DataGeneral

Software Documentation

FORTRAN 5 Programmer's Guide (RDOS)

093-000227-01

FORTRAN 5
Programmer's Guide
(RDOS)

093-000227-01

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000227

Revision 01, January 1984

FORTRAN 5
Programmer's Guide
(RDOS)
093-000227

Revision History:

Original Release - October 1978
First Revision - January 1984

CONTENT UNCHANGED

The content in this revision is unchanged from 093-000227-00. This revision changes only printing and binding details.

Preface

As a programmer fluent in FORTRAN or a similar language and familiar with the Real-Time Disk Operating System (RDOS), you will find this Programmer's Guide a useful companion to the *FORTRAN 5 Reference Manual.*

Two parts comprise this Guide. We convey detailed aspects of operating FORTRAN 5 under RDOS, error handling, and the runtime environment in Part I. If you write your own runtime routines, Chapter 3, *Runtime Environment,* will interest you.

Part I will grow. It's a flexible part of the Guide that will include programming innovations as we document them.

In Part II, we present FORTRAN 5's runtime routines. You can refer to this section for the calling formats of these routines. After you read Part I's chapter on operating instructions, you can use FORTRAN 5's runtime. Understanding all of Part I's information is not essential for you to use Part II.

For your convenience, Appendix A lists system and FORTRAN 5 runtime error messages by error code number.

You may supplement the Programmer's Guide with the *RDOS Reference Manual* and the *Macroassembler User's Manual.*

Reader, Please Note:

We use these conventions for routine name formats in this manual:

ROUTINE NAME required *[optional]* ...

Where	Means
ROUTINE NAME	You must enter the routine name (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\left\{\begin{matrix}\text{required}_1\\ \text{required}_2\end{matrix}\right\}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering some argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
)	Press the NEW-LINE or RETURN key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, we usually show all examples of entries and system responses in THIS TYPEFACE. But, where we *must* clearly differentiate your entries from system responses in a dialog, we will use

THIS TYPEFACE TO SHOW YOUR ENTRY)
THIS TYPEFACE FOR THE SYSTEM RESPONSE

End of Preface

Contents

Part One

Chapter 1 - Using FORTRAN 5 Under RDOS

Compiling a FORTRAN 5 Program Under RDOS I-1-1
Compilation Examples I-1-2
Assembling Your Assembly Language Routines I-1-2
Loading FORTRAN 5 Programs I-1-3
FORTRAN 5 Command Line Examples I-1-3
Limiting the Amount of Memory Available to the FORTRAN 5 Environment I-1-3
Setting Maximum Line Length for Output I-1-4
FORTRAN 5 Unit Numbers I-1-5
FORTRAN 5 I/O Preconnections I-1-5
FORTRAN 5 Overlay Facilities I-1-6

Chapter 2 - Error Handling

Status Variables I-2-1
ERR= and END= Options in FORTRAN 5 Statements I-2-1
Traceback I-2-2
LONGTRACE I-2-2
Short Form Traceback I-2-3
NOTRACE I-2-3
Error Messages: Text Versus Numbers I-2-3
Error Files I-2-3

Chapter 3 - Runtime Environment

Runtime Memory Allocation I-3-1
Configuration of Main Memory I-3-2
Runtime Stack Area I-3-4
FORTRAN 5 Runtime Stack Discipline I-3-4
Using the Stack I-3-5
Using Macros in the FORTRAN 5 Environment I-3-5
What Macros Does FORTRAN 5 Provide? I-3-5
Linkage Conventions for Runtime Routines I-3-5
Assembly Language Routine Called from FORTRAN 5 . . . . . I-3-5
Examples of Assembly Language Routines I-3-6
Calling a FORTRAN 5 or Assembly Language Routine I-3-8

Part Two

Chapter 1 - Introduction

Conventions II-1-1
Aggregates II-1-1
Integer II-1-1
The ier Argument II-1-1
Error Conditions II-1-2

Chapter 2 - Checking for Arithmetic Errors

DVDCHK II-2-1
OVERFL II-2-1

Chapter 3 - Performing Logical Operations with Integers and Words

IAND II-3-1
ICLR II-3-1
IOR II-3-2
ISET II-3-2
ISHIFT (Alias:ISHFT) II-3-3
ITEST II-3-3
IXOR (Alias: IEOR) II-3-4
NOT II-3-4

Chapter 4 - Managing Directories and Devices

CDIR II-4-1
CPART II-4-2
DIR II-4-2
EQUIV II-4-3
GDIR II-4-3
GSYS II-4-4
INIT II-4-4
MDIR II-4-5
RELEASE (Alias: RLSE) II-4-5

Chapter 5 - Maintaining Files

CFILW (Alias: CFIL) II-5-1
CHATR II-5-2
CHLAT II-5-2
CHSTS II-5-3
DFILW (Alias: DFIL) II-5-3
FDELETE II-5-4
.FIOPREP II-5-4
FRENAME II-5-5
GTATR II-5-5
LINK II-5-6
RENAME II-5-6
RSTAT II-5-7
STAT II-5-7
UNLINK II-5-8
UPDATE II-5-8

Chapter 6 - File Input/Output

APPEND II-6-2
BACKSPACE (Alias: FBCKSP) II-6-3
CHRST II-6-3
CHSAV II-6-4
CLOSE II-6-4
FCLOSE II-6-5
FOPEN II-6-5
MTDIO II-6-6
MTOPD II-6-6
OPEN II-6-7
RDBLK II-6-8
RDLIN II-6-9
RDSEQ II-6-10
READRW (Aliases: READR, RDRW) II-6-11
RESET II-6-12
REWIND (Alias: FRWND) II-6-12
WRBLK II-6-13
WRITRW (Alias: WRITR) II-6-14
WRLIN II-6-15
WRSEQ II-6-15

Chapter 7 - Console Input/Output

GCHAR II-7-1
GCIN II-7-2
GCOUT II-7-2
ODIS II-7-3
OEBL II-7-3
PCHAR II-7-4
RCHAR II-7-4
WCHAR (RTOS only) II-7-5

Chapter 8 - Reading the Console Switches

RDSW (Alias: DATSW) II-8-1

Chapter 9 - Using the System Clock and Calendar

DATE II-9-1
FGDAY II-9-2
FGTIME II-9-2
FSDAY II-9-3
FSTIME II-9-3
SDATE II-9-4
STIME II-9-4
TIME II-9-5

Chapter 10 - Multitask Programming in FORTRAN 5

Tasks and Their Resources II-10-1
Memory Partitions in a Multitask Runtime Environment II-10-1
Classes of Suspensions II-10-3

Chapter 11 - Initiating Tasks in a Multitask Environment

FTASK II-11-1
ITASK II-11-2
S?TASK II-11-3

Chapter 12 - Changing Task States in a Multitask Environment

AKILL II-12-2
ARDY II-12-2
ASUSP II-12-3
DESTROY II-12-3
.IWAKE II-12-4
KILL II-12-4
PRI II-12-5
SUSP II-12-5
TIDK (Aliases: TIDKILL, ABORT) II-12-6
TIDP (Aliases: TIDPRI, CHNGE, CHPRI) II-12-6
TIDR (Aliases: TIDRDY, RELSE) II-12-7
TIDS (Aliases: TIDSUSP, HOLD) II-12-7

Chapter 13 - Obtaining Task-Related Information in a Multitask Environment

GETEV II-13-1
GETPRI II-13-1
IDST (Alias: STTSK) II-13-2
MYEV II-13-2
MYID II-13-3
MYPRI II-13-3

Chapter 14 - Intertask Communication

REC II-14-1
XMT II-14-2
XMTW II-14-2

Chapter 15 - Using Overlays

EST II-15-2
OVCLOSE II-15-2
OVEXIT II-15-3
OVKILL II-15-3
OVLOD (Aliases: FOVLY, FOVLD) II-15-4
OVOPN II-15-4
OVREL (Aliases: FOVRL, UNEST) II-15-5

Chapter 16 - Delayed Or Periodic Execution

Requesting Delayed/Periodic Task Initiation II-16-1
Overlay Considerations II-16-1
Premature Termination of a Request II-16-2
Timing II-16-2
ASSOCIATE II-16-4
CANCL II-16-4
CYCLE II-16-5
DQTSK (Alias: DQTASK) II-16-5
FQTSK (Alias: FQTASK) II-16-6
START II-16-6
TRNON II-16-7

Chapter 17 - Task/Operator Communications in a Multitask Environment

IOPC II-17-2
IOPROG II-17-3
TRDOP II-17-4
TWROP II-17-4

Chapter 18 - User/System Clock Commands

FDELAY (Alias: FDELY) II-18-1
GHRZ (Alias: GFREQ) II-18-1
WAIT II-18-2

Chapter 19 - Enabling and Disabling the Multitask Environment

MULTITASK II-19-2
SINGLETASK II-19-2

Chapter 20 - Calling To and Returning From Programs

CHAIN II-20-2
CHECK II-20-2
COMARG II-20-3
EBACK II-20-4
ERROR II-20-4
EXIT II-20-5
FCHAN II-20-5
FSWAP II-20-6
GETERR II-20-6
MESSAGE II-20-7
SWAP II-20-7

Chapter 21 - Using Extended Memory

Using Extended Memory in a Multitask Environment II-21-1
ERDB (Alias: ERDBLK) II-21-3
EWRB (Alias: EWRBLK) II-21-4
MAPDF II-21-5
REMAP II-21-6
VDUMP II-21-7
VFETCH II-21-7
VLOAD II-21-9
VMEM II-21-9
VSTASH (Alias: VS) II-21-10

Chapter 22 - Foreground/Background Programming

EXBG II-22-2
EXFG II-22-3
FGND II-22-3
GROUND (Alias: WHERE) II-22-4
ICMN II-22-4
RDCMN II-22-5
RDOPR II-22-5
WRCMN II-22-6
WROPR II-22-6

Chapter 23 - Multiple Processor Routines

BOOT II-23-1
GMCA II-23-2
GMCA1 . . . . . II-23-2

Chapter 24 - Controlling Spooling

SPEBL II-24-1
SPDIS II-24-1
SPKILL II-24-2

Chapter 25 - User Devices and Interrupts

DUCLK II-25-1
FINRV II-25-2
FINTD II-25-2
RUCLK II-25-3

Appendix A - F5ERR.FR - FORTRAN 5 Runtime Error Parameters

Tables

Part One

Table	Caption	
I-1-1	Default File Preconnections	I-1-5
I-1-2	IBM-Emulated File Preconnections	I-1-6

Part Two

II-16-1	Queue Table	II-16-2

Illustrations

Part One

Figure	Caption	
I-1-1	Limiting Memory in FORTRAN 5	I-1-3
I-1-2	LINESIZE Example	I-1-4
I-2-1	LONGTRACE Output	I-2-2
I-3-1	Configuration of Memory in Runtime	I-3-3
I-3-2	Stack Area	I-3-4
I-3-3	Runtime Stack Frame	I-3-4
I-3-4	Example One--An Assembly Language Routine	I-3-7
I-3-5	Example Two--An Assembly Language Routine	I-3-8
I-3-6	Runtime Stack at Various Stages of a Subroutine Call	I-3-9
I-3-7	Calling a Routine from Assembly Language	I-3-10

Part Two

II-11-1	S?TASK Example	II-11-3

Chapter 1
Using FORTRAN 5 Under RDOS

Data General's RDOS FORTRAN 5 compiler is ten separate files: FORTRAN.SV, FTC1.SV, FTC2.SV, FTC3.SV, FTC4.SV, FTC5.SV, FTC6.SV, FTC7.SV, FTCE.SV, and FTC.TX. Load these files on disk with the following runtime libraries:

F5ENV.LB, the environment support routines;
F5IO.LB, the I/O routines;
F5ISA.LB, the system interface routines;
F5MATH.LB, the math routines;
F5TASK.LB, the multitask support routines.

For more loading information, see the release notice that accompanies your FORTRAN 5 compiler package.

Separately compile each FORTRAN 5 main program, subroutine subprogram, and function subprogram. Once you successfully compile your source program, use RLDR with FLIB or TFLIB, indirect files, to build your executable save file. FLIB and TFLIB name the FORTRAN 5 libraries in the proper order. Following are examples of RDOS CLI commands that compile, load, and execute a FORTRAN 5 program.

Examples

Compile:

```
FORTRAN MAIN)
FORTRAN SUB1)
FORTRAN XFUN)
FORTRAN XSUB)
```

Load:

```
RLDR/N MAIN SUB1 XFUN XSUB @FLIB@)
```

Execute:

```
MAIN)
```

Compiling a FORTRAN 5 Program Under RDOS

Issue the following command to the CLI to compile a FORTRAN 5 program:

```
FORTRAN inputfilename
```

This command produces a relocatable binary file named, inputfilename.RB, with no listing and sends error messages to the console.

The general form of the FORTRAN 5 command line is:

FORTRAN *[global switches]* inputfilename filename *[local switches]*...

Global Switches	Action
/B	Produce a brief listing. This includes the input source program, the storage map, the list of all subprograms called, the cross-reference, and the error list (but not the generated code).
/C	Check the syntax of the source program. If you specified a listing file, the source program and the error list are sent to it. The error list is also sent to the error file, if one exists.
/D	Debug. Compile code that allows the long form error traceback routine to output line numbers. This option doesn't provide more information when an error occurs but provides for a more convenient form. Don't use /D in final versions of programs. See Part I, Chapter 2, for more information concerning this switch.

Global Switches	Action
/I	Don't list source lines from INCLUDE files. /I permits you to include large parameter files in programs without producing bulky listings. Line numbers on /I listings correspond to those on standard listings.
/L	Produce a listing. If you explicitly specify a listing file using the local /L switch, the global /L switch has no effect. Global /L produces a listing filename, inputfilename.LS.
/N	Do not produce an object file.
/M	Generate short Load Effective Address instructions (LEF's) where possible. Use /M when the resulting program runs on a mapped ECLIPSE.
/P	Use punched card input. Only the first 72 characters of each input line are used as FORTRAN 5 source code, but the entire input line is sent to the listing file, if one exists.
/S	Generate code to check subscript references. A runtime routine determines whether or not a reference lies within the array. For singly-subscripted arrays, the check always catches bad references. For arrays with more than one subscript, the check may not catch an out-of-range subscript. For example: DIMENSION A(2,4) B=1(3,2) produces no error since the address calculated for A(3,2) is the same as that for A(1,3), which is within the array.
/X	Compile lines with an X in column one. If you don't use /X, the system treats these lines as comments.

Local Switches	Action
/B	Direct the relocatable binary output to the file named. You receive no listing of the generated code when you use this switch.
/E	Direct any error messages to the file named.
/L	Direct the listing to the specified file. The listing contains the input source program, a storage map, a list of all subprograms called, a listing of the generated code, a cross reference, and a list of errors.

Compilation Examples

```
FORTRAN/B MYPROG)
```

Compiles either MYPROG.FR or MYPROG, depending on the existence of the .FR file. This command sends a brief listing to the current MYPROG.LS file. Since there's no /E switch, all errors are output to the console. The compiler produces the object file, MYPROG.RB.

```
FORTRAN/I PROG.LS/L PROG ERRORS/E)
```

Compiles either PROG.FR or PROG, depending on the existence of the .FR file. This command generates a listing file, PROG.LS. If PROG.LS already exists, the new listing is appended to it. The compilation command sends errors to the file, ERRORS. If ERRORS already exists, the new information is appended to it.

Assembling Your Assembly Language Routines

Use the macroassembler, MAC.SV, to assemble assembly language routines you want included with compiled FORTRAN 5 programs. You can create a file of permanent symbols, MAC.PS, for the assembler by following steps 1 and 2 below.

1. The current directory must contain or have access to the following files:

```
%%
F5SYM.SR
FMAC.SR
NBID.SR
NSKID.SR
LITMACS.SR
NF5SYM.SR (NOVA RDOS only)
NRTF5SYM.SR (NOVA RTOS only)
NF5SYM.AS (NOVA RDOS only)
NRTF5SYM.AS (NOVA RTOS only)
EF5SYM.SR (ECLIPSE RDOS only)
ERTF5SYM.SR (ECLIPSE RTOS only)
EF5SYM.AS (ECLIPSE RDOS only)
ERTF5SYM.AS (ECLIPSE RTOS only)
NEID.SR (ECLIPSE only)
NFPID.SR (ECLIPSE only)
OSID.SR
%%
```

2. Input one of the following commands to the CLI:

For NOVA RDOS:	MAC/S @NF5SYM.AS@
For NOVA RTOS:	@NRTF5SYM.AS@
For ECLIPSE RDOS:	MAC/S @EF5SYM.AS@
For ECLIPSE RTOS:	MAC/S @ERTF5SYM.AS@

   These commands execute only the first pass of the macroassembler to produce MAC.PS.

3. Assemble your assembly language routines with the macroassembler according to the description of the MAC command in the *CLI User's Manual.*

Loading FORTRAN 5 Programs

Load your FORTRAN 5 program with the RLDR command. Use the indirect files, FLIB or TFLIB, to name the libraries in the proper order.

In general, you load your program in the following sequence:

- main FORTRAN 5 program.
- user subprograms and optional user modules.
- support libraries (e.g., Commercial Subroutine Package).

You may need to use the global /N switch with the RLDR command. Since FLIB and TFLIB list SYS.LB, you can prevent RLDR from searching SYS.LB a second time by using global /N. For more information, refer to the RLDR section in the *CLI User's Manual.*

FORTRAN 5 Command Line Examples

```
RLDR/N MYPROG @FLIB@)
```

Loads the main program, MYPROG, and the required FORTRAN 5 runtime routines. Creates the save file, MYPROG.SV.

```
RLDR/N NEWPROG.LM/L NEWPROG.SV/S PROG @FLIB@)
```

Creates the save file, NEWPROG.SV, from the object file, PROG.RB, and loads the required FORTRAN 5 runtime routines. Generates NEWPROG.LM, a listing file.

```
RLDR/N EXAMPLE3 [SUB3,SUB4] @FLIB@)
```

Creates EXAMPLE3.SV, the save file, and EXAMPLE3.OL, the overlay file. The program includes a single overlay area with two overlays, one containing *SUB3* and the other containing *SUB4.*

Limiting the Amount of Memory Available to the FORTRAN 5 Environment

Figure I-1-1 shows you how to prevent the FORTRAN 5 initializer from allocating all of its available memory to a program.

