0) Go To 2000        !command error
            ERROR = IFETCH (DBLK)               !fetch handler
            If (ERROR .ne. 0) Go To 2100        !fetch error
            ERROR = IGFINF (CHAN, DBLK, 10, TIME) !get "time" word
            If (ERROR .eq. -2) Then
                Type *, 'File not found '
                Go To 1000                      !try again
            End If
            If (Error .ne. 0) Go To 2200        !gfinf error
            HOUR = TIME / (60 * (60 / 3))       !expressed in 3 sec units
            TIME = MOD (TIME, (60 * (60 / 3)))  !dump hours part
            MINUTE = TIME / (60 / 3)
            TIME = MOD (TIME, (60 / 3))         !dump minutes part
            SECOND = TIME * 3                   !back to real seconds
            Type 1, HOUR, MINUTE, SECOND
1           Format (' ', I3.2, ':', I2.2, ':', I2.2)
            Go To 1000
2000    Type *, 'CSI error'
        Go To 3000
2100    Type *, 'Fetch error'
        Go To 3000
2200    Type *, 'Gfinf error'
3000    Call Exit (FATAL)
            End
```

GFSTA/IGFSTA

GFSTA/IGFSTA returns the word contents of the directory entry status word (E.STAT) from a file's directory entry. GFSTA is not supported for the distributed special directory handlers LP, LS, MM, MS, MT, MU, and SP.

Form:

```
CALL GFSTA (chan,dblk,istat)
 i = IGFSTA (chan,dblk,istat)
```

where:

chan is a BYTE or INTEGER*2 channel number

dblk is a 4-element INTEGER*2 array containing a 4-word device and file specification in Radix–50; the file specification for which you want to return directory entry status word

istat on return, contains the requested INTEGER*2 directory entry status word

Errors:

Value	Meaning
i = 0	Normal return
= -1	Channel was open
= -2	File not found, or not a file-structured device. (If it is necessary to determine what condition returned the error code, issue a .DSTAT request to determine if a device is file structured.)
= -3	Invalid operation (internal error)
= -4	Invalid offset (internal error)
= -257	Required argument missing

Example:
(See also GFDAT example.)

```
        Program FGFSTA
C
C       This program displays the directory
C       entry status word contents for the selected file.
C       Entries with "?" appended are values that are not expected.
C
        Parameter FATAL = '010'o
        Integer*2 CHAN          !Channel to use
        Integer*2 FILSPC(39)    !DBLK(s)
        Integer*2 DEFTYP(4)     !default extensions
        Data DEFTYP /4*0/       !no defaults
        Integer*2 DBLK(4)       !DBLK
        Integer*2 ERROR         !error/success value
        Integer*2 STATUS        !status word value
        Integer*2 BIT           !sliding bit mask
```

```
        Character*7 BITNAM(0:15)!names for status word bits
        Data BITNAM(0) /'000001?'/, BITNAM(1) /'000002?'/
        Data BITNAM(2) /'000004?'/, BITNAM(3) /'000010?'/
        Data BITNAM(4) /'000020?'/, BITNAM(5) /'000040?'/
        Data BITNAM(6) /'000100?'/, BITNAM(7) /'000200?'/
        Data BITNAM(8) /'E.TENT?'/, BITNAM(9) /'E.MPTY?'/
        Data BITNAM(10)/'E.PERM '/, BITNAM(11)/'E.EOS? '/
        Data BITNAM(12)/'010000?'/, BITNAM(13)/'020000?'/
        Data BITNAM(14)/'E.READ '/, BITNAM(15)/'E.PROT '/
        Equivalence (FILSPC(16), DBLK(1))
C
        CHAN = IGETC ()
1000        Continue
            Call Print (' ')    !cleanup display
            ERROR = ICSI (FILSPC, DEVSPC, , , 0) !get file name
            If (ERROR .ne. 0) Go To 2000
            ERROR = IFETCH (DBLK) !get handler
            If (ERROR .ne. 0) Go To 2100
            ERROR = IGFSTA (CHAN, DBLK, STATUS) !get status word
            If (ERROR .eq. -2) Go To 2200
            If (ERROR .ne. 0) Go To 2300
            BIT = 1             !start of the sliding bit
            DO 1100, I = 0, 15
                If (IAND (STATUS, BIT) .eq. BIT) Type *, BITNAM(I)
                BIT = ISHFT (BIT, 1) !try next bit position
1100            Continue
            Type *, ' '
            Go To 1000
2000    Call Print ('?FGFSTA-F-CSI failed')
        Go To 3000
2100    Call Print ('?FGFSTA-F-FETCH failed')
        Go To 3000
2200    Call Print ('?FGFSTA-W-File not found')
        Go To 1000
2300    Call Print ('?FGFSTA-F-GFSTA failed')
3000    Call Exit (FATAL)
            End
```

GICLOS/GIOPEN/GIREAD/GIWRIT (GIDIS)

GIDIS consists of four FORTRAN system subroutines that can be used on the Professional Series only, instead of using the .SPFUN programmed requests:

GICLOS
GIOPEN
GIREAD
GIWRIT

Each GIDIS subroutine can return error information in a 2-word status array. Relevant error codes are listed with each subroutine. See GIDIS error codes description and sample GIDIS program.

GICLOS

The GICLOS subroutine ends the GIDIS connection to the Professional interface (PI) handler. The output device treats a GICLOS subroutine as an END-PICTURE instruction. Control is returned to the calling program once all data specified by the GIWRIT subroutine has been sent to the output device.

Form:

GICLOS (status,lun)

where:

status is a 2-word integer array used to return a code indicating the results of the requested operation

lun is the unit number assigned by GIOPEN to terminate. If no GIOPEN has been sent for the specified value, *status* is set to (-5,-1)

Errors:
See GIDIS error codes.

Example:
See sample GIDIS program.

GIOPEN

The GIOPEN subroutine initiates contact with the Professional interface (PI) handler and assigns a logical unit number (LUN) for this GIDIS operation. GIOPEN does not affect the current GIDIS state; all attributes currently selected remain in force.

To initialize the Professional video screen, execute the INITIALIZE -1 (complete initialization) instruction followed by the NEW_PICTURE instruction.

Form:

GIOPEN (status,lun[,message][,msglen][,devtype][,driver])

where:

status is a 2-word integer array to return a code indicating the results of the requested operation

lun is the unit number to associate with this GIOPEN; an integer number from 0 through 15

If *lun* is already connected to a GIDIS operation, *status* is set to (-5,-4)

message is a word containing a 0

msglen is the number of words in the message; normally a 1. If *msglen* is less than 0 or greater than 128(decimal) words, *status* is set to (-5,-3)

devtype is 6. Integer values 0 through 5, 7, and 8 are reserved

driver is 0

Errors:
See GIDIS error codes.

Example:
See sample GIDIS program.

GIREAD

The GIREAD subroutine returns a report from GIDIS requested by a report handling instruction sent by GIWRIT. GIREAD waits until GIDIS returns a report, then places the report in the buffer. If the report is longer than the buffer, the excess is lost. If the report is shorter than the buffer, the trailing words of the buffer are left unchanged. The first byte of the report contains the number of data words in the report.

Form:

GIREAD (status,lun,buffer,buflen)

where:

status is a 2-word integer array used to return a code indicating the results of the requested operation

lun is the unit number assigned by GIOPEN. If no GIOPEN has been issued for the specified value, *status* is set to (-5,-1)

buffer is the buffer for the report returned by GIDIS

buflen is the number of words in the report buffer

Errors:
See GIDIS error codes.

Example:
See sample GIDIS program.

GIWRIT

The GIWRIT subroutine sends the buffer of GIDIS command data to the Professional interface (PI) handler. You can pass a maximum of 2048(decimal) words to PI in one

GIWRIT system subroutine. The data in the buffer need not start or end on a command boundary.

Form:

GIWRIT (status,lun,message,msglen)

where:

status is a 2-word integer array used to return a code indicating the results of the requested operation

lun is the unit number assigned by GIOPEN. If no GIOPEN has been executed for the specified value, status is set to (-5,-1)

message is the command data to send

msglen is the number of words in the message.

The *msglen* parameter accepts a value equal to or greater than -1. Specify the -1 value for *msglen* to reset GIDIS. If less than 0 or greater than 2048(decimal) words, status is set to (-5,-3)

Errors:
See GIDIS error codes.

Example:
See sample GIDIS program.

GIDIS Error Codes

GIDIS subroutines can return the following error codes and subcodes in the 2-word status array. The error code specifies the class of error and is returned in the first word of the status array. The subcode specifies the actual error and is returned in the second word of the status array.

Directive error code (-1) can return the following subcode:

-1 No handler. The output device handler is not loaded.

Interface error code (-5) can return the following subcodes:

-1 Channel not open. The logical unit number (LUN) for that GIDIS is not assigned.

-2 DEVTYPE is out of range or invalid. The output device specified in a GIOPEN is invalid.

-3 MSGLEN out of range. The message length in a GIOPEN or GIWRIT is out of range.

-4 Channel in use. The logical unit number (LUN) specified for that GIDIS is already in use.

RT–11 specific error code (-7) can return the following subcodes during a GIDIS operation:

-1 Required argument missing. A required argument in a GIDIS subroutine is not specified.

-2 Handler not loaded. The output device handler is not loaded.

-3 File not found. The indicated file was not found on the device.

-4 File open on nonshareable or non-file-structured device.

-5 An attempt was made to read or write past the end-of-file (EOF) mark.

-6 Hard error. The GIDIS operation experienced a hard error on the output device.

Errors that occur if .SERR is in effect are listed under the .SERR request in the *RT–11 System Macro Library Manual*.

Sample GIDIS Program

The following FORTRAN program fragment uses the GIDIS subroutines to request the current cursor position:

```
        Program FGIDCA !demo GIDIS interface routines
C
C       Declare storage
C
        Integer*2 BUFLEN, LUN, MSGLEN, OCLEN, OPCODE
        Integer*2 BUFFER(3), MESSAG(1), STATUS(2)
C
C       user program here
C
        LUN = 5         !assign logical unit number
        OPCODE = 55*256 !request current position
        OCLEN = 0       !opcode length is 0
C
C       Put OPCODE and OCLEN into MESSAG buffer
C
        MESSAG(1) = OPCODE + OCLEN
        MSGLEN = 1      !length of message
C
C       Send to GIDIS
C
        Call GIWRIT (STATUS, LUN, MESSAG, MSGLEN)
        If (STATUS(1) .le. 0) Go To 999 ! error
C
        BUFLEN = 3      !length of report
C
C       Get report from GIDIS
C
        Call GIREAD (STATUS, LUN, BUFFER, BUFLEN)
        If (STATUS(1) .le. 0) Go To 999 ! error
C
C       Contents of buffer after successful return:
C
C       BUFFER(1) = 258 ((1*256) + 2)
C               1 = report header
C               2 = number of data elements in buffer
C       BUFFER(2) = Current 'X' position
C       BUFFER(3) = Current 'Y' position
C
C       more user program
C
999     Continue                !diagnose errors here
        End
```

GMCX/IGMCX

Mapping

GMCX/IGMCX (get mapping context) returns the mapping status of an extended memory window. Status is returned in the window definition block.

Form:

```
CALL GMCX (iwdb [,ierr])
 i  = GMCX (iwdb)
```

where:

ierr	Error return
iwdb	Address of Window Definition Block

Errors:

Value	Meaning
i = 0	Normal return.
= -5	Invalid window identifier specified.
= -257	Required argument missing.

Example:
See CRAW.

GTDIR/IGTDIR

GTDIR/IGTDIR (get directory) sets up parameters for a wildcard directory search operation on an RT–11 file-structured volume or logical disk file. Subsequent calls to the IGTENT function retrieve the directory entries that meet the criteria specified in GTDIR. GTDIR performs no searches itself and should be followed by calls to the IGTENT function.

GTDIR works outside of the USR. If you call GTDIR in a multijob monitor environment, you may want to explicitly lock the USR by calling the .LOCK and .UNLOCK routines. Locking the USR prevents alteration of the directory being searched.

Along with the required parameters, GTDIR supports numerous optional parameters that determine the criteria used for the directory search. Both required and optional parameter information are stored in the work area and buffer (after error checking is performed), and IGTENT then uses that information for the directory search.

If any optional parameter is specified, then any prior parameter not included in the function call must be marked with a comma pair (, ,) to show the position of that parameter.

Form:

```
CALL GTDIR (arguments)
 i = IGTDIR (arguments)
          (wksize,wkarea,chan,buffer[,header][,dblk][,string]
          [,stvalu][,stmask][,datrel][,datewd][,resrv1]
          [,resrv2][,stofst])
```

where:

wksize is a 1-word INTEGER*2 variable containing the value 64

wkarea is a 64-word INTEGER*2 work area array

chan is a 1-word INTEGER*2 variable. The meaning of *chan* is determined by the value specified for *chan* and the contents of the *dblk* parameter:

- If the first word of *dblk* is specified as nonzero and *chan* is specified as 0 through 255, then *chan* is an RT–11 channel, and *dblk* is a device or file on which to perform the operation.
- If the first word of *dblk* is specified as zero and *chan* is specified as 0 through 255, then *chan* is an RT–11 channel which has been previously opened by a lookup operation. The IGTDIR function is then directed to the device already opened on the specified channel.

- If *dblk* is not specified or is specified as zero and *chan* is specified as a value greater than 255, then *chan* is the address of an externally supplied MACRO–11 read routine. Such an external read routine could be written to provide RT–11 directory segments to IGTDIR in an unconventional manner. The entry environment of the external read routine is:

  R1 = <buffer address>
  R2 = <word count>
  R3 = <block number>

buffer is a 512-word INTEGER*2 array used by GTDIR and IGTENT to contain directory segments

header is a 5-word INTEGER*2 array that, on return, contains the directory segment header

dblk is a 4-word INTEGER*2 array that can be defined in one of the three ways, depending on the value of the first word:

- Word one contains a Radix–50 device specification for the device containing the directory to be searched. Words two, three, and four contain zero.
- Word one contains a Radix–50 device and file specification for a logical disk (.DSK) containing the directory to be searched.
- Word one contains the value 0, specifying a special mode operation. Specifying a special mode operation indicates that a channel is already open on the device you want to search. Therefore, an internal lookup operation is not performed; a device is already open on *chan*.
- Word two must contain the starting block number for the device directory.
- Word three of the array controls which entries are returned by IGTENT:

- If the third word in the array contains the value 0, IGTENT returns only those entries that match the specified GTDIR criteria.
- If the third word in the array contains the value 1, IGTENT returns all entries, governed only by the setting of the status characteristics bits in E.STAT. In this mode, entries returned that do not match the wildcard string are indicated by IGTENT returning the function result -20.

• Word four should contain the value zero and is reserved for Digital.

If *dblk* is not specified, see *chan*.

string is a string of up to eight ASCII RT–11 file specifications, separated by commas (,). Device specifications are not allowed.

Each file specification can contain trailing blanks in either or both the name and extension fields. Each file name and extension must be separated by a period (.).

Each file specification can contain general replacement wildcards (*) or single-character wildcards (%) in either or both the file name and extension. If either or both the file name or extension is left blank, it is treated as a general replacement wildcard.

stvalu is an INTEGER*2 value specifying bit values for the directory entry status (E.STAT) characteristics that apply to the directory search. The default is E.PERM (002000); only permanent files are returned.

stmask is an INTEGER*2 variable that contains the bit mask of the directory entry status (E.STAT) bits that are tested against the value specified in *stvalu*. By default, *stmask* checks the values for the E.PERM, E.TENT, and E.MPTY bits in E.STAT.

datrel is a 2-byte character string that specifies a code to be used in a date relationship directory search. The relationship is between an RT–11 date word you supply for *datewd* and directory entry dates. The default for *datrel* is 'EQ' if *datewd* is specified, or 'AL' if *datewd* is not specified. The following are valid codes:

Value	Meaning
'EQ'	Equal
'NE'	Not equal
'LE'	Less than or equal to
'LT'	Less than
'GE'	Greater than or equal to
'GT'	Greater than
'AL'	All dates

datewd is an INTEGER*2 variable specifying the RT–11 date word to check against directory entry dates.

The default for *datewd* is the current RT–11 system date if *datrel* is specified and all dates if *datrel* is not specified. The format of the RT–11 date word is:

Bits	Contents
0-4	Year minus the base (base specified by bits 14,15)
5-9	Day (1-31)
10-13	Month (1-12)
14,15	Age bits. Age bits extend the directory date by 32(decimal) year increments and have the following meaning:

15	14	Meaning When Set
0	0	Base year 1972
0	1	Base year 2004
1	0	Base year 2036
1	1	Base year 2068

resrv1 is reserved for Digital

resrv2 is reserved for Digital

stofst is a rarely used INTEGER*2 variable that specifies a starting offset from which to begin the directory search.

The value for *stofst* can be supplied from the IGTENT parameter *entofs* value. The *stofst* parameter lets you begin a directory search at the point where you previously stopped searching the directory.

The *stofst* parameter is especially useful when GTDIR/GTENT have previously performed a directory search through to a particular segment of a device directory. IGTENT can return a value representing the current directory search position in parameter *entofs*. In a subsequent search through the directory, specify the *entofs* value from the previous search in the GTDIR parameter *stofst*, to begin the search in the directory at that offset.

i is a returned INTEGER*2 result of function

Errors:

Value	Meaning
i = 0	Success
= -1	Channel in use
= -2	File not found
= -3	Directory already open
= -5	Invalid directory structure
= -7	Error reading directory segment
= -12	Invalid device for operation
= -13	Invalid date relationship code
= -16	Supplied work area is inadequate
= -19	Invalid arguments

Example:

```
       Program FGTDIR
C
C       Display file(s) on SY:
C
        Parameter ERRO R= '010'o
        Integer*2 WKAREA(64)    !area for IGTDIR/IGTENT
        Integer*2 ENTRY(7)      !single directory entry
        Integer*2 BUFFER(512)   !directory buffer
        Integer*2 DBLK(4)       !Device/file to search
        Integer*2 BLOCK2
        Integer*4 BLOCK         !really unsigned 16 bit
        Byte NAME(11)           !file name
        Byte DATSTR(11)         !date in ascii
```

```
        Byte STRING(81)         !filespec(s) w/o device
        Byte Prompt(8)          !command prompt
        Data Prompt /'F', 'i', 'l', 'e', 's', '?', ' ', '200'o/
        Data DBLK /3rSY , 3*0/  !system device
        Data BLOCK /0/
        Equivalence (BLOCK2, BLOCK)
C
        ICHAN = IGETC ()
1000        Continue
            Call RCTRLO         !reset ^O
            Type *, ' '         !blank line
            Call GTLIN (STRING, PROMPT) !get filespec(s)
            IERR = IGTDIR (64, WKAREA, ICHAN, BUFFER, , DBLK, STRING)
            If (IERR .ne. 0) Go To 2000
1100            IERR = IGTENT (WKAREA, ENTRY, , BLOCK2, NAME)
                If (IERR .lt. 0) Go To 1000
                Call DATE4Y (DATSTR, ENTRY(7))
                Type 1, (NAME(I), I = 1, 10), ENTRY(5), DATSTR, BLOCK
1               Format (' ', 10a1, ' ', i6, ' ', 11a1, ' ', i6)
                Go To 1100
2000    Call Print ('?FGTDIR-F-GTDIR failed')
        Call Exit (FATAL)
        End
```

GTDUS/IGTDUS

GTDUS/IGTDUS provides information about a specified MSCP (DU) or TMSCP (MU) class device unit. Issue the IGTDUS function only to MSCP or TMSCP class devices.

Information returned by IGTDUS includes:

- Whether the device unit is offline, available, or online.
- Whether the device media is removable (for example, the RC25).
- Whether the device unit is write-protected.
- Whether the device controller supports bad block replacement (MSCP only).
- The number of physical addressable blocks in the device unit (MSCP only).

  The MSCP volume size returned by IGTDUS is determined by which partition, if any, the unit number is mapped to when you issue IGTDUS. If you issue IGTDUS against the RT–11 unit that is partition zero of the device, it returns the entire volume size. If the unit number is mapped to a particular partition, IGTDUS returns the volume size from the base of that partition to the end of the volume. This change in IGTDUS functionality makes the information it returns more usable with, for example, the JWRITE subroutine.
- The media name, such as RC25, RCF25, RA60, RA80, RA81, RD31, RD32, RD51, RD52, RD53, RD54, RX33, RX50, TK50, or TU81.

Form:

```
CALL GTDUS (dev,ichan,ibuf[,iunit][,itype][,iwork][,isize])
 i = IGTDUS (dev,ichan,ibuf[,iunit][,itype][,iwork][,isize])
```

where:

dev is the Radix–50 device name (MSCP or TMSCP class devices only)

ichan is the integer specification for an RT–11 channel to be opened by IGTDUS

ibuf is a 7-word buffer containing status information returned by IGTDUS. The information returned by IGTDUS in the status buffer includes:

ibuf(1) is the status information word. The following values can be returned in the status information word:

Value	Meaning
0	The device unit is online
1	The device unit is available

2 The device unit is offline

ibuf(2) is the unit bit flag word. One or more of the following values can be returned in the unit flag word:

Value	Meaning
200	Media is removable (always set for TMSCP devices)
20000	Media is write-protected
100000	MSCP device controller supports bad-block replacement (never set for TMSCP devices)

ibuf(3)
ibuf(4)

For MSCP, a unit-size word is returned as a 28-bit value (16 bits in ibuf(3) and 12 bits in ibuf(4)), containing the size of the volume minus the base of the current partitions.

For TMSCP, the size words are undefined and each contains the value -1

ibuf(5)
ibuf(6)
ibuf(7)

Is the media device name in a fixed format 5-byte alphanumeric ASCII string, typically consisting of two characters followed by a space, two numbers, and a null byte. Sometimes the string contains three characters, such as for the RCF25, but is always terminated with a null byte.

iunit is the unit number requested. The meaning of *iunit* is determined by *itype*:

- If the *itype* parameter is 'RT11', *iunit* is optional and is the RT–11 unit number. The valid range is 0 through 77_8. If the unit number is higher than 27_8, include the *isize* parameter along with the *iwork* parameter to allocate a sufficient work area. If *iunit* is not specified, the default unit is that specified in the *dev* parameter.
- If the *itype* parameter is 'MSCP', *iunit* is required and is the decimal MSCP physical unit number. The valid range is 0 through 255.

itype is the type of unit number requested; a character constant specified as 'RT11' (the default) or 'MSCP':

'RT11'	Supports TMSCP and MSCP devices.
'MSCP'	Supports TMSCP and MSCP devices and requires the unit parameter. If MSCP is specified, *iunit* is required.

iwork is a recommended work area used internally by IGTDUS. If you do not define a work area, IGTDUS takes 80 words from the processor stack area. Allow for that when planning stack allocation to avoid stack overflow.

If the DU handler does not support extended device units, you can specify a work area of 80 words and you can omit the *isize* parameter.

You should specify at least a 160-word work area for *iwork*, and include the *isize* parameter if all the following are true:

- The DU handler has been built to support extended device units
- The *itype* parameter is specified as *RT–11*
- The device unit number, *iunit*, is higher than 27_8

isize specifies the work area size, together with *iwork*, when the DU handler has been built to support extended device units, the *itype* parameter is *RT–11*, and the device unit number is higher than 27(octal). The value supplied for *isize* should be the same as that used to declare *iwork*.

Use the IGTDUS function to determine the unit size before issuing the JREAD and JWRITE functions to MSCP devices or anytime you need status information concerning an MSCP or TMSCP device.

IGTDUS implicitly enables .SERR error condition handling while performing the channel lookup operation for the specified MSCP or TMSCP device. Any errors reported during the lookup operation are returned by IGTDUS. After the lookup operation completes, IGTDUS returns error condition handling to any that was previously enabled. Error condition handling by ISERR or IHERR other than during the IGTDUS lookup operation must be explicitly enabled in the program.

Errors:

Value	Meaning
i = 0	Normal return
= 1	Logic error. Retry the operation. If the error persists, submit an SPR to Digital.

= 2	Logic error. Retry the operation. If the error persists, submit an SPR to Digital.
= 3	Logic error. Retry the operation. If the error persists, submit an SPR to Digital.
= 4	Attempt to read or write past end-of-file or invalid function value
= 5	Hard error occurred on channel
= 6	Channel is not open
= 7	Work area is inadequate; specify at least 160 words for *isize* and *iwork* parameters
8-12	Reserved
= 13	Handler is not loaded
= 14	Handler is not installed
= 15	Channel is already in use
= 16	Logic error (.LOOKUP code 1)
= 17	Channel already open on a nonshareable device
= 18	Device does not support MSCP or TMSCP. Inappropriate device for IGTDUS function
= 19	First character of type argument is not an 'M' or 'R'
= 20	Logic error (.LOOKUP code 5)
= 21	Invalid unit
= 22	Reserved
= 23	Required argument missing: *dev, ichan,* or *ibuf* arguments

Other error codes can be returned by IGTDUS since the ISERR function is in effect. See ISERR in this chapter.

Example:

```
      Program FGTDUS  !demo GTDUS
C
C       Display the (T)MSCP status for the devices listed in
C       the DEV array if they are on the system.
C
        Integer*2 DEV(9)        !devices to check
        Integer*2 IBUF(7)       !status info returned
        Integer*2 IWORK(160)    !work area
        Character*8 STATUS(0:2) !status strings
        Character*3 ANAME       !device name in ascii
        Byte DTYPE(5)           !device type
        Equivalence (DTYPE, IBUF(5))
        Data DEV /3rDU0, 3rDU1, 3rDU2, 3rDU3,
        1         3rDU4, 3rDU5, 3rDU6, 3rDU7, 3rMU0/
        Data STATUS /'ONLINE', 'AVAILBL', 'OFFLINE'/
```

```
C
        ICHAN = IGETC ()
        DO 1000, I = 1, 9       !try each device
            Call R50ASC (3, DEV(I), ANAME) !get ascii name
            IERR = IFETCH (DEV(I)) !get handler
            If (IERR .eq. 0) Then
              IERR = IGTDUS (DEV(I), ICHAN, IBUF, , , IWORK, 160)
              If (IERR .eq. 0) Then
                 Type 2, ANAME, STATUS(IBUF(1)), DTYPE, (IBUF(J), J=2,4)
2               Format (' ', 1a3, ': Status=', 1a8, ', Type="', 5a1,
        1       '", IBUF(2--4)=', 3(o6, ' '))
                Else
                 Type *, '?FGTDUS-W-GTDUS failed for ', ANAME, ' with', IERR
                End If
            Else
                Type *, '?FGTDUS-W-Fetch failed for ', ANAME
            End If
1000        Continue
        End
```

GTIM

The GTIM subroutine returns the current time of day. The time is returned in two words and is given in terms of clock ticks past midnight. If the system does not have a line clock, a value of 0 is returned. If an RT–11 monitor TIME command has not been entered, the value returned is the time elapsed since the system was bootstrapped, rather than the time of day.

Form:

CALL GTIM (itime)

where:

itime is the two-word area to receive the time of day

The high-order time is returned in the first word, the low-order time in the second word. The CVTTIM routine can be used to convert the time into hours, minutes, seconds, and ticks. CVTTIM performs the conversion based on the monitor configuration word for 50- or 60-Hz clocks. Under all monitors, except for SB, the time-of-day is automatically reset after 24:00; under the SB monitor it is not.

Errors:
Error message *TRAP $MSARG* will display if *itime* argument is missing.

Example:
See CVTTIM.

GTJB/IGTJB

GTJB/IGTJB returns information about a job in the system.

Form:

```
CALL GTJB (addr[,jobblk[,ierr]])
ierr = IGTJB (addr[,jobblk])
```

where:

addr is the address of an eight-word or twelve-word block into which the parameters are passed.

jobblk is a pointer to a three-word ASCII logical job name for which data is being requested:

0–16: Job number for which information is desired
-1 or 'ME': Information is passed about issuing job
-3: Address of 3-word ASCII job name for which data is desired

ierr is an error return if the job is not running

The values returned are:

Word 1 Job Number = priority level *2 (background job is 0; system jobs are 2, 4, 6, 8, 10, 12; and foreground job is 14 in system job monitors; background job is 0 and foreground job is 2 in non-system job monitors; job number is 0 in all but SB monitor)

Word 2 High-memory limit

Word 3 Low-memory limit

Word 4 Start of channel area

Word 5 Pointer to impure area

Word 6 Low byte: unit number of job's console terminal (used only with multiterminal option; 0 when multiterminal feature is not used)

Word 7 Virtual high limit for a job created with the linker /V option (mapped monitors; 0 in unmapped monitors and where the Linker /V option is not used.)

Word 8-9 Reserved for future use

Word 10-12 ASCII logical job name (system job monitors only)

If one argument is used with the call, only the first eight words will be returned. For example,

Form:

```
INTEGER IJPARM(8)
 CALL GTJB (IJPARM)
 ierr = IGTJB (IJPARM)
```

At least a comma must follow the argument to pass the information into a 12-word block. For example,

Form:

```
INTEGER IJPARM(12)
 CALL GTJB (IJPARM ,)
 I = IGTJB (IJPARM ,)
```

Errors:

Value	Meaning
i = 0	Normal return.
=-1	No such job is currently running.

Error message *TRAP $MSARG* will display if *addr* argument is missing.

Example:

```
        Program FGTJB   !Demo GTJB routine
C
C       Get information about this job
C
        Parameter SUCCS = '001'o
        Parameter FATAL = '010'o
        Integer*2 REPLY(12)
C
        REPLY(10) = '*N'        !assume no job name
        REPLY(11) = 'ON'
        REPLY(12) = 'E*'
        If (IGTJB (REPLY, -1) .ne. 0) Go To 1000
        Type *, 'Job Number =', REPLY(1)
        Type 100, REPLY(4)
100     Format (' ', 'Addr I/O control blocks =', o8)
        Type 110, REPLY(5)
110     Format (' ', 'Addr impure area =', o8)
        Type 120, REPLY(10), REPLY(11), REPLY(12)
120     Format (' ', 'Job name = ', 3a2)
        Call Exit (SUCCS)
1000    Continue
        Type *, '?FGTJB-F-Request failed'
        Call Exit (FATAL)
        End
```

GTLIN/IGTLIN

GTLIN/IGTLIN can transfer a line of input to your program from the terminal or from an active indirect file.

You can force GTLIN to accept input only from the terminal, even if the program is running under the control of an indirect file.

This subroutine requires the USR. The maximum size of the input line is 80 characters. See the .GTLIN programmed request for setting bits in the job status word (JSW) to pass lowercase letters and establish a nonterminating condition.

Form:

```
CALL GTLIN (result[,prompt][,term]
                                [,plain])
 i = IGTLIN (result[,prompt][,term]
                                [,plain])
```

where:

result is the array receiving the string. This LOGICAL*1 array contains a maximum of 80 characters plus 0 as the end indicator and therefore must be dimensioned to at least 81 elements

prompt is a BYTE array containing an optional prompt string to be printed before the input line is received. The string format is the same as that used by the PRINT subroutine. If this argument is not present, no prompt is printed

term is a string constant specified when you want to take input only from the console terminal, even if the program is running under the control of an indirect command file. Use the 'term' argument when direct response from the program user is required. You need specify only the first character ('t'); case is unimportant

plain is a string constant specified when you want to take unaltered input and pass that input to the array specified by the GTLIN result argument. You must specify the 'plain' argument to prevent the operating system from reversing the arguments in the command line.

The *plain* argument is helpful when a program requires command input that is not a file specification, such as a SETUP command. You need specify only the first character ('p'); case is unimportant

GTLIN with the *plain* argument, checks the word in location 510 in the chain area for a byte count higher than 1.

- If location 510 does not contain a byte count higher than 1, GTLIN functions as though the 'plain' argument was not specified.
- If location 510 contains a word count higher than 1, GTLIN copies the ASCIZ string, beginning at location 512, into the result argument as specified in the GTLIN call. GTLIN then clears location 510.

- GTLIN then takes the input (converted by KMON) from the KMON buffer, thereby purging the buffer. Then GTLIN places the input from the KMON buffer into the chain area, beginning at location 512.

At the completion of the GTLIN call:

- The program has the unaltered input.
- Location 510 is clear.
- The KMON buffer is clear.

Notes

To avoid possible problems, Digital recommends that the GTLIN subroutine with the *plain* argument not be used in a program that uses the .CSIGEN and .CSISPC requests or the GTLIN subroutine without the *plain* argument.

GTLIN can be called from MACRO–11 programs if the standard FORTRAN calling convention is followed. All register contents are destroyed across the call. GTLIN has no dependencies on FORTRAN code or routines.

Errors:

Value	Meaning
i = 0	Success.
= -1	Line too long.

Error message *TRAP $MSARG* will display if *result* argument is missing.

Example:

```
C
C       Get input without file (CCL) processing
C
        Byte INPUT(81)
        Byte PROMPT(6)
        Data PROMPT /'N', 'a', 'm', 'e', '?', "200/
        Call GTLIN (INPUT, PROMPT, 'P')
```

HERR/IHERR

HERR/IHERR turns off ISERR (*error interception*) and allows the monitor to abort a job and generate an error message under fatal error conditions. IHERR itself returns no error codes.

Form:

```
CALL HERR ( )
 i = IHERR ( )
```

where:

i is a returned INTEGER*2 result of the function, a flag indicating the previous IHERR/ISERR setting:

Value	Meaning
i = 0	IHERR was in effect
i = 1	ISERR was in effect

Errors:
None.

Example:

```
        Program FHERR !demo HERR and SERR
C
C       Demonstrate how to save, modify and restore the
C       HERR/SERR status in a subroutine
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLK(4)       !unknown device/file
        Integer*2 CHAN          !channel to use
        Integer*2 OLD1          !mainline SERR/HERR setting
        Data DBLK /3rZZZ, 3*3rYYY/ !non existant device
C
        CHAN = IGETC ()
        OLD1 = IHERR ()         !get old setting
        OLD1 = IHERR ()         !confirm it
        Call TEST (CHAN, DBLK)  !call routine that uses SERR
        If (OLD1 .ne. IHERR ()) Then
            Type *, '?FHERR-F-S/HERR status not saved'
            Call Exit (FATAL)
        Else
            Type *, '!FHERR-I-Success'
        End If
        OLD1 = ISERR ()         !get old setting
        OLD1 = ISERR ()         !confirm it
        Call TEST (CHAN, DBLK)  !call routine that uses SERR
        If (OLD1 .ne. IHERR ()) Then
            Type *, '?FHERR-F-S/HERR status not saved'
            Call Exit (FATAL)
        Else
            Type *, '!FHERR-I-Success'
            Call Exit (SUCCS)
        End If
```

```
            End

            Subroutine TEST (CHAN, DBLK)
            Integer*2 DBLK(4)       !unknown device/file
            Integer*2 CHAN          !channel to use
            Integer*2 ERROR         !error code
            Integer*2 OLDERR        !previous SERR/HERR setting
      C
            OLDERR = ISERR ()       !set SERR, save old setting
            ERROR = LOOKUP (CHAN , DBLK) !open that WILL fail
            Type *, 'LOOKUP returned', ERROR
            If (OLDERR .eq. 0) CALL HERR !restore setting
            RETURN
            END
```

IADDR

The IADDR function returns the 16-bit absolute memory address of its argument as the integer function value.

Form:

i - IADDR (arg)

where:

arg is the variable or constant whose memory address is to be obtained. The value obtained by passing an expression as *arg* is unpredictable.

Errors:
Error message *TRAP $MSARG* will display if *arg* is missing.

Example:

```
       Program FADDR ! demo IADDR function
C
C       This is example code of an actual use of IADDR.
C       Mapping requests often need addresses on PAR boundaries
C       and this code demonstrates how you can create such an address
C       dynamically.
C
        Implicit Integer*2 (A-Z)
        Parameter SUCCS = '001'o
        Parameter FATAL = '010'o
        Integer*2 AREA(0:1)     !must disable subscript checking
C
        AREA0 = IADDR (AREA(0)) !find addr of AREA(1)
        PAR1 = ('20000'o - AREA0) / 2 !find element of AREA
C                               ! that is base of Par1
        AREAN = IADDR (AREA(PAR1)) !verify it
        Type 100, AREA0, PAR1, AREAN
100     Format (' ', 'Base address of AREA() is ', o8 /
        1       ' ', 'Address of AREA(', i5, ') is ' o8)
        If (AREAN .ne. '20000'o) Go To 1000
        Type *, '!FADDR-I-Success'
        Call Exit (SUCCS)
C
1000    Continue
        Type *, '?FADDR-F-Failed'
        Call Exit (FATAL)
        End
```

IDATE

The IDATE function returns three INTEGER*2 values representing the current (system) *month, day,* and *year* or an optional RT–11 date word you provide. IDATE was previously located in the distributed FORTRAN subroutine libraries, FORLIB and F77OTS.

Form:

```
i = IDATE (mon,day,year[,opdate])
```

where:

mon is an INTEGER*2 variable that, on return, contains an integer representation of the month. January is represented as 1. December is represented as 12. Returned as zero if the system date has not been set

day is an INTEGER*2 variable that, on return, contains the integer day of the month

year is an INTEGER*2 variable that, on return, contains the positive difference between 1900 and the current year

opdate is an optional RT–11 date word to be converted. Specifying a 0 value for *opdate* returns the current system date. Note that the value you enter for *opdate* is not validated. The format of the date word is:

Bits	Contents
0-4	Year minus the base (base specified by bits 14,15)
5-9	Day (1-31)
10-13	Month (1-12)
14,15	Age bits. Age bits extend the directory date by 32(decimal) year increments and have the following meaning:

15	14	Meaning When Set
0	0	Base year 1972
0	1	Base year 2004
1	0	Base year 2036
1	1	Base year 2068

i is one of the following values returned (only when IDATE is called in the form *i = IDATE*:)

Value	Meaning
i = 0	Success

Error message *TRAP $MSARG* will display if *month, day* or *year* argument is missing.

Example:
See DATE.

IDCOMP

The IDCOMP function compares two RT–11 date words and returns an integer value that reflects the relationship between those dates.

Form:

```
i = IDCOMP (date1[,date2])
```

where:

date1 is the first RT–11 date word; that word is compared against *date2*. The format of the RT–11 date word is:

Bits	Contents
0-4	Year minus the base (base specified by bits 14,15)
5-9	Day (1-31)
10-13	Month (1-12)
14,15	Age bits. Age bits extend the directory date by 32(decimal) year increments and have the following meaning:

15	14	Meaning When Set
0	0	Base year 1972
0	1	Base year 2004
1	0	Base year 2036
1	1	Base year 2068

date2 is the optional second RT–11 date word (default is current system date)

Function Result:

Value	Meaning
i = 0	The dates are the same
< 0	A negative value indicates *date1* is before *date2*
> 0	A positive value indicates *date1* is after *date2*

Errors:

Value	Meaning
i = -257	Invalid or missing argument.

Example:

```
       Program FDCOMP !demo IDCOMP and GFDAT
C
C       Compare the creation DATES of an OBJ and associated SAV
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLKO(4), DBLKS(4), CHAN, ODATE, SDATE
        Data DBLKO /3rOBJ, 3rFDC, 3rOMP, 3rOBJ/
        Data DBLKS /3rSRC, 3rFDC, 3rOMP, 3rFOR/
C
        CHAN = IGETC ()
        IERR = IGFDAT (CHAN, DBLKO, ODATE)
        If (IERR .ne. 0) Go To 300
        IERR = IGFDAT (CHAN, DBLKS, SDATE)
        If (IERR .ne. 0) Go To 300
        If (IDCOMP (ODATE, SDATE)) 110, 120, 130
110     Call Print ('!FDCOMP-I-SAV is newer than OBJ')
        Go To 200
120     Call Print ('!FDCOMP-I-SAV and OBJ have the same date')
        Go To 200
130     Call Print ('!FDCOMP-I-SAV is OLDER than OBJ')
200     Call Exit (SUCCS)
300     Call Print ('?FDCOMP-F-GFDAT returned an error')
        Call Exit (FATAL)
        End
```

IFWILD

The IFWILD function tests a file specification string (file name and extension only) for a match against up to eight wildcard file specifications (file name and extension only). IFWILD returns the result of the test (success or no success) and, if the test is successful, which of the file directory entries produced the match.

Form:

i = IFWILD (tststr,matstr[,explct])

where:

tststr is a character string test file specification; that file name and extension for which you are seeking a match. The test file specification can contain trailing blanks in either or both the name and extension fields, so long as their length does not exceed 6 and 3 characters respectively. The file name and extension must be separated by a period (.), and the file specification is terminated with a NULL character.

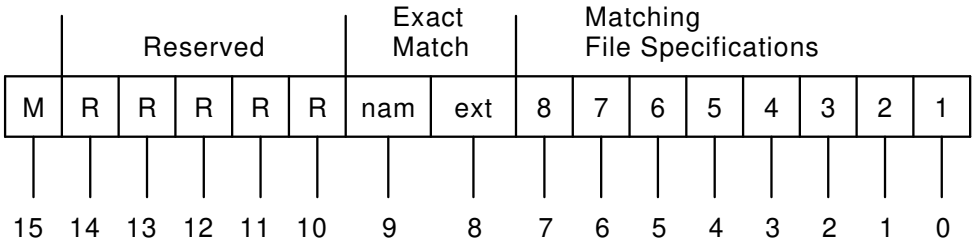
matstr is a character string of up to eight file specifications separated by commas (,) and terminated by a NULL. Device specifications are invalid. Each file specification can contain trailing blanks, general replacement wildcards (*), or single-character wildcards (%) in either or both the file name and extension. If the extension is left blank, it is treated as a general replacement wildcard, unless the optional parameter *explct* is specified as 'E'. Each file name and extension must be separated by a period (.), each file specification must be separated by a comma (,), and the complete string must be terminated by a NULL

explct is either 'E' or 'I'. Specify 'E' to indicate explicit wildcarding; functionally equivalent to the command SET WILD EXPLICIT. Specify 'I' to indicate implicit wildcarding; functionally equivalent to the command SET WILD IMPLICIT (the default). If *explct* is omitted, current monitor setting of SET WILD is used.

i is the INTEGER*2 function result.

Errors:

Value	Meaning
i = -2	Invalid arguments or missing arguments
= -1	No match
= >0	A match occurred and can be interpreted according to the following bitmap:



Bits	Contents
15	Clear indicates a match
10-14	Reserved
9	Successful exact match for file name (*nam*). The test file specification (*tststr*) name exactly matched at least one of the matching file specification (*matstr*) names. A wildcard match for file name does not set this bit.
8	Successful exact match for file extension (ext). The test file specification (*tststr*) extension exactly matched one of the matching file specification (*matstr*) extensions. A wildcard match for file extension does not set this bit.
0-7	Each bit corresponds to a match string file specification. For example, bit 0 corresponds to the first file specification in the match string, and bit 7 corresponds to the eighth. A successful test string match of the test string file name and extension with the corresponding match string file specification sets the corresponding bit. More than one file specification can match, therefore, more than one bit can be set. Each match can be an exact match or a wildcard match. Check bits 8 and 9 to determine if match is exact for file name and/or extension

Example:

```
       Program FIFWIL !demo IFWILD
C
C       Try to match the wildcard file formats:
C        T*.DAT and T*.INP
C       Rejecting all command lines that do not match
C
        Byte FILSPC(81)
        Character*15 MATCH
        Data MATCH /'T*.DAT,T*.INP'/
        Character*14 PROMPT
        Byte P(14)
        Equivalence (PROMPT, P)
        Data PROMPT /'input file? '/
C
        Call SCOPY (MATCH, MATCH, 14)   !null terminate the string
        P(14) = '200'o                  !and no CRLF on prompt
100     Continue
        Call GTLIN (FILSPC, PROMPT)
        If (IFWILD (FILSPC, MATCH) .gt. 0) Then
            Type 1, '!FIFWIL-I-Valid file was: ',  (FILSPC(I), I=1,14)
1           Format (' ', a27, 14a1)
            Call Exit
        Else
            Type *, '!FIFWIL-I-Invalid file name, try again'
            Go To 100
        End If
        End
```

IGTENT

The IGTENT function returns the next directory entry that matches the criteria specified in the IGTDIR function. If there are no remaining matching entries, IGTENT returns an error code.

Form:

i = IGTENT (wkarea,entry[,entofs][,filblk][,ascnam])

where:

wkarea is the 64-word work area array specified in the IGTDIR function *wkarea* parameter

entry is an INTEGER*2 array, the length and contents of which are determined by the IGTDIR *header* parameter.

- If *header* is not specified in IGTDIR, *entry* is a 7-word INTEGER*2 array that, on return, contains the 7-word directory entry matching the criteria specified in IGTDIR.
- If *header* is specified in IGTDIR, *entry* contains the entire directory entry (including any optional extra words) matching the criteria specified in IGTDIR. The fourth word in the directory header returned in the IGTDIR *header* parameter specifies the number of extra bytes in directory entries.

entofs is an INTEGER*2 variable that, on return, contains a value representing the position of the next directory entry (see the IGTDIR function *stofst* parameter.)

filblk is an INTEGER*2 variable that, on return, contains the starting block number for the directory entry that matches the criteria specified in IGTDIR.

ascnam is an 11-byte array that, on return, contains a fixed-format ASCII string file specification of the directory entry matching the criteria specified in IGTDIR. The directory entry is padded in the name and extension fields with blanks up to 6 and 3 characters respectively and is terminated with a NULL.

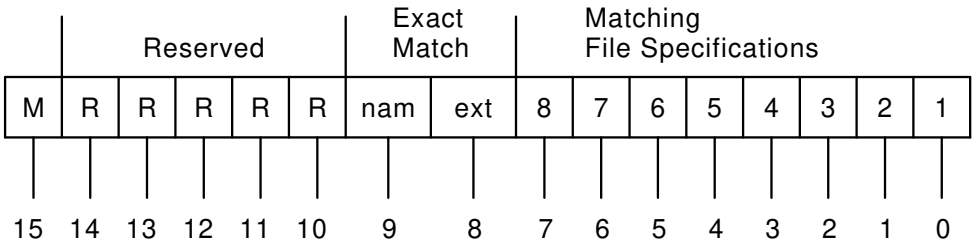
Function Result:

Value	Meaning
i = -7	Error reading directory segment
= -9	No directory open
= -10	End of directory encountered

= -19 Invalid arguments

= -20 Returned file entry does not match (special mode operation)

= >0 A match occurred and can be interpreted according to the following bit fields:



Bits	Contents
15	Clear indicates a match
10-14	Reserved
9	Successful exact match for exact file name (nam). The test file name exactly matched that of at least one of the matching file specifications. A wildcard match for file name does not set this bit
8	Successful exact match for file extension (ext). The test file extension exactly matched that of at least one of the matching file specifications. A wildcard match for file extension does not set this bit
0-7	Each bit corresponds to a match string file specification. For example, bit 0 corresponds to the first file specification in the match string and bit 7 corresponds to the eighth. A successful test string match of the test string file name and extension with the corresponding match string file specification sets the corresponding bit. More than one file specification can match, therefore more than one bit can be set. Each match can be an exact match or a wildcard match. Check bits 8 and 9 to determine if match is exact for file name and/or extension

Example:
See example for GTDIR/IGTDIR.

IJCVT

The IJCVT function converts an INTEGER*4 value to INTEGER*2 format. If you do not specify *ires*, the result returned is the INTEGER*2 value of *jsrc*. If you specify *ires*, the result is stored there.

Form:

i = IJCVT (jsrc[,ires])

where:

jsrc specifies the INTEGER*4 variable or array element whose value is to be converted

ires specifies the INTEGER*2 entity to receive the conversion result

Function Result (if *ires* is specified):

Value	Meaning
i = 0	Normal return; the result is 0.
= 1	Normal return; the result is positive.
= -1	Normal return; the result is negative.
= -2	An overflow occurred during conversion.

Errors:
Unpredictable results will occur if the *jsrc* argument is omitted.

Example:

```
        Program FIJCVT !Demo IJCVT
C        Demonstrate the boundary conditions for IJCVT
C
         Integer*4 JVAL          !Long to convert from
         Integer*2 IVAL          !short to convert into
         Integer*2 IERR          !error/result codes
         Character*8 RESULT(-2:1) !strings describing results
         Data RESULT /'Overflow', 'Negative', 'Zero', 'Positive'/
C
         JVAL = 0
         IERR = IJCVT (JVAL, IVAL)
         Type *, JVAL, IVAL, ' ', RESULT(IERR)
         JVAL = -32768
         IERR = IJCVT (JVAL, IVAL)
         Type *, JVAL, IVAL, ' ', RESULT(IERR)
         JVAL = +32767
         IERR = IJCVT (JVAL, IVAL)
         Type *, JVAL, IVAL, ' ', RESULT(IERR)
         JVAL = -32769
         IERR = IJCVT (JVAL, IVAL)
         Type *, JVAL, IVAL, ' ', RESULT(IERR)
         JVAL = +32768
         IERR = IJCVT (JVAL, IVAL)
         Type *, JVAL, IVAL, ' ', RESULT(IERR)
         End
```

INDEX

INDEX searches a source string for the occurrence of a pattern string and returns the character position of the first occurrence of the pattern within the source.

Form:

```
CALL INDEX (a,pattrn[,i],m)
 i =  INDEX (a,pattrn[,i],m)
```

or

```
m = INDEX (a,pattrn[,i])
```

where:

a	is the array containing the source string to be searched; it must be terminated by a null byte
pattrn	is the string being sought; it must be terminated by a null byte
i	is the integer starting character position of the search in *a*. If *i* is omitted, *a* is searched beginning at the first character position
m	is an integer variable to store the result of the search; *m* is set to the starting character position of pattern in *a*, if found; otherwise, *m* is 0

Errors:
Unpredictable results will occur if required arguments are omitted.

Example:

The following example searches the array STRING for the first occurrence of strings EFG and XYZ and searches the string ABCABCABC for the occurrence of string ABC after position 5.

```
        Program FINDEX !demo INDEX
C
C       Show several forms of INDEX (as function and
C       subroutine) and w/o optional arguments
C
        Byte STRING(10)
C
        Call SCOPY ('ABCDEFGHI', STRING)        !init for test
        Call INDEX (STRING, 'EFG', , M)         !expect 5
        Call INDEX (STRING, 'XYZ', , N)         !expect 0
        Type *, 'M=', M, ' N=', N, ' INDEX =',  !display results
        1       INDEX ('ABCABCABC', 'ABC', 5)   !expect 7
        END
```

INSERT

The INSERT subroutine replaces a portion of one string with another string.

Form:

CALL INSERT (in,out,i[,m])

where:

in is the array containing the string being inserted. The string must be terminated with a null if the number of characters is less than the value of *m* (below), or if *m* is not specified

out is the array containing the string being modified. The string must be terminated with a null

i is the integer specifying the character position in *out* at which the insertion begins

m is the integer maximum number of characters to be inserted

If the maximum number of characters (m) is not specified, all characters to the right of the specified character position (i) in the string being modified are replaced by the string being inserted. The insert string (in) and the string being modified (out) can be in the same array only if the maximum number of characters (m) is specified and is less than or equal to the difference between the position of the insert (i) and the maximum string length of the array.

Errors:
Unpredictable results will occur if required arguments are omitted.

Example:

```
        Program FINSER
C
C        Show various options with INSERT
C
         Byte S1(11), S2(11), S3(11)
C
         Call SCOPY ('ABCDEFGHIJ', S1)    !init test string
         Call SCOPY (S1, S2)              !and another one
         Call SCOPY (S1, S3)              !and another one
         Call INSERT ('123', S1, 6)       !S1 = ABCDE123
         Call INSERT ('123', S2, 6, 2)    !S2 = ABCDE12HIJ
         Call INSERT ('123', S3, 6, 4)    !S3 = ABCDE123IJ
         Call PRINT (S1)
         Call PRINT (S2)
         Call PRINT (S3)
         End
```

IPEEK

The IPEEK function returns the contents of the word located at a specified 16-bit address in the current job's address space. The subroutine can examine device registers if the registers are located in the current job's address space.

Form:

i = IPEEK (iaddr)

where:

iaddr is the integer specification of the 16-bit address in the current job's address space to be examined. If this argument is not an even value, a trap results (except on an LSI–11 or a PDP–11/23)

Function Result:
The function result (i) is set to the value of the word examined.

Errors:
Error message *TRAP $MSARG* will display if argument *iaddr* is missing.

Example:

```
        Program FPEEK
C
C        Use (I)PEEK(B) and (I)POKE(B) to work with
C        the SYSCOM area
C
         Parameter JSW = '44'o   !Job Status Word
         Parameter TTLC = '040000'o !lower case bit in JSW
         Parameter ERRBY = '52'o !Emt eRRor BYte
         Parameter USRRB = '53'o !USer program eRRor Byte
         Parameter SUCCS = '001'o !success bit
         Integer*2 UNKFIL(4)     !DBLK for non-existant file
         Data UNKFIL /3rSY , 3rXXX, 3rXXX, 3rZZZ/
         Integer*2 OLDJSW        !original JSW
         Byte OLDERR             !original ERRBY
         Byte BUFFER(81)         !character buffer
         Byte PROMPT(6)          !input prompt
         Data PROMPT /'t', 'e', 's', 't', ':', '200'o/
C
         OLDJSW = IPEEK (JSW)    !get original JSW
         Type 1, OLDJSW          !display JSW
1        Format (' ', 'JSW=', o7)
         OLDERR = IPEEKB (ERRBY) !get old error byte
         Type 2, OLDERR
2        Format (' ', 'ERRBY=', o3)
         Call Print ('000'o)     !clean up screen
         Call POKE (JSW, IAND (OLDJSW, NOT (TTLC))) !clear TTLC
         Call RCTRLO
         Call GTLIN (BUFFER, PROMPT) !get a line (in uppercase)
         Call Print (BUFFER)     !display it
         Call POKE (JSW, IOR (OLDJSW, TTLC)) !set TTLC
         Call RCTRLO
         Call GTLIN (BUFFER, PROMPT) !get a line (in lowercase)
```

```
        Call Print (BUFFER)     !display it
        Call POKE (JSW, OLDJSW) !restore it to the original state
        ICHAN = IGETC ()
        IERR = LOOKUP (ICHAN, UNKFIL) !look for non-existant file
        Type 2, IPEEKB (ERRBY)  !get error code from SYSCOM
        Call POKEB (USRRB, IOR (IPEEKB (USSRB), SUCCS))
C                               !set success code
C                               !(Call EXIT (SUCCS) is easier
        Type 3, IPEEKB (USRRB)  !display it
3       Format (' ', 'USRRB=', o3)
        End
```

IPEEKB

IPEEKB returns the contents of the byte located at a specified 16-bit address in the current job's address space. Since this subroutine operates in a byte mode, the address supplied can be odd or even. The subroutine can examine device registers if the registers are located in the current job's address space. The return is zero extended; that is, the high byte is 0.

Form:

i = IPEEKB (iaddr)

where:

iaddr is the integer specification of the 16-bit address in the current job's address space to be examined. Unlike the IPEEK subroutine, the IPEEKB subroutine allows odd addresses

Function Result:
The function result (i) is set to the value of the byte examined.

Errors:
Error message *TRAP $MSARG* will display if argument *iaddr* is missing.

Example:
See IPEEK.

IRAD50

The IRAD50 function converts a specified number of ASCII characters to Radix–50 and returns the number of characters converted. Conversion stops on the first non-Radix–50 character encountered in the input, or when the specified number of ASCII characters have been converted.

Form:

n = IRAD50 (icnt,input,output)

where:

n	is the integer number of input characters actually converted
icnt	is the number of ASCII characters to be converted
input	is the area from which input characters are taken
output	is the area in which Radix–50 words are stored

Three characters of text are packed into each word of output. The number of output words modified is computed by the expression (in integer words):

(icnt+2)/3

Thus, if a count of 4 is specified, two words of output are written even if only a one-character input string is given as an argument.

Function Result:
The integer number of input characters actually converted (n) is returned as the function result.

Errors:
Unpredictable results will occur if any required argument is omitted.

Example:

```
Real*8 FSPEC
 Call IRAD50 (12, 'SY SWAP  SYS', FSPEC)
```

ISPY

The ISPY function returns the integer value of the word at a specified offset from the RT–11 resident monitor. This subroutine uses the .GVAL programmed request to return fixed monitor offsets. (See *RT–11 System Macro Library Manual* for information on fixed offset references.)

Form:

```
i = ISPY (ioff[,ierr])
```

where:

ioff is the offset (from the base of RMON) to be examined.

ierr is the optional error return.

Function Result:
The function result (i) is set to the value of the word examined.

Errors:

Value	Meaning
i = 0	Success
= -1	Offset *ioff* is out of range. (Not contained within RMON)
= -2	Trap 4: Odd or nonexistent address.

Error message *TRAP $MSARG* will display if argument *ioff* is missing.

Example:
See PUT.

ISWILD

The ISWILD function checks for matching, either using or not using wildcards, between two ASCII strings. Valid wildcards are the asterisk (*) and percent sign (%).

By default, the strings are terminated by a NULL. Optionally, other terminators can be specified. By default, the string character comparisons are case-insensitive for alphabetic characters. Optionally, the comparison can be made case sensitive.

Form:

i = ISWILD (tststr,matstr[,term][,case][,explct])

where:

tststr is the test string; the string you submit to check against a match string. The test string can contain any ASCII characters and is terminated with a NULL character or a character specified in the *term* parameter

matstr is the match string; the string against which you compare the test string. The match string can contain any ASCII character except an embedded NULL. An asterisk wildcard (*) indicates a match-all sequence of unknown length, while a percent sign wildcard (%) indicates a match-all sequence of exactly one character

term is an optional ASCIZ string containing other terminators made valid for this comparison, such as a period (.), comma (,), blank, or tab. NULL is always recognized as a valid string terminator.

case determines case sensitivity requirements. The default is no sensitivity; all alphabetic characters are forced to uppercase before comparison. Specify 'C' to choose case sensitivity (7-bit ASCII only)

explct is either 'E' or 'I'. Specify 'E' to indicate explicit wildcarding; functionally similar to the command SET WILD EXPLICIT. Specify 'I' to indicate implicit wildcarding; functionally similar to the command SET WILD IMPLICIT. Implicit wildcarding is the default.

i contains the result of the comparison.

Errors:

Value	Meaning
i = 0	Exact match
= 1	Wildcard match
=-1	No match
=-2	Invalid arguments

Example:

```
       Program FISWIL
C
C       Search the specified file for lines containing strings
C       matching the specified wildcard search argument(*).
C
        Byte FILNAM(16), MATCH(80), TEST(80)
        Byte PFILE(7)
        Data PFILE /'F', 'i', 'l', 'e', '?', ' ', '200'o/
        Byte PSTRIN(8)
        Data PSTRIN /'S', 't', 'r', 'i', 'n', 'g', ' ', '200'o/
C
        Call GtLin (FILNAM, PFILE)
        Open (Unit=2, Name=FILNAM, Type='OLD', Err=990)
        Call GtLin (MATCH, PSTRIN)
C
        LINE = 0
100     Continue
            LINE = LINE + 1
            Read (2, 1101, End=999) NCH, TEST
            TEST(NCH+1) = 0             !terminate string
            If (ISWILD (TEST, MATCH) .lt. 0) Go To 100
            Write (5, 1105) LINE, (TEST(K), K=1,MIN (72,NCH))
            Go To 100
C
990     Write (5, 1991) FILNAM
999     Call Exit
C
1101    Format (q, 80a1)
1105    Format (' ', I4, ':', 72a1)
1991    Format (' ', '?FISWIL-F-File not found - ', 16a1)
        End
```

ITLOCK

Multijob

The ITLOCK function is used in a multijob system to attempt to gain ownership of the USR. It is similar to LOCK in that, if successful, the user job returns with the USR in memory. However, if a job attempts to LOCK the USR while the other job is using it, the requesting job is suspended until the USR is free. With ITLOCK, if the USR is not available, control returns immediately and the lock failure is indicated.

Form:

```
i = ITLOCK()
```

For further information on gaining ownership of the USR, see the .TLOCK programmed request.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	USR is already in use.

Example:
See LOCK.

ITTINR

The ITTINR function transfers a character from the console terminal to the user program. If no characters are available, system action is determined by the setting of bit 6 of the Job Status Word.

Form:

```
i = ITTINR()
```

If the function result (i) is less than 0 when execution of the ITTINR function is complete, it indicates that no character was available. ITTINR does not return a result of less than zero unless bit 6 of the Job Status Word was on when the request was issued.

There are two modes of doing console terminal input, and they are governed by bit 12 of the Job Status Word (JSW). The JSW is at octal location 44. If bit 12 is 0, normal I/O is performed under the following conditions:

- The monitor echoes all characters typed.
- CTRL/U and RUBOUT perform line deletion and character deletion, respectively.
- A carriage return, line feed, CTRL/Z or CTRL/C must be struck before characters on the current line are available to the program. When one of these is typed, characters on the line typed are passed one by one to the user program.

If the console is in special mode (bit 12 set to 1), the following conditions apply:

- The monitor does not echo characters typed except for CTRL/C and CTRL/O.
- CTRL/U and RUBOUT do not perform special functions.
- Characters are immediately available to the program.

In special mode, the user program must echo the characters desired. However, CTRL/C and CTRL/O are acted on by the monitor in the usual way.

Bit 12 in the JSW must be set by the user program if special console mode is desired. Bit 14 in the JSW must be set if lowercase characters are desired. These bits are cleared when control returns to RT–11.

Regardless of the setting of bit 12, when a carriage return is entered, both carriage return and line feed characters are passed to the program; if bit 12 is 0, these characters will be echoed.

Lowercase conversion is determined by the setting of bit 14. If bit 14 is 0, lowercase characters are converted to uppercase before being echoed (if bit 12 is 0) and passed to a program; if bit 14 is 1, lowercase characters are echoed (if bit 12 is 0) and passed as received. Bit 14 is cleared when the program terminates.

Notes

To set and/or clear bits in the JSW, do an IPEEK and then an IPOKE (See IPOKE example.) In special terminal mode (JSW bit 12 set), normal FORTRAN formatted I/O from the console is undefined.

If the single-line editor has been enabled with the SET SL ON and SET SL TTYIN commands, input from an ITTINR request can be edited by the single-line editor if JSW bits 4 and 12 are 0. However, if either bit 4 or bit 12 is set, SL will not edit ITTINR input. If SL is editing input, the state of bit 6 (inhibit TT wait) is ignored and an ITTINR request will not return until an edited line is available.

In multijob monitors, CTRL/F and CTRL/B (and CTRL/X in monitors with the system job feature) are not affected by the setting of bit 12. The monitor always acts on these characters if the SET TT FB command is in effect.

Also under the multijob monitor, if a terminal input request is made and no character is available, job execution is normally suspended until a character is ready. If a program requires execution to continue and ITTINR to return a result of less than zero, it must turn on bit 6 of the JSW before the ITTINR. Bit 6 is cleared when a program terminates. The results of ITTINR must be stored in an INTEGER type variable for the purposes of error checking. Once it is known that the call did not have an error return, the result can be moved into a LOGICAL*1 variable or array element. Direct placement into a LOGICAL*1 variable will lead to incorrect results, because the negative flag (bit 15 set) is lost in conversion to a LOGICAL*1 variable.

Function Results:

Value	**Meaning**
i = >0	Character read.
i = <0	No character available.

Example:
See example in Section 1.5.2.

ITTOUR

The ITTOUR function transfers a character from the user program to the console terminal if there is room for the character in the monitor buffer. If it is not currently possible to output a character, an error flag is returned.

Form:

```
i = ITTOUR (char)
```

where:

char is the character to be output, right-justified in the integer (can be LOGICAL*1 entity if desired)

If the function result (*i*) is 1 when execution of the ITTOUR function is complete, it indicates that there is no room in the buffer and that no character was output. ITTOUR normally does not return a result of 1. Instead, the job is blocked until room is available in the output buffer. If a job requires execution to continue and a result of 1 to be returned, it must turn on bit 6 of the JSW before issuing the request.

Notes

If a foreground job has characters in the TT output buffer, they are not output under the following conditions:

- If a background job is doing output to the console TT, the foreground job cannot output characters from its buffer until the background job outputs a line feed character. This can be troublesome if the console device is a graphics terminal and the background job is doing graphic output without sending any line feeds.
- If no background job is running (that is, KMON is in control of background), the foreground job cannot output its characters until the user types a carriage return or a line feed. In the former case, KMON gets control again and locks out foreground output as soon as the foreground output buffer is empty.

Note that the use of PRINT eliminates these problems.

Function Results:

Value	Meaning
i = 0	Character was output.
= 1	Ring buffer is full.

Errors:
Unpredictable results will occur if required arguments are omitted.

Example:
See example in Section 1.5.2.

IWEEKD

The IWEEKD function, supplied with a month, day, and year returns an integer value representing the day of the week for that month, day, and year.

Form:

```
i = IWEEKD (month,iday,iyear)
```

where:

month	is the number of the month between 1 and 12 (no default)
iday	is the number of the day between 1 and 31 (no default)
iyear	is the number of the year between 72 (representing 1972) and 199 (representing 2099), or 1972 through 2099. No default

Function result:

Value	**Meaning**
i = -1	Invalid argument
= 1	Sunday
= 2	Monday
= 3	Tuesday
= 4	Wednesday
= 5	Thursday
= 6	Friday
= 7	Saturday

Example:
See DATE.

JADD

JADD computes the sum of two INTEGER*4 values.

Form:

```
CALL JADD (jopr1,jopr2,jres)
 i  = JADD (jopr1,jopr2,jres)
```

where:

jopr1 is an INTEGER*4 variable

jopr2 is an INTEGER*4 variable

jres is an INTEGER*4 variable that receives the sum of *jopr1* and *jopr2*.

Function Results:

Value	Meaning
i = 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.
= -1	Normal return; the result is negative.

Errors:

Value	Meaning
i = -2	An overflow occurred while computing the result.

Unpredictable results will occur if any argument is omitted.

Example:

```
       Program FJADD !FORTRAN IV
C       using JJCVT and TIMASC to display results
C
        Integer*4 HOUR1         !value of 1 hour
        Integer*4 HOUR12        !value of 12 hours
        Integer*4 JA, JB, JC    !variables
        Logical*1 ASCII(9)      !variables
        Data ASCII(9) /0/       !terminate string with null
C
C       init "constants"
C
        Call JTIME (1, 0, 0, 0, HOUR1)
        Call JTIME (12, 0, 0, 0, HOUR12)
C
C       convert from RT-11 time format to I*4 format
        Call JJCVT (HOUR1)
        Call JJCVT (HOUR12)
        Call JMOV (HOUR1, JA)   !JA = 1hr
        Call JMOV (HOUR12, JB)  !JB = 12hr
        Call JADD (JA, JB, JC)  !JC = JA + JB
        Call JJCVT (JC)         !back to time format
        Call TIMASC (JC, ASCII) !display results
        Call PRINT (ASCII)      !...
        End
```

JAFIX

JAFIX converts a REAL*4 value to INTEGER*4.

Form:

```
CALL JAFIX (asrc,jres)
 i = JAFIX (asrc,jres)
```

where:

asrc is a REAL*4 variable, constant, or expression to be converted to INTEGER*4

jres is an INTEGER*4 variable that is to contain the result of the conversion

Function Results:

Value	Meaning
i = 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.
= -1	Normal return; the result is negative.

Errors:

Value	Meaning
i = -2	An overflow occurred while computing the result.

Unpredictable results will occur if any argument is omitted.

Example:

```
        Program FJAFIX !FORTRAN IV
C        using JJCVT and TIMASC to display results
         Real*4 RTRY              !test value
         Real*4 RPTRY             !previous test value
         Real*4 RNEW              !reconverted value
         Integer*4 JTRY           !integer equivalent
         Integer*4 JA, JB, JC     !variables
         Logical*1 ASCII(9)       !variables
         Data ASCII(9) /0/        !terminate string with null
         RTRY = 1.                !start at the beginning
         RPTRY = RTRY             !remember last value
100      Continue
             IERR = JAFIX (RTRY, JTRY) ! convert to I*4
             If (IERR .eq. -2) Type *, '?FJAFIX-W-Overflow', RTRY
             RNEW = AJFLT (JTRY) !convert back
             If (RNEW .ne. RTRY) Go To 200 !lost some bits
             RPTRY = RTRY
             RTRY = RTRY * 2.
             Go To 100
200      Continue
         Type *, RTRY, RNEW
         End
```

JCMP

The JCMP function compares two INTEGER*4 values and returns an INTEGER*2 value that reflects the signed comparison result.

Form:

i = JCMP (jopr1,jopr2)

where:

jopr1 is the INTEGER*4 variable or array element that is the first operand in the comparison

jopr2 is the INTEGER*4 variable or array element that is the second operand in the comparison

Function Results:

Value	Meaning
i = 0	If *jobpr1* is equal to *jopr2*
= 1	If *jopr1* is greater than *jopr2.*
= -1	If *jopr1* is less than *jopr2.*

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FJCMP   !FORTRAN IV
C
C       Demonstrate J series Integer*4 routines
C       using JJCVT and TIMASC to display results
C
        Integer*4 HOUR1         !value of 1 hour
        Integer*4 HOUR12        !value of 12 hours
        Integer*4 SECON1        !value of 1 second
        Integer*4 JA, JB, JC    !variables
        Logical*1 ASCII(9)      !variables
        Data ASCII(9) /0/       !terminate string with null
C
C       init "constants"
C
        Call JTIME (1, 0, 0, 0, HOUR1)
        Call JTIME (12, 0, 0, 0, HOUR12)
        Call JTIME (0, 0, 1, 0, SECON1)
C
C       convert from RT-11 time format to I*4 format
C
        Call JJCVT (HOUR1)
        Call JJCVT (HOUR12)
        Call JJCVT (SECON1)
C
        Call JMOV (HOUR12, JA)  !JA = 12hr
        Call JMOV (SECON1, JB)  !JB = 1sec
        Call JADD (JA, JB, JC)  !JC = JA + JB
```

```
        If (JCMP (HOUR12, JC) .ge. 0)
        1 Stop '12:00:00 ge 12:00:01'
        Call JJCVT (JC)          !back to time format
        Call TIMASC (JC, ASCII) !display results
        Call PRINT (ASCII)       !...
C
        Call JSUB (JA, JB, JC)  !JC = JA - JB
        If (JCMP (HOUR12, JC) .le. 0)
        1 Stop '12:00:00 le 11:59:59'
        Call JJCVT (JC)          !back to time format
        Call TIMASC (JC, ASCII) !display results
        Call PRINT (ASCII)       !...
        End
```

JDFIX

The JDFIX function converts a REAL*8 (DOUBLE PRECISION) value to INTEGER*4.

Form:

i = JDFIX (dsrc,jres)

where:

dsrc is a REAL*8 variable, constant, or expression to be converted to INTEGER*4

jres is an INTEGER*4 variable to contain the conversion result

Function Results:

Value	Meaning
i = 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.
= -1	Normal return; the result is negative.

Errors:

Value	Meaning
i = -2	An overflow occurred while computing the result.

Unpredictable results will occur if any argument is omitted.

Example:

```
       Program FJDFIX !FORTRAN IV
C       using JJCVT and TIMASC to display results
        Real*8 RTRY             !test value
        Real*8 RPTRY            !previous test value
        Real*8 RNEW             !reconverted value
        Integer*4 JTRY          !integer equivalent
        Integer*4 JA, JB, JC    !variables
        Logical*1 ASCII(9)      !variables
        Data ASCII(9) /0/       !terminate string with null
        RTRY = 1.               !start at the beginning
        RPTRY = RTRY            !remember last value
100     Continue
            IERR = JDFIX (RTRY, JTRY) ! convert to I*4
            If (IERR .eq. -2) Type *, '?FJDFIX-W-Overflow', RTRY
            RNEW = AJFLT (JTRY) !convert back
            If (RNEW .ne. RTRY) Go To 200 !lost some bits
            RPTRY = RTRY
            RTRY = RTRY * 2.
            Go To 100
200     Continue
        Type *, RTRY, RNEW
        End
```

JDIV

JDIV computes the quotient of two INTEGER*4 values.

Form:

```
CALL JDIV (jopr1,jopr2,jres[,jrem])
 i = JDIV (jopr1,jopr2,jres[,jrem])
```

where:

jopr1 is an INTEGER*4 variable that is the dividend of the operation

jopr2 is an INTEGER*4 variable that is the divisor of *jopr1*

jres is an INTEGER*4 variable that receives the quotient of the operation; that is, *jres=jopr1/jopr2*.

jrem is an INTEGER*4 variable that receives the remainder of the operation. The sign is the same as that for *jopr1*

Function Results:

Value	Meaning
i = 0	Normal return; the quotient is 0.
= 1	Normal return; the quotient is positive.
= -1	Normal return; the quotient is negative.

Errors:

Value	Meaning
i = -3	An attempt was made to divide by 0.

Unpredictable results will occur if any required argument is omitted.

Example:

```
        Program FJDIV   !FORTRAN IV
C
C       Demonstrate J series Integer*4 routines
C       using JJCVT and TIMASC to display results
C
        Integer*4 JA, JB                !variable
        Integer*4 JHOUR, JMIN, JSEC     !more vars
        Integer*4 J60                   !constant
        Logical*1 ASCII(8)
C
C       init "constants"
C
        Call JTIME (23, 59, 59, 59, JA)
        Call JICVT (60, J60)
C
C       convert from RT-11 time format to I*4 format
C
        Call JMOV (JA, JB)              !save time format version
        Call JJCVT (JA)                 !make I*4 format version
C
```

JDIV

```
C       split out ticks
C
        Call TIMASC (JB, ASCII)         !convert w/SYSLIB
C
C       convert w/JDIV
C
        Call JDIV (JA, J60, JA)         !dump ticks
        Call JDIV (JA, J60, JA, JSEC)   !get seconds
        Call JDIV (JA, J60, JHOUR, JMIN) !get minutes and hours
C
        Type 100, JHOUR, JMIN, JSEC, ASCII
100     Format (' ', i2, ':', i2, ':', i2, '  ', 8a1)
        End
```

JICVT

JICVT converts a specified INTEGER*2 value to INTEGER*4.

Form:

CALL JICVT (isrc[,jres])
i = JICVT (isrc[,jres])

where:

isrc is the INTEGER*2 quantity to be converted

jres is the INTEGER*4 variable or array element to receive the result

Function Results:

Value	Meaning
i = 0	Normal return; the result is 0.
= 1	Normal return; the result is positive.
= -1	Normal return; the result is negative.

Errors:

Unpredictable results will occur if any argument is omitted.

Example:

```
       Program FJICVT  !FORTRAN IV
C
C       Demonstrate J series Integer*4 routines
C
        Integer*4 JA, JB                !variables
        Integer*2 IA(2), IB(2)          !overlay vars
        Equivalence (JA, IA(1)), (JB, IB(1))
        Data IA /12345, 23456/          !junk patterns
        Data IB /31234, 11111/
C
        Call JICVT (+32000, JA)
        Call JICVT (-2, JB)
C
        Type 100, IA, IB
100     Format (' ', 2o7, '   ', 2o7)
        End
```

JJCVT

The JJCVT subroutine interchanges words of an INTEGER*4 value to form an internal format time (or vice versa) whenever the INTEGER*4 variable is to be used as an argument in a timer-support function such as ITWAIT. When a two-word internal format time is specified to a function such as ITWAIT, it must have the high-order time as the first word and the low-order time as the second word.

Form:

```
CALL JJCVT (jsrc)
```

where:

jsrc is the INTEGER*4 variable whose contents are to be interchanged

Errors:

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See JDIV.

JMOV

JMOV assigns the value of an INTEGER*4 variable to another INTEGER*4 variable and returns the sign of the value moved.

Form:

```
CALL  JMOV  (jsrc,jdest)
 i =  JMOV  (jsrc,jdest)
```

where:

jsrc is the INTEGER*4 variable whose contents are to be moved

jdest is the INTEGER*4 variable that is the target of the assignment

Function Result:

The value of the function is an INTEGER*2 value that represents the sign of the result as follows:

Value	Meaning
i = 0	Normal return; the result is 0.
= 1	Normal return; the result is positive.
= -1	Normal return; the result is negative.

Errors:
Unpredictable results will occur if any argument is omitted.

Example:
See JCMP.

JMUL

JMUL computes the product of two INTEGER*4 values.

Form:

```
CALL JMUL (jopr1,jopr2,jres)
 i =  JMUL (jopr1,jopr2,jres)
```

where:

jopr1	is an INTEGER*4 variable that is the multiplicand
jopr2	is an INTEGER*4 variable that is the multiplier
jres	is an INTEGER*4 variable that receives the product of the operation

Function Results:

Value	Meaning
i = 0	Normal return; the product is 0.
= 1	Normal return; the product is positive.
= -1	Normal return; the product is negative.

Errors:

Value	Meaning
i = -2	An overflow occurred while computing the result.

Unpredictable results will occur if any argument is omitted.

Example:

```
       Program FJMUL   !FORTRAN IV
C
C       Demonstrate J series Integer*4 routines
C       using JJCVT and TIMASC to display results
C
        Integer*4 JA, JB, JC            !variable
        Integer*4 JHOUR, JMIN, JSEC, JTICK !more vars
        Integer*4 J60, J3600            !constant
        Logical*1 ASCII1(8), ASCII2(8)
C
C       init "constants"
C
        Call JTIME (23, 59, 59, 59, JA)
        Call JICVT (59, JTICK)
        Call JICVT (59, JSEC)
        Call JICVT (59, JMIN)
        Call JICVT (23, JHOUR)
        Call JICVT (60, J60)
        Call JICVT (3600, J3600)
C
C       convert w/JMUL
C
        Call JMOV (JTICK, JB)           !put ticks in accum
        Call JMUL (JSEC, J60, JC)       !calc sec value
```

```
        Call JADD (JC, JB, JB)          !add in
        Call JMUL (JMIN, J3600, JC)     !calc min value
        Call JADD (JC, JB, JB)          !add in
        Call JMUL (JHOUR, J3600, JC)    !calc hour value
        Call JMUL (JC, J60, JC)         !...
        Call JADD (JC, JB, JB)          !add in
        Call JJCVT (JB)                 !convert to time format
C
        Call TIMASC (JA, ASCII1)
        Call TIMASC (JB, ASCII2)
        Type 100, ASCII1, ASCII2
100     Format (' ', 8a1, '   ', 8a1)
        End
```

JREAD/JREADC/JREADF/JREADW

JREAD/JREADC/JREADF/JREADW use non-file-structured access to transfer into memory a specified number of words from an MSCP device. They are therefore especially useful because they use a 32-bit starting block number and can read from any block on any DU device.

Use the IQSET function to allocate the extra queue element required with JREAD, JREADC, JREADF functions. JREADW doesn't require an extra queue element since it is synchronous.

JREAD

The JREAD function transfers into memory a specified number of words from an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor returns control to the user program immediately after the JREAD function is initiated. No special action is taken when the transfer is completed.

Form:

```
CALL JREAD (wcnt,buff,jblock,chan[,area][,BMODE=strg])
 i = JREAD (wcnt,buff,jblock,chan[,area][,BMODE=strg])
```

where:

wcnt is the integer number of words to be transferred

buff is an array to be used as the buffer; that array must contain at least *wcnt* words

jblock is a 2-word (32-bit) starting block number of the MSCP device to be read. The first word contains the low order bits. The second word contains the high order bits.

For all currently supported Digital MSCP devices, the high four bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable, and can be manipulated as such.) The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0.

The *jblock* argument refers to a physical block when the MSCP disk has not been partitioned, or to a logical block when the MSCP disk has been partitioned. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77).

	The *jblock* argument must be updated (incremented) when necessary. For example, if the program is reading two sequential blocks at a time, *jblock* should be incremented by two for each read
chan	is the integer specification for the RT–11 channel to be used
area	is accepted and ignored
BMODE=strg	Specify one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value is used to specify the mapping mode for the *buff* argument.

Issue an IWAIT function when the user program needs to access the data read on the specified channel. IWAIT makes sure that the JREAD operation has been completed. IWAIT indicates if a hard error occurs during the transfer.

Errors:

Value	Meaning
i = 0	Normal return.
= -2	Hardware error occurred on channel on last completed operation.
= -3	Specified channel is not open.
= -4	Invalid request (attempted file access), channel is opened to a file.
= -5	Channel not open as a non-file-structured device.
= -6	Address translation not available in monitor.
= -19	Invalid BMODE value.
= -257	Required argument missing.

Example:
See JREADW. See also RCVD.

JREADC

The JREADC function transfers into memory a specified number of words from an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor returns control to the user program immediately after the JREADC function is initiated. When the operation is complete, the monitor enters the specified assembly language routine (crtn) as an asynchronous completion routine.

Form:

```
CALL JREADC (wcnt,buff,jblock,chan,[,area],crtn[,BMODE=strg][,CMODE=strg])
 i = JREADC (wcnt,buff,jblock,chan,[,area],crtn[,BMODE=strg][,CMODE=strg])
```

where:

wcnt is the integer number of words to be transferred

buff is an array to be used as the buffer; that array must contain at least *wcnt* words

jblock is a 2-word (32-bit) starting block number of the MSCP device to be read. The first word contains the low order bits. The second word contains the high order bits. For all currently supported Digital MSCP devices, the high four bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable, and can be manipulated as such.)
The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0. The *jblock* argument refers to a physical block when the MSCP disk has not been partitioned, or to a logical block when the MSCP disk has been partitioned. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77).
The *jblock* argument must be updated (incremented) when necessary. For example, if the program is reading two sequential blocks at a time, *jblock* should be incremented by two for each read

chan is the integer specification for the RT–11 channel to be used

area is accepted and ignored

crtn is an assembly language routine to be activated when the transfer is complete. That routine must be specified in the EXTERNAL statement in the FORTRAN routine that issues the JREADC function

BMODE=strg Specify one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value is used to specify the mapping mode for the BUFF statement.

CMODE=strg Specifying *strg* as string "S" specifies Supervisor address.

Errors:

Value	Meaning
i = 0	Normal return.
= -2	Hardware error occurred on channel on last completed operation.
= -3	Specified channel is not open.
= -4	Invalid request (attempted file access), channel is opened to a file.

= -5	Channel not open as a non-file-structured device.
= -6	Address translation not available in monitor.
= -19	Invalid BMODE or CMODE value.
= -257	Required argument missing.

Example:

See JREADW. See also RCVDC.

JREADF

JREADF transfers into memory a specified number of words from an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor returns control to the user program immediately after JREADF is initiated. When the operation is complete, the monitor enters the specified FORTRAN subprogram (*frtn*) as an asynchronous completion routine.

Form:

```
CALL READF (wcnt,buff,jblock,chan,[,area],lblk,frtn)
i = JREADF (wcnt,buff,jblock,chan,[,area],lblk,frtn)
```

where:

wcnt	is the integer number of words to be transferred
buff	is an array to be used as the buffer; that array must contain at least *wcnt* words
jblock	is a 2-word (32-bit) starting block number of the MSCP device to be read. The first word contains the low order bits. The second word contains the high order bits. For all currently supported Digital MSCP devices, the high four bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable, and can be manipulated as such.) The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0.

The *jblock* argument refers to a physical block when the MSCP disk has not been partitioned, or to a logical block when the MSCP disk has been partitioned. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77).

The *jblock* argument must be updated (incremented) when necessary. For example, if the program is reading two sequential blocks at a time, *jblock* should be incremented by two for each read

chan is the integer specification for the RT–11 channel to be used

area is accepted and ignored

lblk is a 4-word area to be set aside for link information; that area must not be modified by the FORTRAN program or swapped over by the USR. That area can be reused by other FORTRAN completion functions when *frtn* has returned

frtn is a FORTRAN routine activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the routine that issues the JREADF call

Errors:

Value	Meaning
i = 0	Normal return.
= -2	Hardware error occurred on channel on last completed operation.
= -3	Specified channel is not open.
= -4	Invalid request (attempted file access), channel is opened to a file.
= -5	Channel not open as a non-file-structured device.
= -6	Address translation not available in monitor.
= -257	Required argument missing.

Example:

See JREADW. See also RCVDF.

JREADW

JREADW transfers into memory a specified number of words from an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor returns control to the user program when the transfer is complete or when an error is detected.

Form:

```
CALL JREADW(wcnt,buff,jblock,chan[,area][,BMODE=strg])
 i = JREADW(wcnt,buff,jblock,chan[,area][,BMODE=strg])
```

where:

wcnt	is the integer number of words to be transferred
buff	is an array to be used as the buffer; that array must contain at least *wcnt* words
jblock	is a 2-word (32-bit) starting block number of the MSCP device to be read. The first word contains the low order bits. The second word contains the high order bits. For all currently supported Digital MSCP devices, the high 4 bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable and can be manipulated as such.) The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0. The *jblock* argument refers to a physical block when the MSCP disk has not been partitioned or to a logical block when the MSCP disk has been partitioned. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77). Do not specify a block number higher than the physical size of the device (the physical size of an MSCP device is returned by the IGTDUS function). The *jblock* argument must be updated (incremented) when necessary. For example, if the program is writing two sequential blocks at a time, *jblock* should be incremented by two for each write
chan	is the integer specification for the RT–11 channel to be used
area	is accepted and ignored
BMODE=strg	Specify one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value is used to specify the mapping mode for the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return.
= -2	Hardware error occurred on channel on last completed operation.
= -3	Specified channel is not open.
= -4	Invalid request (attempted file access), channel is opened to a file.

= -5	Channel not open as a non-file-structured device.
= -6	Address translation not available in monitor.
= -19	Invalid BMODE value.
= -257	Required argument missing.

Example:

```
        Program FJREAD
C
C       Read the first block following the last user block
C
C       NOTE: writing in this area would be a disaster!!!!
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLK(4)               !device name
        Data DBLK /3rSY , 0, 0, 0/      !no file name
        Integer*2 REPLY(7)              !info from IGTDUS
        Integer*4 BLK                   !block number
        Equivalence (BLK, REPLY(3))     !overlay in reply area
        Parameter WRKSIZ = 80           !size of work area for IGTDUS
        Integer*2 WORK(WRKSIZ)          !work area for IGTDUS
        Parameter WCNT = 256            !I/O word count
        Integer*2 BUFFER(WCNT)          !I/O buffer
C
        ICHAN = IGETC()                 !get a channel
        IERR = IGTDUS (DBLK, ICHAN, REPLY, , , WORK, WRKSIZ)
        If (IERR .ne. 0) Go To 1000     !failed
        Call LOOKUP (ICHAN, DBLK)       !open up system device
        IERR = JREADW (WCNT, BUFFER, BLK, ICHAN)
        If (IERR .ne. 0) Go To 1100     !failed
        Type 100, BLK
100     Format (' ', 'Block number = ', i12)
        Do 500 I = 0, 255/8
            Type 101, I*16, (BUFFER(J), J=I, I+7)
101         Format (' ', o3.3, 8o8.6)
500         Continue
        Close (Unit=5)
        Call EXIT (SUCCS)
C
1000    Type *, '?FJREAD-F-IGTDUS failed, code = ', IERR
        Call EXIT (FATAL)
1100    Type *, '?FJREAD-F-JREAD failed, code = ', IERR
        Call EXIT (FATAL)
        End
```

JSUB

JSUB computes the difference between two INTEGER*4 values.

Form:

```
CALL JSUB (jopr1,jopr2,jres)
 i =  JSUB (jopr1,jopr2,jres)
```

where:

jopr1 is an INTEGER*4 variable that is the minuend of the operation.

jopr2 is an INTEGER*4 variable that is the subtrahend of the operation.

jres is an INTEGER*4 variable that is to receive the difference between *jopr1* and *jopr2*; that is, *jres=jopr1-jopr2*.

Function Results:

Value	Meaning
i = 0	Normal return; the result is 0.
= 1	Normal return; the result is positive.
= -1	Normal return; the result is negative.

Errors:

Value	Meaning
i = -2	An overflow occurred while computing the result.

Unpredictable results will occur if any argument is omitted.

Example:

```
       Program FJSUB !FORTRAN IV
C
C       using JJCVT and TIMASC to display results
C
        Integer*4 HOUR1         !value of 1 hour
        Integer*4 HOUR12        !value of 12 hours
        Integer*4 JA, JB, JC    !variables
        Logical*1 ASCII(9)      !variables
        Data ASCII(9) /0/       !terminate string with null
C
C       init "constants"
C
        Call JTIME (1, 0, 0, 0, HOUR1)
        Call JTIME (12, 0, 0, 0, HOUR12)
C
C       convert from RT-11 time format to I*4 format
C
        Call JJCVT (HOUR1)
        Call JJCVT (HOUR12)
        Call JMOV (HOUR1, JA)   !JA = 1hr
        Call JMOV (HOUR12, JB)  !JB = 12hr
        Call JSUB (JB, JA, JC)  !JC = JB + JA
```

```
Call JJCVT (JC)          !back to time format
Call TIMASC (JC, ASCII) !display results
Call PRINT (ASCII)       !...
End
```

JTIME

The JTIME subroutine converts the time specified to the internal two-word format time.

Form:

CALL JTIME (hrs,min,sec,tick,time)

where:

hrs	is the integer number of hours
min	is the integer number of minutes
sec	is the integer number of seconds
tick	is the integer number of ticks (1/60 of a second for 60-Hz clocks; 1/50 of a second for 50-Hz clocks)
time	is the two-word area to receive the internal format time: time(1) is the high-order time; time(2) is the low-order time.

Errors:
Unpredictable results will occur if any argument is omitted.

Example:
See JMUL.

JWRITE/JWRITC/JWRITF/JWRITW

JWRITE/JWRITC/JWRITF/JWRITW, issued as functions or subroutines, use non-file-structured access to transfer a specified number of words from memory to an MSCP (DU) device. They use a 32-bit starting block number and can, therefore, write to any block on any DU device. JWRITE, JWRITC, and JWRITW have optional arguments that specify mapping for the *buff* argument.

When you don't know the physical size of an MSCP device, use IGTDUS to determine the size of that device, then specify a starting block that is lower than the device size returned by IGTDUS.

CAUTION

If you inadvertently specify a block number higher than the device size returned by IGTDUS, you corrupt formatting and bad-block replacement information contained on the device, making it unusable. You must reformat the device, removing all information contained on it.

Use the IQSET function to allocate the extra queue element required with JWRITE, JWRITC, JWRITF functions. JWRITW does not require an extra queue element, since it is synchronous.

JWRITE

The JWRITE function transfers a specified number of words from memory to an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor returns control to the user program immediately after queuing the request. No special action is taken upon completion of the operation.

Form:

```
CALL JWRITE (wcnt,buff,jblock,chan[,area][,BMODE=strg])
 i = JWRITE (wcnt,buff,jblock,chan[,area][,BMODE=strg])
```

where:

wcnt is the integer number of words to be transferred

buff is an array to be used as the output buffer

jblock is a 2-word (32-bit) starting block number of the MSCP device to be written. The first word contains the low order bits. The second word contains the high order bits. For all currently supported Digital MSCP devices, the high 4 bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable and can be manipulated as such.) The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0. The *jblock* argument refers to a physical block when the channel is opened on the first partition or to a logical block when the channel is opened on any other partition. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77). Do not specify a block number higher than the physical size of the device (the physical size of an MSCP device is returned by the IGTDUS function). The *jblock* argument must be updated (incremented) when necessary. For example, if the program is writing two sequential blocks at a time, *jblock* should be incremented by two for each write

chan is the integer specification for the RT–11 channel to be used. Obtain this channel through an IGETC call or use channel 16(decimal) or higher, if you have obtained extra channels with an ICDFN call

area is accepted and ignored

BMODE=strg Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value specifies the mapping mode of the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return
= -2	Hardware error occurred on channel on last completed operation
= -3	Channel is not open
= -4	Invalid request (attempted file access), channel is opened to a file
= -5	Channel not open as non-file-structured device.
= -6	Address translation not available in monitor.
= -19	Invalid BMODE argument.
= -257	Required argument missing.

Example:
See JREADW. See also SDAT.

JWRITC

JWRITC transfers a specified number of words from memory to an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor queues the request and returns control to the user program. When the transfer is complete, the monitor enters the specified assembly language routine (*crtn*) as an asynchronous completion routine.

Form:

```
CALL JWRITC (wcnt,buff,jblock,chan,[,area],crtn[,BMODE=strg][,CMODE=strg])
 i = JWRITC (wcnt,buff,jblock,chan,[,area],crtn[,BMODE=strg][,CMODE=strg])
```

where:

wcnt is the integer number of words to be transferred

buff is an array to be used as the output buffer

jblock is a 2-word (32-bit) starting block number of the MSCP device to be written. The first word contains the low order bits. The second word contains the high order bits. For all currently supported Digital MSCP devices, the high 4 bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable and can be manipulated as such.) The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0. The *jblock* argument refers to a physical block when the channel is opened on the first partition or to a logical block when the channel is opened on any other partition. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77). Do not specify a block number higher than the physical size of the device (the physical size of an MSCP device is returned by the IGTDUS function). The *jblock* argument must be updated (incremented) when necessary. For example, if the program is writing two sequential blocks at a time, *jblock* should be incremented by two for each write.

chan is the integer specification for the RT–11 channel to be used. Obtain this channel through an IGETC call or use channel 16(decimal) or higher, if you have obtained extra channels with an ICDFN call

area is accepted and ignored

BMODE=strg Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.

CMODE=strg Specifying *strg* as string "S" specifies a Supervisor address.

crtn is an assembly language routine to be activated when the transfer is complete. That routine must be specified in the EXTERNAL statement in the FORTRAN routine that issues the JWRITC function.

Errors:

Value	Meaning
i = 0	Normal return
= -2	Hardware error occurred on channel on last completed operation
= -3	Channel is not open
= -4	Invalid request (attempted file access), channel is opened to a file
= -5	Channel not open as non-file-structured device.
= -6	Address translation not available in monitor.
= -19	Invalid BMODE or CMODE argument.
= -257	Required argument missing.

Example:

See JREADW. See also SDATC.

JWRITF

JWRITF transfers a specified number of words from memory to an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor returns control to the user program immediately after queuing the request. When the transfer is complete, the monitor enters the specified FORTRAN subprogram (crtn) as an asynchronous completion routine.