After assembling the module, load it into the program file by including its name in the RLDR command line.

```
.TITLE          NMAX     ;NAME THIS MODULE
.ENT            .NMAX    ;MAKE THIS SYMBOL KNOWN ELSEWHERE
.NREL 1

.NMAX:          77777    ;HIGHEST ADDRESS
                         ;TO THE FORTRAN 5 ENVIRONMENT.
                         ;(IN THIS CASE, YOU ARE GIVEN ALL
                         ;THE MEMORY AVAILABLE.)

.END                     ;END THIS MODULE.
```

Figure I-1-1. Limiting Memory in FORTRAN 5.

Setting A Maximum Line Length for Output

FORTRAN 5 lets you set a maximum line length for output files and devices. When a line of formatted output exceeds this length, you get the error message, OUTPUT RECORD TOO LONG. When a line of free-formatted output exceeds this length, the excess spills to the next line. Numeric data items are not split across two lines, but Hollerith and string constants (not checked in this way) will split. If a data item other than a Hollerith or string constant spills from one line to a second, and cannot fit on the new line, you get the error message, OUTPUT RECORD TOO LONG.

FORTRAN 5 provides standard default line lengths, but you may define your own defaults, or override the defaults explicitly, file-by-file.

The OPEN statement defines explicit line lengths with the record length option *(LEN=n)* specifying the line length, and the attribute option *(ATT="L")* specifying that the file organization is line-oriented rather than fixed-length records. For example, the statement:

```
OPEN "OUT",ATT="L",LEN=40
```

opens a line-oriented file with a maximum line length of 40 characters. If you attempt to output a longer line, you either get the message, OUTPUT RECORD TOO LONG, or the excess spills to the next line (as previously described).

If you OPEN a file with no options specified, the system assumes that it is line-oriented, and uses the default line length of 132 characters. You may change the default line length by the following method:

1. Edit the file LINESIZE.SR (as shown in Figure I-1-2), changing the default value to the one you want. value to the one you want.
2. Assemble LINESIZE.SR (see *Assembling Your Assembly Language Subprograms).*
3. Include LINESIZE.RB in the RLDR command line when building the save file. It must precede F5IO.LB, named in FLIB and TFLIB.

```
;LINESIZE (COMMON)
;
;DEFINE THE DEFAULT LINE LENGTH

**TITL          LINESIZE

.ENT            L.I.N


.LIN. = MAXLL   ;DEFINE GENERAL DEFAULT LINE LENGTH
                ;AS SYSTEM-DEFINED MAXIMUM LINE LENGTH

L.I.N = .LIN.+FALIN     ;LINE LENGTH + LINE-ORIENTED BIT

END
```

Figure I-1-2. LINESIZE Example

FORTRAN 5 Unit Numbers

FORTRAN 5 manages unit numbers on a per-program basis. A program may use up to 64 files, numbered from 0 to 63.

FORTRAN 5 I/O Preconnections

Under FORTRAN 5 there are conventional statement/unit number and unit number/filename preconnections. You can alter these implicit connections by editing one of two preconnection source files, IBMPCT.SR or DGCPCT.SR.

The statements, TYPE, ACCEPT, PUNCH, PRINT, and READ, don't allow you to explicitly mention unit numbers. But you may edit the preconnection source file, directing the statements to the unit numbers of your choice.

Editing the same source file enables you to specify unit number/filename and device name preconnections. When you open a file explicitly with an OPEN statement, the unit number that you provide is associated with the file. If, however, you specify a unit number in an I/O statement before it is associated with a particular file, a file preconnection table is checked, giving one of two results:

1. If a filename is preconnected to the unit number you specified, then that file is opened and its name is associated with the unit number.

2. If a filename is not preconnected to the unit number you specified, an error is signaled.

The preconnections that exist by default are shown in Table I-1-1.

Table I-1-1. Default File Preconnections

Statement	Unit Number	
	DGC	IBM
READ	9	5
PRINT	12	6
PUNCH	14	7
TYPE	10	10
ACCEPT	11	11

Unit Number	Device Name	Meaning
6	$PLT	Incremental plotter
8	$TTP	Teletype tape punch
9	$CDR	Card reader
10	$TTO or $TTO1	Console output device
11	$TTI or $TTI1	Console input device
12	$LPT	Lineprinter (has P attribute)
13	$PTR	Paper tape reader
14	$PTP	Paper tape punch
15	$TTR	Teletype tape reader

FORTRAN 5 provides you with two ways to change the default file preconnections. If you would like file preconnections similar to those used in IBM FORTRAN IV, name the file IBMPCT.RB in your RLDR command line. This defines file preconnections as shown in Table I-1-2.

Table I-1-2. IBM-Emulated File Preconnections

Unit Number	Device Name	Meaning
5	$CDR	Card reader
6	$LPT	Line printer (has P attribute)
10	$TTO or $TTO1	Console output
11	$TTI or $TTI1	Console output

If you want to define your own I/O preconnections, follow this procedure:

1. Create an assembly language source file of your own, specifying the I/O preconnections you want. You may provide most of the same information in the file preconnection table that you provide in the OPEN statement. You build the table by calling the macro PRECON defined in DGCPCT.SR or IBMPCT.SR. In producing a source file, you'll find it simplest to edit either of these two files.
2. Assemble your preconnection file as described previously in *Assembling Your Assembly Language Subprograms.*
3. Specify the object filename, produced by the assembly, in your RLDR command line before the FORTRAN 5 libraries.

FORTRAN 5 Overlay Facilities

FORTRAN 5 gives you access to the standard RDOS overlay mechanism when you call FORTRAN 5 interface routines such as OVLOD, OVREL, and OVEXIT. You must load a specific overlay before calling any routine it contains. Release the overlay when you no longer need it.

Another FORTRAN 5 overlay facility is the *load-on-call* mechanism. When you use this mechanism, any routine that resides in the overlay is automatically loaded if you call it. The load-on-call overlay is released when you return from the overlay.

To use the load-on-call mechanism you must set up a table of information for the load-on-call overlay manager. You must also open the overlay file with OVOPN. See LOCO.SR in your FORTRAN 5 software package for more information on the load-on-call mechanism.

End of Chapter

Chapter 2
Error Handling

FORTRAN 5 never ignores errors. It either acts on them or returns error codes so you can act on them. This error-handling chapter describes the actions FORTRAN 5 takes, your control of those actions, and the actions you take when a routine returns an error code.

FORTRAN 5 acts upon three kinds of errors. Mathematical errors are always reported by sending appropriate information to the error files you specified. The program continues execution. You can neither intercept control when a mathematical error occurs nor suppress the reporting of the error. The second kind of error occurs when FORTRAN 5 detects an error from which recovery is impossible or undesirable. In this case FORTRAN 5 sends a message to the error files and terminates your program. Errors of this type currently include: stack overflow, subscript out-of-bounds, and extended memory reference out-of-bounds.

FORTRAN 5 may act upon all other errors or may pass them on to you for action. When you call any particular FORTRAN 5 routine, the calling sequence determines your choice of error-handling alternatives. For a routine returning a status variable, FORTRAN 5 will pass back a 1 in that variable if the routine is completed successfully. If a problem occurred it will return an error code. FORTRAN 5 will never act on an error which occurs in such a routine, but will always leave the action up to you.

Some routines' calling sequences do not include a status variable. FORTRAN 5 acts on errors in these routines by sending an error message to the error files. Then your program terminates.

If FORTRAN 5 statements such as DELETE, RENAME, or WAKEUP detect errors, FORTRAN 5 will handle them because it can't pass an error code to you. Errors detected in I/O statements or task statements with ERR= or END= clauses will cause a transfer of control to the statement you name in the appropriate clause. After the transfer of control, you may determine what error occurred by calling the routine GETERR.

Status Variables

In a subroutine call the status variable is always the last argument. The subroutine returns a status code that is either 1 or an error code. A 1 indicates that no error occurred.

The Instrument Society of America (ISA) convention requires all error codes to be greater than or equal to 3. Since the system starts its error codes at 0, FORTRAN 5 must add 3 to all system-defined error codes in order to comply with the standard.

Never ignore status variables. You should always check them for information about occurring errors. To accomplish this, use the following format:

```
CALL CHECK (error code)
```

When CHECK sees a value of 1, program execution continues. If the value is not 1, CHECK invokes the error reporter, and the program stops.

If you check the status variables yourself, you can control error processing. The file in Appendix A, F5ERR.FR is useful. Use its definitions of error names to check for specific errors. Instead of referring to Appendix A, you can incorporate all of F5ERR.FR (or only those error definitions you need) into your program with the INCLUDE statement.

You can also signal an error by calling CHECK with any error code defined in F5ERR.FR. The FORTRAN 5 runtime error reporter will process it and terminate your program.

ERR= and END= Options in FORTRAN 5 Statements

If a FORTRAN 5 statement does not include an ERR= or END= clause, errors that occur will terminate your program. ERR= specifies a statement that receives control if any error occurs during execution of the statement. END= specifies a statement that receives control when an end-of-file condition is detected during execution of the statement.

If both clauses occur in FORTRAN 5 statements, END= takes control of an end-of-file condition, and ERR= takes control on all other cases. If END= is not present, ERR= takes control of an end-of-file condition.

You can examine the error code causing the most recent ERR= or END= branch by calling the routine, GETERR. GETERR accepts one argument, an integer variable, in which it returns the error code.

A call to GETERR clears the internally saved error code. It's the sole method in which the internal error code is cleared. If neither an ERR= nor an END= branch has occurred, GETERR returns 1.

Traceback

Traceback is an error-handling mechanism that indicates where an error occurred in a program. This mechanism provides output when an error occurs in either a routine not returning a status variable or internal runtime code. You may choose one of three approaches to using traceback -- LONGTRACE, NOTRACE, and short form traceback.

In selecting a form of traceback for an error-handling mechanism, consider the following guidelines. The short form gives you the same information as LONGTRACE, but occupies far less memory. LONGTRACE outputs routine names. It also outputs line numbers if you compile your routines with the global switch, /D.

Incorporating line numbers has both an advantage and a disadvantage. The advantage of using line numbers is that you don't need a full listing of your program's code to determine where an error occurred. The disadvantage of using line numbers is that it slows down your program's execution.

LONGTRACE

Inserting LONGTRACE into your RLDR command line gives you more readable traceback. Follow these command changes:

```
@FLIB@ = >F5ISA.LB SYS.LB LONGTRACE F5IO.LB
  F5MATH.LB F5ENV.LB
@TFLIB@ = >F5TASK.LB F5ISA.LB SYS.LB
  LONGTRACE F5IO.LB F5MATH.LB F5ENV.LB
```

If you include the global switch, /D, on the RLDR command causing the loading of the debugger, RLDR issues an error message stating that DEBUG has been defined more than once. This is normal for the load sequence.

See Figure I-2-1 for a sample of LONGTRACE output.

```
**ERROR** REPORTED BY DSQRT
        CALLED AT OFFSET 2E IN PROGRAM UNIT SUBR1
        CALLED AT OFFSET 13 (LINE 2) IN PROGRAM UNIT .MAIN
ILLEGAL ARGUMENT FOR SQRT
```

Figure I-2-1. LONGTRACE Output

Here is an interpretation of the output. An attempt to take the square root of a negative number caused an error. DSQRT, the double precision square root function, reported the error. The subroutine, SUBR1, called DSQRT. At offset 26 (octal) from the start of SUBR1, the compiler generated a call operation. The main program, .MAIN, called SUBR1 on line 2 (decimal). Since .MAIN was compiled with the global /D switch, the traceback output includes the line number.

In a multitask environment, LONGTRACE provides output for a reported error in the following way:

```
**ERROR** IN TASK 3 REPORTED BY DSQRT
```

The number 3 indicates the task identifier of the task reporting the error. Tasks with no identifier are reported as Task 0.

Short Form Traceback

If you don't select LONGTRACE, FORTRAN 5 provides the short form traceback. Notice the lack of line numbers in the following example of the short form.

```
**ERROR** REPORTED BY (50)
        CALLED AT 470+26
        CALLED AT 445+13
ILLEGAL ARGUMENT FOR SQRT
```

The short form traceback requires some information from the listing file that RLDR produces when the program is loaded. Following is a portion of the listing file with the information you needed.

```
.SSE        000042
.SOV        000043
DSQRT       000050
.ERET       000052
.RTER       000053
  .
  .
.LEFE       000401
.MAIN       000445
SUBR1       000470
.PCT        000537
.F5         000701
  .
  .
```

NOTRACE

In addition to the error-handling alternatives, LONGTRACE and short form traceback, you can also choose NOTRACE and produce no traceback output at all. In your RLDR command line, specify NOTRACE before F5ENV.LB. Since you'll receive no indication where an error occurs with this alternative, use NOTRACE only in completely debugged code.

Error Messages: Text Versus Numbers

You receive text associated with a reported error when you have access to FORTRAN.ER and choose the default version of the error reporter, RTER. When memory is a scarce resource, you can forego output of error message text. Select the error reporter, SHORTRTER, which outputs error numbers.

In decimal, the numbers follow the ISA error convention explained previously in *Status Variables;* they are greater than or equal to 3. These numbers have corresponding definitions contained in the file in Appendix A, F5ERR.FR.

F5ERR.FR has twofold use. First, you may refer to reported error codes. Secondly, your program may compare returned error values to specific errors by using the names that F5ERR.FR defines.

Error Files

As we mentioned in the introduction to this chapter, you may direct error output to any number of error files. By default, error message output goes to the console. You may add or remove files by editing either DGCPCT.SR or IBMPCT.SR. Decide what file to edit by reading the preconnection discussion in Chapter 1.

In order to prevent errors from going to the console, you remove the line:

```
EFILE CONOUT
```

from the source file. To send errors to a disk filename EFILE1, you add:

```
EFILE "EFILE1"
```

If you name a link in your error file definitions, you can unlink and relink the error file before each program run. This lets you produce a different error file each time your program runs without changing the program.

End of Chapter

Chapter 3
Runtime Environment

As you read this chapter on the runtime environment, you will realize more effective ways to use FORTRAN 5. The general areas we will discuss are: runtime memory allocation, runtime stack discipline, and the use of macros in the FORTRAN 5 runtime environment. For those of you who write your own assembly language routines, we explain subprogram linkage conventions.

If you write multitask programs or assembly language routines, you must understand the runtime environment. Without reading this chapter, however, you can refer to Part II of the Guide and call any of the runtime routines documented there.

Before you proceed to read our discussion of the runtime environment, you will need to know the definitions of three terms: process, task, and routine activation.

A *process* is the set of system resources allocated to the execution of a program. These resources include main memory, I/O devices, the floating point unit, and the CPU. Particular operating systems handle processes differently. Under AOS many processes can exist simultaneously, whereas RDOS supports only two processes -- the background and the foreground.

A *task* uses system resources such as memory and CPU control. A task is a logically complete unit of program execution. The program contains the instruction sequences -- the code paths -- that a task executes.

In discussing the concept of a task, we must explain both singletask and multitask operations. Singletask operation consists of a single flow of control through a program. The complexity of the program's branching structure does not affect this single flow.

Multitask operation consists of multiple, concurrent flows through a program. The various flows or tasks compete for CPU control. The multitask scheduler controls this competition by allocating resources to the highest priority task that is ready to execute.

Tasks may execute different code paths or the same code paths. For several tasks to execute the same code paths, the programs must permit these tasks to enter and execute a routine before prior executions are complete; this means the routines must be re-entrant.

A re-entrant routine allows several tasks to flow concurrently through it. The multitask scheduler coordinates task flow, while the tasks switch rapidly from one to another. Within the re-entrant environment all tasks use the routine's executable code; this code never changes. Each task in turn has its own code that provides data for task flow through the routine.

A *routine activation* is a single execution of a program unit. The main program is active throughout the entire program's execution. Accordingly, when you issue a call to a subroutine or function from the main program, the subroutine or function is active until it returns.

More than one activation of a single routine may occur at one time. If the activations belong to different tasks, the routine must be re-entrant. If the activations belong to the same task, the routine must be recursive.

A routine is recursive if it is re-entrant and it stores all of its data on the stack. In order for a routine to call itself or other routines, it must be recursive.

Runtime Memory Allocation

In order to acquire a basic understanding of the runtime environment, you must know how the system allocates memory at runtime.

There are three levels of data that determine memory allocation in the FORTRAN 5 runtime environment: per-process, per-task, and per-routine-activation.

Per-process data includes executable code, COMMON blocks and STATIC variables, subroutine linkage pointers in page zero, and information maintained by the operating system and FORTRAN 5. Each process has its own copy of per-process data. All tasks and all routines in each task can access this data.

Per-task data includes the task's status and priority, the contents of the accumulators, the carry, the floating point unit, the program counter (pointing to the next machine instruction to execute), and the task's memory partition (the stack, the I/O control blocks, and the task global area). Each task also has a copy of all per-task data in addition to a copy of per-process data. When several tasks comprise a process, there may be multiple copies of the per-task data; the task scheduler coordinates the use of each copy.

In addition, per-task data encompasses the FORTRAN 5 state variables: .SP, .FP, .SSE, .RP, .GP, .ND1, and .ND2. Controlling the runtime stack are .SP, .FP, and .SSE. Internal runtime support routines use .RP for a temporary storage. The variable, .GP points at the task global area. This area contains information that the task needs. This includes a pointer to the current I/O control block, and the last error causing an ERR= or END= branch to be taken. The state variables .ND1 and .ND2 are NOVA specific; routines emulating ECLIPSE instructions use these as temporaries.

Per-routine-activation data includes pointers to routine arguments, local variables (including arrays), and temporary storage of intermediate results from calculations.

Configuration of Main Memory

The stacks pictured in Figure I-3-1 represent the logical layout of memory for a runtime routine in either a singletask or multitask environment. Showing only the memory allocated to your program, the diagram excludes memory maintained by both the operating system and other processes. As you study the diagram, read the following explanations of the various logical parts.