Form:

```
CALL JWRITF (wcnt,buff,jblock,chan,[,area],lblk,frtn)
 i = JWRITF (wcnt,buff,jblock,chan,[,area],lblk,frtn)
```

where:

wcnt is the integer number of words to be transferred

buff is an array to be used as the output buffer

jblock is a 2-word (32-bit) starting block number of the MSCP device to be written. The first word contains the low order bits. The second word contains the high order bits. For all currently supported Digital MSCP devices, the high 4 bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable and can be manipulated as such.) The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0. The *jblock* argument refers to a physical block when the channel is opened on the first partition or to a logical block when the channel is opened on any other partition. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77). Do not specify a block number higher than the physical size of the device (the physical size of an MSCP device is returned by the IGTDUS function). The *jblock* argument must be updated (incremented) when necessary. For example, if the program is writing two sequential blocks at a time, *jblock* should be incremented by two for each write

chan is the integer specification for the RT–11 channel to be used. Obtain this channel through an IGETC call or use channel 16(decimal) or higher, if you have obtained extra channels with an ICDFN call

area is accepted and ignored

lblk is a 4-word area to be set aside for link information; this area must not be modified by the FORTRAN program or swapped over by the USR. This area can be reused by other FORTRAN completion functions when crtn has returned

frtn is a FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the routine that issues the JWRITF call

Errors:

Value	Meaning
i = 0	Normal return
= -2	Hardware error occurred on channel on last completed operation
= -3	Channel is not open
= -4	Invalid request (attempted file access), channel is opened to a file
= -5	Channel not open as non-file-structured device.
= -6	Address translation not available in monitor.
= -257	Required argument missing.

Example:
See JREADW. See also SDATF.

JWRITW

JWRITW transfers a specified number of words from memory to an MSCP device associated with the indicated channel. The channel must be opened to the MSCP device in a non-file-structured manner. The monitor returns control to the user program when the transfer is complete.

Form:

```
CALL JWRITW (wcnt,buff,jblock,chan[,area][,BMODE=strg])
 i = JWRITW (wcnt,buff,jblock,chan[,area][,BMODE=strg])
```

where:

wcnt is the integer number of words to be transferred

buff is an array to be used as the output buffer

jblock is a 2-word (32-bit) starting block number of the MSCP device to be written. The first word contains the low order bits. The second word contains the high order bits. For all currently supported Digital MSCP devices, the high 4 bits of the second word must be zero. (In FORTRAN–77, *jblock* can be expressed as an INTEGER*4 variable and can be manipulated as such.) The first block of the MSCP device is physical block 0. The first block of each MSCP device partition is logical block 0. The *jblock* argument is offset from physical or logical block number 0. The *jblock* argument refers to a physical block when the channel is opened on the first partition or to a logical block when the channel is opened on any other partition. When an MSCP disk has been partitioned, the block you address with the *jblock* argument depends on which partition is assigned to the RT–11 unit (DU0 through DU7 or D10 through D77). Do not specify a block number higher than the physical size of the device (the physical size of an MSCP device is returned by the IGTDUS function). The *jblock* argument must be updated (incremented) when necessary. For example, if the program is writing two sequential blocks at a time, *jblock* should be incremented by two for each write

chan is the integer specification for the RT–11 channel to be used. Obtain this channel through an IGETC call or use channel 16(decimal) or higher, if you have obtained extra channels with an ICDFN call

area is accepted and ignored

BMODE=strg Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return
-2	Hardware error occurred on channel on last completed operation
-3	Channel is not open
-4	Invalid request (attempted file access), channel is opened to a file
-5	Channel not open as non-file-structured device.
-6	Address translation not available in monitor.
-19	Invalid BMODE argument.
-257	Required argument missing.

Example:
See JREADW. See also SDATW.

KPEEK

The KPEEK function returns the contents of the word located at a specified 16-bit address. The function can examine any location in Kernel memory. For description of PS, see KPOKE.

Form:

i = KPEEK (iaddr[,ierr])

where:

iaddr is the integer specification of the 16-bit (Kernel-mapped) address to be examined. As a special case, if the value -2 is specified for *iaddr*, KPEEK returns the PS associated with the calling program, rather than the monitor. However, condition codes are undefined.

ierr is a returned error condition.

Function result:

Value	Meaning
i = n	Value (n) of the location in Kernel mapping.

Errors:

Value	Meaning
ierr = 0	Normal return
= -2	Odd or nonexistent address

Error message *TRAP $MSARG* will display if argument *iaddr* is missing.

Example:

```
        Program FKPEEK
C
C       Demo program to show using KPEEK and KPOKE accessing the PS
C
C       WARNING: invoking KPEEK uses an EMT which is executed at PR0!
C
        Integer*2 PS, PR7       !Proc status register, priority 7 mask
        Data PS /"177776/, PR7 /"000340/
        Integer*2 OLDPS, NEWPS  !copies of PS
        Integer*2 SHARED        !location shared with a
        Common /STATUS/ SHARED  !  completion routine
        Integer*2 OLDSHR        !local copy of old value
C
        OLDPS = KPOKE (PS, PR7, 'BIS') !Set PS to PR7 (BIS)
        NEWPS = KPEEK (PS)      !get modified PS
        OLDSHR = SHARED         !get current value
        SHARED = SHARED + 3     !change shared location, protected
C                               !  with PR7 from interference
        CALL KPOKE (PS, OLDPS, 'MOV') !Put old PS back
        Type 100, OLDPS, NEWPS, KPEEK (PS) !display PSs
100     Format (' ', 3o10)
        End
```

KPOKE

KPOKE, either as a function or as a subroutine, stores a specified 16-bit integer value into a 16-bit (Kernel mapped) address. The subroutine can store values at any location in Kernel memory.

Form:

```
CALL KPOKE (iaddr,ivalue[,type][,ierr])
 i =  KPOKE (iaddr,ivalue[,type][,ierr])
```

where:

iaddr is the integer specification of the 16-bit address to be modified. As a special case, if the value -2 is specified for *iaddr*, KPEEK returns the PS associated with the calling program, rather than the monitor. However, condition codes are undefined.

ivalue is the integer value to be stored in the address specified by the *iaddr* argument.

type is the operation to be used to modify the address to the value specified by the *ivalue* argument:

- MOV = Replace with a MOV operation (default).
- BIC = Change with a BIC operation.
- BIS = Change with a BIS operation.

ierr is a returned error condition.

Function result:

Value	Meaning
i = n	Value (n) of the location in Kernel mapping.

Errors:

Value	Meaning
i = 0	Normal return
= -2	Odd or nonexistent address (has meaning only if *ierr* is specified.

Error message *TRAP $MSARG* will display if any required argument is missing or if *type* argument is incorrect.

Altering the Processor Status Word

The KPOKE request can successfully alter priority bits in the processor status (PS) word:

- A KPOKE function returns the contents of the PS with undefined condition codes.
- A KPOKE can modify all bits in the PS except the 000020 (trace trap) and 140000 (current mode) bits. However, modifying the 000400 (instruction suspension)

or 004000 (register set) bits can cause unexpected results and, although not prohibited, is not recommended.

Modifying the priority bits is supported. However, changing processor priority with KPOKE automatically lowers processor priority to zero (PR0) during the time KPOKE is executing. Therefore, a period of lowest processor priority exists between the time the processor is running at a given priority and the time the processor priority change takes effect.

Setting the carry bit is not recommended as it causes KPOKE to return an error.

- The priority bits of the modified PS are preserved at the completion of KPOKE.

Example:
See KPEEK.

LEN

The LEN function returns the number of characters currently in the string contained in a specified array. This number is computed as the number of characters preceding the first null byte encountered. If the specified array contains only a null string, a value of 0 is returned.

Form:

```
i = LEN (a)
```

where:

a specifies the array containing the string, which must be terminated by a null byte

Errors:
Unpredictable results will occur if argument *a* is omitted.

Example:
See SDAT.

LOCK/UNLOCK

The LOCK subroutine, in a multijob environment, keeps the USR in memory for a series of operations involving various RT–11 file management functions. The UNLOCK subroutine releases the User Service Routine (USR) from memory, if it was placed there by the LOCK routine.

LOCK

If all the conditions that cause swapping are satisfied, a portion of the user program is written out to the disk file SWAP.SYS and the USR is loaded. Otherwise, the USR in memory is used, and no swapping occurs. The USR is not released until an UNLOCK is given. (Note that in a multijob system, calling the CSI can also perform an implicit UNLOCK.) To save time in swapping, a program that makes multiple USR requests can LOCK the USR in memory, make all the requests, and then UNLOCK USR.

Form:

```
CALL LOCK
```

In a multijob environment, LOCK inhibits another job from using USR. USR should be locked only for as long as necessary.

Notes

If any job does a LOCK, it can cause the USR to be unavailable for other jobs for a considerable period of time. The USR is not reentrant and only one job has use of the USR at a time, which should be considered for systems requiring concurrent foreground and background jobs. This is particularly true when magtape and/or cassette are active.

File operations by the USR require a sequential search of the tape for magtape and cassette. This could lock out the foreground job for a long time while the background job does a tape operation. The programmer should keep this in mind when designing such systems. The multijob monitors supply the ITLOCK routine, which permits the job to check for the availability of the USR.

After a LOCK has been executed, the UNLOCK routine must be executed to release the USR from memory. The LOCK/UNLOCK routines are complementary and must be matched. That is, if three LOCKs are issued, at least three UNLOCKs must be done, otherwise the USR is not released. More UNLOCKs than LOCKs can occur without error; the extra UNLOCKs are ignored.

The LOCK call must not come from within the area into which the USR will be swapped. If this should occur, the return from the USR request would not be to the user program, but to the USR itself, since the LOCK function causes part of the user program to be saved on disk and replaced in memory by the USR. Furthermore, subroutines, variables, and arrays in the area where the USR is swapping should not be referenced while the USR is locked in memory.

LOCK/UNLOCK

Once a LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an UNLOCK is done.

LOCK cannot be called from a completion or interrupt routine.

If a SET USR NOSWAP command has been issued, LOCK and UNLOCK do not cause the USR to swap. However, LOCK still inhibits the other job from using the USR, and UNLOCK allows the other job access to the USR.

The USR cannot accept argument lists, such as device file name specifications, located in the area into which it has been locked.

Errors:
None.

Example:

```
        Program FLOCK
C
C       Demo ITLOCK, LOCK, & UNLOCK
C
        If (ITLOCK () .eq. 0) Go To 100 !USR available
C
C       do stuff not requiring the USR
C
100     Continue
        Call LOCK               !USR now required
C
C       do stuff requiring the USR
C
        Call UNLOCK
        End
```

UNLOCK

The UNLOCK subroutine releases the User Service Routine (USR) from memory if it was placed there by the LOCK routine. If the LOCK required a swap, the UNLOCK loads the user program back into memory. If the USR does not require swapping, the UNLOCK involves no I/O. The USR is always resident in mapped monitors.

Form:

```
CALL UNLOCK
```

Notes

- You should give at least as many UNLOCK calls as LOCK calls. Otherwise, the USR remains locked in memory. Extra UNLOCK calls are ignored.
- When running in a multijob system, use the LOCK/UNLOCK pairs only when absolutely necessary. If one job locks the USR, the other job cannot use the USR until it is unlocked.
- In a multijob system, calling the CSI (ICSI) with input coming from the console terminal performs a temporary implicit UNLOCK.

For further information on releasing the USR from memory, see the .LOCK /.UNLOCK programmed requests.

Errors:
None.

Example:
See LOCK.

LOOKUP

The LOOKUP function associates a specified channel with a device and/or file for the purpose of performing I/O operations. The channel used is then busy until one of the following functions is executed:

CLOSEC/ICLOSE
CLOSZ
ISAVES
PURGE

Form:

```
CALL LOOKUP (chan,dblk[,seqnum,])
i = LOOKUP (chan,dblk[,seqnum,])
CALL LOOKUP (chan,jobdes)
i = LOOKUP (chan,jobdes)
```

where:

chan is the integer specification for the RT–11 channel to be associated with the file. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call

dblk is the four-word area specifying the Radix–50 file descriptor. Note that unpredictable results occur if the USR swaps over this four-word area

seqnum is a file number.

For magtape, it describes a file sequence number. The action taken depends on whether the file name is given or null. The sequence number can have the following values:

-1 Suppress rewind and search for the specified file name from the current tape position. If a file name is given, a file-structured lookup is performed (do not rewind). If the file name is null, a non-file-structured lookup is done (tape is not moved). You must specify a -1 and no other negative number.

0 Rewind to the beginning of the tape and do a non-file-structured lookup.

n Where *n* is any positive number. Position the tape at file sequence number n and check that the file names match. If the file names do not match, an error is generated. If the file name is null, a file-structured lookup is done on the file designated by *seqnum*.

jobdes is an argument that allows communication between jobs in a system job environment. It is a pointer to a four-word job descriptor of the job to which messages will be sent or received. The syntax is:

```
jobdes: .RAD50  /MQ/
        .ASCII  /logical-job-name/
```

where the logical-job-name is six characters long (right-padded with nulls). If the logical-job-name is zero, the channel will be opened only for .READ/C/W requests, and such requests will accept messages from any jobs.

Non-File-Structured Lookup

The handler for the selected device must be in memory for a LOOKUP. If the first word of the file name in *dblk* is 0 and the device is a file-structured device, absolute block 0 of the device is designated as the beginning of the file. This technique, called a non-file-structured lookup, allows I/O to any physical block on the device. If a file name is specified for a device that is not file structured (such as NL:FILE.TYP), the name is ignored.

Since a non-file-structured lookup allows I/O to any physical block on the device, in this mode, you could possibly overwrite the RT–11 device directory, destroying all information on the device. Position the LOOKUP arguments so that the USR does not swap over them.

Function Result:

Value	Meaning
i = >0	Successful file-structured lookup on a random-access storage volume. Value is length in blocks of file just opened. Note: Value can be of sufficient size to create a negative value.
= 0	Successful non-file-structured lookup on both random-access and non-file-structured volumes, or a successful file-structured lookup on magtape.

Errors:

Value	Meaning
i = -1	Channel specified is already open.
= -2	File specified was not found on the device.
= -3	Device in use.
= -6	Invalid argument error with a non-file-structured volume.
= -7	Invalid unit number.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See also CHCPY.

```
        Program FLOOKU
C
C       demo JOB LOOKUP
C
        Integer*2 JBLK(4)       !jobblk
        Logical*1 JNAM(6)       !name part
        Equivalence (JBLK(2), JNAM(1))
        Data JBLK(1) /3rMQ /
        Data JNAM /'S', 'P', 'O', 'O', 'L', "000/
C
C       open a message channel to SPOOL
C
        ICHAN = IGETC ()
        If (LOOKUP (ICHAN, JBLK) .lt. 0)
        1 Call PRINT ('?FLOOKU-F-SPOOL is not running')
        End
```

MAP

Mapping

The MAP (MAP window) function maps a previously defined address window into a dynamic region of extended memory or into the static region in the lower 28 KW.

Form:

```
ierr= MAP (iwdb)
 CALL MAP (iwdb [,ierr])
```

where:

ierr	Error return
iwdb	Address of Window Descriptor Block

Errors:

Value	Meaning
i = 0	Function completed successfully.
= -3	Invalid region identifier specified.
= -4	Invalid region identifier specified.
= -5	Specified region not mapped because of one of the following: • Offset is beyond the end of the region. • Window is larger than the region. • Window would extend beyond the bounds of the region.
= -16	Mode/space not available.
= -257	Required argument missing.

Example:
See CRAW.

MRKT

Timer (SYSGEN Option)

MRKT schedules an assembly language completion routine to be entered after a specified time interval has elapsed. An optional parameter, *CMODE*, can be used to specify a Supervisor address.

Form:

```
CALL MRKT (id,crtn,time[,CMODE=strg])
 i =  MRKT (id,crtn,time[,CMODE=strg])
```

where:

id	is an integer identification number to be passed to the routine being scheduled
crtn	is the name of the assembly language routine to be entered when the time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the MRKT call
time	is the two-word internal format time interval; when this interval elapses, the routine is entered. If considered as a two-element INTEGER*2 array: time(1) is the high-order time. time(2) is the low-order time.
CMODE=strg	Specifying *strg* as string "S" sets on low bit of *crtn* address to indicate an S-I address.

Notes

- MRKT requires a queue element, which should be considered when the IQSET function is executed.
- If the system is busy, the time interval that elapses before the completion routine is run can be greater than that requested.

For further information on scheduling completion routines, see the .MRKT programmed routine discussion in the *RT–11 System Macro Library Manual*.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No queue element was available; unable to schedule request.
= -19	Invalid CMODE value.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
       Program FMRKT ! Demo MRKT and CMKT
C
C       Run two timers at 5 and 50 ticks each, shutdown
C       first timer after 5 times and wait for second.
C
        Integer*2 COUNT
        Integer*4 TIME, TIME2, RTIME
        Common /TIMING/ COUNT, TIME
        External FAMRKT         !macro completion routine
C
        Call IQSET (10)         !get some queue elements
        COUNT = 1               !init count
        TIME = 5                !init at 5 ticks
        Call JJCVT (TIME)       !and convert from int to time
        TIME2 = 50              !init at 50 ticks
        Call JJCVT (TIME2)      !and convert from int to time
        Call MRKT (12345, FAMRKT, TIME) !timer 1
        Call MRKT (02222, FAMRKT, TIME2)!timer 2
100     Continue
        Call SUSPND             !wait for completion
        If (COUNT .eq. 5) Then
            Call CMKT (12345, RTIME) !on 5th, shutdown timer 1
            Go To 200
        Else
            Go To 100
        End If
C
200     Continue
        Call PRINT ('!FMRKT-I-Normal Termination')
        End

        .TITLE  FAMRKT  macro completion routine for FMRKT
        .MCALL  .RSUM   .PRINT  .MRKT

FAMRKT::
        .PRINT  #HERE           ;indicate entered
        INC COUNT               ;indicate entered
        .RSUM                   ;resume suspended mainline
        .MRKT   #AREA,#TIME,#FAMRKT ;reissue the request
        RETURN

AREA:   .BLKW   3

HERE:   .ASCIZ  "... in FAMRKT"

.PSECT  TIMING,RW,D,GBL,REL,OVR ;Common /TIMING/ ...
COUNT:  .BLKW   1
TIME:   .BLKW   2
        .END
```

MSDS

Full Mapping

The MSDS subroutine controls the linkage of Supervisor and User data space.

Form:

```
CALL MSDS (ival [,iold] [,ierr])
 ierr = MSDS (ival [,iold])
```

where:

ierr	Error return
ival	New value for CMAP status
iold	Previous CMAP status

Errors:

Value	Meaning
i = 0	Success.
i = -257	Required argument missing.

Example:
See CMAP.

MTATCH

Multiterminal Option

The MTATCH subroutine attaches a terminal for exclusive use by the requesting job. This operation must be performed before any job can use a terminal with multiterminal programmed requests. An optional argument, *amode*, lets you specify an S-D address.

Form:

```
CALL MTATCH (unit[,addr][,jobnum][,AMODE=strg])
 i =  MTATCH (unit[,addr][,jobnum][,AMODE=strg])
```

where:

unit	is the unit number of the terminal
addr	is the optional address of an asynchronous terminal status word. Omit this argument. For example: `I = MTATCH (unit,,jobnum)`
jobnum	is the job number associated with the terminal if the terminal is not available
AMODE=strg	Specify *amode* as the string "S" to set on the low bit of the address of *iflag* to indicate an S-D address.

Errors:

Value	Meaning
i = 0	Normal return.
= 3	Nonexistent unit number.
= 5	Unit attached by another job (job number returned in *jobnum*.)
= 6	In XM monitor, the optional status word address is not in a valid user virtual address space.
= 7	Unit attached by handler (name returned in *jobnum*.)
=-19	Invalid AMODE value.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTDTCH

Multiterminal Option

The MTDTCH subroutine is the complement of the MTATCH subroutine. Its function is to detach a terminal from a particular job and make it available for other jobs.

Form:

```
CALL MTDTCH(unit)
 i = MTDTCH(unit)
```

where:

unit is the unit number of the terminal to be detached

Errors:

Value	Meaning
i = 0	Normal return.
= 2	Invalid unit number; terminal is not attached.
= 3	Nonexistent unit number.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTGET

Multiterminal Option

The MTGET subroutine furnishes the user with information about a specific terminal in a multiterminal system. You do not need to do an MTATCH before using MTGET.

Form:

```
CALL MTGET (unit,addr[,jobnum])
 i = MTGET (unit,addr[,jobnum])
```

where:

unit	is the unit number of the line and terminal whose status is desired
addr	is the four-word area to receive the status information.
jobnum	is the job number associated with the terminal if the terminal is not available

Status information including bit definitions for the terminal configuration words and the terminal state byte are described in detail under the .MTGET programmed request.

Errors:

Value	Meaning
i = 0	Normal return.
= 3	Nonexistent unit number.
= 5	Unit attached by another job (job number returned in *jobnum*.)
= 6	In XM monitor, the optional status word address is not in a valid user virtual address space.
= 7	Unit attached by handler (name returned in *jobnum*.)

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTIN

Multiterminal Option

MTIN transfers characters from a specified terminal to the user program. This subroutine is a multiterminal form of ITTINR. If no characters are available, an error flag is set to indicate an error upon return from the subroutine. If no character count argument is specified, one character is transferred.

Form:

```
CALL MTIN (unit,char[,chrcnt][,ocnt])
 i =  MTIN (unit,char[,chrcnt][,ocnt])
```

where:

unit	is the unit number of the terminal
char	is the variable to contain the characters read in from the terminal indicated by the unit number
chrcnt	is an optional argument that indicates the number of characters to be read
ocnt	is an optional argument that indicates the number of characters actually transferred

When a request for a multiple-character transfer is requested, if the optional fourth argument (ocnt) is specified and bit 6 of the M.TSTS word is set, the variable specified as the argument will have a value equal to the actual number of characters transferred upon return from the subroutine.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No input available.
= 2	Unit not attached.
= 3	Nonexistent unit number.
= 6	In XM monitor, the optional status word address is not in a valid user virtual address space.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTOUT

Multiterminal Option

The MTOUT subroutine transfers characters to a specified terminal. This subroutine is a multiterminal form of ITTOUR. If no room is available in the output ring buffer, an error flag is set to indicate an error upon return from the subroutine. If no character count argument is specified, one character is transferred.

Form:

```
CALL MTOUT (unit,char[,chrcnt][,ocnt])
 i =  MTOUT (unit,char[,chrcnt][,ocnt])
```

where:

unit	is the unit number of the terminal
char	is the variable or array containing the characters to be output, right-justified in the integer (can be LOGICAL*1 if desired)
chrcnt	is an optional argument that indicates the number of characters to be output
ocnt	is an optional argument that indicates the number of characters actually transferred

When a request for a multiple-character transfer is requested, if the optional fourth argument *ocnt* is specified and bit 6 of the M.TSTS word is set, the *ocnt* will have a value equal to the actual number of characters transferred upon return from the subroutine.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No room in output ring buffer.
= 2	Unit not attached.
= 3	Nonexistent unit number.
= 6	In the XM monitor, the address of the user buffer is outside the valid program limits.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTPRNT

Multiterminal Option

The MTPRNT subroutine allows output to be printed at any terminal in a multiterminal environment. This subroutine has the same effect as the PRINT subroutine. See PRINT.

Form:

```
CALL MTPRNT (unit,string)
 i = MTPRNT (unit,string)
```

where:

unit	is the unit number associated with the terminal
string	is the character string to be printed. Note that all quoted literals used in FORTRAN subroutine calls are in ASCIZ format, which ends in null (0) for a RETURN instruction or a 200_8 to suppress RETURN instruction

Errors:

Value	Meaning
i = 0	Normal return.
= 2	Unit not attached.
= 3	Nonexistent unit number.
= 6	In the XM monitor, the address of the character string is outside the valid program limits.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTRCTO

Multiterminal Option

The MTRCTO subroutine resets the CTRL/O command typed at the specified terminal in a multiterminal environment. This subroutine has the same effect as the .MTRCTO programmed request.

Form:

```
CALL MTRCTO(unit)
 i = MTRCTO(unit)
```

where:

unit is the unit number associated with the terminal

Errors:

Value	Meaning
i = 0	Normal return.
= 2	Unit not attached.
= 3	Nonexistent unit number.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTSET

Multiterminal Option

The MTSET subroutine sets terminal and line characteristics. The set conditions remain in effect until the system is booted or the terminal and line characteristics are reset. See the .MTSET programmed request for more details.

Form:

```
CALL MTSET (unit,addr)
 i = MTSET (unit,addr)
```

where:

unit is the unit number of the line and terminal whose characteristics are to be changed

addr is a four-word area to pass the status information. The area is a four-element INTEGER*2 array

Errors:

Value	Meaning
i = 0	Normal return.
= 2	Unit not attached.
= 3	Nonexistent unit number.
= 6	In the XM monitor, the address of the status block is outside the valid program limits.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MTSTAT

Multiterminal Option

The MTSTAT subroutine returns multiterminal system status in an eight-word status block.

Form:

```
CALL MTSTAT (addr)
 i = MTSTAT (addr)
```

where:

addr is the address of an eight-word array where multiterminal status information is returned. The status block contains the following information:

addr(1)	Offset from the base of the resident monitor to the first Terminal Control Block (TCB).
addr(2)	Offset from the base of the resident monitor to the terminal control block of the console terminal for the program.
addr(3)	The value (0-16 decimal) of the highest logical unit number (LUN) built into the system.
addr(4)	The size of the terminal control block in bytes.
addr(5)-(8)	Reserved.

Errors:

Value	Meaning
i = 0	Normal return.
= 6	In the XM monitor, the address of the status block is not in valid user address space.

Example:
Refer to the *RT–11 System Macro Library Manual* for an example.

MWAIT

Multijob

The MWAIT subroutine suspends main program execution of the current job until all messages sent to or from the other job have been transmitted or received. It provides a means for ensuring that a required message has been processed. MWAIT is used primarily in conjunction with the IRCVD and ISDAT calls, where no action is taken when a message transmission is completed. This subroutine requires a queue element, which should be considered when the IQSET function is executed.

Form:

```
CALL MWAIT
```

Errors:
None.

Example:
See SDAT.

POKE/IPOKE

IPOKE stores a specified 16-bit integer value into a 16-bit address in the current job's address space. The subroutine can store values in device registers if the register is in the current job's address space.

Form:

```
CALL POKE (iaddr,ivalue)
 i = IPOKE (iaddr,ivalue)
```

where:

iaddr is the integer specification of the 16-bit address in the current job's address space to be modified. If this argument is not an even value, a trap results (except on an LSI-11 or a PDP–11/23)

ivalue is the integer value to be stored in the address specified by the iaddr argument

Errors:
None.

Example:

To set bit 12 in the JSW without zeroing any other bits in the JSW, refer to example under PEEK.

POKEB/IPOKEB

POKEB/IPOKEB stores a specified 8-bit integer value into a 16-bit address in the current job's address space. Since this subroutine operates in a byte mode, the address supplied can be odd or even. The subroutine can store values in device registers if the register location is in the current job's address space.

Form:

```
CALL POKEB(iaddr,ivalue)
 i = IPOKEB(iaddr,ivalue)
```

where:

iaddr is the integer specification of the 16-bit address in the current job's address space to be modified. Unlike the IPOKE subroutine, the IPOKEB subroutine allows odd addresses

ivalue is the integer value to be stored in the address specified by the iaddr argument. Only the low byte of ivalue is actually stored

Errors:
None.

Example:
See PEEK.

PRINT

Refer to WRITE, MWRITC, MWRITW.

The PRINT subroutine prints output from a specified string to the terminal. This routine can be used to print messages from completion routines without using the FORTRAN formatted I/O system. Control returns to the user program after all characters have been placed in the output buffer.

Form:

CALL PRINT (string)

where:

string is the string to be printed. Note that all quoted literals used in FORTRAN subroutine calls are in ASCIZ format, as are all strings produced by the SYSLIB string-handling package (The CONCAT routine can be used to append an octal 200 to an ASCIZ string; see example.)

The string to be printed can be terminated with either a null (0) byte or a 200(octal) byte. If the null (ASCIZ) format is used, the output is automatically followed by a carriage return/line feed pair (octal 15 and 12). If a 200 byte terminates the string, no carriage return/line feed pair is generated.

In the FB monitor, a change in the job that is controlling terminal output is indicated by a B> or F>. Any text following the message has been printed by the job indicated (foreground or background) until another B> or F> is printed. In a system job monitor the job name is printed; for example, SPOOL>. When PRINT is used by the foreground job, the message appears immediately, regardless of the state of the background job. Thus, for urgent messages, PRINT should be used rather than ITTOUR.

Errors:
If an argument is missing, a random portion of memory is printed, as if it were text.

Example:

```
        Program FPRINT
C
C       This shows calls to PRINT with normal CRLF and with
C       it suppressed.
C
        Byte PROMPT (80)
C
        Call PRINT ('This is a normal line of output')
        Call CONCAT ('Name? ', '200'o, PROMPT)
        Call PRINT (PROMPT)     !and one w/o CRLF
        Call Print ('Frank Johnson') !"answer" prompt
        End
```

PROTE/IPROTE

PROTE/IPROTE lets a job obtain exclusive control of a two-word vector in the region 0 to 474. If PROTE returns successfully:

- Specified locations are not currently in use by another job or the monitor.
- Calling job can place an interrupt address and priority in the protected 2-word vector and begin the device associated with those vectors.

Form:

```
CALL PROTE (addr)
i = IPROTE (addr)
```

where:

addr is the address of the 2-word vector pair to be protected

Errors:

Value	Meaning
i = 0	Normal return
= -1	Protect failure; location already in use.
= -2	*Addr* is greater than 474 or not a multiple of 4.
= -257	Required argument *addr* missing.

Example:

```
        Program FPROTE
C
C       Try to protect all the vectors, see which are
C       already protected.
C       Then unprotect all of them.
C
        Parameter MAXVEC = '474'o
        Integer*2 VEC           !vector to try
C
        Do 100, VEC = 0, MAXVEC, 4
            If (IPROTE (VEC) .ne. 0) Type 1, VEC
1           Format (' ', o3, ' was protected')
            Call UNPRO (VEC)    !just unprotect everything (we can)
100         Continue
        End
```

PURGE

The PURGE subroutine deactivates a channel. Any tentative file currently associated with the channel is not made permanent. This subroutine prevents entered (IENTER or .ENTER) files from becoming permanent directory entries.

Form:

CALL PURGE(chan)

where:

chan is the integer specification for the RT–11 channel to be deactivated

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

Example:
Refer to the example under ENTER/IENTER.

PUT/IPUT

PUT replaces the value of a monitor fixed offset. PUT uses the monitor .PVAL programmed request.

Form:

```
CALL PUT (ioff,value[,operation][,ierr])
 ierr = IPUT (ioff,value[,operation])
```

where:

ioff is the offset (from the base of RMON) to be modified

value Specifies an integer replacement value to replace current contents of offset location or specifies a BIC/BIS mask, if *operation* is specified.

operation If specified, one of the following:

- MOV = Replace with a MOV operation (default).
- BIC = Change with a BIC operation.
- BIS = Change with a BIS operation.

ierr the word that will get the error status of the request.

Errors:

Value	Meaning
i = 0	Success, except for trap message.
= -1	Offset out of range.
= -2	Trap 4 (odd address/non-existent memory.) location.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FPUT
C        Change the default max file size to 10 and try
C        a ENTER to verify the effect, then put it back.
         Parameter MAXBL = '314'o !fixed offset for max block
         Integer*2 OLDMAX        !old value
         Integer*2 DBLK(4)
         Data DBLK /3rDK , 3rTES, 3rT  , 3rTMP/
         OLDMAX = ISPY (MAXBL)   !get old value
         Type 1, OLDMAX
1        Format (' ', 'MAXBL=', o7)
         Call PUT (MAXBL, 10)    !set it to 10
         IERR = IENTER (IGETC (), DBLK, 0) !ask for default space
         Type 2, IERR
2        Format (' ', 'Created file size', o7)
         Type 1, IPUT (MAXBL, OLDMAX) !put it back
         Type 1, ISPY (MAXBL)    !verify
         End
```

R50ASC

The R50ASC subroutine converts a specified number of Radix–50 characters to ASCII.