Page zero contains certain FORTRAN 5 runtime routine linkages, locations reserved by the operating system and the hardware, and the task state variables -- .RP, .FP, .SP, .SSE, .RP, and .GP.

The *operating system tables* include the User Status Table (UST) and an overlay directory (if you're using overlays). This area also includes the task control blocks (TCBs).

The *executable code* includes your FORTRAN 5 main program, subroutines and functions, assembly language routines you write, FORTRAN 5 runtime library routines, and routines from the system library (SYS.LB). Since all executable code is per-process data, the routine may not modify any instructions. A routine modifies per-task and per-routine-activation data.

Page zero location .FT points to the FORTRAN 5 file table. This location contains information about all FORTRAN 5 I/O units. All of this information is per-process data.

Task control block extensions contain additional per-task information for each FORTRAN 5 task in multitasking environments. Examples of this per-task information are the floating point unit and certain of the FORTRAN 5 state variables. In a re-entrant routine the program modifies only per-task and per-routine-activation data. Each task's stack contains local data for compiled FORTRAN 5 program units and runtime routines.

The I/O control block (IOCB) for each task contains information about FORTRAN I/O statements in progress.

Each task's global area contains I/O and task control information. Page zero state variable .GP points to the task global area for the currently executing task.

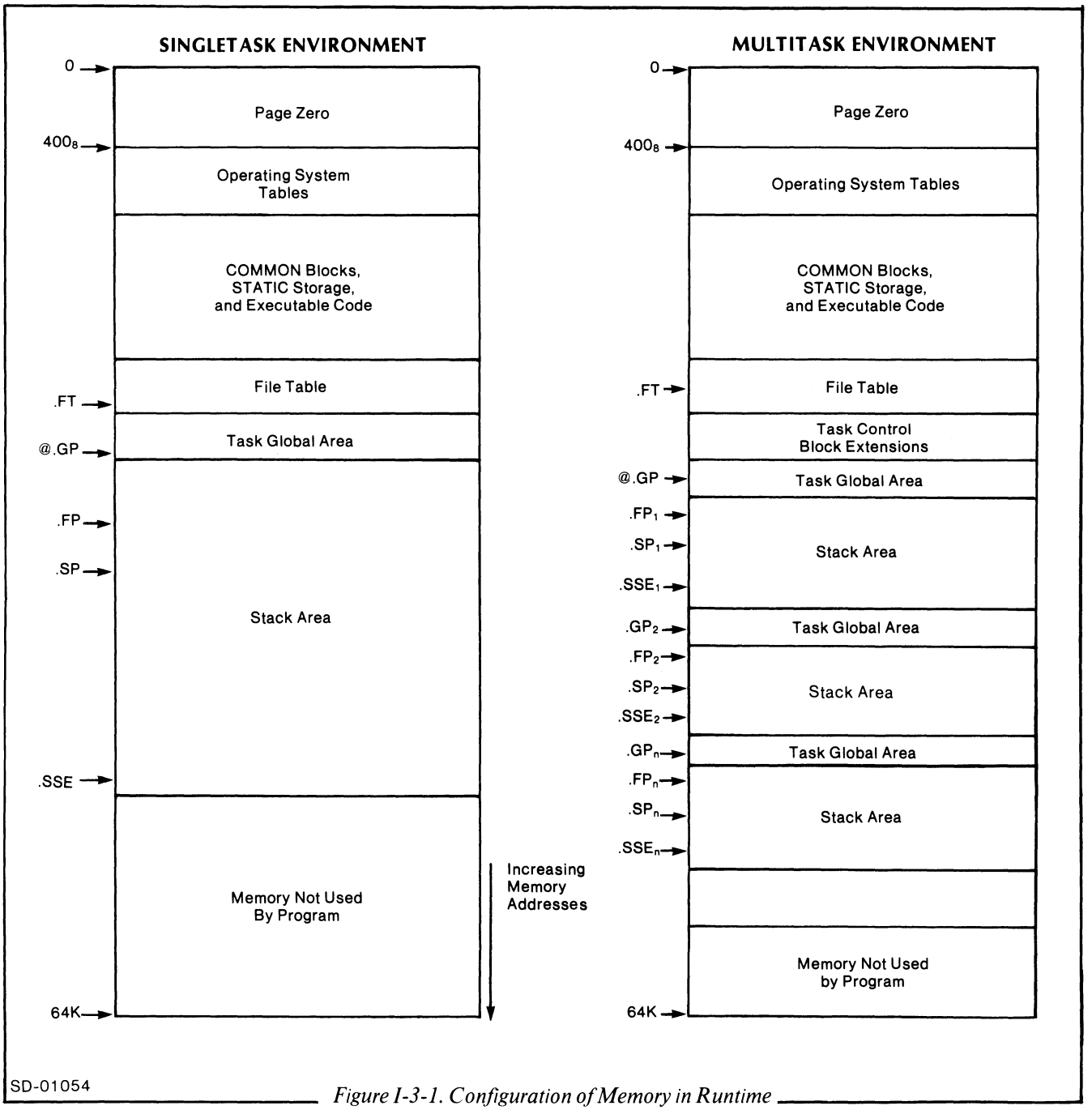


Figure I-3-1. Configuration of Memory in Runtime

Runtime Stack Area

Within the logical layout of memory is a runtime stack area. Each task has exactly one stack area. Runtime routines use the same stack as compiled FORTRAN 5 programs.

A stack extends upward in memory from the beginning of the runtime stack area. (Notice the direction of increasing memory addresses in Figure I-3-1.) A task uses its stack for several activities. Most FORTRAN 5 runtime routines use part of the runtime stack to pass arguments and to store information about the calling program, such as its return address and state of carry. If temporary storage is required by the runtime routine, it is allocated accordingly. I/O control blocks at the upper end of the runtime stack area hold input/output information during an I/O operation. When the I/O operation ends, it frees the stack space that the IOCB previously occupied.

The page zero state variables -- .SP and .SSE -- indicate the current allocation of a task's runtime stack area. Locating the current extent of the stack, the stack pointer, .SP, points to the most recent entry on the stack. The other stack indicator, .SSE, points to the highest location to which the stack can extend. It therefore reflects both the total size of the runtime stack area and the amount of space currently allocated to I/O control blocks.

Figure I-3-2 illustrates a task's runtime stack area in more detail than Figure I-3-1. The end zone, a buffer zone between the upper extent of the stack and the beginning of IOCB storage, protects the IOCB data in case of stack overflow.

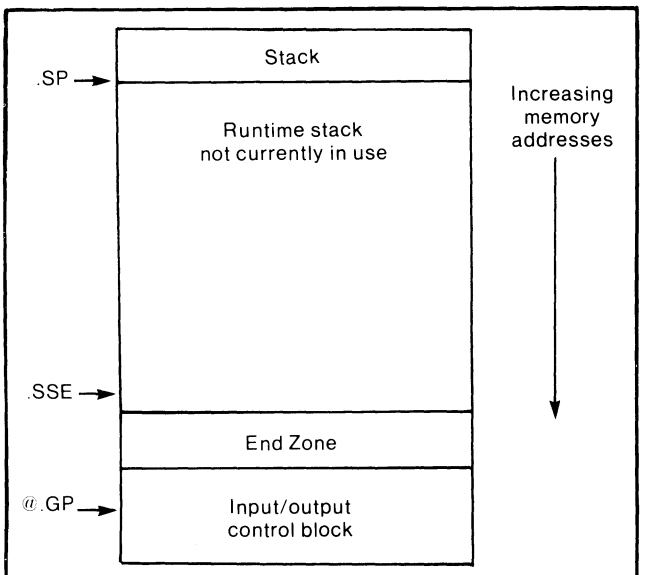


Figure I-3-2. Stack Area

FORTRAN 5 Runtime Stack Discipline

Certain page zero state variables define the stack activities. When the task scheduler gives control to a task, it sets up three words in page zero. The contents of these words -- .SP, .SSE, and .FP -- describe the runtime stack area that the executing task will use. When a task is not executing, its values for .SP, .FP, and .SSE are stored in the task control block.

The stack pointer, .SP, points to the last word of the stack that the task scheduler most recently allocated.

The stack limit, .SSE, points to the last word that the stack may include. If you attempt to make the stack pointer greater than the stack limit, stack overflow will result.

The frame pointer, .FP, points to the current stack frame. See Figure I-3-3 to get a picture of the runtime stack. Part of the stack, a stack frame, belongs to a single activation of a routine. All FORTRAN 5 routines and most runtime routines create stack frames.

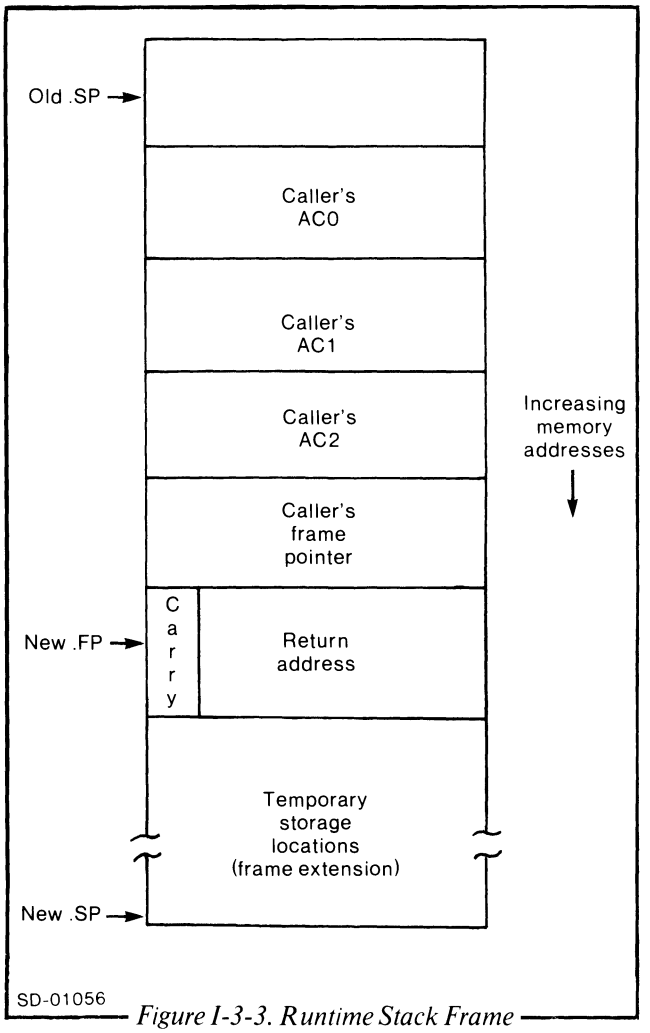


Figure I-3-3. Runtime Stack Frame

The stack frame contains the contents of the calling program unit's AC0, AC1, and AC2. It also contains the contents of the frame pointer when the routine was called. After the call, the stack frame stores the state of the carry bit and the contents of bits 1-15 of AC3. These bits contain the address of the instruction where the CPU will transfer control when the routine returns.

The routine that allocated the stack frame has space for local storage. This information is an extension of the information stored in the accumulators.

Using the Stack

There are several guidelines for using the stack. When you use the stack you issue the instructions -- PSH, SAVE, and POP. The PSH instruction puts new data in the stack. It provides a space for this data by incrementing the stack pointer and storing the data in the allocated words.

With the SAVE instruction, you direct a routine to build a stack frame. The routine allocates space on the stack for the new frame, succeeding the previous stack frame. Since the previous frame pointer is saved in the frame, you can execute a RTN instruction to restore the old environment. After executing a SAVE, the frame pointer points to the return address and carry. If the routine allocates memory for the extension mentioned in the stack description, the stack pointer points to the last word of that extension.

The POP instruction removes data from the stack. It decrements the stack pointer and loads the data into accumulators; then the words it occupied are free.

A stack overflow error occurs when you attempt to make the stack pointer greater than the stack limit. FORTRAN 5 reports the error and terminates your program.

Using Macros in the FORTRAN 5 Environment

If you are on a NOVA system, you use PSH, POP, SAVE, and RTN *macros,* not instructions. FORTRAN 5 provides macros to help you write assembly language routines.

A *macro* is a named sequence of instructions which you can include in an assembly language program by writing the name of the macro as if it were an instruction. Often when we refer to an instruction in this chapter, we mean instruction on an ECLIPSE, but a macro on the NOVA. If you use our macros, you can run ECLIPSE routines on the NOVA.

There are several advantages to using the macros we give you. Macros decrease the probability of errors occurring in your program. They check their arguments to ensure that you use them correctly, whereas there is no mechanism to ensure that you use a sequence of instructions correctly.

For information about the use and definition of macros and general information about writing assembly language code, see the *Macroassembler User's Manual.* When you need definitions of instructions and examples of their use, see the *ECLIPSE-Line Computers Programmer's Reference Manual.*

What Macros Does FORTRAN 5 Provide?

In addition to the PSH, POP, SAVE, and RTN macros, FORTRAN 5 provides macros that guide you in writing assembly language routines. We describe these other macros -- TITLE, DEFARGS, DEF, DEFTMPS, FENTRY, and END -- in the next section.

Linkage Conventions for Runtime Routines

If you write your own assembly language routines, you need to read about linkage conventions. In this section we will discuss how you call assembly language written routines from FORTRAN 5. We follow the discussion with example routines that demonstrate calling sequences and macro use. They will guide you in writing your own routines.

Assembly Language Routine Called from FORTRAN 5

When you write an assembly language routine to be called from FORTRAN 5, you must use the following macros. If you want to know the definitions of these macros, refer to the FORTRAN 5 parameter files, F5SYM.SR and FMAC.SR.

TITLE is the first line of your routine; comments, however, may precede TITLE. This macro specifies the title of the routine you are writing and initializes the environment for other macros.

DEFARGS immediately follows TITLE. Use this macro to start the definitions of your routine's arguments; you define each argument with the DEF macro. Even if your routine has no arguments, use DEFARGS.

DEF names each of your routine's arguments and temporaries. You name your arguments by using DEFARGS once and then writing:

```
DEF name
```

to name each of your routine's arguments. You must name the arguments in the order in which they appear when the routine is called.

DEFTMPS follows DEFARGS and its pertinent DEFs. Use DEFTMPS to start the definitions of your routine's temporaries; you define each temporary with the DEF macro. Even if your routine has no temporaries, write DEFTMPS into your routine.

DEF names each of your routine's arguments and temporaries. You name your temporaries by using DEFTMPS once and then writing:

DEF name *[size]*

to name each of your routine's temporaries. The *size* is optional and specifies the number of words you want to allocate for the named temporary. If you specify size, it must be a parenthetical expression.

FENTRY follows DEFTMPS or the last DEF. FENTRY generates a SAVE instruction, so you do not write a SAVE yourself. The SAVE generated by FENTRY allocates memory for all of the temporaries you named with DEFTMPS and DEF. AC3 contains the new frame pointer when the first instruction after FENTRY is executed.

FRET returns your routine to its caller. This macro generates an RTN instruction. It restores AC0, AC1, and AC2 to the values they had upon entry to your routine. FRET also transfers control to the appropriate location. You need not set up any registers or memory locations before using FRET.

END must be the last line of your routine. This macro generates the .END that the assembler needs. END terminates the environment prepared by the preceding macros. Do not substitute .END for END.

As you write your assembly language routine, you will address arguments and temporaries. Addressing requires a stack frame. When you issue the FENTRY macro, it generates the SAVE that creates the stack frame. After the routine executes the first instruction, the frame pointer is in AC3. If you need to load the frame pointer into an accumulator later in your routine, use the LDAFP macro,

```
LDAFP ac
```

where ac is the accumulator into which you load the frame pointer.

Examples of Assembly Language Routines

We present examples of assembly language routines in this section. Read through the FORTRAN 5 routine and then compare it with its translation into assembly language. In the assembly language example, we illustrate the use of the macros: DEFARGS, DEFTMPS, FENTRY, FRET, and END. In this example we address only arguments, whereas in a final, more detailed assembly language routine, we address both arguments and temporaries. These examples will provide guidelines for writing your own routines to be called from FORTRAN 5.

The FORTRAN 5 routine is:

```
SUBROUTINE FRED (B, A, C)
   .
   .
   .
RETURN
END
```

In assembly language you would write this routine as shown in Figure I-3-4.

```
                        ;TITLE FRED

DEFARGS         ;START DEFINING ARGUMENTS

DEF B           ;FIRST ARGUMENT

DEF A           ;SECOND ARGUMENT

DEF C           ;THIRD ARGUMENT

DEFTMPS         ;DONE DEFINING ARGS, START DEFINING TEMPORARIES
                ;NO TEMPORARIES

FENTRY FRED     ;DEFINE THE ENTRY POINT AND GENERATE A 'SAVE'
        .
        .
        .
        FRET

        END
```

Figure I-3-4. Example One--An Assembly Language Routine

```
;BASIC EXAMPLE OF AN ASSEMBLY LANGUAGE ROUTINE
;
;CALLING SEQUENCE:
;
;     CALL EX1 (  <JANE>  )

            TITLE     EX1      ;NAME THIS MODULE

DEFARGS     ;START DEFINING ARGUMENTS

   DEF JANE             ;NAME THE ARGUMENT

DEFTMPS     ;DONE DEFINING ARGS, START DEFINING TEMPORARIES

   DEF T1                    ;ONE WORD
   DEF T2          (10.)     ;TEN WORDS

FENTRY   EX1     ;DEFINE THE ENTRY POINT AND GENERATE A 'SAVE'

         LDA     0,JANE,3          ;ACC <- ADDRESS OF 'JANE'
         .
         LDA     0,@JANE,3         ;ACC <- VALUE OF 'JANE'
         .
         STA     0,T1,3            ;SAVE VALUE IN TEMPORARY T1
         .
         LDA     0,C.T1            ;ACC <- FRAME OFFSET OF T1
         ADD     3,0               ;ACC <- ADDRESS OF T1
         .
;NOW ASSUME THAT AC3 HAS BEEN CHANGED AND AC0 MUST BE LOADED
;WITH THE VALUE OF 'JANE'

   LDAFP          3          ;AC3 -> FRAME

         LDA     0,@SECOND,3     ;ACC <- VALUE OF 'JANE'
         .
         FRET                ;RETURN TO CALLER

C.T1:    T1                  ;DEPOSIT VALUE OF FRAME OFFSET

                END
```

Figure I-3-5. Example Two--An Assembly Language Routine

The second example of an assembly language routine, Figure I-3-5, illustrates addressing both arguments and temporaries with the macros.

Calling a FORTRAN 5 or Assembly Language Routine

To conclude this chapter on the runtime environment, we will explain how to write an assembly language routine that either calls a FORTRAN 5 or another assembly language routine. The previous section contains guidelines for writing a routine to be called from FORTRAN 5. From the other perspective, there are several conventions you must follow. Figure I-3-6 illustrates changes occurring on the stack when you implement these conventions.

First, load the stack pointer into AC2 with the LDASP macro. This is called marking the stack. Marking the stack allows the routine to count its arguments and to use the ISA status variable mechanism. See *Stage 1* in Figure I-3-6.

Secondly, you must push the addresses of all but the first two arguments onto the stack. Push the addresses in reverse order. See *Stage 2* in Figure I-3-6.

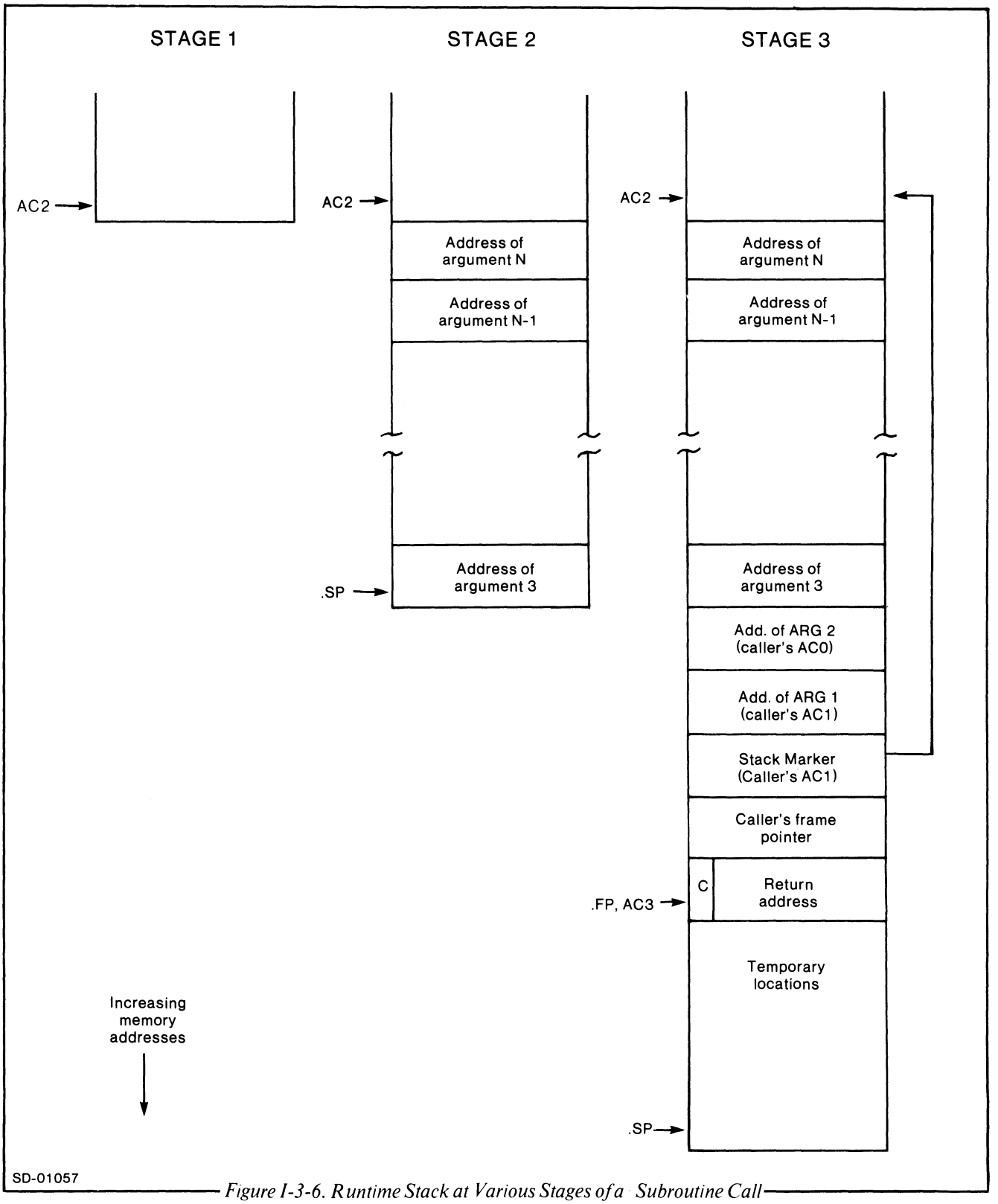


Figure I-3-6. Runtime Stack at Various Stages of a Subroutine Call

Load AC0 with the address of the second argument and load AC1 with the address of the first argument.

Call the routine with the FCALL macro. See *Stage 3* in Figure I-3-6. When the routine returns, AC3 will contain the frame pointer (*Stage 2* in I-3-6).

Store AC2 in the stack pointer with the STASP macro. This removes any pushed addresses from the stack. Always issue the STASP command when you call a FORTRAN 5 or assembly language routine. Even though you did not push any addresses, FORTRAN 5 may have pushed some addresses when FCALL was performed (*Stage 1*, I-3-6).

Follow the example in Figure I-3-7 for the application of these conventions.

```
;DEMONSTRATION OF CALLING A ROUTINE FROM ASSEMBLY LANGUAGE
;
;CALLING SEQUENCE;
;
;     CALL EX2  (  <JOE>, <BILL>, <SALLY>  )

              TITLE   EX2

DEFARGS

      DEF JOE
      DEF BILL
      DEF SALLY

DEFTMPS

      DEF TEMP

FENTRY   EX2

          .
          .
;SET UP TO DO:
;
;     CALL FRED  ( BILL, JOE, TEMP, SALLY  )

      LDASP    2              ;MARK THE STACK
      LDA      0,SALLY,3      ; -> LAST ARGUMENT FOR 'FRED'
      LDA      1,C.TEMP       ;FRAME OFFSET TO TEMP
      ADD      3,1            ; -> TEMPORARY, 3RD ARG FOR 'FRED'
      PSH      0,1            ;PUSH BOTH AT ONCE
      LDA      0,JOE,3        ; -> SECOND ARGUMENT FOR 'FRED'
      LDA      1,BILL,3       ; -> FIRST ARGUMENT FOR 'FRED'
      FCALL FRED              ;CALL 'FRED'
      STASP    2              ;CLEAN OFF STACK
      .
      .
      FRET          ;RETURN, ALL DONE

C.TEMP:  TEMP    ;FRAME OFFSET

             END
```

Figure I-3-7. Calling a Routine from Assembly Language

End of Chapter

Part II

Chapter 1
Introduction

Runtime comprises an important facet of a computer language. Here in Part II we organize FORTRAN 5's runtime routines into a usable format. In Part II, we also group functionally related runtime routines into chapters.

Equipped with an understanding of the operating instructions described in Part I, Chapter 1, you can call the subroutines detailed in this part of the Guide.

There are 24 chapters of calls to runtime routines. Within each chapter an alphabetical list of the calls precedes their descriptions. We then present the routines.

Where appropriate, we introduce the chapter with a functional discussion of the subroutines. In these introductions we direct you to supplementary references.

Conventions

In Part II, we set up all of the runtime routines in a similar fashion. You will first encounter the routine's name. Equivalent routine names or aliases will sometimes follow. (You can use these names interchangeably when calling the routine.) Next we explain the purpose of each routine. Then we document the format that you must use to call the routine. The format specifies the order in which you must enter the arguments.

We explain the arguments of the calling sequence. Various Instrument Society of America (ISA) and documentation conventions characterize these arguments.

In the following routine descriptions, we use two sections, *Rules* and *Notes,* sparingly. *Rules* is a section reserved for items essential to you for calling the routine, whereas *Notes* further clarifies a routine's purpose.

The final section of the routine descriptions is *Reference.* We document a system reference by stating the RDOS-defined name of the call.

Aggregates

As we detail the arguments, we often employ the terms, aggregate and integer.

The type of argument you pass to a subprogram must match the type that the subprogram expects. There are situations where the argument type is irrelevant to the subprogram. We refer to an argument as an aggregate when the subprogram treats the argument merely as a sequence of contiguous words or bytes. In this case, we do not specify its type.

An aggregate's use pertains to either a source or a destination of data. It may be of any length.

If you provide an aggregate with a string of ASCII characters, you must insure that a null terminates the string.

When a routine returns information in an aggregate, you must provide an aggregate large enough to contain all returned data. Overflowing an aggregate produces disastrous results.

Integer

An integer appears as a constant, variable, array element, or expression. If we don't specify the type of integer, assume it takes on any of its legal forms outlined in the ANSI standard.

The ier Argument

The last argument for many of the routines is ier, which indicates an integer status variable. ier receives a numeric status code. ISA defines these codes as follows:

Negative or 0	Undefined.
1	No error, successful completion.
2	Currently unused.
3 and up	An error occurred. Look up the (decimal) error code in F5ERR.FR (see Appendix A).

You can incorporate all of the file in Appendix A, F5ERR.FR, or some of its error definitions into your program with the INCLUDE statement.

If an error occurs in a routine without ier as its last argument, the FORTRAN 5 runtime error reporter will process the error and terminate your program.

For details on error-handling in FORTRAN 5, read Chapter 2 in Part I.

Error Conditions

In each presentation of a runtime routine, we list the error conditions by name and description.

You can refer to the error code values listed in the file, F5ERR.FR, which is listed in Appendix A. Mnemonic names beginning with E indicate system errors; those beginning with F indicate FORTRAN 5 errors.

End of Chapter

Chapter 2
Checking for Arithmetic Errors

The following is an alphabetical list of the runtime routines you will find in this chapter.

DVDCHK	Determines whether or not a floating point divide-by-zero occurred since the last call to DVDCHK or since the start of the program.	OVERFL	Determines whether or not a floating point overflow or underflow occurred since the last call to OVERFL or since the start of the program.

DVDCHK

determines whether or not a floating point divide-by-zero has occurred since the last call to DVDCHK or since the start of the program.

Format

CALL DVDCHK (code)

Argument

code is an integer variable that receives one of the following:

1 If a division-by-zero occurred.
2 If a division-by-zero didn't occur.

Error Conditions

No error conditions are currently defined.

Example

```
CALL DVDCHK (ICODE)
;BRANCH IF DIVIDE-BY-ZERO OCCURRED
IF (ICODE .EQ. 1) GO TO 99
```

OVERFL

determines whether or not a floating point overflow or underflow has occurred since the last call to OVERFL or since the start of the program.

Format

CALL OVERFL (code)

Argument

code is an integer variable that receives one of the following:

1 If overflow occurred.
2 If neither overflow nor underflow occurred.
3 If underflow occurred, but overflow didn't.

If both overflow and underflow occurred, overflow is signaled (code 1).

Error Conditions

No error conditions are currently defined.

Examples

```
CALL OVERFL (J)
;BRANCH IF OVERFLOW OR UNDERFLOW OCCURRED
IF (J .NE. 2) GO TO 80
```

End of Chapter

Chapter 3
Performing Logical Operations with Integers and Words

Each of the runtime routines in this chapter permits manipulation of or access to the bits of an integer or a word. The arguments of the routines are 16-bit integers (sign is ignored) and each routine returns an integer value to the function reference that invoked it.

The bit numbering scheme in the runtime routines, ICLR, ISET, and ITEST, is 0 for the least significant bit (the rightmost) and 15 for the most significant bit (the leftmost).

For more information on logical operations with integers and words, see Chapter 8 in the *FORTRAN 5 Reference Manual.*

The following is an alphabetical list of all runtime routines you will find in this chapter:

IAND	Produces the logical AND of two integers.
ICLR	Clears a bit in a word.
IEOR	Alias for IXOR.
IOR	Produces the logical inclusive OR of two integers.
ISET	Sets a bit in a word.
ISHFT	Alias for ISHIFT.
ISHIFT	Shifts an integer.
ITEST	Tests a bit in a word.
IXOR	Produces the logical exclusive OR of two integers.
NOT	Produces the logical complement of an integer.

IAND

produces the bit-by-bit AND of two integers.

Format

```
IAND (int1,int2)
```

Arguments

int1 is the first integer operand.

int2 is the second integer operand.

Error Conditions

No error conditions are currently defined.

Examples

1.
```
I = IAND (I,J)
```

2.
```
;CHECK FOR A NULL RIGHT BYTE
IF (IAND (J,377K) .EQ. 0) GO TO 50
```

ICLR

clears a bit in a word.

Format

```
CALL ICLR (word,bit)
```

Arguments

word is a one-word aggregate that contains the bit you want cleared.

bit is an integer that specifies the position of the bit you want to clear (bits are numbered from 0, the rightmost, to 15, the leftmost).

Error Conditions

No error conditions are currently defined.

Example

```
CALL ICLR (K,3)
```

Notes

If you issue a call to ICLR giving a bit that is outside the legal range, the operation isn't performed but no error is signaled. When ICLR clears a bit, it sets it to zero.

IOR

produces the bit-by-bit inclusive OR of two integers.

Format

IOR (int1,int2)

Arguments

int1 is the first integer operand.

int2 is the second integer operand.

Error Conditions

No error conditions are currently defined.

Examples

1. K = IOR (J,IDEF)

2. IF (IOR (M,J) .NE. 1) GO TO 100

ISET

sets a bit in a word.

Format

CALL ISET (word,bit)

Arguments

word is a one-word aggregate that contains the bit you want set.

bit is an integer that specifies the position of the bit you want to set (bits are numbered from 0, the rightmost, to 15, the leftmost).

Error Conditions

No error conditions are currently defined.

Example

CALL ISET (I,3)

Notes

If you issue a call to ISET giving a bit that is outside the legal range, the operation isn't performed, but no error is signaled.

ISHIFT (Alias: ISHFT)
shifts an integer.

Format

ISHIFT (integer,count)

Arguments

integer is the integer you want to shift.

count is an integer (n) that specifies the number and the direction of bits you want to shift.

Given integer n:

n=0	no shift.
n>0	shift left n bits, bringing in zeros from the right.
n<0	shift right n bits, bringing in zeros from the left.

Error Conditions

No error conditions are currently defined.

Examples

1.
```
INEW = ISHIFT (IOLD,-5)
```

2.
```
;CHECK FOR A NULL LEFT BYTE
IF (ISHIFT (J,-8) .EQ. 0) GO TO 60
```

ITEST
tests a bit in a word.

Format

ITEST (word,bit)

Arguments

word is the integer you want to test.

bit is an integer that specifies the position of the bit you want to test (bits are numbered from 0, the rightmost, to 15, the leftmost).

Error Conditions

No error conditions are currently defined.

Examples

1.
```
J = ITEST (I,K)
```

2.
```
LOGICAL ITEST
      .
      .
;PASS CONTROL TO STMNT LBL 70
IF BIT 3 OF J IS SET
IF (ITEST (J,3)) GO TO 70
```

Notes

If you issue a call to ITEST giving a bit that is outside the legal range, the result is as if the bit were zero.

If you declare ITEST to be LOGICAL, the result is .TRUE. if the bit is set (1) and .FALSE. if it's clear (0).

IXOR (Alias: IEOR)

produces the bit-by-bit exclusive OR of two integers.

Format

IXOR (int1,int2)

Arguments

int1 is the first integer operand.

int2 is the second integer operand.

Error Conditions

No error conditions are currently defined.

Examples

1. IOTHR = IXOR (IONE,ITWO)

2. IF (IXOR (I,J) .EQ. 1) GO TO 200

NOT

produces the logical complement of an integer.

Format

NOT (integer)

Argument

integer is the integer you want to complement.

Error Conditions

No error conditions are currently defined.

Examples

1. ICAN = NOT (IMAY)

2. ;CHECK FOR ALL 1 BITS IN J
   IF (NOT (J) .EQ. 0) GO TO 40

End of Chapter

Chapter 4
Managing Directories and Devices

RDOS can simultaneously manage multiple directory devices. The system identifies each device by a name; you must initialize the device using its name before you can access it. Call the FORTRAN 5 runtime routine, INIT, for device initialization.

If you want to access a file that isn't in the current default directory, you may issue a call to runtime routine, DIR, to make the appropriate directory the new default directory. DIR also initializes the directory if you haven't already done so.

To prevent further I/O activities in a directory, magnetic tape, or cassette unit, issue a call to the runtime routine RELEASE. If you want to release a removable device or close any open files, then issue the call, RELEASE, before you physically remove the device from the unit. The release of a master directory releases all directories. (The master directory is that directory specified in a BOOT command or in a response to the HIPBOOT query; you can obtain its name by using the runtime routine MDIR.) The release of a primary partition releases all directories within that partition. The release of a secondary partition releases all subdirectories within that partition.

The following is an alphabetical list of all runtime routines you will find in this chapter:

CDIR	Creates a subdirectory.
CPART	Creates a secondary partition.
DIR	Changes the current default directory.
EQUIV	Assigns a temporary name to a global specifier.
GDIR	Obtains the current default directory name.
GSYS	Obtains the current system name.
INIT	Initializes a directory, partition, or device.
MDIR	Obtains the current master device name.
RELEASE	Releases a directory, partition, or device from current use.
RLSE	Alias for RELEASE.

CDIR
creates a subdirectory.

Format

```
CALL CDIR (subdirectory name,ier)
```

Arguments

subdirectory name is an aggregate that contains the name of the subdirectory you want to create.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal subdirectory name.
ERCRE	Attempt to create an existing subdirectory.
ERDSN	Directory specifier unknown.
ERDDE	Attempt to create a subdirectory in a subdirectory.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.

Example

```
CALL CDIR ("NEWSTUFF",IER)
CALL CHECK (IER)
```

Reference

.CDIR (System call)

CPART

creates a secondary partition.