Form:

CALL R50ASC (icnt,input,output)

where:

icnt	is the integer number of ASCII characters to be produced
input	is the area from which words of Radix–50 values to be converted are taken. Note that (icnt+2)/3 words are read for conversion
output	is the area into which the ASCII characters are stored

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

If an input word contains illegal Radix–50 codes—that is, if the input word is greater (unsigned) than 174777(octal)—the routine outputs question marks for the value.

Example:
See CSI.

RAD50

The RAD50 function provides a method of encoding RT–11 file descriptors in Radix–50 notation. The RAD50 function converts six ASCII characters from the specified area, returning a REAL*4 result that is the two-word Radix–50 value.

Form:

a = RAD50 (input)

where:

input is the area from which the ASCII input characters are taken

The RAD50 call:

A = RAD50 (LINE)

is exactly equivalent to the IRAD50 call:

CALL IRAD50 (6,LINE,A)

Function Results:
The two-word Radix–50 value is returned as the function result.

Errors:
Unpredictable results will occur if any required argument is omitted.

Example:

```
Real*8 FSPEC
 Call IRAD50 (12, 'SY SWAP  SYS', FSPEC)
```

RAN/RANDU

RT–11 adds the RAN function and RANDU subroutine, uniform pseudo-random number generators to SYSLIB. These are the same default routines as those previously supplied with FORTRAN–77. RAN and RANDU were previously located in the FORTRAN IV and FORTRAN–77 object time system libraries. RAN and RANDU are different from the routines previously supplied with FORTRAN IV.

RAN and RANDU generate a floating-point number that is evenly distributed in the range between 0.0 inclusive and 1.0 exclusive (1.0 is never generated).

When you provide a 32-bit seed number, that number is automatically updated according to the following:

```
SEED = 69069 * SEED +1 (MOD 2**32)
```

The value of SEED is a 32-bit number. The high-order 24 bits of SEED are converted to floating point and returned as the result *f*.

Form:

```
f = RAN (jseed)
 f = RAN (iseed1,iseed2)
 CALL RANDU(iseed1,iseed2,f)
```

where:

f	is a 32-bit floating-point variable
jseed	is an INTEGER*4 seed number
iseed1	is the low-order INTEGER*2 seed variable
iseed2	is the high-order INTEGER*2 seed variable

RAN and RANDU are multiplicative-congruential, general-random-number generators. They are fast, but prone to nonrandom sequences if you construct and analyze triples of generated numbers.

There are no restrictions on the seed. The seed should be initialized to different values on separate runs to obtain different random sequences.

Errors:
Unpredictable results will occur if any required argument is omitted.

Example:

This example illustrates a simple way to get a uniform random integer selector. Multiply the value returned by the RAN function by the number of cases—in this example, five.

```
GO TO (1,2,3,4,5), (1 + IFIX(5.*RAN(JSEED)))
```

The explicit IFIX is necessary before adding 1 to avoid possible rounding during the normalization after the addition of floating-point numbers.

RCHAIN

The RCHAIN subroutine allows a program to determine whether it has been chained to and to access variables passed across a chain. If RCHAIN is used, it must be used in the first executable FORTRAN statement in a program.

Form:

CALL RCHAIN (flag,var,wcnt)

where:

flag is an integer variable that RCHAIN will set to -1 (true) if the program has been chained to; otherwise, it is 0 (false)

var is the first variable in a sequence of variables with increasing memory addresses to receive the information passed across the chain (See CHAIN).

wcnt is the number of words to be moved from the chain parameter area to the area specified by var. RCHAIN moves *wcnt* words into the area beginning at *var*.

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See also CHAIN.

```
        Program FRCHAI  ! test RCHAIN routine
C
C       Receive .CHAIN and expect 100,200,301 as input
C
        Logical*2 FLAG
        Parameter SUCCS = '001'o, ERROR = '004'o, FATAL = '010'o
        Common /RCHAIN/ L, M, N
C
        Call RCHAIN (FLAG, L, 3)
        If (.not. FLAG) Go To 1000      !not chained to
        If (L .ne. 100) Go To 2000      !wrong value
        If (M .ne. 200) Go To 2000      !wrong value
        If (N .ne. 301) Go To 2000      !wrong value
        Type *, ' !FRCHAI-I-Successful chain entry'
        Call EXIT (SUCCS)
C
1000    Type *, ' ?FRCHAI-F-Not chained to'
        Call EXIT (FATAL)
C
2000    Type *, ' ?FRCHAI-E-Wrong value'
        Call EXIT (ERROR)
        End
```

RCTRLO

The RCTRLO subroutine resets the effect of any console terminal CTRL/O command that was typed. After an RCTRLO call, any output directed to the console terminal prints until another CTRL/O is typed. It should also be issued after the program changes any JSW bits, to synchronize internal monitor data structures.

Form:

```
CALL RCTRLO
```

Errors:
None.

Example:
See PEEK.

*RCVD/*RCVDC/*RCVDF/*RCVDW

Multijob

Four forms of *RCVD can be used in conjunction with the ISDAT (send data) functions to allow a general data/message transfer system. All forms of *RCVD, issued either as a function or subroutine, issue RT–11 receive-data programmed requests. These functions require a queue element which should be a consideration when the IQSET function is executed.

Specify mapping for MRCVD, MRCVDC and MRCVDW by optional parameters *BMODE* and *CMODE*.

RCVD/IRCVD/MRCVD

RCVD/IRCVD/MRCVD requests data and continues execution. The operation is queued and the issuing job continues execution. When the job has to receive the transmitted message, an MWAIT should be executed. This causes the job to be suspended until all pending messages have been received.

Form:

```
CALL RCVD (buff,wcnt)
 i = IRCVD (buff,wcnt)
CALL MRCVD (buff,wcnt[,BMODE=strg])
 i = MRCVD (buff,wcnt[,BMODE=strg])
```

where:

buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVD is complete.
wcnt	is the maximum integer number of words that can be received
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value specifies the mapping mode of the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No other job exists in the system. A job exists as long as it is loaded, whether or not it is active.
= -19	Invalid BMODE value.

Error message *TRAP $MSARG* will display if *buff* or *wcnt* argument is missing.

Example:

```
       Program FRCVD !demo RCVD (use with FSDATx)
C
C       Try to get a message from the other job, when
C       you get it, display it and exit.
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter RCVCNT = 0, RCVMSG = 1, WCNT = 42
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/
        Integer*2 BUFFER(RCVCNT:WCNT)
C
100     Continue
            IERR = IRCVD (BUFFER, WCNT) !try for a message
            If (IERR .eq. 1) Go To 100  !wait for the job
            If (IERR .ne. 0) Go To 200  !unknown error
            Call MWAIT                  !wait for a message
            If (BUFFER(RCVMSG) .eq. CTRLZ) Go To 300 !"EOF"
            Call PRINT (BUFFER(RCVMSG)) !display message (skip count)
            Go To 100                   !get another
C
200     Call PRINT ('?FRCVD-F-Unknown error code from IRCVD')
        Call EXIT (FATAL)
C
300     Call PRINT ('!FRCVD-I-Normal termination')
        Call EXIT (SUCCS)               !and done
        End
```

RCVDC/IRCVDC/MRCVDC

RCVDC/IRCVDC/MRCVDC requests data and enters an assembly language completion routine when the message is received. RCVDC/IRCVDC/MRCVDC is queued, and program execution stays with the issuing job. When the other job sends a message, the completion routine specified is queued and run according to standard scheduling of completion routines.

Form:

```
CALL RCVDC (buff,wcnt,crtn)
 i = IRCVDC (buff,wcnt,crtn)
CALL MRCVDC (buff,wcnt,crtn[,BMODE=strg][,CMODE=strg])
 i = MRCVDC (buff,wcnt,crtn[,BMODE=strg][,CMODE=strg])
```

where:

buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVDC is complete
wcnt	is the maximum integer number of words to be received

crtn is the assembly language completion routine to be entered. This name must be specified in a FORTRAN EXTERNAL statement in the routine that issues the IRCVDC call

BMODE=strg Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.

CMODE=strg Specify *strg* as string 'S' to specify a Supervisor address.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No other job exists in the system. A job exists as long as it is loaded, whether or not it is active.
=-19	Invalid BMODE or CMODE value.

Error message *TRAP $MSARG* will display if *buff, wcnt,* or *crtn* argument is missing.

Example:
See SFDAT examples.

RCVDF/IRCVDF

RCVDF/IRCVDF requests data and enters a FORTRAN completion subroutine when the message is received. The RCVDF/IRCVDF is queued, and program execution continues with the issuing job. When the other job sends a message, the FORTRAN completion routine specified is entered.

Form:

```
CALL RCVDF (buff,wcnt,area,frtn)
 i = IRCVDF (buff,wcnt,area,frtn)
```

where:

buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVDF is complete
wcnt	is the maximum integer number of words to be received
area	is a four-word area to be set aside for linkage information. This area must not be modified by the FORTRAN program and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion routines when crtn has been entered
frtn	is the FORTRAN completion routine to be entered. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the RCVDF call

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No other job exists in the system. A job exists as long as it is loaded, whether or not it is active.

Error message *TRAP $MSARG* will display if *buff, wcnt, area* or *frtn* argument is missing.

Example:
See also SDAT*.

```
        Program FRCVDF !demo RCVDF (use with FSDATx)
C
C       Try to get a message from the other job, when
C       you get it, display it and exit.
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter RCVCNT = 0, RCVMSG = 1, WCNT = 42
        Integer*2 BUFFER(RCVCNT:WCNT)
        Common /DATA/ BUFFER
        Integer*2 LINKAG(4)             !area for linkage
        External FRCVDG                 !completion routine
C
100     Continue
```

```
            IERR = IRCVDF (BUFFER, WCNT, LINKAG, FRCVDG) !try for a message
            If (IERR .eq. 1) Go To 100  !wait for the job
        If (IERR .ne. 0) Go To 200      !unknown error
C
C       other processing not requiring message could be done here
C
        Call SUSPND                     !wait for completion routine to
C                                       ! resume us
        Call PRINT ('!FRCVDF-I-Termination successfully completed')
        Call EXIT (SUCCS)               !and done
C
200     Call PRINT ('?FRCVDF-F-Unknown error code from IRCVDF')
        Call EXIT (FATAL)
        End

        Subroutine FRCVDG       !completion routine for FRCVDF
        Parameter FATAL = '010'o
        Parameter RCVCNT = 0, RCVMSG = 1, WCNT = 42
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/
        Integer*2 BUFFER(RCVCNT:WCNT)
        Common /DATA/ BUFFER
        Integer*2 LINKAG(4)     !area for linkage
        External FRCVDH         !fakeout the recursion detection
C
        If (BUFFER(RCVMSG) .eq. CTRLZ) Go To 100 !"EOF"
        Call PRINT (BUFFER(RCVMSG))     !print the data
        IERR = IRCVDF (BUFFER, WCNT, LINKAG, FRCVDH) !try for next message
        If (IERR .ne. 0) Go To 200      !unknown error
        Return
C
100     Call RESUME                     !restart mainline so it can exit
        Return
C
200     Call PRINT ('?FRCVDG-F-Unknown error code from IRCVDF')
        Call EXIT (FATAL)               !can't really exit cleanly from
C                                       !completion, but dying is ok anyway
        End

        .TITLE  FRCVDH  -- Just call FRCVDG

FRCVDH::CALLR   FRCVDG  ;slip around recursion detection
                        ;in the compiler
        .END
```

RCVDW/IRCVDW/MRCVDW

RCVDW/IRCVDW/MRCVDW requests data and waits until it is available. This function queues a message request and suspends the job issuing the request until the other job sends a message. When execution of the issuing job resumes, the message has been received, and the first word of the buffer indicates the number of words transmitted.

Form:

```
CALL RCVDW (buff,wcnt)
 i = IRCVDW (buff,wcnt)
CALL MRCVDW (buff,wcnt[,BMODE=strg])
 i = MRCVDW (buff,wcnt[,BMODE=strg])
```

where:

buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVDW is complete
wcnt	is the maximum integer number of words to be received
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No other job exists in the system. A job exists as long as it is loaded, whether or not it is active.
=-19	Invalid BMODE value.

Error message *TRAP $MSARG* will display if *buff* or *wcnt* argument is missing.

Example:
See also SDAT*.

```
        Program FRCVDW !demo RCVDW (use with FSDATx)
C
C       Try to get a message from the other job, when
C       you get it, display it and exit.
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter RCVCNT = 0, RCVMSG = 1, WCNT = 42
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/
        Integer*2 BUFFER(RCVCNT:WCNT)
C
100     Continue
            IERR = IRCVDW (BUFFER, WCNT) !try for a message
            If (IERR .eq. 1) Go To 100  !wait for the job
            If (IERR .ne. 0) Go To 200  !unknown error
            If (BUFFER(RCVMSG) .eq. CTRLZ) Go To 300 !"EOF"
```

```
            Call PRINT (BUFFER(RCVMSG)) !display message (skip count)
            Go To 100
C
200     Call PRINT ('?FRCVDW-F-Unknown error code from IRCVDW')
        Call EXIT (FATAL)
C
300     Call PRINT ('!FRCVDW-I-Success')
        Call EXIT (SUCCS)               !and done
        End
```

*READ/*READC/*READF/*READW

Multijob

Four forms of *READ, issued as a function or as a subroutine, transfer a specified number of words from a file into memory. These functions require a queue element, which should be considered when the IQSET function is executed.

Specify mapping for MREAD, MREADC and MREADW by optional parameters *BMODE* and *CMODE*.

READ/IREAD/MREAD

READ/IREAD/MREAD issued either as a function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. Control returns to the user program immediately after the READ/IREAD/MREAD function is initiated. No special action is taken when the transfer is completed.

Form:

```
CALL READ (wcnt,buff,blk,chan)
 i = IREAD (wcnt,buff,blk,chan)
CALL MREAD (wcnt,buff,blk,chan[,BMODE=strg])
 i = MREAD (wcnt,buff,blk,chan[,BMODE=strg])
```

where:

wcnt	is the relative integer number of words to be transferred
buff	is the array to be used as the buffer; this array must contain at least *wcnt* words
blk	is the integer block number of the file to be read. The first block of a file is block number 0. The blk argument must be updated when necessary. For example, if the program is reading two blocks at a time, blk should be updated by 2
chan	is the integer specification for the RT–11 channel to be used
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI.' This value specifies the mapping mode of the *buff* argument.

Function Return:

i = Normal return; *i* equals the number of words requested (0 for non-file-structured read, multiple of 256 for file-structured read). If the read is from a magtape, the number of words requested is returned. For example:

- If *wcnt* is a multiple of 256 and less than that number of words remain in the file, *i* is shortened to the number of words that remain in the file; thus, if *wcnt* is 512 and only 256 words remain, i=256.

- If *wcnt* is not a multiple of 256 and more than *wcnt* words remain in the file, *i* is rounded up to the next block; thus, if *wcnt* is 312 and more than 312 words remain, i = 512, but only 312 are read.
- If *wcnt* is not a multiple of 256 and less than *wcnt* words remain in the file, *i* equals a multiple of 256 that is the actual number of words being read.

Errors:

Value	Meaning
i = -1	Attempt to read past end-of-file; no words remain in the file.
= -2	Hardware error occurred on channel.
= -3	Specified channel is not open.
= -19	Invalid BMODE value.

Error message *TRAP $MSARG* will display if any argument is missing.

Notes

If an asynchronous operation on a channel (for example, IREAD) results in end-of-file, the following IWAIT will not detect it. IWAIT detects only hard error conditions. A subsequent operation on that channel will detect end-of-file and returns the end-of-file error code. Under these conditions, the subsequent operation is not initiated.

Example:

```
        Program FREAD
C
C        Demonstrate IREAD and IWAIT
C        Scan SRC: and try to find this source file
C
         Parameter SUCCS = '001'o, FATAL = '010'o
         Parameter NULL = '000'o, HT = '011'o
         Parameter LF = '012'o, CR = '015'o
         Integer*2 BUF(256)      !block buffer
         Byte CBUF(512)          !char overlay for BUF
         Equivalence (CBUF(1), BUF(1))
         Integer*2 CHAN          !channel number to use
         Integer*2 SRC(2)        !device dblk
         Data SRC /3rSRC, 0/     !whole device (no file name)
         Integer*2 BLK           !current block number
         Integer*2 IERR          !function return value
         Byte Search(17)         !our first few chars
         Data Search /HT, 'P', 'r', 'o', 'g', 'r', 'a', 'm',
         1   ' ', 'F', 'R', 'E', 'A', 'D', CR, LF, NULL/
C
         CHAN = IGETC()          !allocate a channel
         BLK = 0                 !begin at the beginning
         Call LOOKUP (CHAN, SRC) !open the device
```

```
100     Continue
            IERR = IREAD (256, BUF, BLK, CHAN)
            If (IERR .ne. 256) Go To 200 !failure
C
C           The program could do something not dependent on
C           the I/O here
C
            IERR = IWAIT (CHAN) !wait for the data
            If (IERR .ne. 0) Go To 300 !failure
            Call SCOPY (CBUF, CBUF, 16) !truncate the string
            If (ISCOMP (CBUF, SEARCH) .eq. 0)
        1       Type 101, BLK   !Eureka!
101         Format (' ', '!FREAD-I-Self discovery at block ', i6)
            BLK = BLK + 1       !try next block
            Go To 100
C
200     Continue
        If (IERR .eq. -2) Go To 400 !hard err is expected
        If (IERR .eq. -1) Type 102
102     Format (' ', '?FREAD-F-EOF from IREAD')
        If (IERR .ne. -1) Type 103, IERR
103     Format (' ', '?FREAD-F-Unexpected error from IREAD', i6)
        Call EXIT (FATAL)
300     Continue
        If (IERR .eq. 2) Go To 500 !hard err is expected
        Type 104, IERR
104     Format (' ', '?FREAD-F-Unexpected error from IWAIT', i6)
        Call EXIT (FATAL)
400     Type 105
105     Format (' ', '!FREAD-I-Success (probably)')
        Call EXIT (SUCCS)
500     Type 106
106     Format (' ', '?FREAD-F-Hard error from IREAD')
        Call EXIT (SUCCS)
        End
```

READC/IREADC/MREADC

READC/IREADC/MREADC, issued either as a function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. Control returns to the user program immediately after the READC /IREADC/MREADC function is initiated. When the operation is complete, the specified assembly language routine (*crtn*) is entered as an asynchronous completion routine.

Form:

```
CALL READC (wcnt,buff,blk,chan,crtn)
i = IREADC (wcnt,buff,blk,chan,crtn)
CALL MREADC (wcnt,buff,blk,chan,crtn[,BMODE=strg][,CMODE=strg])
i = MREADC (wcnt,buff,blk,chan,crtn[,BMODE=strg][,CMODE=strg])
```

where:

wcnt	is the integer number of words to be transferred
buff	is the array to be used as the buffer; this array must contain at least wcnt words
blk	is the integer block number of the file to be read. The user program normally updates *blk* before it is used again. The first block of a file is block number 0.
chan	is the integer specification for the RT–11 channel to be used
crtn	is the assembly language routine to be activated when the transfer is complete. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IREADC call
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.
CMODE=strg	Specify *strg* as string 'S' to specify Supervisor address.

Function Return:

i = Normal return; *i* equals the number of words requested (0 for non-file-structured read, multiple of 256[decimal] for file-structured read). If the read is from a magtape, the number of words requested is returned. For example:

- If *wcnt* is a multiple of 256 and less than that number of words remain in the file, *i* is shortened to the number of words that remain in the file; thus, if *wcnt* is 512 and only 256 words remain, i=256.

- If *wcnt* is not a multiple of 256 and more than *wcnt* words remain in the file, *i* is rounded up to the next block; thus, if *wcnt* is 312 and more than 312 words remain, i = 512, but only 312 are read.
- If *wcnt* is not a multiple of 256 and less than *wcnt* words remain in the file, *i* equals a multiple of 256 that is the actual number of words being read.

Errors:

Value	Meaning
i = -1	Attempt to read past end-of-file; no words remain in the file.
= -2	Hardware error occurred on channel.
= -3	Specified channel is not open.
= -19	Invalid BMODE or CMODE value.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See RCVDF for FRCVDM routine.

```
        Program FREADC
C
C       This program demonstrates READC and WRITC
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLKI(4), DBLKO(4)    !file names
        Data DBLKI /3rSRC, 3rFRE, 3rADC, 3rFOR/
        Data DBLKO /3rDK , 3rFRE, 3rADC, 3rTMP/
        Parameter WCNT = 256            !buffer size
        Integer*2 BUFFER (WCNT)         !buffer itself
        Integer*2 ICHAN, OCHAN          !channel numbers
        Integer*2 BLK                   !block number
        External FRCVDM
C
        ICHAN = IGETC()                 !get channel numbers
        OCHAN = IGETC()                 ! ...
        ISIZE = LOOKUP (ICHAN, DBLKI)   !open input file
        If (ISIZE .lt. 0) Go to 1000    !error
        IERR = IENTER (OCHAN , DBLKO, ISIZE) !open output file
        If (IERR .lt. 0) Go To 1100     !error
        Do 500, BLK = 0, ISIZE-1        !copy all the blocks
            IERR = IREADC (WCNT, BUFFER, BLK, ICHAN, FRCVDM)
            If (IERR .ne. WCNT) Go To 1200
C           here you could do something else while waiting for I/O
            Call SUSPND                 !wait for completion routine
            IERR = IWRITC (WCNT, BUFFER, BLK, OCHAN, FRCVDM)
            If (IERR .ne. WCNT) Go To 1300
C           here you could do something else while waiting for I/O
            Call SUSPND                 !wait for completion routine
500         Continue
        Call CLOSEC (ICHAN)
        Call CLOSEC (OCHAN)
        Call EXIT (SUCCS)
C
1000    Type *, '?FREADC-F-LOOKUP failed, code = ', ISIZE
        Call EXIT (FATAL)
1100    Type *, '?FREADC-F-ENTER failed, code = ', IERR
        Call EXIT (FATAL)
1200    Type *, '?FREADC-F-READC failed, code = ', IERR
        Call EXIT (FATAL)
1300    Type *, '?FREADC-F-WRITC failed, code = ', IERR
        Call EXIT (FATAL)
        End
```

READF/IREADF

READF/IREADF issued as either as a function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. Control returns to the user program immediately after the IREADF function is initiated. When the operation is complete, the specified FORTRAN subprogram (*frtn*) is entered as an asynchronous completion routine.

Form:

```
CALL READF (wcnt,buff,blk,chan,area,frtn)
 i = IREADF (wcnt,buff,blk,chan,area,frtn)
```

where:

wcnt is the integer number of words to be transferred

buff is the array to be used as the buffer; this array must contain at least *wcnt* words

blk is the integer block number of the file to be used. The user program normally updates *blk* before it is used again. The first block of a file is block number 0

chan is the integer specification for the RT–11 channel to be used

area is a four-word area to be set aside for link information; this area must not be modified by the FORTRAN program or swapped over by the USR. This area can be reclaimed by other FORTRAN completion functions when *frtn* has been activated

frtn is the FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the routine that issues the READF call. See description of completion routines.

Function Return:

i = Normal return; *i* equals the number of words requested (0 for non-file-structured read, multiple of 256[decimal] for file-structured read). If the read is from a magtape, the number of words requested is returned. For example:

- If *wcnt* is a multiple of 256 and less than that number of words remain in the file, *i* is shortened to the number of words that remain in the file; thus, if *wcnt* is 512 and only 256 words remain, i=256.

- If *wcnt* is not a multiple of 256 and more than *wcnt* words remain in the file, *i* is rounded up to the next block; thus, if *wcnt* is 312 and more than 312 words remain, i = 512, but only 312 are read.
- If *wcnt* is not a multiple of 256 and less than *wcnt* words remain in the file, *i* equals a multiple of 256 that is the actual number of words being read.

Errors:

Value	Meaning
i = -1	Attempt to read past end-of-file; no words remain in the file.
= -2	Hardware error occurred on channel.
= -3	Specified channel is not open.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
       Program FREADF
C
C       demonstrate READF and WRITF routines
C
        Integer*2 ICHAN, OCHAN          !channel numbers
        Integer*2 BLK                   !current block number
        Integer*2 SIZE                  !file size (hi BLK+1)
        Parameter WCNT = 256            !word cound
        Integer*2 BUFFER(WCNT)          !buffer
        Integer*2 ERROR                 !error indicator
        Common /JFWCCW/ ICHAN, OCHAN, BLK, SIZE, BUFFER, ERROR
        Integer*2 DBLK1(4), DBLK2(4)    !file names
        Data DBLK1 /3rSY , 3rRT1, 3r1XM, 3rSYS/
        Data DBLK2 /3rDK , 3rRT1, 3r1XM, 3rTMP/
C
        ICHAN = IGETC()
        OCHAN = IGETC()
        SIZE = LOOKUP (ICHAN, DBLK1)    !open input
        If (SIZE .lt. 0) Go To 1000
        IERR = IENTER (OCHAN, DBLK2, SIZE) !open output
        If (IERR .lt. 0) Go To 1100
        SIZE = SIZE - 1                 !highest block number
        BLK = -1                        !since we preincrement in FREADG
        Call FREADG (0)                 !start the I/O
C       here we could do other stuff while to I/O is happening
        Call SUSPND                     !wait for I/O to finish
        If (ERROR .eq. 0) Go To 900     !success
        Type *, '?FREADF-F-A completion routine reported code = ', ERROR
        Call EXIT (FATAL)
C
900     Type *, '!FREADF-I-Success'
```

```
        Call EXIT (SUCCS)
1000    Type *, '?FREADF-F-LOOKUP failed, code = ', SIZE
        Call EXIT (FATAL)
1100    Type *, '?FREADF-F-ENTER failed, code = ', IERR
        Call EXIT (FATAL)
        End

        Subroutine FREADG (STATUS)
C
C       Completion routine for WRITF (does a READF)
C
        Integer*2 STATUS                !status of previous operation
        Integer*2 ICHAN, OCHAN          !channel numbers
        Integer*2 BLK                   !current block number
        Integer*2 SIZE                  !file size (hi BLK+1)
        Parameter WCNT = 256            !word cound
        Integer*2 BUFFER(WCNT)          !buffer
        Integer*2 ERROR                 !error indicator
        Common /JFWCCW/ ICHAN, OCHAN, BLK, SIZE, BUFFER, ERROR
        Integer*2 AREA(4)               !linkage area
        External FREADH
C
        If (IAND (STATUS, 1) .ne. 0) Go To 1000
        BLK = BLK + 1                   !read next block
        If (BLK .gt. SIZE) Go To 900
        IERR = IREADF (WCNT, BUFFER, BLK, ICHAN, AREA, FREADH)
        If (IERR .ne. WCNT) Go To 1100
        Return
C
900     ERROR = 0
        Call RESUME
        Return
1000    ERROR = -2                      !hard error
        Call RESUME
        Return
1100    ERROR = IERR                    !returned error
        Call RESUME
        Return
        End

        Subroutine FREADH (STATUS)
C
C       Completion routine for READF (does a WRITF)
C
        Integer*2 STATUS                !status of previous operation
        Integer*2 ICHAN, OCHAN          !channel numbers
        Integer*2 BLK                   !current block number
        Integer*2 SIZE                  !file size (hi BLK+1)
        Parameter WCNT = 256            !word cound
        Integer*2 BUFFER(WCNT)          !buffer
        Integer*2 ERROR                 !error indicator
        Common /JFWCCW/ ICHAN, OCHAN, BLK, SIZE, BUFFER, ERROR
        Integer*2 AREA(4)               !linkage area
        External FREADG
C
        If (IAND (STATUS, 1) .ne. 0) Go To 1000
        IERR = IWRITF (WCNT, BUFFER, BLK, ICHAN, AREA, FREADG)
```

```
        If (IERR .ne. WCNT) Go To 1100
        Return
C
1000    ERROR = +2                      !hard error
        Call RESUME
        Return
1100    ERROR = -IERR                   !returned error
        Call RESUME
        Return
        End
```

READW/IREADW/MREADW

READW/IREADW/MREADW issued either as a function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. Control returns to the user program when the transfer is complete or when an error is detected.

Form:

```
CALL READW (wcnt,buff,blk,chan)
 i = IREADW (wcnt,buff,blk,chan)
CALL MREADW (wcnt,buff,blk,chan[,BMODE=strg])
 i = MREADW (wcnt,buff,blk,chan[,BMODE=strg])
```

where:

wcnt	is the integer number of words to be transferred
buff	is the array to be used as the buffer; this array must contain at least *wcnt* words
blk	is the integer block number of the file to be read. The user program normally updates *blk* before it is used again
chan	is the integer specification for the RT–11 channel to be used
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.

Function Return:

i = Normal return; *i* equals the number of words requested (0 for non-file-structured read, multiple of 256[decimal] for file-structured read). If the read is from a magtape, the number of words requested is returned. For example:

- If *wcnt* is a multiple of 256 and less than that number of words remain in the file, *i* is shortened to the number of words that remain in the file; thus, if *wcnt* is 512 and only 256 words remain, i=256.

- If *wcnt* is not a multiple of 256 and more than *wcnt* words remain in the file, *i* is rounded up to the next block; thus, if *wcnt* is 312 and more than 312 words remain, i = 512, but only 312 are read.

- If *wcnt* is not a multiple of 256 and less than *wcnt* words remain in the file, *i* equals a multiple of 256 that is the actual number of words being read.

Errors:

	Value	Meaning
i	= -1	Attempt to read past end-of-file; no words remain in the file.
	= -2	Hardware error occurred on channel.
	= -3	Specified channel is not open.
	= -19	Invalid BMODE value.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FREADW
C        Demonstrate IREADW
C        Scan SRC: and try to find this source file
         Parameter SUCCS = '001'o, FATAL = '010'o
         Parameter NULL = '000'o, HT = '011'o
         Parameter LF = '012'o, CR = '015'o
         Integer*2 BUF(256)      !block buffer
         Byte CBUF(512)          !char overlay for BUF
         Equivalence (CBUF(1), BUF(1))
         Integer*2 CHAN          !channel number to use
         Integer*2 SRC(2)        !device dblk
         Data SRC /3rSRC, 0/     !whole device (no file name)
         Integer*2 BLK           !current block number
         Integer*2 IERR          !function return value
         Byte Search(18)         !our first few chars
         Data Search /HT, 'P', 'r', 'o', 'g', 'r', 'a', 'm',
         1   ' ', 'F', 'R', 'E', 'A', 'D', 'W', CR, LF, NULL/
C
         CHAN = IGETC()          !allocate a channel
         BLK = 0                 !begin at the beginning
         Call LOOKUP (CHAN, SRC) !open the device
100      Continue
             IERR = IREADW (256, BUF, BLK, CHAN)
             If (IERR .ne. 256) Go To 200 !failure
             Call SCOPY (CBUF, CBUF, 17) !truncate the string
             If (ISCOMP (CBUF, SEARCH) .eq. 0)
         1       Type 101, BLK   !Eureka!
101          Format (' ', '!FREADW-I-Self discovery at block ', i6)
             BLK = BLK + 1       !try next block
             Go To 100
200      Continue
         If (IERR .eq. -2) Go To 400 !hard err is expected
         If (IERR .eq. -1) Type 102
102      Format (' ', '?FREADW-F-EOF from IREADW')
         If (IERR .ne. -1) Type 103, IERR
103      Format (' ', '?FREADW-F-Unexpected error from IREADW', i6)
         Call EXIT (FATAL)
400      Type 105
105      Format (' ', '!FREADW-I-Success (probably)')
         Call EXIT (SUCCS)
         End
```

RENAM/IRENAM

Multijob

RENAM/IRENAM causes an immediate change of the name of a specified file.

Form:

```
CALL RENAM (chan,dblk)
 i = IRENAM (chan,dblk)
```

where:

chan is the integer specification for the RT–11 channel to be used for the operation. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call. The channel is again available for use once the rename operation is completed

dblk is the eight-word area specifying the name of the existing file and the new name to be assigned. If considered as an eight-element INTEGER*2 array, *dblk* has the form:

- Words 1-4 specify the Radix–50 file descriptor for the old file name
- Words 5-8 specify the Radix–50 file descriptor for the new file name

NOTE

IRENAM arguments must be positioned so USR does not swap over them.

If a file already exists with the same name as the new file on the indicated device, it is deleted. IRENAM requires that the handler to be used be resident at the time the IRENAM is issued. If it is not, a monitor error occurs. The device names specified in the file descriptors must be the same.

For more information on renaming files, see the .RENAME programmed request.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Specified channel is already open.
= 2	Specified file was not found.
= 3	Invalid operation.
= 4	A file by that name already exists and is protected.

Error message *TRAP $MSARG* will display if *chan* or *dblk* argument is missing.

Example:

```
       Program FRENAM
C
C       Demonstrate renaming a file
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 CHAN          !i/o channel
        Integer*2 IERR          !error code
        Integer*2 DBLK2(8)      !a pair of DBLKS
        Data DBLK2
        1 /3rDK , 3rFAT, 3r    , 3rTMP,
        2  3rDK , 3rPOR, 3rTLY , 3rTMP/
C
        CHAN = IGETC ()         !get a channel
        IERR = IRENAM (CHAN, DBLK2) !apply euphemism
        If (IERR .ne. 0) Go To 100 !problem
        Call PRINT ('!FRENAM-I-Success')
        Call EXIT (SUCCS)
100     Continue
        Type *, '?FRENAM-F-Unexpected error', IERR
        Call EXIT (FATAL)
        End
```

REOPN/IREOPN

Multijob

REOPN/IREOPN identifies a specified channel with a file on which an ISAVES was performed. The ISAVES/IREOPN combination is useful when a large number of files must be operated on at one time. Necessary files can be opened with LOOKUP and their status preserved with ISAVES. When data is required from a file, an IREOPN enables the program to read from the file. IREOPN is not required to reference same channel as the original LOOKUP and ISAVES.

Form:

```
CALL REOPN (chan,cblk)
 i = IREOPN (chan,cblk)
```

where:

chan is the integer specification for the RT–11 channel to be associated with the reopened file; this channel must be initially inactive

cblk is the five-word block where the channel status information was stored by a previous ISAVES. This block, considered as a five-element INTEGER*2 array, has the following format:

cblk (1)	Channel status word.
cblk (2)	Starting block number of the file; zero for non-file-structured devices.
cblk (3)	Length of file (in 256-word blocks).
cblk (4)	Reserved for future use.
cblk (5)	Two information bytes: • Even byte: I/O count of the number of requests outstanding on this channel. • Odd byte: Unit number of the device associated with the channel.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Specified channel is already in use.

Error message *TRAP $MSARG* will display if *chan* or *cblk* argument is missing.

Example:

```
       Program FREOPN
C
C       Demo SAVES and REOPN operations on channels
C
        Parameter WCNT = 256
        Integer*2 DBLK1(4), DBLK2(4)    !file names
        Data DBLK1 /3rSRC, 3rFRE, 3rOPN, 3rFOR/
        Data DBLK2 /3rBIN, 3rFRE, 3rOPN, 3rSAV/
        Integer*2 DBLK3(4)              !TT:
        Data DBLK3 /3rTT , 0, 0, 0/
        Integer*2 CBLK1(5), CBLK2(5)    !channel blocks
        Integer*2 ICHAN, OCHAN          !channel numbers
        Integer*2 BUFFER(WCNT)          !block buffer
        Integer*2 BLK                   !block number
C
        ICHAN = IGETC()                 !get a channel
        IERR = LOOKUP (ICHAN, DBLK1)    !lookup the source file
        CALL SAVES (ICHAN, CBLK1)       !save the src lookup
        IERR = LOOKUP (ICHAN, DBLK2)    !lookup the sav file
        Call SAVES (ICHAN, CBLK2)       !save the sav lookup
        Type 100, CBLK1(3), CBLK2(3)    !compare sizes
100     Format (' ', 'Source = ', i6, '  save = ', i6///)
        Call IREOPN (ICHAN, CBLK1)      !now open the src again
        OCHAN = IGETC()                 !get an output channel
        Call LOOKUP (OCHAN, DBLK3)      !open TT:
        Do 200 BLK = 0, CBLK1(3)-1      !reveal ourselves
            Call READW (WCNT, BUFFER, BLK, ICHAN)
            Call WRITW (WCNT, BUFFER, BLK, OCHAN)
200         Continue
        Call CLOSEC (ICHAN)
        Call CLOSEC (OCHAN)
        End
```

REPEAT

The REPEAT subroutine concatenates a specified string with itself to produce the indicated number of copies. REPEAT places the resulting string in a specified array.

Form:

CALL REPEAT (in,out,i[,len][,err])

where:

in is the array containing the string to be repeated; it must be terminated with a null byte

out is the array into which the resultant string is placed. This array must be at least one element longer than the value of len, if len is specified. Because string handling always creates null terminated strings, *out* will be a null terminated string when it returns.

i is the integer number of times to repeat the string

len is the integer number representing the maximum length of the output string

err is the logical error flag set if the output string is truncated to the length specified by *len*

Input and output strings can specify the same array only if the repeat count (i) is 1 or 0. When the repeat count is 1, this routine is the equivalent of SCOPY; when the repeat count is 0, out is replaced by a null string. The old contents of *out* are lost when this routine is called.

Errors:

Error conditions are indicated by *err*, if specified. If *err* is given and the output string will be longer than *len* characters, then *err* is set to .TRUE.; otherwise, *err* is unchanged.

NOTE

The argument *err* must be specified as BYTE in FORTRAN 77. It can be any logical type in FORTRAN IV and any integer type in PDP–11 C.

Error message *TRAP $MSARG* will display if argument *a, b,* or *out* is missing.

Example:

```
       Program SREPEA
C
C       This will build a ruler of length 1 to 99
C
        Integer*2 SIZE          !length of ruler
        Byte LINE1(100), LINE2(100) !strings for ruler
C
        SIZE = 12
        Call Ruler (SIZE, LINE1, LINE2)
        Call PRINT (LINE1)
        Call PRINT (LINE2)
        SIZE = 36
        Call Ruler (SIZE, LINE1, LINE2)
        Call PRINT (LINE1)
        Call PRINT (LINE2)
        End

        Subroutine RULER (SIZE, LINE1, LINE2)
        Integer*2 SIZE          !length of ruler
        Byte LINE1(*), LINE2(*) !ruler strings
        Byte NUM19(11)          !constant
        Data NUM19 /'1', '2', '3', '4', '5',
        1           '6', '7', '8', '9', '0', '000'o/
C
        Call REPEAT ('0', LINE1(1), 9, SIZE)
        Do 100 I = 1, SIZE/10
            ICHAR = NUM19(I)    !get single char with null term
            Call REPEAT (ICHAR, LINE1(I*10), 10, SIZE+1-(I*10))
100         Continue
        Call REPEAT (NUM19, LINE2(1), 10, SIZE)
        Return
        End
```

RESUME

The RESUME subroutine allows a job to resume execution of the main program. A RESUME call is normally issued from an asynchronous FORTRAN routine entered on I/O completion or because of a schedule request (See SUSPND subroutine).

Form:

```
CALL RESUME
```

Errors:
None.

Example:
See RCVDF.

SAVES/ISAVES

Multijob

SAVES/ISAVES stores five words of channel status information into a user-specified array. These words contain all the information that RT–11 requires to completely define an RT–11 file. (Special directory devices cannot have their file status saved with this request.) When an ISAVES is finished, the data words are placed in memory and the specified channel is closed, so that it is again available for use. When the saved channel data is required, the IREOPN function is used.

ISAVES can be used only if a file was opened with a LOOKUP call. If IENTER was used, ISAVES returns an error. Note that ISAVES is not valid on magtape or cassette files.

Form:

```
CALL SAVES (chan,cblk)
 i = ISAVES (chan,cblk)
```

where:

chan is the integer specification for the RT–11 channel whose status is to be saved. You must obtain this channel through an IGETC call, or you can use channel 16 or higher if you have done an ICDFN call

cblk is a five-word block in which the channel status information describing the open file is stored (See IREOPN for format of this block)

The ISAVES/IREOPN combination is very useful, but care must be exercised when using it. In particular, the following cases should be avoided.

If an ISAVES is performed on a file and the same file is then deleted before it is reopened, the space occupied by the file becomes available as an empty space which could then be used by another file. If this sequence occurs, there is a change in the contents of the file whose status was supposedly saved.

Although the handler for the required peripheral need not be in memory for execution of an IREOPN, a fatal error is generated if the handler is not in memory when an IREAD or IWRITE is executed.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	The specified channel is not currently associated with any file.
= 2	The file was opened with an IENTER call.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See REOPN.

SCCA/ISCCA/MSCCA

SCCA/ISCCA provides a CTRL/C intercept to:

- Inhibit a CTRL/C abort.
- Indicate that a CTRL/C has been entered.
- Distinguish between single and double CTRL/C command.

Global support provides an ISCCA function variant. An optional parameter, *itype*, provided for both SCCA and ISCCA subroutines lets you set local or global SCCA support. An optional parameter, *AMODE*, lets you specify a Supervisor data space address. See *RT–11 System Macro Library Manual* for .SCCA information on local and global SCCA support.

Form:

```
CALL SCCA [([iflag][,itype])]
 i = ISCCA ([iflag][,itype])
CALL MSCCA [([iflag][,itype][,AMODE=strg])]
 i = MSCCA ([iflag][,itype][,AMODE=strg])
```

where:

iflag is an integer terminal status word that must be tested and cleared to determine if two CTRL/Cs were typed at the console terminal; the *iflag* must be an INTEGER*2 variable (not LOGICAL*1).

itype is the mode of SCCA operation.

Specify 0 (the default) or 1 to select LOCAL or GLOBAL SCCA support:

itype = 0	LOCAL (default) mode of SCCA operation
itype = 1	GLOBAL mode of SCCA operation

i is the returned address of the previous SCCA terminal status word

AMODE=strg Specify *AMODE* string "S" to select a Supervisor-Data address.

Notes

When a CTRL/C is typed, if SCCA is in effect, it is placed in the input ring buffer. While residing in the buffer, the character can be read by the program. The program must test and clear the *iflag* to determine if two CTRL/C commands were typed consecutively. The *iflag* is set to non-zero when two CTRL/Cs are typed together. It is the responsibility of the program to abort itself, if appropriate, on an input of CTRL/C from the terminal. The SCCA subroutine with no argument disables the CTRL/C intercept. A CTRL/C from indirect command files is not intercepted by SCCA.

Errors:

Value	Meaning
i = -19	Invalid AMODE value.

Error message *TRAP $MSARG* will display if argument *itype* is missing.

Example:

```
        Program FSCCA
C
C       Demonstrate SCCA preventing job termination by ^C
C
        Type 1
1       Format (' ', 'Enter chars, including a ^C or ^C^C and ret' /)
        IFLAG = 0               !init flag word
        Call SCCA (IFLAG)       !protect from ^C
10      If (ITTINR() .ne. '003'o) Go To 10 !look for ^C
        Type 2
2       Format (' ', 'A ^C was typed' /)
        If (IFlag .eq. 0) Go To 10 !continue until ^C^C
        Type 3
3       Format (' ', 'Actually a double ^C^C' /)
        Call SCCA ()            !unprotect from ^C
        Type 4
4       Format (' ', 'Enter ^C to terminate program' /)
20      Go To 20                !loop, see loop
        End
```

SCHED/ISCHED

SB Timer (SYSGEN Option)

SCHED/ISCHED schedules a specified FORTRAN subroutine to be run as an asynchronous completion routine at a specified time of day.

Form:

```
CALL SCHED (hrs,min,sec,tick,area,id,frtn)
 i = ISCHED (hrs,min,sec,tick,area,id,frtn)
```

where:

hrs is the integer number of hours

min is the integer number of minutes

sec is the integer number of seconds

tick is the integer number of ticks (1/60 of a second on 60-Hz clocks; 1/50 of a second on 50-Hz clocks)

area is a four-word area that must be provided for link information; this area must never be modified by the FORTRAN program, and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion functions when crtn has been activated

id is the integer identification to be passed to the routine being scheduled

frtn is the name of the FORTRAN subroutine to be entered at the time of day specified. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISCHED call. The subroutine has one argument. For example:

```
SUBROUTINE frtn(id)
INTEGER id
```

When the routine is entered, the value of the integer argument is the value specified for *id* in the appropriate ISCHED call

Notes

- The scheduling request made by ISCHED can be canceled at a later time by an ICMKT function call.
- If the system is busy, the actual time of day that the completion routine is run may be later than the requested time of day.
- A FORTRAN subroutine can periodically reschedule itself by issuing its own ISCHED or ITIMER calls from within the routine.
- ISCHED requires a queue element; this should be considered when the IQSET function is executed.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No queue elements available; unable to schedule request.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FSCHED
C
C       This program uses timer support to announce lunch.
C
        Integer*2 LINK1(4)      !linkage area for FTN completion
C
        IERR = IQSET (2)        !add some queue elements
        IERR = ISCHED (12, 0, 0, 0, LINK1, 73, NOON)
C
        Type *, 'Do some work'
        Call SUSPND
        Type *, 'Off to lunch'
        End

        Subroutine NOON (ID)
        Type *, 'Time for LUNCH'
        Call RESUME
        Return
        End
```

SCOMP/ISCOMP

The SCOMP/ISCOMP routine compares two character strings and returns the integer result of the comparison.

Form:

```
CALL SCOMP (a,b,i)
 i = ISCOMP (a,b)
```

where:

a	is the array containing the first string; it must be terminated with a null byte
b	is the array containing the second string; it must be terminated with a null byte
i	is the integer variable that receives the result of the comparison

The strings are compared from left to right, one character at a time, using the collating sequence specified by the ASCII codes for each character. If the two strings are not equal, the absolute value of variable i (or the result of the function ISCOMP) is the character position of the first inequality found. Strings are terminated by a null (0) character.

If the strings are not the same length, the shorter one is treated as if it were padded on the right with blanks to the length of the other string. A null string argument is equivalent to a string containing only blanks.

Function Results:

i < 0	If *a* is less than *b*.
= 0	If *a* is equal to *b*.
> 0	If *a* is greater than *b*.

Errors:
Unpredictable results will occur if any argument is omitted.

Example:
See SDTTM.

SCOPY

The SCOPY routine copies a character string from one array to another. Copying stops either when a null (0) character is encountered or when a specified number of characters have been moved.

Form:

CALL SCOPY (in,out[,len][,err])

where:

in is the array containing the string to be copied; it must be terminated with a null byte if *len* is not specified, or if the string is shorter than *len*

out is the array to receive the copied string. This array must be at least one element longer than the value of *len*, if *len* is specified. It also must be terminated with a null byte if *len* is specified

len is the integer number representing the maximum length of the output string. The effect of *len* is to truncate the output string to a given length, using null termination

err is a logical variable that receives the error indication if the output string was truncated to the length specified by len

NOTE

The argument *err* must be specified as LOGICAL*1 in FORTRAN 77. It can be any logical type in FORTRAN IV and any integer type in PDP–11C.

The input *(in)* and output *(out)* arguments can specify the same array. The string previously contained in the output array is lost when this subroutine is called.

Errors:

Error conditions are indicated by *err*, if specified. If *err* is given and the output string was truncated to the length specified by *len*, then *err* is set to .TRUE.; otherwise, *err* is unchanged.

Unpredictable results will occur if either *in* or *out*argument is omitted.

Example:
See INSERT.

*SDAT/*SDATC/*SDATF/*SDATW

Multijob Only

Four forms of *SDAT, are used with the IRCVD, IRCVDC, IRCVDF, and IRCVDW calls to allow message transfers under the FB or XM monitor. Note that the buffer containing the message should not be modified or reused until the message has been received by the other job. These functions require a queue element, which should be considered when the IQSET function is executed.

Specify mapping for MSDAT, MSDATC and MSDATW by optional parameters *BMODE* and *CMODE*.

SDAT/ISDAT/MSDAT

SDAT/ISDAT/MSDAT transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued. When your program needs to wait until the other program has received the data, an IWAIT function should be issued to ensure that the ISDAT operation has been completed. If an error occurred during the transfer, the IWAIT function indicates the error.

Form:

```
CALL SDAT (buff,wcnt)
 i = ISDAT (buff,wcnt)
CALL MSDAT (buff,wcnt[,BMODE=strg])
 i = MSDAT (buff,wcnt[,BMODE=strg])
```

where:

buff	is the array containing the data to be transferred
wcnt	is the integer number of data words to be transferred
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode for the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No such job currently exists in the system. A job exists as long as it is loadable, whether or not it is active.
= -19	Invalid BMODE value.

Error message *TRAP $MSARG* will display if *buff* or *wcnt* argument is missing.

Example:
See also *RCVD.

*SDAT/*SDATC/*SDATF/*SDATW

```
        Program FSDAT !demo SDAT (use with FRCVDx)
C
C       Try to send a message to the other job
C       send a null message for EOF and exit
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter WCNT = 42
        Integer*2 SNDCNT
        Integer*2 BUFFER(WCNT)
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/             !^Z null string equiv
        External LEN                    !use RT-11 SYSLIB LEN func
C
100     Continue
            Call GTLIN (BUFFER)         !get a line
            SNDCNT = LEN (BUFFER)       !get byte count (-1)
            If (SNDCNT .eq. 0) Then     !is it a null string
                BUFFER(1) = CTRLZ       !yes, send ^Z
                SNDCNT = 1              !...
            End If
            SNDCNT = (SNDCNT + 2) / 2   ! get word count
200             Continue
                IERR = ISDAT (BUFFER, SNDCNT) !try sending a message
                If (IERR .eq. 1) Go To 200 !wait for the job
            If (IERR .ne. 0) Go To 300  !unknown error
            Call MWAIT                  !wait for a message
            If (BUFFER(1) .ne. CTRLZ) Go To 100
        Call PRINT ('!FSDAT-I-Normal termination')
        Call EXIT (SUCCS)               !and done
C
300     Call PRINT ('?FSDAT-F-Unknown error code from ISDAT')
        Call EXIT (FATAL)
        End
```

SDATC/ISDATC/MSDATC

SDATC/ISDATC/MSDATC transfers a specified number of words from one job to another. Control returns to the user program immediately after the transfer is queued. When the other job accepts the message through a receive data request, the specified assembly language routine (*crtn*) is activated as an asynchronous completion routine.

Form:

```
CALL SDATC (buff,wcnt,crtn)
i = ISDATC (buff,wcnt,crtn)
CALL MSDATC (buff,wcnt,crtn[,BMODE=strg][,CMODE=strg])
i = MSDATC (buff,wcnt,crtn[,BMODE=strg][,CMODE=strg])
```

where:

buff	is the array containing the data to be transferred
wcnt	is the integer number of data words to be transferred
crtn	is the name of an assembly language routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATC call
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.
CMODE=strg	Specify *strg* as string "S" to specify a Supervisor address.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No such job currently exists in the system. A job exists as long as it is loadable, whether or not it is active.
= -19	Invalid BMODE or CMODE value.

Error message *TRAP $MSARG* will display if *buff, wcnt* or *crtn* argument is missing.

Example:

```
        Program FSDATC !demo SDATC (use with FRCVDx)
C
C       Try to send a message to the other job
C       after sending a null message, exit
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter WCNT = 42
        Integer*2 SNDCNT
        Integer*2 BUFFER(WCNT)
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/             !^Z null string equiv
        External FRCVDM                 !completion routine
        External LEN                    !use RT-11 SYSLIB LEN func
C
```

```
100     Continue
            Call GTLIN (BUFFER)
            SNDCNT = LEN (BUFFER)       !get byte count (-1)
            If (SNDCNT .eq. 0) Then     !is it a null string
                BUFFER(1) = CTRLZ       !yes, send ^Z
                SNDCNT = 1              !...
            End If
            SNDCNT = (SNDCNT + 2) / 2   ! get word count
200             Continue
                IERR = ISDATC (BUFFER, SNDCNT, FRCVDM) !try sending a message
                If (IERR .eq. 1) Go To 200 !wait for the job
            If (IERR .ne. 0) Go To 400  !unknown error
C
C       other processing not requiring message could be done here
C
            Call SUSPND                 !wait for completion routine to
C                                       ! resume us
            If (BUFFER(1) .ne. CTRLZ) Go To 100
300     Call PRINT ('!FSDATC-I-Successful normal termination')
        Call EXIT (SUCCS)               !and done
C
400     Call PRINT ('?FSDATC-F-Unknown error code from ISDATC')
        Call EXIT (FATAL)
        End
```

SDATF/ISDATF

SDATF/ISDATF transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued and execution continues. When the other job accepts the message through a receive data request, the specified FORTRAN subprogram (*frtn*) is activated as an asynchronous completion routine.

Form:

```
CALL SDATF (buff,wcnt,frtn)
 i = ISDATF (buff,wcnt,area,frtn)
```

where:

buff is the array containing the data to be transferred

wcnt is the integer number of data words to be transferred

area is a four-word area to be set aside for link information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion functions when *crtn* has been activated

frtn is the name of a FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATF call

Function Return:

i = Normal return; *i* equals the number of words requested (0 for non-file-structured read, multiple of 256[decimal] for file-structured read). If the read is from a magtape, the number of words requested is returned. For example:

- If *wcnt* is a multiple of 256 and less than that number of words remain in the file, *i* is shortened to the number of words that remain in the file; thus, if *wcnt* is 512 and only 256 words remain, i=256.
- If *wcnt* is not a multiple of 256 and more than *wcnt* words remain in the file, *i* is rounded up to the next block; thus, if *wcnt* is 312 and more than 312 words remain, i = 512, but only 312 are read.
- If *wcnt* is not a multiple of 256 and less than *wcnt* words remain in the file, *i* equals a multiple of 256 that is the actual number of words being read.

Errors:

Value		Meaning
i =	0	Normal return.

= 1 No such job currently exists in the system. A job exists as long as it is loadable, whether or not it is active.

= -19 Invalid BMODE value.

Error message *TRAP $MSARG* will display if *buff* or *wcnt* argument is missing.

Example:

```
        Program FSDATF !demo SDATF (use with FRCVDx)
C
C       Try to send a message to the other job
C       after sending a null message, exit
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter WCNT = 42
        Integer*2 SNDCNT
        Integer*2 BUFFER(WCNT)
        Common /DATA/ SNDCNT, BUFFER
        Integer*2 LINKAG(4)             !area for linkage
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/             !^Z null string equiv
        External FSDATG                 !completion routine
        External LEN                    !use RT-11 SYSLIB LEN function
C
100     Continue
            Call GTLIN (BUFFER)         !get a line to send
            SNDCNT = LEN (BUFFER)       !get byte count (-1)
            If (SNDCNT .eq. 0) Then     !is it a null string
                BUFFER(1) = CTRLZ       !yes, send ^Z
                SNDCNT = 1              !...
            End If
            SNDCNT = (SNDCNT + 2) / 2   ! get word count
200             Continue
                IERR = ISDATF (BUFFER, SNDCNT, LINKAG, FSDATG)
C                                       !try sending a message
                If (IERR .eq. 1) Go To 200 !wait for the job
        If (IERR .ne. 0) Go To 300      !unknown error
C
C       other processing not requiring message could be done here
C
        Call SUSPND                     !wait for completion routine to
C                                       ! resume us
        Call PRINT ('!FSDATF-!-Termination successfully completed')
        Call EXIT (SUCCS)               !and done
C
300     Call PRINT ('?FSDATF-F-Unknown error code from ISDATF')
        Call EXIT (FATAL)
        End

        Subroutine FSDATG       !completion routine for FSDATF
        Parameter FATAL = '010'o
        Parameter WCNT = 42
        Integer*2 SNDCNT
        Integer*2 BUFFER(WCNT)
        COMMON /DATA/ SNDCNT, BUFFER
        Integer*2 LINKAG(4)             !area for linkage
```

```
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/             !^Z null string equiv
        External FSDATH                 !fakeout the recursion detection
        External LEN                    !use RT-11 SYSLIB LEN func
C
        Call GTLIN (BUFFER)             !get a line for sending
        SNDCND = LEN (BUFFER)           !get length
            If (SNDCNT .eq. 0) Then     !is it a null string
                BUFFER(1) = CTRLZ       !yes, send ^Z
                SNDCNT = 1              !...
            End If
            SNDCNT = (SNDCNT + 2) / 2   ! get word count
        IERR = ISDATF (BUFFER, SNDCNT, LINKAG, FSDATH) !try for next message
        If (BUFFER(1) .eq. CTRLZ) Go To 100 !"EOF"
        If (IERR .ne. 0) Go To 200      !unknown error
        Return
C
100     Call RESUME                     !restart mainline so it can exit
        Return
C
200     Call PRINT ('?FSDATG-F-Unknown error code from ISDATG')
        Call EXIT (FATAL)               !can't really exit cleanly from
C                                       !completion, but dying is ok anyway
        End

        .TITLE  FSDATH  -- Just call FSDATG

FSDATH::CALLR   FSDATG  ;slip around recursion detection
                        ;in the compiler
        .END
```

SDATW/ISDATW/MSDATW

SDATW/ISDATW/MSDATW transfers a specified number of words from one job to the other. Control returns to the user program when the other job has accepted the data through a receive data request.

Form:

CALL SDATW (buff,wcnt)
i = ISDATW (buff,wcnt)
CALL MSDATW (buff,wcnt[,BMODE=strg])
i = MSDATW (buff,wcnt[,BMODE=strg])

where:

buff	is the array containing the data to be transferred
wcnt	is the integer number of data words to be transferred
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode of the *buff* argument.

Function Return:

i = Normal return; *i* equals the number of words requested (0 for non-file-structured read, multiple of 256[decimal] for file-structured read). If the read is from a magtape, the number of words requested is returned. For example:

- If *wcnt* is a multiple of 256 and less than that number of words remain in the file, *i* is shortened to the number of words that remain in the file; thus, if *wcnt* is 512 and only 256 words remain, i=256.
- If *wcnt* is not a multiple of 256 and more than *wcnt* words remain in the file, *i* is rounded up to the next block; thus, if *wcnt* is 312 and more than 312 words remain, i = 512, but only 312 are read.
- If *wcnt* is not a multiple of 256 and less than *wcnt* words remain in the file, *i* equals a multiple of 256 that is the actual number of words being read.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No such job currently exists in the system. A job exists as long as it is loadable, whether or not it is active.
= -19	Invalid BMODE value.

Example:

```
        Program FSDATW !demo SDATW (use with FRCVDx)
C
C       Try to send a message to the other job
C       after sending a null message, exit
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Parameter WCNT = 42
        Integer*2 SNDCNT
        Integer*2 BUFFER(WCNT)
        Integer*2 CTRLZ
        Data CTRLZ /'032'o/             !^Z null string equiv
        External LEN                    !use RT-11 SYSLIB LEN func
C
100     Continue
            Call GTLIN (BUFFER)         !get a line to send
            SNDCNT = LEN (BUFFER)       !get byte count (-1)
            If (SNDCNT .eq. 0) Then     !is it a null string
                BUFFER(1) = CTRLZ       !yes, send ^Z
                SNDCNT = 1              !...
            End If
            SNDCNT = (SNDCNT + 2) / 2   ! get word count
200             Continue
                IERR = ISDATW (BUFFER, SNDCNT) !try sending a message
```

```
            If (IERR .eq. 1) Go To 200 !wait for the job
          If (IERR .ne. 0) Go To 300  !unknown error
          If (BUFFER(1) .eq. CTRLZ) Go To 400 !"EOF"
          Go To 100
C
300     Call PRINT ('?FSDATW-F-Unknown error code from ISDATW')
        Call EXIT (FATAL)
C
400     Call PRINT ('!FSDATW-I-Success')
        Call EXIT (SUCCS)               !and done
        End
```

SDTTM/ISDTTM

SDTTM/ISDTTM sets the system date and time. An argument of -1 leaves the corresponding value unchanged.

Form:

```
CALL SDTTM (date,hitime,lotime)
 i = ISDTTM (date,hitime,lotime)
```

where:

date	is the new system date
hitime	is the high-order time of day, in ticks past midnight
lotime	is the low-order time of day, in ticks past midnight

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FSDTTM
C
C       Accept date in German and set it
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        Byte MONATE(4, 12)
        Data MONATE             !month names
        1 /'J', 'A', 'N', 0,
        2  'F', 'E', 'B', 0,
        3  'M', 'A', 'R', 0,
        4  'A', 'P', 'R', 0,
        5  'M', 'A', 'I', 0,
        6  'J', 'U', 'N', 0,
        7  'J', 'U', 'L', 0,
        8  'A', 'U', 'G', 0,
        9  'S', 'E', 'P', 0,
        1  'O', 'K', 'T', 0,
        2  'N', 'O', 'V', 0,
        3  'D', 'E', 'C', 0/
        Byte INPUT(81)          !raw input
        Byte PROMPT(6)          !prompt
        Data PROMPT /'D', 'a', 't', 'e', '?', '200'o/
        Integer*2 DAY, YEAR     !day and year integer values
        Byte MONTH(4)           !month string value
        Data MONTH(4) /0/       !with suffixed null
        Character*10 DMMMY      !DD-MON-YY
```

```
C
        Call GTLIN (INPUT, PROMPT) !ask for date
        If (INPUT(2) .eq. '-') Then
            Call CONCAT ('0', INPUT, DMMMY) !D-MON-YY
            Else
            Call SCOPY (INPUT, DMMMY) !DD-MMM-YY
        End If
        Read (DMMMY, 101, ERR=200) DAY, DUMMY1, (MONTH(I), I=1,3),
        1   DUMMY2, YEAR
101     Format (i2, a1, 3a1, a1, i2)
        If ((DUMMY1 .ne. '-') .or. (DUMMY2 .ne. '-')) Go To 200
        DO 100 IMON = 1, 12
            If (ISCOMP (MONATE(1, IMON), MONTH) .eq. 0) Go To 300
100         Continue
200     Continue
        Type *, '?FSDTTM-F-Invalid input'
        Call EXIT (FATAL)
300     Continue
        If (YEAR .lt. 73) Go To 200
        If (DAY .gt. 31) Go To 200
        Call SDTTM (IMON*1024+DAY*32+YEAR-72, -1, -1)
        Call EXIT (SUCCS)
        End
```

SERR/ISERR

SERR/ISERR performs the following functions:

- Inhibits the monitor from aborting a job
- Causes an error return to the EMT that produced the error.

  Those error conditions are listed at the end of this section. ISERR itself returns no error codes.

If SERR is in effect and an error occurs on a channel, the channel must be closed by a CLOSEC or PURGE call. Otherwise, subsequent operations on that channel will fail.

Form:

```
CALL SERR ()
i = ISERR ()
```

where:

i is a returned INTEGER*2 result of the function, a flag indicating the previous IHERR/ISERR setting:

Value	Meaning
i = 0	IHERR was in effect
= 1	ISERR was in effect

Errors:

The following list contains error codes that can be returned to other EMT requests if SERR is in effect:

Value	Meaning
i = -129	Called USR from completion routine
= -130	No device handler; this operation needs one
= -131	Error doing directory I/O
= -132	.FETCH error. An I/O error occurred while the handler was being used, or an attempt was made to load the handler over USR or KMON
= -133	Error reading an overlay
= -134	No more room for files in the directory
= -135	Reserved
= -136	Invalid channel number; number is greater than actual number of channels that exist

= -137	Invalid EMT, and invalid function code has been decoded
= -138	Reserved
= -139	Reserved
= -140	Invalid directory
= -141	Unloaded XM handler
= -142 through = -146	Reserved

Example:
See HERR.