Format

CALL CPART (partition name,size,ier)

Arguments

partition name is an aggregate that contains the name of the secondary partition you want to create.

size is an integer that contains the number of contiguous disk blocks you want to allocate to the secondary partition.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal secondary partition name.
ERCRE	Attempt to create an existing secondary partition.
ERICB	Insufficient number of free contiguous disk blocks available.
ERDSN	Directory specifier unknown.
ERD2S	Directory too small (must have at least 48 (decimal) blocks).
ERDDE	Attempt to create a secondary partition in a secondary partition (i.e., a tertiary partition).
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.

Example

```
CALL CPART ("IPART1",80,IER)
CALL CHECK (IER)
```

Reference

.CPAR (System call)

DIR

changes the current default directory.

Format

CALL DIR (directory name,ier)

Arguments

directory name is an aggregate that contains the name of the new default directory.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERDLE	Directory doesn't exist.
ERDNM	Device or directory not in system.
ERNMD	Insufficient number of extra buffers specified at SYSGEN time.
ERIDS	Illegal directory specifier.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
EROVF	System stack overflow due to excessive number of chained directory specifiers.

Example

```
CALL DIR ("USMAN",IER)
CALL CHECK (IER)
```

Reference

.DIR (System call)

EQUIV

assigns a temporary name to a global specifier (device name), permitting unit independence during the execution of your program.

Format

```
CALL EQUIV (new name,old name,ier)
```

Arguments

new name is an aggregate that contains a new, temporary name for a global specifier.

old name is an aggregate that contains the current name of the global specifier.

ier is an integer variable that receives the routine's completion status code.

Rules

You must issue a call to EQUIV before you initialize the device.

You can't assign a temporary name to a master device.

Error Conditions

Error codes that may return in ier include:

ERDSN	Directory specifier unknown.
ERDIU	Device in use (i.e., already initialized).
ERMPR	Address is outside address space (mapped systems only).

Example

```
CALL EQUIV ("QDISK","DP3",IER)
CALL CHECK (IER)
CALL INIT ("QDISK",IER)
CALL CHECK (IER)
```

Notes

Temporary name assignments persist until you reboot the system, release the device, or assign a new temporary name.

Reference

.EQIV (System call)

GDIR

obtains the current default directory name.

Format

```
CALL GDIR (directory name,ier)
```

Arguments

directory name is an aggregate that receives the default directory name.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERRD	Attempted system overwrite.
ERMPR	Address is outside address space (mapped systems only).

Example

```
INTEGER CURDIR(6)
      .
      .
CALL GDIR (CURDIR,IER)
CALL CHECK (IER)
```

Reference

.GDIR (System call)

GSYS

obtains the current system name.

Format

CALL GSYS (system name,ier)

Arguments

system name is an aggregate that receives the current system name.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERRD	Attempted system overwrite.
ERMPR	Address is outside address space (mapped systems only).

Example

```
INTEGER SYSNM(7)
      .
      .
CALL GSYS (SYSNM,IER)
CALL CHECK (IER)
```

Reference

.GSYS (System call)

INIT

initializes a directory, partition, or device.

Format

CALL INIT (directory name,ier)

Arguments

directory name is an aggregate that contains the name of a directory, partition, or device.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERWRP	Device is write-protected.
ERDLE	Directory doesn't exist.
ERSEL	Unit improperly selected.
ERDNM	Device not in system.
ERIBS	Device already initialized.
ERNMD	Insufficient number of extra buffers specified at SYSGEN time.
ERIDS	Illegal directory specifier.
ERLDE	Link depth exceeded.
ERMPR	Address is outside address space (mapped systems only).
ERSDE	Error detected in SYS.DR of nonmaster device.
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).
EROVF	System stack overflow due to excess number of chained directory specifiers; this can occur only when you use links in the specifier string.

Example

```
CALL INIT ("DP0",IER)
CALL CHECK (IER)
```

Reference

.INIT (System call)

MDIR

obtains the current master device name.

Format

CALL MDIR (device name,ier)

Arguments

device name is an aggregate that receives the name of the current master device.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERRD	Attempted system overwrite.
ERMPR	Address is outside address space (mapped systems only).

Examples

```
INTEGER MSTDEV(6)
      .
      .
CALL MDIR (MSTDEV,IER)
CALL CHECK (IER)
```

Reference

.MDIR (System call)

RELEASE (Alias: RLSE)

releases a directory, partition, or device from current use.

Format

CALL RELEASE (directory name,ier)

Arguments

directory name is an aggregate that contains the name of a directory, partition, or device.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERSEL	Unit improperly selected.
ERDNM	Device not in system.
ERDIU	Directory in use.
ERDNI	Directory not initialized.
EROPD	Released directory in use by other program.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERNIR	Attempted release of a unit containing an open file.

Example

```
CALL RELEASE ("MT1",IER)
CALL CHECK (IER)
```

Reference

.RLSE (System call)

End of Chapter

Chapter 5
Maintaining Files

The following is an alphabetical list of all runtime routines you will find in this chapter:

CFIL	Alias for CFILW.
CFILW	Creates a disk file.
CHATR	Changes the attributes of an opened file.
CHLAT	Changes link attributes of an opened file.
CHSTS	Obtains the status for an opened unit (file) in the current directory.
DFIL	Alias for DFILW.
DFILW	Deletes an unopened disk file.
FDELETE	Deletes an unopened disk file.
.FIOPREP	Obtains an RDOS channel number corresponding to a FORTRAN 5 unit number using assembly language.
FRENAME	Renames an unopened disk file.
GTATR	Obtains the attributes of an opened file.
LINK	Creates a link entry in the current directory.
RENAME	Renames an unopened disk file.
RSTAT	Gets the status information for a file in the current directory. If the file is a link entry, the information returned describes the resolution file.
STAT	Gets the status information for a file in the current directory. If the file is a link entry, the information returned tells to which file the entry is linked.
UNLINK	Deletes a link entry.
UPDATE	Updates a file's directory entry on disk.

CFILW (Alias: CFIL)
creates a disk file.

Format

CALL CFILW (filename,file type, *[size,]* ier)

Arguments

filename is an aggregate that contains the name of the disk file to create.

file type has one of the following integer values:

1 sequential file
2 random file
3 contiguous file

size is an integer that specifies the number of 256-word blocks you want to allocate. This argument is meaningful only for contiguous files; it's ignored for other types.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERCRE	Attempt to create an existing file.
ERICB	Insufficient number of free contiguous disk blocks available (contiguous file only).
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.

Example

```
CALL CFILW ("FILNM.DC",1,IER)
CALL CHECK (IER)
```

References

.CREATE	(System call) - If the file is to be organized sequentially.
.CRAND	(System call) - If the file is to be organized randomly.
.CCONT	(System call) - If the file is to be organized contiguously.

CHATR
changes the attributes of an opened file.

Format

CALL CHATR (unit number,attributes,ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the file whose attributes are to be changed.

attributes is an integer that specifies the new attributes.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERCHA	Illegal attempt to change file attributes.
ERFOP	Attempt to change attributes of an unopened file.
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
CALL OPEN (3,"MATH.DC",2,IER)
CALL CHECK (IER)
        .
        .
        .
;MAKE "MATH.DC" PERMANENT
CALL CHATR (3,10K,IER)
CALL CHECK (IER)
```

Reference

.CHATR (System call)

CHLAT
changes link attributes of an opened file.

Format

CALL CHLAT (unit number,attributes,ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the link file whose attributes are to be changed.

attributes is an integer that specifies the new attributes.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNO	Illegal channel number.
ERCHA	Resolution entry is attribute protected.
ERFOP	Attempt to change attributes of an unopened file.
ERDTO	Ten second disk time-out occurred.

Example

```
CALL OPEN (3,"MATH.DC",2,IER)
CALL CHECK (IER)
        .
        .
        .
;MAKE "MATH.DC" PERMANENT
;FOR ANYONE LINKING TO IT
CALL CHLAT (3,10K,IER)
CALL CHECK (IER)
```

Reference

.CHLAT (System call)

CHSTS

obtains the status for an opened FORTRAN unit (file) in the current directory.

Format

CALL CHSTS (unit number,status,ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number.

status is an aggregate that receives 18 words of channel status information.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFOP	No file opened on the given channel.
ERRD	Attempted system overwrite.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
INTEGER ISTAT (18)
      .
      .
CALL CHSTS (3,ISTAT,IER)
CALL CHECK (IER)
```

Notes

The directory information is updated (the equivalent of CALL UPDATE) before its status is reported.

References

.UPDAT	(System call) - To update the current file size.
.CHSTS	(System call)

DFILW (Alias: DFIL)

deletes an unopened disk file.

Format

CALL DFILW (filename,ier)

Arguments

filename is an aggregate that contains the name of the disk file you want to delete.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERDLE	Attempt to delete a nonexisting file.
ERDE1	Attempt to delete a permanent file.
ERDSN	Directory specifier unknown.
ERDIU	Directory in use.
ERLDE	Link depth exceeded.
ERFIU	File in use.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERMDE	Error detected in MAP.DR of nonmaster device.
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).

Example

```
CALL DFILW ("TEST.SV",IER)
CALL CHECK (IER)
```

Reference

.DELET	(System call)

FDELETE

deletes an unopened disk file.

Format

```
CALL FDELETE (filename)
```

Argument

filename is an aggregate that contains the name of the disk file you want to delete.

Error Conditions

The error conditions that may result are:

ERFNM	Illegal filename.
ERDLE	Attempt to delete a nonexisting file.
ERDE1	Attempt to delete a permanent file.
ERDSN	Directory specifier unknown.
ERDIU	Directory in use.
ERLDE	Link depth exceeded.
ERFIU	File in use.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERMDE	Error detected in MAP.DR of nonmaster device.
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).

Example

```
CALL FDELETE ("TEST.SV")
```

Notes

The error conditions given above cause program termination when they occur.

Reference

.DELET	(System call)

.FIOPREP

obtains an RDOS channel number corresponding to a FORTRAN 5 unit number using assembly language.

Format

```
(AC1 = FORTRAN unit number)
JSR @.FIOPREP
error return
normal return
```

Returned Information

AC1	FORTRAN 5 unit number
AC2	RDOS channel number
AC3	frame pointer

Error Return

AC2	error code

Error Conditions

The error conditions that may result are:

ERFOP	Attempt to refer to an unopened unit.
FEIFN	Illegal unit number.

Example

```
;GET UNIT NUMBER
LDA 1,C3
        .
        .
        .
JSR @.FIOPREP
;RETURN ERROR CODE IN LAST
;ARGUMENT TO THIS ROUTINE
ISA.ERR
        .
        .
        .
        C3: 3
```

Notes

AC0, AC1, and the carry remain unchanged.

The values of system error names are different when used in assembly language, but the meanings are the same as in a FORTRAN 5 program.

FRENAME

renames an unopened disk file.

Format

```
CALL FRENAME (old name,new name)
```

Arguments

old name is an aggregate that contains the current name of the disk file you want to rename.

new name is an aggregate that contains the new name of the disk file.

Error Conditions

The error conditions that may result are:

ERFNM	Illegal filename.
ERCRE	Attempt to create an existing file.
ERDLE	Attempt to rename a nonexisting file.
ERDE1	Attempt to rename a permanent file.
ERDIR	Files specified on different directories.
ERDSN	Directory specifier unknown.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERFIU	File in use.

Example

```
CALL FRENAME ("MATH.DC","CHAP4.DC")
```

Notes

The error conditions given above cause abnormal termination of your program when they occur.

Reference

.RENAM (System call)

GTATR

obtains the attributes of an opened file.

Format

```
CALL GTATR (unit number,attributes,
  device characteristics,ier)
```

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of an opened file.

attributes is a 1-word aggregate that receives the file's attributes.

device characteristics is a 1-word aggregate that receives the file's device characteristics.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFOP	Attempt to get attributes of an unopened file.
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
INTEGER ATTS,DEVCH
        .
        .
CALL GTATR (1,ATTS,DEVCH,IER)
CALL CHECK (IER)
```

Reference

.GTATR (System call)

LINK

creates a link entry in the current directory.

Format

CALL LINK (linkname, *[filename,]* ier)

Arguments

linkname is an aggregate that contains the name of the link entry you want to create.

filename is an aggregate that contains the name of the file onto which you want to link.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERCRE	Link entry name already exists.
ERDSN	Directory specifier unknown.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).

Example

```
CALL LINK ("SYS.SV","ASYS.SV",IER)
CALL CHECK (IER)
```

Reference

.LINK (System call)

RENAME

renames an unopened disk file.

Format

CALL RENAME (old name,new name,ier)

Arguments

old name is an aggregate that contains the current name of the disk file you want to rename.

new name is an aggregate that contains the new name of the disk file.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERCRE	Attempt to create an existing file.
ERDLE	Attempt to rename a nonexisting file.
ERDE1	Attempt to rename a permanent file.
ERDIR	Files specified on different directories.
ERDSN	Directory specifier unknown.
ERFIU	File in use.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.

Example

```
CALL RENAME ("MATH.DC","CHAP4.DC",IER)
CALL CHECK (IER)
```

Reference

.RENAM (System call)

RSTAT

gets the current resolution file directory information for a file, opened or not.

Format

CALL RSTAT (filename,status,ier)

Arguments

filename is an aggregate that contains the name of the file whose information is to be obtained.

status is an aggregate that receives 18 words of file directory information.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERDLE	File doesn't exist.
ERRD	Attempted system overwrite.
ERDNM	Device not in system.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.

Example

```
INTEGER STAT (18)
      .
      .
CALL RSTAT ("AMFIL.SV",STAT,IER)
CALL CHECK (IER)
```

Reference

.RSTAT (System call)

STAT

gets the status information for a file, opened or not, in the current directory.

Format

CALL STAT (filename,status,ier)

Arguments

filename is an aggregate that contains the name of the file whose information is to be obtained.

status is an aggregate that receives 18 words of file information.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERDLE	File doesn't exist.
ERRD	Attempted system overwrite.
ERDNM	Device not in system.
ERDSN	Directory specifier unknown.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.

Example

```
INTEGER FILUFD (18)
      .
      .
CALL STAT ("IOLINK",FILUFD,IER)
CALL CHECK (IER)
```

Reference

.STAT (System call)

UNLINK

deletes a link entry.

Format

CALL UNLINK (linkname,ier)

Arguments

linkname is an aggregate that contains the name of the link entry you want to delete.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERDLE	Attempt to delete a nonexisting file.
ERDSN	Directory specifier unknown.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERNLE	Not a link entry.
ERDTO	Ten second disk time-out occurred.

Example

```
CALL UNLINK ("SYS.SV",IER)
CALL CHECK (IER)
```

Reference

.ULNK (System call)

UPDATE

updates a file's directory entry on disk. It ensures that the directory contains the current length of the file.

Format

CALL UPDATE (unit number,ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the file whose directory entry is to be updated.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFOP	File not opened.
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
CALL UPDATE (13,IER)
CALL CHECK (IER)
```

Reference

.UPDAT (System call)

End of Chapter

Chapter 6
File Input/Output

You have several options when opening, closing, reading, and writing files.

Before you issue any I/O commands concerning a file, you must connect a FORTRAN 5 unit number to that file or device by opening the file. You may open the file by calling one of the following runtime routines: OPEN, FOPEN, APPEND, or MTOPD. You may also use the OPEN statement to open a file.

You can disconnect a FORTRAN 5 unit number from a file by issuing a call to CLOSE or FCLOSE; the CLOSE statement is another option that you may use. To close all open files, use RESET.

There are five modes in which you can read and write files. The FORTRAN 5 runtime libraries provide you with runtime routine calls that correspond to these modes.

- Line Mode - Data consists of an ASCII character string terminated either by a carriage return, form feed, or null character. (Use runtime routines RDLIN and WRLIN.)

- Sequential Mode - Data is unedited; it's transmitted exactly as read or written from a file or a device. (Use runtime routines RDSEQ and WRSEQ.)

- Random Record Mode - Data consists of fixed length records within a disk file and is accessed randomly. (Use runtime routines READRW (READR, RDRW) and WRITRW (WRITR).)

- Direct Block Mode - Data exists in a contiguous group of blocks in a random or contiguous file and is transmitted without the use of an intervening system buffer. Only entire blocks of disk space may participate and, if more than one block is involved, you may transfer only an unbroken sequence of blocks. (Use runtime routines RDBLK and WRBLK.)

- Mag-Tape Direct Mode - Data is read or written on a block basis to cassette or magnetic tape. (Use runtime routine MTDIO.)

For more information on I/O commands, see the *RDOS Reference Manual.*

The following is an alphabetical list of all runtime routines you will find in this chapter:

APPEND	Opens a file for appending.
BACKSPACE	Backspaces a file to the previous logical record.
CHRST	Restores the position of a file saved by a call to CHSAV.
CHSAV	Saves the position of a file.
CLOSE	Closes a file.
FBCKSP	Alias for BACKSPACE.
FCLOSE	Closes a file.
FOPEN	Opens a file.
FRWND	Alias for REWIND.
MTDIO	Performs free-form I/O to or from a magnetic tape or cassette unit.
MTOPD	Opens a magnetic tape or cassette unit for free-form I/O.
OPEN	Opens a file.
RDBLK	Reads a series of blocks from a randomly or contiguously organized disk file.
RDLIN	Reads a line in a file.
RDRW	Alias for READRW.
RDSEQ	Performs a sequential read.
READR	Alias for READRW.
READRW	Reads a series of records from a file.
RESET	Closes all open files.
REWIND	Rewinds a file.
WRBLK	Writes a series of blocks to a randomly or contiguously organized disk file.
WRITE	Alias for WRITRW.
WRITRW	Writes a series of records to a file.
WRLIN	Writes a line to a file.
WRSEQ	Performs a sequential write.

In reference to the three calls, OPEN, FOPEN, and APPEND, OPEN lets you control access to a file and set a record length. But FOPEN does not allow this control. APPEND opens a file for appending information and, like the OPEN statement, lets you set a record length.

Files opened in these ways do not respond to carriage control in FORMAT statements. You must either use the OPEN statement (with ATT="P") to allow carriage control, or use default opening.

APPEND

opens a file for appending information; that is, to position to the end of a file so that any WRITEing will be done after the existing data in the file.