SETCMD

The SETCMD routine allows a user program to pass a command line to the keyboard monitor to be executed after the program exits. This routine can be used in a program running in the background. The command lines are passed to the chain information area (500-777 octal) and stored beginning at location 512(octal). No check is made to determine if the string extends into the stack space. For this reason, the command line should be short and the subroutine call should be made in the main program unit near the end of the program just before completion.

The monitor commands REENTER, START, and CLOSE are not allowed if the SETCMD feature is used.

Form:

CALL SETCMD (string)

where:

string is a keyboard monitor command line ASCIZ format with no embedded carriage returns or line feeds

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FSETCM
C
C        Issue any command supplied as input
C
         Byte INPUT(81), PROMPT(8)
         Data PROMPT /'P', 'r', 'o', 'm', 'p', 't', '?', '200'o/
C
         Call GTLIN (INPUT, PROMPT)      !get a command
         Call SETCMD (INPUT)
         End
```

SFDAT/ISFDAT

SFDAT/ISFDAT allows user programs to modify the creation date of an RT–11 file. The device must have an RT–11 file structure.

Form:

```
CALL SFDAT (chan,dblk[,idate][,iold])
 i = ISFDAT (chan,dblk[,idate][,iold])
```

where:

chan	is the integer value of the RT–11 channel to be used for the operation. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call
dblk	is the four word RT–11 file specification, in Radix–50, of the file whose date is being changed
idate	is the integer date in RT–11 date format
iold	is the INTEGER*2 result of the request; the original creation date (E.DATE) of the specified file

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Channel in use.
= 2	File not found.
= 3	Invalid operation for magtape.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
This example changes the date of the file DY1:OLD23.DAT to July 4, 1976.

```
        Program FSFDAT
C
C       This program sets the file date of 4 July 1976 on the
C       file DK:BI100L.TMP.
        Parameter SUCCS = '001'o, FATAL = '010'o
        Integer*2 DBLK(4)               !file name
        Data DBLK /3rDK , 3rBI1, 3r00L, 3rTMP/
C
        ICHAN = IGETC()                 !get a channel
        IDATE = 7*1024 + 4*32 + (1976-1972) !RT-11 format date
        IERR = ISFDAT (ICHAN, DBLK, IDATE) !set it
        If (IERR .ne. 0) Go To 100
        Call PRINT ('!FSFDAT-I-Success')
        Call EXIT (SUCCS)
100     Continue
        Call PRINT ('?FSFDAT-F-Revolting development')
        Call EXIT (FATAL)
        End
```

SFINF/ISFINF

SFINF/ISFINF function saves and then modifies the contents of the directory entry offset you specify from a file's directory entry. ISFINF is not supported for the distributed special directory handlers LP, LS, MM, MS, MT, MU, and SP.

Form:

```
CALL SFINF (chan,dblk,value,oper,offset,[iold])
 i = ISFINF (chan,dblk,value,oper,offset,[iold])
```

where:

chan is a BYTE or INTEGER*2 channel number

dblk is a 4-element INTEGER*2 array containing a 4-word device and file specification in Radix–50; the file specification for which you want to return directory entry information.

value is the value to be placed in the specified offset location.

For RT–11 file structured volume directories, if the offset is 0 (E.STAT) and the operation is a BIC or BIS, E.STAT bits 000400, 001000, and 004000 must be clear. If the offset is E.STAT and the operation is a MOV, only the bottom 4 bits of E.STAT are moved. For special directory volumes, no bit restrictions are enforced

oper is the name or octal value indicating the type of operation to be performed:

Name/Value	Type	Meaning
'G'	GET	Get value, an IGFINF operation
'C'	CLEAR	Bit clear (BIC) operation
'S'	SET	Bit set (BIS) operation
'M'	MOVE	Word move (MOV) operation
128-255	USER	Reserved for the user

offset is the octal byte offset for the directory entry word for this operation. The offset must be even, and cannot be 8 (E.LENG). For example, specifying offset 10 saves the current contents of E.USED in *iold* and opens that location for modification.

iold is the returned INTEGER*2 previous value in the modified directory entry word

Errors:

Value	Meaning
i = 0	Normal return
= -1	Channel was open
= -2	File not found
= -3	Invalid operation argument
= -4	Invalid offset
= -257	Required argument missing

Example:

```
       Program FSFINF !demo SFINF
C
C       This program modifies the directory entry at offset 10.,
C       setting it to a value based on the value entered for the
C       /T switch.  The /T switch is used as follows:
C
C       /T:11.          11:00:00
C       /T:11.:22.      11:22:00
C       /T:11.:22.:33.  11:22:33
C
C       The value placed in the entry is the number of seconds
C       past midnight in the switch value divided by 3.
C
        Parameter FATAL = '010'o
        Integer*2 CHAN          !Channel number
        Integer*2 FILSPC(39)    !DBLK(s) area
        Integer*2 DEFTYP(4)     !default types
        Integer*2 DBLK(4)       !DBLK
        Integer*2 SW(4,6)       !switch parsing table
        Parameter LETTER = 1, FLAG = 2, FILE = 3, VALUE = 4
        Integer*2 ERROR         !error / success value
        Integer*2 TIME          !time value to store
        Integer*2 HOUR          !hour part
        Integer*2 MINUTE        !minute part
        Integer*2 SECOND        !second part
        Equivalence (FILSPC (16), DBLK(1)) !use first input file only
        Data DEFTYP /4*0/       !no default types
        Data SW(LETTER,1) /'T'/, SW(LETTER,4) /'t'/
        Data SW(LETTER,2) /'T'/, SW(LETTER,5) /'t'/
        Data SW(LETTER,3) /'T'/, SW(LETTER,6) /'t'/
C
        CHAN = IGETC ()
1000        Continue
            Call Print (' ')            !clean up display
            ERROR = ICSI (FILSPC, DEFTYP, , SW, 6) !get file and switches
            If (ERROR .ne. 0) Go To 2100
            ERROR = IFETCH (DBLK)       !Get handler into memory
            If (ERROR .ne. 0) Go To 2200
            If (SW(FLAG,1) .eq. 0) Then !lower case 't' used
                Do 1100, I = 1, 3       !copy info from LC to UC entries
                    SW(FLAG,I) = SW(FLAG,I + 3)
                    SW(VALUE,I) = SW(VALUE,I + 3)
```

```
1100                Continue
                End If
                MINUTE = 0                  !assume nothing
                SECOND = 0
                If ((SW(FLAG,2) .eq. 0).and.(SW(FLAG,3) .eq. 0)) Then
                    HOUR = SW(VALUE,1)
                End If
                If ((SW(FLAG,2) .ne. 0).and.(SW(FLAG,3) .eq. 0)) Then
                    MINUTE = SW(VALUE,1)
                    HOUR = SW(VALUE,2)
                End If
                If ((SW(FLAG,2) .ne. 0).and.(SW(FLAG,3) .ne. 0)) Then
                    SECOND = SW(VALUE,1)
                    MINUTE = SW(VALUE,2)
                    HOUR = SW(VALUE,3)
                End If
                If ((HOUR .gt. 23).or.(HOUR .lt. 0)) Go To 2300
                If ((MINUTE .gt. 59).or.(MINUTE .lt. 0)) Go To 2400
                If ((SECOND .gt. 59).or.(SECOND .lt. 0)) Go To 2500
                TIME = (HOUR * (60*60/3) + MINUTE * (60/3) + SECOND /3)
                ERROR = ISFINF (CHAN, DBLK, TIME, 'M', '12'o) !set "time"
                If (ERROR .eq. -2) Go To 2600
                If (ERROR .ne. 0) Go To 2700
                Go To 1000                  !loop for next
2100        Call Print ('?FSFINF-F-CSI error')
            Go To 3100
2200        Call Print ('?FSFINF-F-Fetch error')
            Go To 3100
2300        Call Print ('?FSFINF-W-Hour out of range (0-23)')
            Go To 1000
2400        Call Print ('?FSFINF-W-Minute out of range (0-59)')
            Go To 1000
2500        Call Print ('?FSFINF-W-Second out of range (0-59)')
            Go To 1000
2600        Call Print ('?FSFINF-W-File not found')
            Go To 1000
2700        Call Print ('?FSFINF-F-SFINF error')
3100        Call Exit (FATAL)
            End
```

SFSTA/ISFSTA

SFSTA/ISFSTA saves and then modifies the contents of the directory entry status word (E.STAT) from a file's directory entry. ISFSTA is not supported for the distributed special directory handlers LP, LS, MM, MS, MT, MU, and SP.

Form:

```
CALL SFSTA (chan,dblk,value,oper,[iold])
 i = ISFSTA (chan,dblk,value,oper,[iold])
```

where:

chan is a BYTE or INTEGER*2 RT–11 channel number

dblk is a 4-element INTEGER*2 array containing a 4-word device and file specification in Radix–50; the file specification for which you want to return directory entry information

value is the value to be placed in the directory entry status word.

- If the operation is a BIC or BIS, E.STAT bits 000400, 001000, and 004000, must be clear.
- If the operation is a MOV, only the bottom 4 bits of E.STAT are moved.

For special directory volumes, no bit restrictions are enforced.

oper is the name or octal value indicating the type of operation to be performed:

Name/Value	Type	Meaning
'C'	CLEAR	A bit clear (BIC) operation
'S'	SET	A bit set (BIS) operation
'M'	MOVE	A word move (MOV) operation
128-255	USER	Reserved for the user

iold is the returned INTEGER*2 previous value in the modified directory entry status word

Error:

Value	Meaning
i = 0	Normal return
= -1	Channel was open
= -2	File not found
= -3	Invalid operation argument (internal error)
= -4	Internal error.
= -257	Required argument missing

Example:
See also GFSTA.

```
        Program FSFSTA
C
C       This program modifies the directory status entry
C       setting it to indicate either read-write or read-only.
C       The command is a file name and a switch.
C               /R      read-only
C               /W      read-write
C
        Parameter EREAD = '040000'o !read-only bit
        Integer*2 CHAN          !Channel number to use
        Integer*2 FILSPC(39)    !DBLKs return area
        Integer*2 DBLK(4)       !The interesting DBLK
        Equivalence (FILSPC(16), DBLK(1)) !First input file
        Integer*2 DEFEXT(4)     !Default extensions
        Data DEFEXT /0,0,0,0/   !None
        Integer*2 SWITCH(4, 4)  !Switch parsing table
        Parameter LETTER = 1, FLAG = 2, FILE = 3, VALUE = 4
        Data SWITCH(LETTER,1) /'R'/, SWITCH(LETTER,2) /'r'/
        Data SWITCH(LETTER,3) /'W'/, SWITCH(LETTER,4) /'w'/
        Integer*2 TYPE          !Type of operation flag
        Byte OPER(2)            !Operations to perform
        Data OPER /'S', 'C'/    !set for first / clear for second
        Integer*2 ERROR         !Error/success flag
C
        CHAN = IGETC()          !get a channel
1000        Continue            !main loop
            Call PRINT (0)      !blank line
            ERROR = ICSI (FILSPC, DEFTYP, , SWITCH, 4) !get command
            If (ERROR .ne. 0) Go To 1100 !failed
            ERROR = IFETCH (DBLK(1)) !load handler
            If (ERROR .ne. 0) Go To 1200 !failed
            TYPE = 0            !assume no switch specified
            If ((SWITCH(FLAG,1).ne.0).or.(SWITCH(FLAG,2).ne.0)) TYPE = 1
            If ((SWITCH(FLAG,3).ne.0).or.(SWITCH(FLAG,4).ne.0)) TYPE = 2
            If (TYPE .eq. 0) Go To 1300 !no switch
            ERROR = ISFSTA (CHAN, DBLK, EREAD, OPER(TYPE)) !set/clr bit
            If (ERROR .eq. 0) Go To 1000
        If (ERROR .eq. -2) Go To 1400
        Call PRINT ('?FSFSTA-F-Unexpected error from SFSTA')
        Call EXIT (FATAL)
```

```
1300        Call PRINT ('?FSFSTA-W-Switch required, supply /R or /W')
            Go To 1000
1400        Call PRINT ('?FSFSTA-W-File not found')
            Go To 1000
1100    Call PRINT ('?FSFSTA-F-Unexpected error from CSI')
        Call EXIT (FATAL)
1200    Call PRINT ('?FSFSTA-F-Unexpected error from FETCH')
        Call EXIT (FATAL)
        End
```

SLEEP/ISLEEP

Timer Support

SLEEP/ISLEEP suspends the main program execution of a job for a specified amount of time. The specified time is the sum of hours, minutes, seconds, and ticks specified in the ISLEEP call. All completion routines continue to execute.

Form:

```
CALL SLEEP (hrs,min,sec,tick)
 i = ISLEEP (hrs,min,sec,tick)
```

where:

hrs	is the integer number of hours
min	is the integer number of minutes
sec	is the integer number of seconds
tick	is the integer number of ticks (1/60 of a second on 60-Hz clocks; 1/50 of a second on 50-Hz clocks)

Notes

- When you execute IQSET, remember that SLEEP requires an extra queue element.
- If the system is busy, time execution may be suspended longer than specified.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No queue element available.

Unpredictable results will occur if any arguments are omitted.

Example:
See also TIMER for FTIMEA and FTIMEC.

```
        Program FSLEEP
C
C       demonstrate the SLEEP routine
C
        Integer*2 HRS, MIN, SEC, TIC
        Integer*2 AREAA(4)
        External FTIMEA         !fast timer completion
C
        Call IQSET (10)         !allocate extra queue elements
        Call TIMER (0, 0, 1, 0, AREAA, 12345, FTIMEA)
        CALL ISLEEP (0, 0, 5, 0)!sleep for 5 seconds
        Call PRINT ('!FSLEEP-I-Exiting')
        End
```

SPCPS/ISPCPS

SYSGEN Option

SPCPS/ISPCPS lets a program's completion routine change the flow of control of the mainline code. ISPCPS saves the mainline PC and PS and changes the mainline PC to a new value.

If ISPCPS is issued by a program running under a monitor for which .SPCPS support was not generated, no action is taken and no error is returned.

Form:

```
CALL SPCPS (addr)
 i = ISPCPS (addr)
```

where:

addr is the address of the three-word block in user memory. The first word contains the new mainline PC. The following two words, on return, contain the saved mainline PC and PS

Errors:

Value	Meaning
i= 0	Normal return
= -1	ISPCPS was issued from mainline code rather than from a completion routine
= -2	A previous ISPCPS call is outstanding
= -257	Required argument missing

Example:

```
        Program FSPCPS
C
C       This program demonstrates SPCPS and its ability to change
C       the main-line flow of control in an async completion routine.
C
        Parameter SUCCS = '001'o, FATAL = '010'o
        External FSPCPC
        Integer*2 DBLK(4)       !file name
        Data DBLK /3rSY , 3rSWA, 3rP  , 3rSYS/
        Integer*2 IAREA(4)      !argument area
        Integer*2 BUFF(4096)    !buffer
        Data WCNT /4096/        !transfer word count
        Data BLK /0/            !beginning block
        Integer*2 NEWPC         !address to set PC to
        Integer*2 OLDPC         !previous PC value
        Integer*2 OLDPS         !old PS
        Common /SPCPS/ NEWPC, OLDPC, OLDPS !argument block for SPCPS
C
```

```
        ICHAN = IGETC()         !get a channel
        IERR = LOOKUP (ICHAN, DBLK) !open a known file
        If (IERR .lt. 0) Go To 300 !error?
        Assign 200 to NEWPC     !get PC for SPCPS
C
        IERR = IREADF (WCNT, BUFF, BLK, CHAN, AREA, FSPCPC)
        If (IERR .lt. 0) Go To 400 !error?
C
C       here we hang (wasting CPU cycles)
C
100     Go To 100               !loop, infinite: see infinite loop
C
C       here we derail to from completion routine
C
200     Continue
        Type *, '!FSPCPS-I-Got back via derail'
        Type 101, OLDPC, NEWPC, OLDPS
101     Format (' ', 'OLDPC=', o6, 'NEWPC=', o6, 'OLDPS=', o6)
        Call EXIT (SUCCS)
C
C       error messages
C
300     Continue
        Type *, '?FSPCPS-F-LOOKUP failed', IERR
        Call EXIT (FATAL)
C
400     Continue
        Type *, '?FSPCPS-F-IREADF failed', IERR
        Call EXIT (FATAL)
        End

        SUBROUTINE FSPCPC
        Integer*2 NEWPC         !address to set PC to
        Integer*2 OLDPC         !previous PC value
        Integer*2 OLDPS         !old PS
        Common /SPCPS/ NEWPC, OLDPC, OLDPS !argument block for SPCPS
        Type *, '!FSPCPS-I-In completion routine'
        Call SPCPS (NEWPC)
        Return
        End
```

*SPFN/*SPFNC/*SPFNF/*SPFNW

*SPFN/*SPFNC/*SPFNF/*SPFNW, issued either as functions or subroutines, are used in conjunction with special functions to various handlers having special device-dependent characteristics. Asterisk prefixes indicate generic SPFN forms can begin with the letter *I* when issued as a function or with the letter *M* when the function or subroutine specifies mapping.

You can specify mapping for MSPFN, MSPFNC, and MSPFNW by adding optional arguments *BMODE* and *CMODE*. For details on programming for specific devices, see the *RT–11 Device Handlers Manual*.

All *SPFN forms provide a means of doing device-dependent functions, such as rewind and backspace, to those devices. If ISPFN function calls are made to any other devices, the function call is ignored. To use these functions, the handler must be in memory, and a channel must be associated with a device. These functions require a queue element; this should be a consideration when the IQSET function is executed.

Functions—ISPFN 376, 377

Two ISPFN subroutine calls, 376 and 377, perform a non-file-structured write and read operation, using a 32-bit starting block number. See Table 2–1.

Table 2–1: Device Support (SF.AWR/SF.ARD)

Device	Code	Name	Action
DL/DM/DU	376	SF.AWR	Write operation without doing bad block replacement; returns definitive error data.
	377	SF.ARD	Read operation without doing bad block replacement; returns definitive error data.
DW	376	SF.AWR	Write
	377	SF.ARD	Read
DX/DY/DZ	376	SF.AWR	Write absolute sector
	377	SF.ARD	Read absolute sector

DU Support

The DU handler has support for ISPFN 376 (SF.AWR) and 377 (SF.ARD). For DU, SF.AWR performs a write to the specified sector, and SF.ARD performs a read from the specified sector. Those writes and reads are not absolute; bad-block replacement and block vectoring remain in effect. See *RT–11 Device Handlers Manual* for information.

For DU, SF.AWR and SF.ARD can return the following error code (in addition to those returned by DM):

Code	Explanation
140000	A forced error occurred. • If the device is a disk drive that supports bad block replacement (BBR), the device controller or DU handler discovered bad data on a good (replaced) block. (Bad-block replacement was performed, but no data was recovered.) • If the device does not support BBR, this is an unexpected condition. Refer to *RT–11 Device Handlers Manual* for appropriate action.

DW Support

For DW, ISPFN 376 (SF.AWR) and 377 (SF.ARD) use logical block numbers rather than physical block numbers in the *blk* argument. Therefore, to address a write or read to physical block zero, specify -1 for the *blk* argument. That is necessary because the physical block number of a DW device is one less than the logical block number.

MU Support

MU, ISPFN 374 (SF.MWE, write with extended file gap) executes as ISPFN 371 (SF.MWR, write). The TK50 magtape device does not support SF.MWE functionality; however, future release of RT–11 may add support for this functionality for other MU devices. See Table 2–2.

Table 2–2: Device Support (SF.MWE/SF.MWR)

Device	Code	Name	Action
DM	374	SF.SIZ	Return unit size. Parameter arguments for SF.SIZ for LD and DM are identical.
MU	371	SF.MWR	After initial non-file-structured .LOOKUP and SF.USR, perform write operations of variable word count blocks.
MU	374	SF.MWE	Write with extended gap.

SPFN/ISPFN/MSPFN

SPFN/ISPFN/MSPFN queues the specified operation and immediately returns control to the user program. The IWAIT function can be used to ensure completion of the operation.

Form:

```
CALL SPFN (code,chan[,wcnt,buff,blk])
 i = ISPFN (code,chan[,wcnt,buff,blk])
CALL MSPFN (code,chan[,wcnt,buff,blk][,BMODE=strg])
 i = MSPFN (code,chan[,wcnt,buff,blk][,BMODE=strg])
```

where:

code is the integer numeric code of the function to be performed

chan is the integer specification for the RT–11 channel to be used for the operation. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call

wcnt is the integer number of data words in the operation. This parameter is optional with some ISPFN calls, depending on the particular function. Default value is 0. In magtape operations, it specifies the number of records to space forward or backward. For a backspace operation (*wcnt=0*), the tape drive backspaces to a tape mark or to the beginning-of-tape. For a forward space operation (*wcnt=0*), the tape drive forward spaces to a tape mark or the end-of-tape.

buff is the array to be used as the data buffer. This parameter is optional with some ISPFN calls, depending on the particular function. Default value is 0

blk This parameter is optional with some ISPFN calls, depending on the particular function. Default value is 0.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to zero.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block, execute the following instructions:

```
INTEGER*2       ERRADR, ERRBLK(4)
DATA ERRBLK     /0,0,0,0,/
                .
                .
                .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF
                        !THE 4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

BMODE=strg Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode for the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.
= -19	Invalid BMODE value.

Example:
See SPFNW. See also SDAT.

SPFNC/ISPFNC/MSPFNC

SPFNC/ISPFNC/MSPFNC queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified assembly language routine (*crtn*) is entered as an asynchronous completion routine.

Form:

```
CALL SPFNC (code,chan,wcnt,buff,blk,crtn)
 i = ISPFNC (code,chan,wcnt,buff,blk,crtn)
CALL MSPFNC (code,chan,wcnt,buff,blk,crtn[,BMODE=strg][,CMODE=strg])
 i = MSPFNC (code,chan,wcnt,buff,blk,crtn[,BMODE=strg][,CMODE=strg])
```

where:

code is the integer numeric code of the function to be performed

chan is the integer specification for the RT–11 channel to be used for the operation. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call

wcnt is the integer number of data words in the operation; the default value for this argument is 0

buff is the array to be used as the data buffer; the default value for this argument is 0

blk This parameter is optional with some ISPFN calls, depending on the particular function. Default value is 0.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to zero.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block, execute the following instructions:

```
INTEGER*2       ERRADR, ERRBLK(4)
DATA ERRBLK     /0,0,0,0,/
                .
                .
                .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF
                        !THE 4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

crtn is the name of an assembly language routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNC call

BMODE=strg Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode for the *buff* argument.

CMODE=strg Specify *strg* as string "S" to specify a Supervisor address.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Attempt of read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.
= -19	Invalid BMODE or CMODE value.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See SPFNW. See also SDATC.

SPFNF/ISPFNF

SPFNF/ISPFNF queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified FORTRAN subprogram (*frtn*) is entered as an asynchronous completion routine.

Form:

```
CALL SPFNF (code,chan,wcnt[,buff][,blk],area,crtn)
 i = ISPFNF (code,chan,wcnt[,buff][,blk],area,crtn)
```

where:

code is the integer numeric code of the function to be performed

chan is the integer specification for the RT–11 channel to be used for the operation. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call

wcnt is the integer number of data words in the operation; this argument must be 0 if not required

buff is the array to be used as the data buffer; this argument must be 0 if not required

blk This parameter is optional with some ISPFN calls, depending on the particular function. Default value is 0.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to zero.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block, execute the following instructions:

```
INTEGER*2       ERRADR, ERRBLK(4)
DATA ERRBLK     /0,0,0,0,/
              .
              .
              .
              .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF
                        !THE 4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

area is a four-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program, and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion functions when *frtn* has been activated

frtn is the name of a FORTRAN routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNF call.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        REAL*4 MTNAME(2),AREA(2)
         DATA MTNAME/3RMT0,0./
         EXTERNAL DONSUB
         .
         .
         .
         I=IGETC()                !ALLOCATE CHANNEL
         CALL IFETCH(MTNAME)      !FETCH MT HANDLER
         CALL LOOKUP(I,MTNAME)    !NON-FILE-STRUCTURED LOOKUP ON MT0
         IERR-ISPFNF("373,I,0,0,0,AREA,DONSUB)   !REWIND MAGTAPE
         .
         .
         .
         END
         SUBROUTINE DONSUB
C
C        RUNS WHEN MT0 HAS BEEN REWOUND
C
         .
         .
         .
         END
```

SPFNW/ISPFNW/MSPFNW

SPFNW/ISPFNW/MSPFNW queues the specified operation and returns control to the user program when the operation is complete.

Form:

```
CALL SPFNW (code,chan[,wcnt,buff,blk])
 i = ISPFNW (code,chan[,wcnt,buff,blk])
CALL MSPFNW (code,chan[,wcnt,buff,blk][,BMODE=strg])
 i = MSPFNW (code,chan[,wcnt,buff,blk][,BMODE=strg])
```

where:

code is the integer numeric code of the function to be performed

chan is the integer specification for the RT–11 channel to be used for the operation. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call

wcnt is the integer number of data words in the operation. This parameter is optional with some ISPFNW calls, depending on the function

buff is the array to be used as the data buffer. This parameter is optional with some ISPFNW calls, depending on the function

blk This parameter is optional with some ISPFN calls, depending on the particular function. Default value is 0.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to zero.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block, execute the following instructions:

```
INTEGER*2       ERRADR, ERRBLK(4)
DATA ERRBLK     /0,0,0,0,/
                .
                .
                .
                .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF
                        !THE 4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

BMODE=strg Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. This value specifies the mapping mode for the *buff* argument.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.
= -19	Invalid BMODE argument.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FSPFNW
C
C       Demonstate SPFUN
C
        Parameter VARSZ = '000400'o !variable size bit
        Parameter SFSIZ = '373'o !spfun code for size
        Integer*2 DBLK(4)       !device name (no file)
        Data DBLK /3rSY , 0, 0, 0/ !system device
        Integer*2 REPLY(4+1)    !DSTATUS reply area
        Integer*4 SIZE          !trick for unsigned display
        Equivalence (SIZE, REPLY(4))
C
        ICHAN = IGETC ()        !Get a channel
        Call DSTAT (DBLK, REPLY) !check on the device
        REPLY(5) = 0            !clear high part of I4 Var
        If (IAND (REPLY(1), VARSZ) .eq. VARSZ) Then
            Call LOOKUP (ICHAN, DBLK) !open (CAN'T fail)
            Call SPFNW (SFSIZ, ICHAN, 1, REPLY(4), 0)
            Type *, 'Volume size (Variable) = ', SIZE
            Else
            Type *, 'Volume size (Constant) = ', SIZE
            End If
        End
```

STRPAD

The STRPAD routine pads a character string with right-most blanks until that string is a specified length. This padding is done in place; the result string is contained in its original array. If the present length of the string is greater than or equal to the specified length, no padding occurs.

Form:

CALL STRPAD (a,len[,err])

where:

a	is the array containing the string to be padded. This array must be one element longer than the value of *len* if *len* is specified. It must be terminated by a null byte
len	is the integer length of the desired result string
err	is the logical error flag that is set to .TRUE. if the string specified by a exceeds the value of *len* in length

Errors:

Error conditions are indicated by *err*, if specified. If *err* is given and the string indicated is longer than *len* characters, *err* is set to .TRUE.; otherwise, the value of *err* is unchanged.

NOTE

The argument *err* must be specified as LOGICAL*1 in FORTRAN 77. It can be any logical type in FORTRAN IV and any integer type in PDP–11C.

Unpredictable results will occur if any argument is omitted.

Example:

```
       Program FSTRPA
C
C       This program demonstrates the STRPAD function
C
        Byte BUFFER(61)
        Byte BUF1(81)
C
        Do 100 I = 1, 60
            BUFFER(I) = '?'
100         Continue
        BUFFER(61) = '000'o
        Call SCOPY ('test', BUFFER)
        Type 101, (BUFFER(I), I=1,60)
101     Format (' ', '(', 60a1, ')')
C
        Call STRPAD (BUFFER, 60)
        Type 101, (BUFFER(I), I=1,60)
        End
```

SUBSTR

The SUBSTR routine copies a substring from a specified position in a character string. If desired, the substring can then be placed in the same array as the string from which it was taken.

Form:

CALL SUBSTR (in,out,i[,len])

where:

in is the array from which the substring is taken; it is terminated by a null byte

out is the array to contain the substring result. This array must be one element longer than *len*, if *len* is specified. It also is terminated by a null byte if *len* is specified

i is the integer character position in the input string of the first character of the desired substring

len is the integer number of characters representing the maximum length of the substring

Errors:
Unpredictable results will occur if any argument is omitted.

SUSPND

The SUSPND subroutine suspends main program execution of the current job and allows only completion routines for I/O and scheduling requests to run.

Form:

```
CALL SUSPND
```

The monitor maintains a suspension counter for each job. This count is decremented by SUSPND and incremented by RESUME. A job will actually be suspended only if this counter is negative. When RESUME is issued before a SUSPND, the latter routine will return immediately.

A program must issue an equal number of SUSPND and RESUME calls.

A SUSPND subroutine call from a completion routine decrements the suspension counter but does not suspend the main program. If a completion routine does a SUSPND, the main program continues until it also issues a SUSPND, at which time it is suspended. Two RESUME calls are then required to proceed.

Because SUSPND and RESUME are used to simulate an ITWAIT in the monitor, a RESUME issued from a completion routine and not matched by a previously executed SUSPND can cause the main program execution to continue past a timed wait before the entire time interval has elapsed.

For further information on suspending main program execution of the current job, see the .SPND programmed request.

Errors:
None.

Example:
See MRKT.

$SYTRP

Trap Handler

$SYTRP processes SYSLIB-generated TRAP instructions for programs not linked with FORTRAN libraries. $SYTRP should not be called by any program that will be linked with FORTRAN libraries F77OTS.OBJ or FORLIB.OBJ.

You include the $SYTRP trap handler module by writing the following line in the program:

Form:

.GLOBL $SYTRP

When some SYSLIB routines detect an error condition, they execute a TRAP instruction, using a unique error number symbol, $MSARG. $SYTRP evaluates the $MSARG value, prints an error message, and terminates the execution. If the error condition, and therefore the TRAP instruction, is caused by a routine being called with an invalid argument list (too few parameters), $SYTRP prints the following error message:

?SYSLIB-F-Invalid argument

Any other TRAP instruction produces the following error message:

?SYSLIB-F-Unknown error

If no trap handler module is called, execution of a TRAP instruction causes the program to terminate.

The following is the source listing for $SYTRP:

```
.MCALL .MODULE
.MODULE SYTRP,VERSION=04,COMMENT=^\SYSLIB/$SYTRP\,IDENT=NO,LIB=YES

        .NLIST  BEX

        .SBTTL  Definitions:
        .SBTTL  . $MSARG Definition

        .WEAK   $MSARG              ; Invisible definition
        .GLOBL  $SYSLB            ; Include system library work area.

$MSARG  ==:       128.+1.                       ; Global definition of $MSARG

        .SBTTL  . Macro references

        .MCALL  .PRINT, .EXIT

        .LIBRARY "SRC:SYSTEM.MLB"
        .MCALL  .SAVDF    .SYCDF      .UEBDF
        .SYCDF
        .UEBDF
        .SAVDF  E==:

        .SBTTL  . TRAP Vector Contents
```

```
        .ASECT
        . = SV.NID                  ;any .SAV file linked
        .WORD   000001               ;to CMPLT.
        .=34                ; TRAP vector
        .WORD   $SYTRP               ; PC after TRAP
        .WORD   030000               ; PS = USER-USER, priority 0

        .SBTTL  . Error Message Text

        .PSECT  SYS$S,D
MSG1:   .ASCIZ     /?SYSLIB-F-Invalid argument/
MSG2:   .ASCIZ     /?SYSLIB-F-Unknown error/
        .EVEN
        .SBTTL  $SYTRP - TRAP Handler Code

        .PSECT  SYS$I,I

$SYTRP::                                ; Trap handler entry point
        MOV     @SP,R0                     ; get old PC
        CMP     -2(R0),#TRAP+$MSARG  ; was it OUR trap?
        BNE     10$
        MOV     #MSG1,R0          ; print error message,
        BR      20$

10$:    MOV #MSG2,R0  ; print error message,
20$:    .PRINT
        CMP     (SP)+,(SP)+    ; eat PS and PC caused by TRAP
..WARN::BISB    #ERROR$,@#$USRRB    ; announce error condition
        .EXIT            ; exit program.