Format

CALL APPEND (unit number,filename,mode,
[record size,] ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the file you want to open.

filename is an aggregate that contains the name of the file.

mode is unconditionally ignored. It is included so that the argument list is identical to that of the OPEN routine.

record size is an integer that specifies the size of a record in bytes.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERICD	Illegal command for device.
ERDLE	File doesn't exist.
ERUFT	No channels are free (returned by .GCHN).
ERSEL	Unit improperly selected.
ERDNM	Device not in system.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERFIU	File in use.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).
ERDOP	Attempt to open a file that is already open.
FEIFN	Illegal unit number.

Example

```
CALL APPEND (13,"LOGOFL",DUMMY,IER)
```

Notes

The call to APPEND is identical to the call to OPEN except:

1. APPEND allows you to open and extend a file that already exists.

2. The mode argument is ignored in the call to APPEND, because you open the file specifically for appending.

Reference

.APPEND (System call)

BACKSPACE (Alias: FBCKSP)

backspaces a file to the previous record.

Format

CALL BACKSPACE (unit number)

Argument

unit number is an integer that specifies the FORTRAN 5 unit number of the file you want backspaced.

Error Conditions

The error conditions that may result are:

ERFOP	Attempt to refer to an unopened unit.
ERSCP	File position error.
FEIFN	Illegal unit number.
FESEK	Record-oriented file required for backspace.

Example

```
CALL BACKSPACE (3)
```

References

.GPOS	(System call) - Obtains the system file pointer, which is the next character position where system processing occurs.
.SPOS	(System call) - Sets the system file pointer to a given value, i.e., back one record.

CHRST

restores the position of a file saved by a call to CHSAV.

Format

CALL CHRST (unit number,position)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number.

position is a 2-word aggregate that contains the position of the file when saved by a call to CHSAV.

Error Conditions

The error conditions that may result are:

ERFOP	Attempt to refer to an unopened unit.
ERSCP	File position error.
FEIFN	Illegal unit number.

Example

```
INTEGER POSTN (2)
      .
      .
CALL CHRST (2,POSTN)
```

Reference

.SPOS	(System call)

CHSAV
saves the position of a file.

Format

CALL CHSAV (unit number,position)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number.

position is a 2-word aggregate that receives the current position of the specified file.

Error Conditions

The error conditions that may result are:

ERFOP	Attempt to refer to an unopened unit.
FEIFN	Illegal unit number.

Example

```
INTEGER POSTN (2)
      .
      .
CALL CHSAV (2,POSTN)
```

Reference

.GPOS (System call)

CLOSE
closes a file.

Format

CALL CLOSE (unit number,ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the file you want closed.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFOP	Attempt to refer to an unopened unit.
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
CALL CLOSE (7,IER)
```

Notes

If you prepared this file for printing (ATT="P") when you OPENed it, FORTRAN 5 will append a line feed and a carriage return to the last line of the file before closing the file.

Reference

.CLOSE (System call)

FCLOSE
closes a file.

Format

CALL FCLOSE (unit number)

Argument

unit number is an integer that specifies the FORTRAN 5 unit number of the file you want closed.

Error Conditions

The error conditions that may result are:

ERFOP	Attempt to refer to an unopened unit.
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
CALL FCLOSE(2)
```

Notes

If you prepared this file for printing (ATT="P") when you OPENed it, FORTRAN 5 will append a line feed and a carriage return to the last line of the file before closing the file.

Reference

.CLOSE (System call)

FOPEN
opens a file.

Format

CALL FOPEN (unit number,filename)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the file you want opened.

filename is an aggregate that contains the name of the file.

Error Conditions

The error conditions that may result are:

ERFNM	Illegal filename.
ERUFT	No channels are free.
ERSEL	Unit improperly selected.
ERDNM	Device not in system.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERFIU	File in use due to exclusive OPEN.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERDOP	Attempt to open a tape file which is already open.
FEIFN	Illegal unit number.

Example

```
CALL FOPEN (6, "INPUT")
```

Notes

If the named file doesn't exist, a randomly organized file is created.

Reference

.OPEN (System call)

MTDIO

performs free-form I/O to or from a magnetic tape or cassette unit.

Format

CALL MTDIO (unit number,command,data array,
status word, *[count,]* ier)

Arguments

unit number is an integer that specifies the number of a unit opened for direct I/O.

command is a one-word aggregate that contains the command the unit will perform.

data array is an aggregate that receives or contains data.

status word is an integer variable that receives a code describing the status of the hardware if the return is normal; however, if a software error occurs, this argument receives a zero. The *RDOS Reference Manual* further defines these codes.

count is an integer variable that receives the number of records spaced, read, or written (depends on command); this count is returned upon a premature end-of-file condition.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device (i.e., improper open).
ERFOP	Attempt to refer to an unopened unit.
ERSEL	Unit improperly selected (check status word).
EROVA	File not accessible by free-form I/O.
ERMPR	Address is outside address space (mapped systems only).
FEIFN	Illegal unit number.

Example

```
INTEGER ARAY1 (100)
      .
      .
;WRITE 100 WORDS WITH EVEN PARITY
CALL MTDIO (3,150144K,ARAY1,ISTAT,ICOUNT,IER)
CALL CHECK (IER)
```

Reference

.MTDIO (System call)

MTOPD

opens a magnetic tape or cassette unit for free-form I/O.

Format

CALL MTOPD (unit number,filename,mask,ier)

Arguments

unit number is an integer that specifies the number of the unit you want opened.

filename is an aggregate that contains the name of the file you want to open.

mask is a 1-word aggregate that contains the device characteristics you want to inhibit for the duration of the open.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
ERDLE	File doesn't exist.
ERUFT	Attempt to use a channel already in use.
ERSEL	Unit improperly selected.
ERDNM	Device not in system.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDOP	Attempt to open a file that is already open.
FEIFN	Illegal unit number.

Example

```
CALL MTOPD (4,"MT0:3",0,IER)
CALL CHECK (IER)
```

Reference

.MTOPD (System call)

OPEN

opens a file.

Format

CALL OPEN (unit number,filename,mode,
[record size,] ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the file you want opened.

filename is an aggregate that contains the name of the file.

mode is an integer with one of the following values:

0	append
1	read only
2	shared
3	exclusive write access
4	exclusive write access

record size is an integer that specifies the size of a record in bytes.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERDLE	File doesn't exist.
ERUFT	No channels are free (returned by .GCHN).
ERSEL	Unit improperly selected.
ERDNM	Device not in system.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERFIU	File in use due to .EOPEN.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).
ERDOP	Attempt to open a tape file which is already open.
FEIFN	Illegal unit number.
FEFNU	Unit number in use.
FEMOP	Illegal mode for open.

Example

```
;OPEN FOR READING ONLY
CALL OPEN (6,"MYFILE",1,IER)
;OPEN FOR APPENDING WITH 20-BYTE RECORDS
CALL OPEN (13,"OUTPUT",0,20,IER)
CALL CHECK (IER)
```

Notes

If you OPEN a file in exclusive write mode, no other user can access that file for any purpose except reading. If the file does not exist, it will not be created, but the error, ERDLE, will return.

References

.OPEN	(System call) - For shared mode.
.ROPEN	(System call) - For read-only mode.
.EOPEN	(System call) - For exclusive write mode.

RDBLK

reads a series of blocks from a randomly or contiguously organized disk file.

Format

CALL RDBLK (unit number,starting block,data array, count, *[returned count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the randomly or contiguously organized disk file.

starting block is an integer that specifies the block with which to start reading (the first block is numbered 0).

data array is an aggregate that receives the data read.

count is an integer that specifies the total number of blocks you want to read.

returned count is an integer variable that receives the total number of blocks successfully read only when an end-of-file condition is encountered; otherwise, this argument isn't set.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
ERSV1	Not a randomly or contiguously organized file.
EREOF	End of file (for contiguous files only).
ERRPR	File is read protected.
ERFOP	Attempt to refer to an unopened unit.
ERSPC	Disk space is exhausted.
ERFIL	File read error.
ERRD	Attempt to read into system area.
EROVA	File not accessible by free-form I/O.
ERMPR	Address is outside address space (for mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
INTEGER RDARAY (1024)
        .
        .
OPEN 7,"DATA"
        .
        .
;READ BLOCKS #5 THRU #8 INTO RDARAY
CALL RDBLK (7,5,RDARAY,4,IER)
CALL CHECK (IER)
```

Reference

.RDB (System call)

RDLIN
reads a line from a file.

Format

CALL RDLIN (unit number,data array, *[returned count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number from which you want data read.

data array is an aggregate that receives the line read; it should generally be 133 bytes long to accommodate the longest possible line.

returned count is an integer variable that receives the number of bytes read (including the terminator).

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
EREOF	End of file.
ERRPR	Attempt to read a read-protected file.
ERFOP	Attempt to refer to an unopened unit.
ERLLI	Line limit (132 nonterminator characters) exceeded.
ERPAR	Parity error.
ERFIL	File read error.
ERRD	Attempt to read into system area.
ERDIO	File accessible by direct block I/O only.
ERSIM	Simultaneous reads on same QTY line.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERCLO	QTY/MCA input terminated by channel close.
FEIFN	Illegal unit number.

Example

```
INTEGER ARAYD(67)
      .
      .
CALL RDLIN (5,ARAYD,ICNT,IER)
;SIGNAL ANY ERROR
CALL CHECK (IER)
;BRANCH IF TERMINATOR IS THE ONLY CHARACTER
IF (ICNT .EQ. 1) GO TO 15
```

Notes

Reading from a terminal results in typed characters echoing. RDSEQ will not echo the typed characters.

Reference

.RDL (System call)

RDSEQ
performs a sequential read.

Format

CALL RDSEQ (unit number,data array,count,
[returned count,] ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number from which you want data read.

data array is an aggregate that receives the data read.

count is an integer that specifies the number of bytes you want read.

returned count is an integer variable that receives the partial read count in bytes only when an end-of-file condition is encountered; otherwise, this argument isn't set.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
EREOF	End of file.
ERRPR	Attempt to read a read-protected file.
ERFOP	Attempt to refer to an unopened unit.
ERFIL	File read error.
ERRD	Attempt to read into system area.
ERDIO	File accessible by direct block I/O only.
ERSIM	Simultaneous reads on same QTY line.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERMCA	Attempt to perform an MCA read on a channel where reading is currently in progress.
ERCLO	QTY/MCA input terminated by channel close.
FEIFN	Illegal unit number.

Example

```
INTEGER AREA20(20)
      .
      .
CALL RDSEQ (FILNBR,AREA20,120,ICNT,IER)
CALL CHECK (IER)
```

Notes

RDSEQ does not echo characters read from terminals.

Reference

.RDS (System call)

READRW (Aliases: READR, RDRW)
reads a series of records from a file.

Format

CALL READRW (unit number,starting record,
data array,count, *[returned count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number from which you want data read.

starting record is an integer that specifies the number of the first record you want read (the first record of a FORTRAN 5 file is numbered 1).

data array is an aggregate that receives the records which are read.

count is an integer that specifies the number of records you want read.

returned count is an integer variable that receives the number of bytes in a record that are read only when an end-of-file condition is encountered; otherwise, this argument isn't set.

ier is an integer variable that receives the routine's completion status code.

Rules

Before calling READRW, you must have specified the record length for this file in either an OPEN statement or in a call to the runtime routines, OPEN or APPEND.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
EREOF	End of file.
ERRPR	Attempt to read a read-protected file.
ERFOP	Attempt to refer to an unopened unit.
ERFIL	File read error.
ERRD	Attempted read into system area.
ERDIO	File accessible by direct block I/O only.
ERSIM	Simultaneous reads on same QTY line.
ERSCP	File position error (returned by .SPOS).
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERMCA	Attempt to perform an MCA read on a channel where reading is currently in progress.
ERCLO	QTY/MCA input terminated by channel close.
FEIFN	Illegal unit number.

Example

```
INTEGER PDQARY (400)
        .
        .
;FOUR-WORD RECORDS
OPEN 14,"INPUTDATA",LEN=8
        .
        .
CALL READRW (14,100,PDQARY,5,ICNT,IER)
CALL CHECK (IER)
```

Reference

.RDS (System call)

RESET
closes all open files.

Format
CALL RESET

Arguments
None.

Error Condition
The error condition that may result is:

ERDTO	Ten second disk time-out occurred.

Notes
When you call this routine in a multitask environment, invoke the SINGLETASK routine first. This freezes the environment. Call RESET, then reinstates the multitask environment with a call to MULTITASK.

Reference
.CLOSE (System call)

REWIND (Alias: FRWND)
rewinds a file.

Format
CALL REWIND (unit number)

Argument
unit number is an integer that specifies the FORTRAN 5 unit number of the file you want to rewind.

Error Conditions
The error conditions that may result are:

ERFOP	Attempt to refer to an unopened unit.
ERSCP	File position error.
FEIFN	Illegal unit number.

Example
```
CALL REWIND (22)
```

Reference
.SPOS (System call)

WRBLK

writes a series of blocks to a randomly or contiguously organized disk file.

Format

CALL WRBLK (unit number,starting block,
data array,count, *[returned count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the randomly or contiguously organized disk file you want written.

starting block is an integer that specifies the first block you want written (the first block is numbered 0).

data array is an aggregate that contains the data you want written.

count is an integer that specifies the number of blocks you want written.

returned count is an integer variable that receives the number of blocks successfully written when disk space is exhausted.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
ERSV1	Not a randomly or contiguously organized file.
EREOF	End of file (for contiguous files only).
ERWPR	File is write-protected.
ERFOP	Attempt to refer to an unopened file.
ERSPC	Disk space is exhausted.
ERFIL	File read error.
EROVA	File not accessible by direct I/O.
ERMPR	Address is outside address space (mapped systems only).
FEIFN	Illegal unit number.

Example

```
INTEGER AREA9 (1024)
      .
      .
OPEN 9,"FILE9"
      .
      .
;WRITE BLOCKS #4 THRU #6 FROM
;AREA9 INTO FILE9
CALL WRBLK (9,4,AREA9,3,IER)

CALL CHECK (IER)
```

Reference

.WRB (System call)

WRITRW (Alias: WRITR)
writes a series of records to a file.

Format

```
CALL WRITRW (unit number,starting record,
  data array,count,ier)
```

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number to which you want data written.

starting record is an integer that specifies the number of the first record you want written (the first record of a FORTRAN 5 file is numbered 1).

data array is an aggregate that contains the records you want to write.

count is an integer that specifies the number of records you want to write.

ier is an integer variable that receives the routine's completion status code.

Rules

Before calling WRITRW, you must specify the record length for this file. Do this in either an OPEN statement or in a call to the runtime routines, OPEN or APPEND.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
ERWPR	Attempt to write to a write-protected file.
ERFOP	Attempt to refer to an unopened file.
ERSPC	Disk space is exhausted.
ERDIO	File accessible by direct block I/O only.
ERSIM	Simultaneous writes to the same QTY line.
ERSCP	File position error (returned by .SPOS).
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERMCA	Attempt to issue an MCA write to a channel on which writing is currently in progress.
ERSRR	Incomplete MCA transmissions due to short receiver request.
ERCLO	MCA/QTY output terminated by channel close.
ERNMC	No outstanding receive request.
FEIFN	Illegal unit number.

Example

```
COMPLEX MESAREA
      .
      .
;FOUR-BYTE RECORDS
CALL OPEN (7,"OUTDATA",3,4,IER)
      .
      .
;WRITE RECORDS 14 AND 15
CALL WRITRW (7,14,MESAREA,2,IER)
CALL CHECK (IER)
```

Reference

.WRS (System call)

WRLIN
writes a line to a file.

Format

CALL WRLIN (unit number,data array, *[returned count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number to which you want data written.

data array is an aggregate that contains the line you want written.

returned count is an integer variable that receives the total number of bytes you want written.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
ERWRP	Attempt to write to a write-protected file.
ERFOP	Attempt to refer to an unopened unit.
ERLLI	Line limit (132 characters) exceeded.
ERSPC	Disk space is exhausted.
ERDIO	File accessible by direct block I/O only.
ERSIM	Simultaneous writes to same QTY line.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERCLO	QTY output terminated by channel close.
FEIFN	Illegal unit number.

Example

```
DOUBLE PRECISION COMPLEX SIGNET
      .
      .
SIGNET="TEST LINENUL"
      .
      .
CALL WRLIN (4,SIGNET,ICNT,IER)
CALL CHECK (IER)
```

Reference

.WRL (System call)

WRSEQ
performs a sequential write.

Format

CALL WRSEQ (unit number,data array,count,ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number to which you want data written.

data array is an aggregate that contains the data you want written.

count is an integer that specifies the number of bytes you want to write.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
ERWPR	Attempt to write to a write-protected file.
ERFOP	Attempt to refer to an unopened unit.
ERSPC	Disk space is exhausted.
ERDIO	File accessible by direct block I/O only.
ERSIM	Simultaneous writes to the same QTY line.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERMCA	Attempt to issue an MCA write to a channel on which writing is currently in progress.
ERSRR	Incomplete MCA transmissions due to short receiver request.
ERCLO	MCA/QTY output terminated by channel close.
ERNMC	No outstanding receive request.
FEIFN	Illegal unit number.

Example

```
INTEGER USAREA (10)
      .
      .
CALL WRSEQ (3,USAREA,360,IER)
CALL CHECK (IER)
```

Reference

.WRS (System call)

End of Chapter

Chapter 7
Console Input/Output

The FORTRAN 5 runtime routines, GCHAR, RCHAR, and PCHAR, handle the buffered transfer of single characters from the program console. GCHAR and RCHAR operate like a read sequential of one character and PCHAR operates like a write sequential of one character. These commands refer to $TTI/$TTO in the background and to $TTI1/$TTO1 in the foreground.

RDOS allows you to halt or suspend the operation of a currently executing program at the program console. Depressing CTRL-A interrupts the current program and gives you program control; depressing CTRL-C interrupts the current program, writes a copy of the program to disk file BREAK.SV (or FBREAK.SV for a foreground program) and gives you program control. Depressing CTRL-F terminates a foreground program's operation. You may disable any of these interrupts by issuing a call to the FORTRAN 5 runtime routine ODIS.

These console interrupts are enabled when you first bootstrap the system. However, because you can disable them, the runtime routine OEBL is provided to re-enable the interrupts in the current program environment.

The following is an alphabetical list of all runtime routines you will find in this chapter:

GCHAR	Reads a single character from the console.
GCIN	Obtains the input console name.
GCOUT	Obtains the output console name.
ODIS	Disables console interrupts.
OEBL	Enables console interrupts.
PCHAR	Writes a single character to the console.
RCHAR	Reads and echoes a single character from the console.
WCHAR	Waits for a character from the console (RTOS only).

GCHAR

reads a single character from the console without echoing it.

Format

CALL GCHAR (character,ier)

Arguments

character is a 1-word aggregate that receives the character from the console; the character is stored in ASCII format in the right byte, with zeros in the left byte.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERCID	Console not in system.

Example