        .END
```

TIMASC

The TIMASC subroutine converts a two-word internal format time into an ASCII string, **hh:mm:ss**.

where:

hh	is the two-digit hours indication
mm	is the two-digit minutes indication
ss	is the two-digit seconds indication

Form:

```
CALL TIMASC (itime,strng)
```

where:

itime	is the two-word internal format time to be converted, where • *itime(1)* is the high-order time • *itime(2)* is the low-order time
strng	is the eight-element array to contain the ASCII time

Errors:
Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See JDIV.

TIME

The TIME subroutine returns the current system time of day as an eight-character ASCII string, **hh:mm:ss**

where:

hh	is the two-digit hours indication
mm	is the two-digit minutes indication
ss	is the two-digit seconds indication

Form:

```
CALL TIME (strng)
```

where:

strng	is the eight-element array to receive the ASCII time

NOTE

A 24-hour clock is used (for example, 1:00 p.m. is represented at 13:00:00).

Errors:
Unpredictable results will occur if any argument is omitted.

Example:
See TWAIT.

TIMER/ITIMER

SYSGEN Option for SB

TIMER/ITIMER schedules a specified FORTRAN subroutine to be run as an asynchronous completion routine after a specified time interval has elapsed. For SB monitor, you must select timer support during SYSGEN.

Form:

```
CALL TIMER (hrs,min,sec,tick,area,id,scrtn)
 i = ITIMER (hrs,min,sec,tick,area,id,scrtn)
```

where:

hrs is the integer number of hours

min is the integer number of minutes

sec is the integer number of seconds

tick is the integer number of ticks (1/60 of a second on 60-Hz clocks; 1/50 of a second on 50-Hz clocks)

area is a four-word area that must be provided for link information; this area must never be modified by the FORTRAN program, and the USR must never swap over it. This area can be reclaimed by other FORTRAN completion functions when *scrtn* has been activated

id is the identification integer to be passed to the routine being scheduled

scrtn is the name of the FORTRAN subroutine to be entered when the specified time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that references ITIMER. The subroutine has one argument. For example:

```
SUBROUTINE scrtn(id)
INTEGER id
```

When the routine is entered, the value of the integer argument is the value specified for *id* in the appropriate ITIMER call.

Notes

- This function can be canceled at a later time by an ICMKT function call.
- If the system is busy, the actual time interval after which the completion routine is run can be longer than the time interval requested.
- FORTRAN subroutines can periodically reschedule themselves by issuing ISCHED or ITIMER calls.
- ITIMER requires a queue element, which should be considered when the IQSET function is executed.

For more information on scheduling completion routines, see program routines and .MRKT programmed request.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No queue elements available; unable to schedule request.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
       Program FTIMER
C
C       demonstrate the FORTRAN timer completion routine
C
        Integer*2 HRS, MIN, SEC, TIC
        Integer*2 AREAA(4), AREAB(4)
        External FTIMEA         !fast timer completion
        External FTIMEB         !slow timer completion
C
        Call IQSET (10)         !allocate extra queue elements
        Call TIMER (0, 0, 1, 0, AREAA, 12345, FTIMEA)
        Call TIMER (0, 0, 5, 0, AREAB, 23456, FTIMEB)
        Call SUSPND
        Call PRINT ('!FTIMER-I-Exiting')
        End

        Subroutine FTIMEA (ID)
C
C       demonstrate the FORTRAN timer completion routine
C
        Integer*2 HRS, MIN, SEC, TIC
        Integer*2 AREAA(4)
        External FTIMEC         !fast timer completion
C
        Call TIMER (0, 0, 1, 0, AREAA, 12345, FTIMEC)
        Call PRINT ('!FTIMEA-I-Entered')
        End

        Subroutine FTIMEB (ID)
C
C       demonstrate the FORTRAN timer completion routine
C
        Call RESUME
        Call PRINT ('!FTIMEB-I-Entered')
        End

.TITLE  FTIMEC - bypass recursion test in F77
        .GLOBL  FTIMEA
FTIMEC::JMP     FTIMEA  ;let FTIMEA refer to itself
        .END
```

TRANSL

The TRANSL routine performs character translation on a specified string and requires approximately 64(decimal) words on the R6 stack for its execution. This space should be considered when allocating stack space.

Form:

CALL TRANSL (in,out,r[,p])

where:

in is the array containing the input string; it is terminated by a null byte

out is the array to receive the translated string; it is not terminated by a null byte

r is the array containing the replacement string; it is terminated by a null byte

p is the array of characters in *in* to be translated; it is terminated by a null byte

The string specified by array *out* is replaced by the string specified by array *in*, modified by the character translation process specified by arrays *r* and *p*. If any character position in *in* contains a character that appears in the string specified by *p*, it is replaced in *out* by the corresponding character from string *r*. If the array *p* is omitted, it is assumed to be the 127 seven-bit ASCII characters arranged in ascending order, beginning with the character whose ASCII code is 001. If strings *r* and *p* are given and differ in length, the longer string is truncated to the length of the shorter. If a character appears more than once in string *p*, only the last occurrence is significant. A character can appear any number of times in string *r*.

Errors:
Unpredictable results will occur if any argument is missing.

Examples:

```
       Program FTRAN1
C
C       demonstrate translating characters
C       The following uppercases all the letters
C       and changes all numbers to #.
C
        Byte IN(81), OUT(81)    !string arguments
        Byte FROM(81), TO(81)   !translation table
C
        Call SCOPY ('qwertyuiopasdfghjklzxcvbnm', FROM)
        Call SCOPY ('QWERTYUIOPASDFGHJKLZXCVBNM', TO)
        Call CONCAT (FROM, '0123456789', FROM)
        Call CONCAT (TO  , '##########', TO)
C
        Call SCOPY ('abcABC123', IN)
```

```
        Call TRANSL (IN, OUT, TO, FROM)
        Call PRINT (IN)
        Call PRINT (OUT)
        End
```

The following is an example of TRANSL being used to format character data.

```
        Program FTRAN2
C
C       This shows a way to shuffle a string using TRANSL
C
        Byte STRING(27), RESULT(27), PATTER(27)
C
C       000000000111111111112222222
C       123456789012345678901234567
C       The horn blows at midnight
C
        Data PATTER
        1 /16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 15,
        2  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0/
C
        Call SCOPY ('The Horn Blows at Midnight', STRING)
C
C       The following call to TRANSL rearranges the chars of
C       the input string  to the order specified by the pattern.
C
        Call TRANSL (PATTER, RESULT, STRING)
C
C       RESULT now contains the string 'at Midnight the Horn Blows'.
C       In general, this method can be used to format strings of up
C       to 127 chars.  The resultant string will be as long as the
C       pattern string.
C
        Call PRINT (STRING)
        Call PRINT (RESULT)
        End
```

TRIM

The TRIM routine shortens a specified character string by removing all trailing blanks. A trailing blank is a blank that has no non-blanks to its right. If the specified string contains all blank characters, it is replaced by the null string. If the specified string has no trailing blanks, it is unchanged.

Form:

CALL TRIM (a)

where:

a is the array containing the string to be trimmed; it is terminated by a null byte on input and output

Errors:
Unpredictable results will occur if the argument is omitted.

Example:

```
       Program FTRIM
C
C       This demonstrates the TRIM function
C
        External LEN            !use the RT-11 LEN function
        Byte INPUT(81)          !input buffer
C
        Accept 100, (INPUT(I), I=1,80)
100     Format (80a1)
        Call SCOPY (INPUT, INPUT, 80)   !punch in a null
        Type *, LEN (INPUT)             !length before trimming
        Call TRIM (INPUT)               !trim trailing blanks
        Type *, LEN (INPUT)             !length after trimming
        End
```

TWAIT/ITWAIT

SYSGEN Option for SB

TWAIT/ITWAIT suspends the main program execution of the current job for a specified time interval. All completion routines continue to execute. For SB monitor, you must select timer support during SYSGEN.

Form:

```
CALL TWAIT (itime)
i = ITWAIT (itime)
```

where:

itime	is the two-word internal format time interval
itime(1)	is the high-order time
itime(2)	is the low-order time

Notes

TWAIT requires a queue element, which should be considered when the IQSET function is executed.

If the system is busy, the actual time interval during which execution is suspended may be longer than the time interval specified.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No queue element available.

Unpredictable results will occur if any argument is omitted.

Example:

```
        Program FTWAIT
C
C       Wait 10 seconds
C
        Integer*4 TIMEWD        !time variable
        Byte ASCTIM(9)
C
        Call JTIME (0, 0, 10, 0, TIMEWD) !set for 10 sec
        Call TIME (ASCTIM)      !get current time
        Call PRINT (ASCTIM)     !display it
        Call TWAIT (TIMEWD)     !wait 10 seconds
        Call TIME (ASCTIM)      !get current time
        Call PRINT (ASCTIM)     !display it
        End
```

UNMAP/IUNMAP

Mapping

UNMAP/IUNMAP is used to eliminate the mapping window. See MAP subroutine. See also .WDBDF in the *RT–11 System Macro Library Manual*.

Form:

```
CALL UNMAP (iwdb [,ierr])
 ierr = IUNMAP (iwdb)
```

where:

ierr Error return

iwdb Address of Window Descriptor Block

Errors:

Value	Meaning
ierr = 0	Function completed successfully.
= -4	Invalid window identifier.
= -6	Specified window was not already mapped.
= -16	Mode/space not available.
= -257	Required argument missing.

Example:
See CRAW.

UNPRO/IUNPRO

UNPRO/IUNPRO cancels any protection for the specified 2-word vector in the 0 to 474 area. UNPRO is the complement of PROTE. If the specified two-word vector is currently not protected, UNPRO is ignored.

Form:

```
CALL UNPRO (addr)
i = IUNPRO (addr)
```

where:

addr is the address of the two-word vector pair for which protection is to be canceled.

Errors:

Value	Meaning
i = 0	Normal return.
= -2	Addr is greater than 474 or not a multiple of 4.
= -257	Required argument *addr* missing.

Example:
See PROTE/IPROTE.

UNTIL/IUNTIL

The UNTIL/IUNTIL suspends main program execution of the job until the time-of-day specified. All completion routines continue to run. For SB monitor, you must select timer support during SYSGEN.

Form:

```
CALL UNTIL (hrs,min,sec,tick)
 i = IUNTIL (hrs,min,sec,tick)
```

where:

hrs	is the integer number of hours
min	is the integer number of minutes
sec	is the integer number of seconds
tick	is the integer number of ticks (1/60 of a second on 60-Hz clocks; 1/50 of a second on 50-Hz clocks)

NOTES

- IUNTIL requires a queue element, which should be considered when the IQSET function is executed.
- If the system is busy, the actual time of day that the program resumes execution may be later than that requested.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	No queue element available.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
        Program FUNTIL
C
C        This program wait until the next even minute (sec=00)
C
         Integer*4 NOW
         Integer*2 HOURS, MINUTE, SECOND, TICKS
         Byte ASCTIM(9)
         Data ASCTIM(9) /0/
C
         Call GTIM (NOW)          !get current time
         Call CVTTIM (NOW, HOURS, MINUTE, SECOND, TICKS) !dissect it
         MINUTE = MINUTE + 1
         If (MINUTE .eq. 60) Then
             MINUTE = 0
```

```
    HOUR = HOUR + 1
    If (HOUR .eq. 24) HOUR = 0
    End If
SECOND = 0
TICKS = 0
Call UNTIL (HOURS, MINUTES, SECOND, TICKS)
Call TIME (ASCTIM)
Call PRINT (ASCTIM)
End
```

VERIFY/IVERIFY

VERIFY/IVERIFY checks that a given string is composed entirely of characters from a second string. If a character does not exist in the string being examined, VERIFY returns the position of the first character in the string being examined that is not in the source string. If all characters exist, VERIFY returns a 0.

Form:

```
CALL VERIFY (a,b,i)
 i = IVERIF (a,b)
```

where:

a is the array containing the string to be scanned; it is terminated by a null byte

b is the array containing the string of characters to be accepted in a; it is terminated by a null byte

Function Result:

Value	Meaning
i = 0	If all characters of *a* exist in *b*; also if *a* is a null string.
i = n	Where *n* is the character position of the first character in array *a* that does not appear in array *b*; if *b* is a null string and *a* is not, *i* equals 1.

Errors: Error message *TRAP $MSARG* will display if any argument is missing.

Example:

```
       Program FVERIF
C
C       Verify that the entered string contains only "hex"
C       characters (0-9, a-f, A-F).
C
        Character*81 INPUT      !input buffer
        Byte VALID(23)          !valid chars
        Data VALID
        1 /'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
        2  'a', 'b', 'c', 'd', 'e', 'f',
        3  'A', 'B', 'C', 'D', 'E', 'F', '000'o/
C
100     Continue
            Call GTLIN (INPUT)  !get a command line
            IERR = IVERIF (INPUT, VALID) !check it
            If (IERR .eq. 0) Then
                Type *, '!FVERIF-I-Valid input'
                Else
                Type *, '?FVERIF-W-Invalid input - ', INPUT(IERR:IERR)
                End If
            Go To 100
        End
```

WAIT/IWAIT

WAIT/IWAIT suspends execution of the main program until all input/output operations on the specified channel are complete. This function is used with *READ, *WRITE, and *SPFN calls. Completion routines continue to execute.

Form:

```
CALL WAIT (chan)
 i = IWAIT (chan)
```

where:

chan is the integer specification for the RT–11 channel to be used. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call

For further information on suspending execution of the main program, see the .WAIT programmed request in the *RT–11 System Macro Library Manual*.

Errors:

Value	Meaning
i = 0	Normal return.
= 1	Channel specified is not open.
= 2	Hardware error occurred during the previous I/O operation on this channel.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See READ.

*WRITE/*WRITC/*WRITF/*WRITW

*WRITE/*WRITC/*WRITF/*WRITW, issued either as function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. The *WRITE functions require queue elements; this should be considered when the IQSET function is executed.

Specify mapping for MWRITE, MWRITC and MWRITW by adding optional parameters *BMODE* and *CMODE*.

WRITE/IWRITE/MWRITE

WRITE/IWRITE/MWRITE transfers a specified number of words from memory to the specified channel. Control returns to the user program immediately after the request is queued. No special action is taken upon completion of the operation.

Form:

```
CALL WRITE (wcnt,buff,blk,chan)
i = IWRITE (wcnt,buff,blk,chan)
CALL MWRITE (wcnt,buff,blk,chan[,BMODE=strg])
i = MWRITE (wcnt,buff,blk,chan[,BMODE=strg])
```

where:

wcnt	is the integer number of words to be transferred
buff	is the array to be used as the output buffer
blk	is the integer block number of the file to be written. The user program normally updates *blk* before it is used again.
chan	is the integer specification for the RT–11 channel to be used. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value specifies the mapping mode for the *buff* argument.

Errors:

Value	Meaning
i = n	Normal return; n equals the number of words written, rounded to a multiple of 256 (0 for non-file-structured writes). If the word count returned is less than that requested, an implied end-of-file has occurred, although the normal return is indicated.
= -1	Attempt to write past end-of-file; no more space is available in the file.
= -2	Hardware error occurred.
= -3	Channel specified is not open.
=-19	Invalid BMODE or CMODE value.

Error message *TRAP $MSARG* will display if any argument is missing.

Example:
See ABTIO.

WRITC/IWRITC/MWRITC

WRITC/IWRITC/MWRITC issued either as function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. The request is queued and control returns to the user program. When the transfer is complete, the specified assembly language routine (*crtn*) is entered as an asynchronous completion routine.

Form:

```
CALL WRITC (wcnt,buff,blk,chan,crtn)
 i = IWRITC (wcnt,buff,blk,chan,crtn)
CALL MWRITC (wcnt,buff,blk,chan,crtn[,BMODE=strg][,CMODE=strg])
 i = MWRITC (wcnt,buff,blk,chan,crtn[,BMODE=strg][,CMODE=strg])
```

where:

wcnt	is the relative integer number of words to be transferred
buff	is the array to be used as the output buffer
blk	is the relative integer block number of the file to be written. The user program normally updates *blk* before it is used again (for example, if the program is writing two blocks at a time, *blk* should be updated by 2)
chan	is the relative integer specification for the RT–11 channel to be used. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call
crtn	is the name of the assembly language routine to be activated upon completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IWRITC call
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value specifies the mapping mode for the *buff* argument.
CMODE=strg	Specifying *strg* as the string "S" specifies a Supervisor address.

Errors:
Same errors as *WRITE.

Example:
See READC.

WRITF/IWRITF

WRITF/IWRITF issued either as function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. The transfer request is queued and control returns to the user program. When the operation is complete, the specified FORTRAN subprogram (*frtn*) is entered as an asynchronous completion routine.

Form:

```
CALL WRITF (wcnt,buff,blk,chan,area,frtn)
 i = IWRITF (wcnt,buff,blk,chan,area,frtn)
```

where:

wcnt	is the integer number of words to be transferred
buff	is the array to be used as the output buffer
blk	is the integer block number of the file to be written. The user program normally updates *blk* before it is used again
chan	is the integer specification for the RT–11 channel to be used. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call
area	is a four-word area to be set aside for link information; this area must not be modified by the FORTRAN program, and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion functions when *frtn* has been activated
frtn	is the name of the FORTRAN routine to be activated upon completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IWRITF call.

Errors:
See the errors under *WRITE.

Example:
See IREADF.

WRITW/IWRITW/MWRITW

WRITW/IWRITW/MWRITW issued either as function or subroutine, transfers a specified number of words from memory to the device or file specified by channel. Control returns to the user program when the transfer is complete.

Form:

```
CALL WRITW (wcnt,buff,blk,chan)
 i = IWRITW (wcnt,buff,blk,chan)
CALL MWRITW (wcnt,buff,blk,chan[,BMODE=strg])
 i = MWRITW (wcnt,buff,blk,chan[,BMODE=strg])
```

where:

wcnt	is the integer number of words to be transferred
buff	is the array to be used as the output buffer
blk	is the integer block number of the file to be written. The user program normally updates *blk* before it is used again
chan	is the integer specification for the RT–11 channel to be used. You must obtain this channel through an IGETC call, or you can use channel 16(decimal) or higher if you have done an ICDFN call
BMODE=strg	Specify *strg* with one of the following: 'UI'/'UD'/'SI'/'SD'/'CD'/'CI'. Value specifies the mapping mode for the *buff* argument.

Errors:
See the errors under *WRITE.

Example:
See REOPN.

Index

A

ABTIO/IABTIO
  description, 2–2
  example, 2–2
Accessing addresses
  under mapped monitors, 1–14
  using KPEEK, KPOKE, 1–14
Additional queue elements
  allocating, 1–14
  subroutines requiring, 1–14
AJFLT/IAJFLT
  description, 2–3
  example, 2–3

C

Calculating workspace
  formula, 1–30
CALL$F
  description, 2–5
  example, 2–5
Calling SYSLIB
  subroutines or functions, 1–15
Calling SYSLIB routines
  FORTRAN subroutine linkage, 1–15
  PDP–11 C support, 1–15
Calling the command string interpreter, 1–5
CHAIN
  description, 2–6
  example, 2–6
Character string
  allocating variables, 1–37
  ASCII code, 1–37
  err argument, 1–37
  len argument, 1–37
  length, 1–37
  multidimensional arrays, 1–39
  passing to subprograms, 1–38
  quoted strings, 1–39
  restrictions, 1–37
  storage, 1–36
  unknown length, 1–39
Character string functions
  description, 1–36
  table, 1–36
CHCPY/ICHCPY
  description, 2–8
  example, 2–8
CLOSEC/ICLOSE
  description, 2–10
  example, 2–10
CLOSZ/ICLOSZ
  description, 2–12
  example, 2–12
CMAP/ICMAP
  description, 2–14
  example, 2–14
  mapping control, 2–14
CMKT/ICMKT
  description, 2–16
  example, 2–16
CNTXS
  description, 2–17
  example, 2–17
Completion routine
  certain restrictions, 1–4
Completion routines
  error handling, 1–4
  written in FORTRAN, 1–4
CONCAT
  description, 2–18
  example, 2–18
Conventions
  return values, 1–24
CRAW/ICRAW
  description, 2–20
  example, 2–20
CRRG/ICRRG
  description, 2–24
  example, 2–24
CSI/ICSI
  description, 2–25
  example, 2–25
CSTAT/ICSTAT

CSTAT/ICSTAT (Cont.)
description, 2–29
example, 2–29
CVTTIM
description, 2–31
example, 2–31

D

DATE/DATE4Y
description, 2–32
example, 2–32
DELET/IDELET
description, 2–34
example, 2–34
Device blocks
setting up in FORTRAN, 1–5
DEVICE/IDEVICE
description, 2–36
example, 2–36
DJFLT
See also IDJFLT
description, 2–37
example, 2–37
DSTAT/IDSTAT
description, 2–38
example, 2–38

E

ELAW/IELAW
description, 2–40
example, 2–40
ELRG/IELRG
description, 2–41
example, 2–41
ENTER/IENTER
See also CLOSEC, CLOSZ
description, 2–42
example, 2–42

F

FORTRAN
workspace for FB program, 1–30, 1–31
FORTRAN/MACRO interface
description, 1–23
FORTRAN OTS
interfacing user-written routines, 1–6
FORTRAN Programs
in FB environment, 1–29
FPROT/IFPROT
description, 2–44
example, 2–44
FREER/IFREER
description, 2–45
example, 2–45
Functions
invoking, 1–23

G

GCLOS
example, 2–57
GCMAP/IGCMAP
description, 2–46
example, 2–46
GETR/IGETR
description, 2–48
example, 2–48
GFDAT/IGFDAT
description, 2–52
example, 2–52
GFINF/IGFINF
description, 2–53
example, 2–53
GFSTA/IGFSTA
description, 2–55
example, 2–55
GICLOS
description, 2–57
GIDIS
error codes, 2–59
example, 2–59
GIOPEN
description, 2–57
example, 2–57
GIREAD
description, 2–58
example, 2–58
GIWRIT
description, 2–58
example, 2–58
Global regions
See IGETR/MGETR, IFREER
attaching to, 1–36
control of, 1–33
detaching from, 1–36
GMCX/IGMCX
See also CRAW
description, 2–62

GTDIR/IGTDIR
description, 2–63
example, 2–63
GTDUS/IGTDUS
description, 2–69
example, 2–69
GTIM
See also CVTTM
description, 2–74
GTJB/IGTJB
description, 2–75
example, 2–75
GTLIN/IGTLIN
description, 2–77
example, 2–77

H

HERR/IHERR
description, 2–79
example, 2–79

I

IADDR
description, 2–81
IDATE
See also DATE
description, 2–82
IDCOMP
description, 2–84
example, 2–84
IFWILD
description, 2–86
example, 2–86
IGTENT
See also GTDIR/IGTDIR
description, 2–89
IJCVT
description, 2–91
example, 2–91
INDEX
description, 2–92
example, 2–92
INSERT
description, 2–93
example, 2–93
INTEGER*4
arithmetic operation, 1–34
two-word support, 1–34
INTEGER*4 support functions
how to initialize, 1–5
IPEEK
description, 2–94
example, 2–94
IPEEKB
description, 2–96
example, 2–96
IPROTE
example, 2–164
IRAD50
description, 2–97
example, 2–97
IRCVD/MRCVD
See also SFDAT
description, 2–172
example, 2–172
ISPY
description, 2–98
example, 2–98
ISWILD
description, 2–99
example, 2–99
ITLOCK
description, 2–101
example, 2–101
ITTINR
description, 2–102
example, 2–102
ITTOUR
description, 2–104
IWEEKD
See also DATE
description, 2–105

J

JADD
description, 2–106
example, 2–106
JAFIX
description, 2–107
example, 2–107
JCMP
description, 2–108
example, 2–108
JDFIX
description, 2–110
example, 2–110
JDIV

JDIV (Cont.)
description, 2–111
example, 2–111
JICVT
description, 2–113
example, 2–113
JJCVT
See also JDIV
description, 2–114
JMOV
See also JCMP
description, 2–115
JMUL
description, 2–116
example, 2–116
JREAD
See also RCVD
description, 2–118
JREADC
See also RCVDF
description, 2–119
JREADF
See also RCVDF
description, 2–121
JREADW
See also RCVDC
description, 2–122
example, 2–122
JSUB
description, 2–125
example, 2–125
JTIME
See also JMUL
description, 2–127
JWRITC
See also SDATC
description, 2–131
JWRITE
See also SDAT
description, 2–129
JWRITF
See also SDATF
description, 2–133
JWRITW
See also SDATW
description, 2–135

K

KPEEK
KPEEK (Cont.)
See also KPOKE
description, 2–137
example, 2–137
KPOKE
See KPEEK
description, 2–138

L

LEN
See also SDAT
description, 2–140
LOCK
description, 2–141
example, 2–141
LOOKUP
See also CHCPY
description, 2–144
example, 2–144

M

MACRO subroutines
See also DOFOR
See FINITA
called by FORTRAN programs, 1–25
MAP
See CRAW
description, 2–147
MRKT
description, 2–148
example, 2–148
MSDS
See also CMAP
description, 2–150
MTATCH
description, 2–151
MTDTCH
description, 2–152
MTGET
description, 2–153
MTIN
description, 2–154
MTOUT
description, 2–155
MTPRNT
description, 2–156
MTRCTO
description, 2–157
MTSET

MTSET (Cont.)
description, 2–158
MTSTAT
description, 2–159
MWAIT
See also SDAT
description, 2–160

O

Operations>
Character string, 1–35

P

POKE
See also PEEK
POKEB/IPOKEB
See also PEEK
description, 2–162
POKE/IPOKE
description, 2–161
PRINT
description, 2–163
example, 2–163
Program suspension
ITWAIT, ISLEEP, IUNTIL, 1–33, 1–34
PROTE
description, 2–164
PSECT ordering
avoiding USR swapping, 1–8
for PDP–11 C, 1–11
for RTL programs, 1–11
PSET
allocating, 1–8
PURGE
See also ENTER/IENTER
description, 2–165
PUT/IPUT
description, 2–166
example, 2–166

R

R50ASC
See also CSI
description, 2–167
RAD50
description, 2–168
example, 2–168
RAN/RANDU
RAN/RANDU (Cont.)
See also CHAIN, RCHAIN
description, 2–169
example, 2–169
RCHAIN
See also CHAIN
description, 2–170
example, 2–170
RCTRLO
See also IPEEK
description, 2–171
RCVDC/IRCVDC/MRCVDC
description, 2–173
example, 2–173
RCVDF/IRCVDF
description, 2–175
example, 2–175
RCVDW/IRCVDW/MRCVDW
description, 2–177
example, 2–177
READC/IREADC/MREADC
description, 2–182
example, 2–182
READF/IREADF
description, 2–185
example, 2–185
READ/IREAD/MREAD
description, 2–179
example, 2–179
READW/IREADW/MREADW
description, 2–189
example, 2–189
Register
See also CALL$F
save and restore, 1–24
RENAM/IRENAM
description, 2–191
example, 2–191
REOPN/IREOPN
description, 2–193
example, 2–193
REPEAT
description, 2–195
example, 2–195
RESUME
description, 2–197
example, 2–197
Routines
invoking, 1–23

S

SAVES/ISAVES
  See also REOPN
  description, 2–198
SCCA/ISCCA
  description, 2–199
  example, 2–199
SCHED/ISCHED
  description, 2–201
  example, 2–201
SCOMP/ISCOMP
  See also SDTTM
  description, 2–203
SCOPY
  description, 2–204
  example, 2–204
SDATC/ISDATC/MSDATC
  description, 2–207
  example, 2–207
SDATF/ISDATF
  description, 2–209
  example, 2–209
SDAT/ISDAT/MSDAT
  description, 2–205
  example, 2–205
SDATW/ISDATW/MSDATW
  description, 2–211
  example, 2–211
SDTTM/ISDTTM
  description, 2–214
  example, 2–214
SERR/ISERR
  See also HERR
  description, 2–216
SETCMD
  description, 2–218
  example, 2–218
SFDAT/ISFDAT
  description, 2–219
  example, 2–219
SFINF/ISFINF
  description, 2–220
  example, 2–220
SFSTA/ISFSTA
  description, 2–223
  example, 2–223
SLEEP/ISLEEP
  See also TIMER
  description, 2–226
SPCPS/ISCPS
  description, 2–227
  example, 2–227
SPFN
  Added support, 2–229
    DU support, 2–229
    DW support, 2–229
    MU support, 2–229
SPFNC/ISPFNC/MSPFNC
  description, 2–233
SPFNF/ISPFNF
  See also SDATF
  description, 2–235
SPFN/ISPFN/MSPFN
  description, 2–231
SPFNW/ISPFNW/MSPFNW
  description, 2–237
  example, 2–237
STRPAD
  description, 2–239
  example, 2–239
Subroutine
  register usage, 1–24
Subroutines
  added queue elements, 1–14
  invoking, 1–23
SUBSTR
  description, 2–240
  example, 2–240
SUSPND
  See also MRKT
  description, 2–241
SYSLIB
  applicability of routines, 1–2
  conversion calls, 1–34
  FORTRAN naming conventions, 1–3
  functional organization, 1–2
  list of functions, 1–16
  list of subroutines, 1–16
  services not provided, 1–33
  system conventions, 1–2
SYSLIB conversion calls
  table, 1–34
System conventions
  allocating channels, 1–4
  channel numbers, 1–3
  completion routines, 1–4
  functions, 1–3
  subroutines, 1–3
$SYTRP

$SYTRP (Cont.)
description, 2–242
example, 2–242

T

TIMASC
See also JDIV
description, 2–244
TIME
See also TWAIT
description, 2–245
TIMER/ITIMER
description, 2–246
example, 2–246
TRANSL
description, 2–248
example, 2–248
TRIM
description, 2–250
example, 2–250
TWAIT/ITWAIT
description, 2–251
example, 2–251

U

UNLOCK
description, 2–143
example, 2–143
UNMAP/IUNMAP
See also CRAW
description, 2–252
UNPRO/IUNPRO
See also PROTE/IPROTE
description, 2–253
UNTIL/IUNTIL
description, 2–254
example, 2–254
USR
See also LOCK, UNLOCK, ITLOCK
allowing swapping, 1–7
automatic swapping, 1–8
controlling swapping, 1–7
Keeping USR resident, 1–7
preventing swapping, 1–7
PSECT order table, 1–8
requests for functions, 1–6
restrictions, 1–10
SET USR NOSWAP, 1–7
SET USR SWAP, 1–7
USR (Cont.)
subroutines requiring, 1–6
swapping over data, 1–6
USR requirements
in mapped monitors, 1–6
in unmapped monitors, 1–6
USR restrictions
examining link map, 1–11

V

VERIFY/IVERIFY
description, 2–256
example, 2–256

W

WAIT/IWAIT
See also READ
description, 2–257
WRITC/IWRITC/MWRITC
See also READC
description, 2–260
WRITE/IWRITE/MWRITE
See also ABTIO
description, 2–258
WRITF/IWRITF
See also IREADF
description, 2–261
WRITW/IWRITW/MWRITW
See also REOPN
description, 2–262