```
CALL GCHAR (ICHAR,IER)
CALL CHECK (IER)
;BRANCH TO STMNT LBL 70 IF
;CHARACTER IS A SPACE
IF (ICHAR .EQ. 32) GO TO 70
```

Reference

.GCHAR (System call)

GCIN

obtains the input console name.

Format

```
CALL GCIN (filename,ier)
```

Arguments

filename is a 3-word aggregate that receives the input console name.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERRD	Attempt to read into a system area.
ERMPR	Address is outside address space (mapped systems only).

Example

```
INTEGER CONM(3)
      .
      .
CALL GCIN (CONM,IER)
CALL CHECK (IER)
```

Reference

.GCIN (System call)

GCOUT

obtains the output console name.

Format

```
CALL GCOUT (filename,ier)
```

Arguments

filename is a 3-word aggregate that receives the output console name.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Example

```
INTEGER OUTCON(3)
      .
      .
CALL GCOUT (OUTCON,IER)
CALL CHECK (IER)
```

Reference

.GCOUT (System call)

ODIS
disables console interrupts.

Format
CALL ODIS (ier)

Argument
ier is an integer variable that receives the routine's completion status code.

Error Conditions
No error conditions are currently defined.

Example
```
CALL ODIS (IER)
CALL CHECK (IER)
```

Reference
.ODIS (System call)

OEBL
enables console interrupts.

Format
CALL OEBL (ier)

Argument
ier is an integer variable that receives the routine's completion status code.

Error Conditions
No error conditions are currently defined.

Example
```
CALL OEBL (IER)
CALL CHECK (IER)
```

Reference
.OEBL (System call)

PCHAR

writes a single character to the console.

Format

CALL PCHAR (character,ier)

Arguments

character is the ASCII character to output to the console; it's taken from the right byte of the word.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERCID Console not in system.

Example

```
;WRITE THE LETTER "A" TO THE CONSOLE
CALL PCHAR (1AR,IER)
CALL CHECK (IER)
```

Reference

.PCHAR (System call)

RCHAR

reads and echoes a single character from the console.

Format

CALL RCHAR (character,ier)

Arguments

character is a 1-word aggregate that receives the character; the character is stored in ASCII format in the right byte, with zeros in the left byte.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERCID Console not in system.

Example

```
CALL RCHAR (ICHAR,IER)
CALL CHECK (IER)
```

References

.GCHAR (System call) - Reads a character from the console.

.PCHAR (System call) - Writes a character to the console.

/CHAR (RTOS only)
waits for a character from the console.

ɔrmat

ALL WCHAR (wait character,device code,ier)

.rguments

'ait character is either the console character for which ıe calling task must wait or -1 which terminates ıother task's outstanding wait and readies that ıspended task.

evice code is an integer variable that receives either ıe device code of the console that issued the wait haracter or -1 indicating that another task terminated ıe previous wait. (This Argument is returned only if o error occurs. See *Notes* below.)

ər is an integer variable that receives the routine's ompletion status code.

Error Condition

The error code that may return in ier is:

ERSIM	A previous wait character request is outstanding.

Example

```
CALL WCHAR (14K,ICODE,IER)
CALL CHECK (IER)
```

Notes

A call to this routine suspends you until the specified character is typed onto the console or another task issues the call (with -1 as the first argument) to terminate your wait. Only one task may be suspended in this way at any time.

References

.WCHAR (System call)

RTOS Reference Manual, Chapter 2, *Teletypewriter and Video Display Commands.*

End of Chapter

Chapter 8
Reading the Console Switches

RDSW (Alias: DATSW)
reads the console switches.

Format
CALL RDSW (switches,ier)

Arguments
switches is a 1-word aggregate that receives the state of the console switches.

ier is an integer variable that receives the routine's completion status code.

Error Conditions
No error conditions are currently defined.

Example
```
CALL RDSW (ISWICH,IER)
CALL CHECK (IER)
```

Reference
.RDSW (System call)

End of Chapter

Chapter 9
Using the System Clock and Calendar

The FORTRAN 5 runtime routines described in this chapter allow you to get the time of day, set the time, get the current date, and set the date. You can call one of two routines to perform each of these functions -- an SA formatted routine with an argument of 3-word nteger array or a routine that takes three separate nteger arguments. For example, to set the time you nay use either STIME or FSTIME.

Valid dates are those between January 1, 1968 and December 31, 2099. The time of day is based on a 24-hour clock. Midnight is 0,0,0; the second before midnight is 23,59,59.

The following is an alphabetical list of all runtime routines you will find in this chapter:

DATE	Obtains the current date.
FGDAY	Obtains the current date.
FGTIME	Obtains the current time.
FSDAY	Sets the date.
FSTIME	Sets the time.
SDATE	Sets the date.
STIME	Sets the time.
TIME	Obtains the current time.

DATE
obtains the current date.

Format

```
CALL DATE (date)
```

Argument

date is an integer array whose first three elements receive the current month, day, and year.

Error Conditions

No error conditions are currently defined.

Example

```
DIMENSION IDATE(3)
      .
      .
CALL DATE (IDATE)
```

Notes

The year is returned as a 4-digit integer.

Reference

.GDAY (System call)

FGDAY
obtains the current date.

Format

CALL FGDAY (month,day,year)

Arguments

month is an integer variable that receives the current month by number.

day is an integer variable that receives the current day.

year is an integer variable that receives the current year.

Error Conditions

No error conditions are currently defined.

Example

```
CALL FGDAY (IMON,IDAY,IYEAR)
```

Notes

The year is returned as a 2-digit integer.

Reference

.GDAY (System call)

FGTIME
obtains the current time.

Format

CALL FGTIME (hour,minute,second)

Arguments

hour is an integer variable that receives the current hour (0 - 23).

minute is an integer variable that receives the current minute (0 - 59).

second is an integer variable that receives the current second (0 - 59).

Error Conditions

No error conditions are currently defined.

Example

```
CALL FGTIME (IHOUR,IMIN,ISEC)
```

Reference

.GTOD (System call)

SDAY
sets the date.

ormat
:ALL FSDAY (month,day,year)

ırguments
ıonth is an integer that specifies the month by number.

lay is an integer that specifies the day.

ear is an integer that specifies the year.

rror Condition
'he error condition that may result is:

:RTIM Illegal month, day, or year.

xample
SET THE DATE TO OCTOBER 17, 1976
:ALL FSDAY (10,17,76)

Notes
'ou specify only two digits to express the year, whereas vith SDATE you must specify all four digits.

Reference
SDAY (System call)

FSTIME
sets the time.

Format
CALL FSTIME (hour,minute,second)

Arguments
hour is an integer that specifies the hour (0 - 23).

minute is an integer that specifies the minute (0 - 59).

second is an integer that specifies the second (0 - 59).

Error Condition
The error condition that may result is:

ERTIM Illegal time of day.

Example
;SET THE CLOCK TO 3:30 PM
CALL FSTIME (15,30,0)

Reference
.STOD (System call)

SDATE
sets the date.

Format
```
CALL SDATE (date,ier)
```

Arguments
date is an integer array whose first three elements contain the date as month, day, and year.

ier is an integer variable that receives the routine's completion status code.

Error Condition
The error code that may return in ier is:

ERTIM	Illegal month, day, or year.

Example
```
DIMENSION IDATE(3)
        .
        .
IDATE(1) = 7
IDATE(2) = 4
IDATE(3) = 1976
;SET THE DATE TO JULY 4, 1976
CALL SDATE (IDATE,IER)
CALL CHECK (IER)
```

Notes
You must specify four digits to express the year.

Reference
.SDAY	(System call)

STIME
sets the time.

Format
```
CALL STIME (time,ier)
```

Arguments
time is an integer array whose first three elements contain the time based on a 24-hour clock in hours, minutes, and seconds.

ier is an integer variable that receives the routine's completion status code.

Error Condition
The error code that may return in ier is:

ERTIM	Illegal time of day.

Example
```
DIMENSION IARRAY(3)
        .
        .
IARRAY(1) = 10
IARRAY(2) = 0
IARRAY(3) = 0
        .
        .
;SET THE CLOCK TO 10:00 AM
CALL STIME (IARRAY,IER)
CALL CHECK (IER)
```

Reference
.STOD	(System call)

'IME
obtains the current time.

ormat

:ALL TIME (time,ier)

ırguments

ime is an integer array whose first three elements eceive the current time of day based on a 24-hour lock in hours, minutes, and seconds.

er is an integer variable that receives the routine's ompletion status code.

Error Conditions

No error conditions are currently defined.

Examples

```
DIMENSION ITIME(3)
        .
        .
CALL TIME (ITIME,IER)
CALL CHECK (IER)
```

Reference

.GTOD (System call)

End of Chapter

Chapter 10
Multitask Programming in FORTRAN 5

This chapter presents general concepts of multitasking, while Chapters 11 through 19 detail the multitasking runtime routines. Before reading this chapter, read Chapter 9 in the *FORTRAN 5 Reference Manual.*

FORTRAN 5 supports nearly all multitask capabilities of RDOS. In addition, FORTRAN 5 provides you with the event mechanism explained in the *FORTRAN 5 Reference Manual.*

Tasks and Their Resources

The current state of a task's resources defines the task. In a multitask environment, tasks share physical resources that are both abundant and scarce. Depending on a particular task's resource requirements, the operating system and FORTRAN 5 manage the resources.

RDOS handles the scarce resources that most tasks need. These are the accumulators, carry, user stack pointer (USP), the hardware stacks for both the ECLIPSE and NOVA/3, and the program counter. The system manages these resources through the task control block (TCB) for each task.

For example, RDOS allocates an area of memory for each task to store its copy of the accumulators' values when the task isn't executing. The multitask scheduler is responsible for task state save and restore.

FORTRAN 5 tasks require resources in addition to those which RDOS handles. These are abundant resources such as memory partitions. FORTRAN 5 handles the resources FORTRAN 5 tasks need.

In addition to managing the abundant resource, memory, FORTRAN 5 manages the scarce resources -- floating point unit and page zero locations which are called *task state variables* (.SP, .FP, .SSE, .GP, .RP, .ND1, .ND2). FORTRAN 5's management of both abundant and scarce resources utilizes the TCB extension.

If you write all your tasks in FORTRAN 5, then RDOS and FORTRAN 5 together handle the resources.

You may also write non-FORTRAN 5 tasks. For instance, you may write tasks in assembly language. If you don't designate which resources a task can use, it has access to both RDOS-managed and FORTRAN-managed resources. Runtime routines, for instance, have access to both types of resources.

When you write a task in assembly language that doesn't need FORTRAN 5 resources, differentiate the resources it can access; this prevents a waste of resources, memory, and time. It's unnecessary to store copies of unused resources, and the multitask scheduler wastes time saving and restoring unused resources.

You can specify whether or not a task you initiate needs access to FORTRAN 5 resources. Specify the stack size as 100000K in the stack size parameter. This stack size specification prevents the task from receiving a TCB extension or a FORTRAN 5 memory partition. It must not use these FORTRAN 5 resources or it will affect FORTRAN 5 tasks.

Memory Partitions in a Multitask Runtime Environment

Each FORTRAN 5 task in a multitask environment has its own memory partition consisting of a task global area and a runtime stack area. The runtime stack area contains a stack, end zone, and I/O control blocks (IOCB). See F5SYM.SR for sizes of the task global area, end zone, and fixed portion of the IOCB.

By default, the memory available at runtime initialization time is divided into partitions numerically equal in size to the TCB's. (You specify this number at RLDR time with the local /K switch.) Default allocation of memory may not satisfy certain combinations of tasks. You can specify more precisely the method by which memory should be allocated to the various tasks in the system.

Control memory partitioning by using the PARTITION macro defined in PARTITION.SR. With this macro, you build a partition specification table in assembly language. You assemble and load the table with your FORTRAN 5 programs and runtime routine. The

runtime initializer apportions available memory according to the partition specification table. When you initiate a FORTRAN 5 task use the stack size parameter to assign a partition to the task. The amount of required stack area depends on the nesting of subprogram calls and on the individual stack requirements for each of the subprograms.

When you kill a FORTRAN 5 task, its partition returns to the pool of available partitions from where you may later allocate it to another task.

The PARTITION macro enables you to specify the number and size of partitions that the runtime initializer will construct. You may specify a fixed or default size for each partition to be constructed. All partitions with fixed sizes are allocated first; the remaining memory is divided equally among all partitions which are assigned the default size. Thus, the default size for partitions depends on the total amount of memory available, the amount of memory you allocate to fixed size partitions, and the number of default size partitions that share the remaining memory.

The number of default size partitions the runtime initializer constructs depends on one of two specifications. If you explicitly specify zero or more default size partitions, the number you specify will be the number constructed. If you don't explicitly specify any default size partitions, the runtime initializer will attempt to construct a number of default size partitions such that the total number of partitions (both fixed and default sizes) is equal to the maximum number of tasks that can exist (i.e., the number of TCBs you allocated at load time via the RLDR /K switch). If there are already as many (or more) partitions of fixed size TCBs, then the runtime initializer doesn't allocate any default size partitions.

To accomplish memory partitioning, you must first edit DPART.SR, and then assemble it to produce the partition specification table. This source file will include several invocations (calls) of the PARTITION macro in the following form:

```
PARTITION
PARTITION S1 [N1]
PARTITION S2 [N2]
        .
        .
        .
PARTITION Sk [Nk]
PARTITION
```

The first and last invocations must be only the word PARTITION; the first line begins the partition specification table and the last line terminates it. Any number of invocations with either one or two decimal arguments may appear between these two calls. The first argument defines a partition size (0 or 1 means the default size). The second optional argument is a repetition count; it specifies the number of partitions to be constructed with the size you gave in the first argument. If you omit the second argument, one partition of the size you gave will be constructed.

You then assemble this source file as described previously in Part I, Chapter 1, in the section, *Assembling your Assembly Language Subprograms.* The assembled macro calls expand into a partition specification table.

You load the assembled partition specification table with your main program and subprograms, before the FORTRAN 5 runtime libraries.

The FORTRAN 5 runtime initializer apportions available memory according to the partition specification table. You may restrict the amount of memory the initializer treats as available by redefining the .NMAX symbol. All fixed size partitions are allocated first, then the remaining memory is divided equally among the default size partitions.

When you initiate a FORTRAN 5 task, the task is allocated a partition from the pool of available partitions. (If none is available, you get an error message.) The task is assigned a partition according to its stack size parameter. The stack size is specified in one of a number of ways:

- If you initiate the task using the TASK statement, you may designate the stack size with the STK= option. (See the *FORTRAN 5 Reference Manual* for details.) If you do not use the option, a stack size parameter value of 0 is used.

- For runtime routines ASSOCIATE, FTASK,IOPROG, and ITASK, you may specify the stack size as an argument to the call. The stack size parameter is optional for FTASK, IOPROG, and ITASK. If you don't specify your stack size, a stack size parameter value of 0 is used.

- If you initiate the task from assembly language with the macro call, S?TASK, the stack size is passed in an accumulator.

- DPART.SR specifies the stack size for the FORTRAN 5 main program as a parameter. By default the stack size parameter value of 0 is used.

The following table shows the interpretation of the stack size parameter.

Stack Size Parameter	Effect on Partition Selection
0 or 1	Select an available default size partition.
2	Select the smallest available partition.
3	Select the largest available partition.
>3	Select an available partition of the size given. (Note: The size must match exactly.)
100000K	Select no partition.

You get an error message if an available partition can't be found to meet the criteria you specified for the stack size.

Each called FORTRAN 5 subprogram requires an amount of stack space, in words, equal to the sum of the following:

- the number of arguments passed to it (plus one additional word if the program is a function subprogram).
- 5 words for the stack frame header.
- the number of words designated by the second word of its SAVE operation.

The stack requirements of runtime routines vary; a typical routine needs less than 20 words.

Classes of Suspensions

There are various classes of task suspensions. For information on multiple suspensions refer to the *RDOS Reference Manual* or the *FORTRAN 5 Reference Manual.* Each task suspension acts independently. When you issue a call to suspend a task, you must issue its corresponding call to ready the task. A task will not resume until you lift all suspensions.

End of Chapter

Chapter 11
Initiating Tasks in a Multitask Environment

When you initiate a task, you must assign a priority to it. You can also give the task an optional identification number.

A task's priority may range from 0, the highest, to 255 decimal, the lowest. The task scheduler allocates CPU control to the highest priority ready task.

Tasks may have equal priorities; these tasks will receive CPU control on a round robin basis. The task which most recently received control will be the last to receive control again, unless the other tasks at this same priority aren't ready when their turns come.

A task's ID must be unique, constant, and in the range 1 through 255 decimal. Any number of tasks may have no ID (indicated as 0).

The following is an alphabetical list of all runtime routines you will find in this chapter:

FTASK	Initiates a task.
ITASK	Initiates a task.
S?TASK	Initiates a task from assembly language.

Although FTASK and ITASK both initiate a task, we recommend that you call ITASK. It permits you to specify an ID and gives you a completion status code.

FTASK
initiates a task.

Format

CALL FTASK (subroutine,error return label,priority *[,stack size]*)

Arguments

subroutine is the initial subroutine the task will execute.

error return label is a statement label to which control is transferred if an error occurs during task initiation.

priority is an integer that specifies the initial priority for the task.

stack size is an integer that specifies the stack size for the task. If you omit stack size, a stack size parameter value of 0 is used.

Error Conditions

The error conditions that may result are:

ERNOT	No TCB available.
ERTID	A task with the requested ID (except 0) already exists.
FEPNA	Requested partition unavailable.
FEPRI	Illegal task priority.
FESTK	Illegal stack size.

Example

```
CALL FTASK (SU,$99,20,400)
```

Reference

.TASK	(Task call)

ITASK
initiates a task.

Format

CALL ITASK (subroutine,task ID,priority,
[stack size,] ier)

Arguments

subroutine is the initial subroutine the task executes.

task ID is an integer specifying the identification number you want to assign to the task. Zero indicates you didn't want to specify a task ID.

priority is an integer that specifies the initial priority for the task.

stack size is an optional integer argument that specifies the stack size parameter for the task. If you omit stack size, a stack size parameter value of 0 is used.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERNOT	No TCB available.
ERTID	Task with requested ID (except 0) already exists.
FEPNA	Requested partition unavailable.
FEPRI	Illegal task priority.
FESTK	Illegal stack size.

Example

```
CALL ITASK (SUBR2,11,2,500,IER)
CALL CHECK (IER)
```

Reference

.TASK (Task call)

?TASK
initiates a task from assembly language.

ormat

?TASK
rror return
ormal return

nput

AC0 (left byte) contains the task identification number.

AC0 (right byte) contains the initial task priority.

AC1 contains the task entry point.

AC2 contains the stack size.

error return is the instruction receiving control if an error occurs.

normal return is the instruction receiving control if no error occurs.

Output

The accumulators and carry remain the same.

Error Conditions

If an error occurs, AC2 may receive one of the following codes:

ERNOT	No TCB available.
ERTID	A task with the ID you requested (except 0) already exists.

Notes

S?TASK is a macro that calls the runtime routine, .TASK.

Reference

.TASK (Task call)

Example

```
          .
          .EXTN    HARRY    ;NAME OF INITIAL SUBROUTINE
                            ;FOR TASK TO EXECUTE
          .
          .
          LDA      0,TASKID         ;GET TASK ID VALUE
          MOVS     0,0              ;MOVE IT TO LEFT BYTE
          LDA      1,TASKPRI        ;GET TASK PRIORITY VALUE
          ADD      1,0              ; TASK ID / TASK PRI
          LDA      1,.HARRY         ;GET SUBROUTINE ADDRESS
          SUB      2,2              ;SIGNAL DEFAULT-SIZE STACK
          S?TASK                    ;START THE TASK
          JMP      ERROR            ;PROCESS ANY ERROR
          .
          .
.HARRY:   HARRY                     ;ADDRESS OF SUBROUTINE HARRY
TASKID:   2                         ;TASK IDENTIFIER VALUE
TASKPRI:  3                         ;TASK PRIORITY VALUE
          .
          .
```

Figure II-11-1. S?TASK Example

End of Chapter

Chapter 12
Changing Task States in a Multitask Environment

You can use the runtime routines in this chapter to suspend and ready tasks. These calls are of like class. This means if you call one of the following routines to suspend a task, there are appropriate routines here to ready the task.

The calls, TIDS, TIDR, TIDK, and TIDP approach tasks by their ID to suspend, ready, kill, or change their priority. Other calls, such as AKILL, ARDY, and ASUSP instead approach tasks by their priorities to kill, ready, and suspend.

If you use ASUSP to suspend a task, you can ready that task with ARDY or TIDR.

In addition to calls which ready tasks of a specific ID or priority, there is an event-oriented system call, .IWAKE, that readies a task. You can call .IWAKE to perform the WAKEUP function from the interrupt world. Read about the event mechanism in the *FORTRAN 5 Reference Manual.*

There are several methods you can designate for terminating tasks. A task can terminate itself. The best way for you to design a program in which a task terminates itself is to write a RETURN from the task's initial subroutine. If you use this method to kill the task, the initial subroutine can then be used as a normal subroutine or as a task.

Another way in which a task terminates itself is through CALL KILL. With TIDK, AKILL, the KILL statement and DESTROY, a task can terminate itself or other tasks. Your call to DESTROY is the most abrupt of the calls; it aborts the system call.

The following is an alphabetical list of all runtime routines you will find in this chapter:

ABORT	Alias for TIDK.
AKILL	Kills all tasks of a given priority.
ARDY	Readies all tasks of a given priority.
ASUSP	Suspends all tasks of a given priority.
CHNGE	Alias for TIDP.
CHPRI	Alias for TIDP.
DESTROY	Kills the task specified by an identification number.
HOLD	Alias for TIDS.
.IWAKE	Performs a wakeup on an event number from the interrupt world.
KILL	Kills the calling task.
PRI	Changes the priority of the calling task.
RELSE	Alias for TIDR.
SUSP	Suspends the calling task.
TIDK	Kills the task specified by an identification number.
TIDKILL	Alias for TIDK.
TIDP	Changes the priority of the task specified by an identification number.
TIDPRI	Alias for TIDP.
TIDR	Readies the task specified by an identification number.
TIDRDY	Alias for TIDR.
TIDS	Suspends the task specified by an identification number.
TIDSUSP	Alias for TIDS.

AKILL
kills all tasks of a given priority.

Format
CALL AKILL (priority)

Argument
priority is an integer that specifies a task priority number.

Error Conditions
No error conditions are currently defined. If the task's priority is outside the range, 0 to 255 (decimal), or if no task exists at the priority specified, AKILL has no effect.

Examples
CALL AKILL (4)

Reference
.AKILL (Task call)

ARDY
readies all tasks of a given priority.

Format
CALL ARDY (priority)

Argument
priority is an integer that specifies a task priority number.

Error Conditions
No error conditions are currently defined. If no tasks with the given priority exist, ARDY has no effect.

Example
CALL ARDY (PRI)

Reference
.ARDY (Task call)

ASUSP

suspends all tasks of a given priority.

Format

CALL ASUSP (priority)

Argument

priority is an integer that specifies a task priority number.

Error Conditions

No error conditions are currently defined. If a task's priority is outside the range, 0 to 255 (decimal), or if no task exists at the priority specified, ASUSP has no effect.

Example

```
CALL ASUSP (2)
```

Reference

.ASUSP (Task call)

DESTROY

aborts a task's system call and kills the task.

Format

CALL DESTROY (task ID,ier)

Arguments

task ID is an integer that specifies the identification number of the task you want to kill.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERTID	Either you specified a task ID of 0, or no such task ID was found.
ERABT	The task you specified was performing either QTY, MCA I/O, or a system read/write operator message call.
FETID	You specified an illegal ID.

Example

```
CALL DESTROY (201,IER)
CALL CHECK (IER)
```

Notes

DESTROY doesn't release any open channels used by the killed task, nor does it release any overlays. DESTROY terminates outstanding operations performed by the task and aborts most system calls. (See the error code, ERABT, for the exceptions.)

Reference

.ABORT (Task call)

.IWAKE

performs a wakeup on an event number from the interrupt world.

Format

```
JSR @.IWAKE
return
```

Input

AC0 contains the event number.

Output

The contents of carry and all accumulators are lost.

Error Conditions

No error conditions are currently defined. If the event number is illegal, then .IWAKE takes no action.

Example

```
.EXTD .IWAKE

LDA 0,EVNUM     ;GET EVENT NUMBER
JSR @.IWAKE     ;CALL THE ROUTINE

EVNUM: 3        ;EVENT 3
```

KILL

kills the calling task.

Format

```
CALL KILL
```

Arguments

None.

Error Conditions

No error conditions are currently defined.

Reference

.KILL (Task call)

PRI

changes the priority of the calling task.

Format

CALL PRI (priority)

Argument

priority is an integer that specifies the new priority of the calling task.

Error Condition

The error condition that may result is:

FEPRI	You specified an illegal task priority.

Example

```
CALL PRI (5)
```

Reference

.PRI (Task call)

SUSP

suspends the calling task.

Format

CALL SUSP

Arguments

None.

Error Conditions

No error conditions are currently defined.

Reference

.SUSP (Task call)

IDK (Aliases: TIDKILL, ABORT)

kills the task specified by an identification number.

rmat

LL TIDK (task ID,ier)

guments

sk ID is an integer that specifies the identification mber of the task you want to kill.

is an integer variable that receives the routine's mpletion status code.

ror Conditions

ror codes that may return in ier include:

RTID	Either you specified a task ID of 0, or no such task ID was found.
ETID	You specified an illegal ID to a task.

ample

```
ALL TIDK (99,IER)
ALL CHECK (IER)
```

eference

IDK (Task Call)

TIDP (Aliases: TIDPRI, CHNGE, CHPRI)

changes the priority of a task specified by an identification number.

Format

CALL TIDP (task ID,priority,ier)

Arguments

task ID is an integer that specifies the identification number of the task receiving the new priority.

priority is an integer that specifies the new priority of the task.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERTID	Either you specified a task ID of 0, or no such task ID was found.
FEPRI	You specified an illegal task priority.
FETID	You designated an illegal ID for a task.

Example

```
CALL TIDP (100,1,IER)
CALL CHECK (IER)
```

Reference

.TIDP (Task call)

)R (Aliases: TIDRDY, RELSE)

readies a task specified by an identification number.

mat

.L TIDR (task ID,ier)

;uments

‹ ID is an integer that specifies the identification ıber of the tasks you want to ready.

is an integer variable that receives the routine's ıpletion status code.

or Conditions

or codes that may return in ier include:

ΓID	Either you specified a task ID of 0, or no such task was found.
ΓID	You designated an illegal ID for a task.

ımple

```
LL TIDR (101,IER)
LL CHECK (IER)
```

ference

)R	(Task call)

TIDS (Aliases: TIDSUSP, HOLD)

suspends a task specified by an identification number.

Format

```
CALL TIDS (task ID,ier)
```

Arguments

task ID is an integer that specifies the identification number for the task you want to suspend.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERTID	Either you specified a task ID of 0, or no such task ID was found.
FETID	You designated an illegal ID for a task.

Example

```
CALL TIDS (55,IER)
CALL CHECK (IER)
```

Reference

.TIDS	(Task call)

End of Chapter

Chapter 13
Obtaining Task-Related Information in a Multitask Environment

The following is an alphabetical list of all runtime routines you will find in this chapter:

GETEV	Obtains a task's event number.
GETPRI	Obtains a task's priority.
IDST	Obtains a task's status.
MYEV	Obtains the calling task's event number.
MYID	Obtains the calling task's identification number.
MYPRI	Obtains the calling task's priority.
STTSK	Alias for IDST.

GETEV
obtains a task's event number.

Format

```
CALL GETEV (task ID,event,ier)
```

Arguments

task ID is an integer that specifies the task's identification number.

event is an integer variable that receives the event number associated with the task. Zero represents the lack of event association.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

The error code that may return in ier is:

ERTID	Either you specified a task ID of 0, or no such task ID was found.
FEEVT	Illegal event usage (task is not a FORTRAN 5 task).

Example

```
CALL GETEV (100,IEVENT,IER)
CALL CHECK (IER)
```

GETPRI
obtains a task's priority.

Format

```
CALL GETPRI (task ID,priority,ier)
```

Arguments

task ID is an integer that specifies the task's identification number.

priority is an integer variable that receives the task's priority.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERTID	Either you specified a task ID of 0, or no such task ID was found.

Example

```
CALL GETPRI (100,IPRI,IER)
CALL CHECK (IER)
```

IDST (Alias: STTSK)
obtains a task's status.

Format
CALL IDST (task ID,status)

Arguments
task ID is an integer that specifies the task's identification number.

status is an integer variable that receives the task's status, and may have one of the following values:

0	Ready.
1	Suspended by a .SYSTM call or .TRDOP.
2	Suspended by a .SUSP, .ASUSP, or .TIDS.
3	Waiting for a message to be sent or received.
4	Waiting for an overlay area.
5	Suspended by .ASUSP, .SUSP, or .TIDS and by .SYSTM.
6	Suspended by .XMTW or .REC and by .SUSP, .ASUSP, or .TIDS.
7	Waiting for an overlay area and suspended by .ASUSP, .SUSP, or .TIDS.
10	No task exists with this ID.

Error Conditions
No error conditions are currently defined.

Example
```
CALL IDST (124,I)
```

Reference
.IDST (Task call)

MYEV
obtains the calling task's event number.

Format
CALL MYEV (event)

Argument
event is an integer variable that receives the task's event number. Zero represents a lack of event association.

Error Conditions
No error conditions are currently defined.

Example
```
CALL MYEV (J)
```

MYID

obtains the calling task's identification number.

Format

CALL MYID (task ID)

Argument

task ID is an integer variable that receives the task's identification number. Zero represents a lack of ID.

Error Conditions

No error conditions are currently defined.

Example

CALL MYID (I)

MYPRI

obtains the calling task's priority.

Format

CALL MYPRI (priority)

Argument

priority is an integer variable that receives the task's priority.

Error Conditions

No error conditions are currently defined.

Example

CALL MYPRI (I)

End of Chapter

Chapter 14
Intertask Communication

FORTRAN 5 provides for the transmission and reception of one-word messages between tasks. If several tasks attempt to receive the same message, only the highest priority task will receive the message.

The following is an alphabetical list of all runtime routines you will find in this chapter:

REC	Receives a one-word message from another task.
XMT	Transmits a one-word message to a task.
XMTW	Transmits a one-word message to a task and waits for it to be received.

REC

receives a one-word message from another task.

Format

```
CALL REC (mailbox,message)
```

Arguments

mailbox is a one-word aggregate through which the message passes.

message is an integer variable that receives the message.

Rules

Never directly change or examine a mailbox.

Error Conditions

No error conditions are currently defined.

Example

```
CALL REC (AREA9,IMSG)
```

Reference

.REC (Task call)

XMT

transmits a one-word message to a task.

Format

CALL XMT (mailbox,message,error)

Arguments

mailbox is one word through which the message passes.

message is a nonzero integer value you want to transmit to another task.

error is a statement label to which control is transferred when an error occurs.

Rules

Never directly change or examine a mailbox.

Error Conditions

The error conditions that may result are:

ERXMT	Message address is in use.
ERXMZ	Zero message word.

Example

```
CALL XMT (ITSK10,IMSG,$100)
```

Notes

The difference between XMT and XMTW is that XMT deposits a message, whereas XMTW suspends you until the transmitted message is received.

Transmitting and receiving messages occurs in the same mailbox. Messages pass under the control of the multitask scheduler in the mailbox.

Reference

.XMT (Task call)

XMTW

transmits a one-word message to a task and waits for it to be received.

Format

CALL XMTW (mailbox,message,error)

Arguments

mailbox is one word through which the message passes.

message is a nonzero integer value you want to transmit to another task.

error is a statement label to which control is transferred when an error occurs.

Rules

Never directly change or examine a mailbox.

Error Conditions

The error conditions that may result are:

ERXMT	Message address is in use.
ERXMZ	Zero message word.

Example

```
CALL XMTW (ITSKIO,IABC,$1050)
```

Reference

.XMTW (Task call)

End of Chapter

Chapter 15
Using Overlays

An overlay file is a disk file containing parts of a program which are called overlays. Each overlay consists of one or more subroutines or functions used by a memory-resident program. When the resident program needs a particular routine, it first calls the operating system to load the appropriate overlay into memory and then calls the routine it needs. FORTRAN 5 provides interfaces to all of the overlay facilities provided by the operating system. FORTRAN 5 also provides a different overlay management system called *load-on-call overlays* . See Part I, Chapter 1, for more information about using FORTRAN 5 under RDOS.

There are two types of overlays in RDOS: ordinary overlays and virtual overlays. Ordinary overlays are copied into memory from a disk file whenever they are needed. Virtual overlays are mapped into your program from extended memory, where they are kept by RDOS. Since they are in memory at all times, virtual overlays are much faster than ordinary overlays.

You may request that overlays be loaded conditionally or unconditionally. A conditional request loads the overlay unless it is already resident. Use it when no data in the overlay must be reinitialized before the overlay routine is called. An unconditional request causes the overlay to be loaded even if it is already present, reinitializing any data in the overlay to their original values. You may place data in an overlay by declaring it STATIC, or by placing it in a COMMON block which is defined only within the overlay. Note that a DATA-initialized variable which is not in COMMON will be placed in STATIC storage.

There is an important difference between ordinary overlays and virtual overlays: loading a virtual overlay does not reinitialize data in the overlay. A virtual overlay is not copied into your program; the operating system redefines the set of memory blocks your program can access to bring in the overlay. Changes made in a virtual overlay persist for all uses of that virtual overlay, as if the overlay routines were not even in an overlay.

In either singletask or multitask environment you may use all of the routines this chapter describes. When you use an overlay in a multitask environment, you must release the overlay when you finish with it so the overlay area may be reused. The routine OVREL releases an overlay in a multitask environment, but does nothing in a singletask environment. The singletask version exists so you can develop programs within a singletask environment and run them in a multitask environment without change or recompilation.

Many routines in this chapter require you to specify an overlay name. You declare an overlay name in an OVERLAY statement by exactly one subroutine or function in the overlay. Declare the name EXTERNAL in all other routines using it.

For example, in the main program:

```
EXTERNAL OFRED
      .
CALL OVLOD (OFRED, -1, IER)
CALL CHECK (IER)
CALL FRED (Y)
      .
END
```

and in the subroutine:

```
SUBROUTINE FRED (X)
OVERLAY OFRED
      .
      .
END
```

Several routines in this chapter let you specify a *unit number* as their first argument. All of the routines ignore this argument because they don't need the information. The *unit number* is allowed as an optional argument because the FORTRAN IV versions of the routines require the information.

Following is an alphabetical list of the routines you will find in this chapter:

EST	Loads an overlay unconditionally.
FOVLD	Alias for OVLOD.
FOVLY	Alias for OVLOD.
FOVRL	Alias for OVREL.
OVCLOSE	Closes an overlay file.
OVEXIT	Releases an overlay and returns to the overlay's caller.
OVKILL	Kills task and releases the overlay in which it is currently executing.
OVLOD	Loads an overlay.
OVOPN	Opens an overlay file.
OVREL	Releases an overlay.
UNEST	Alias for OVRL.

EST
unconditionally loads an overlay.

Format

CALL EST (*[unit number,]* overlay name,ier)

Arguments

unit number is ignored.

overlay name is the name of the overlay.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

EREOF	End of file.
ERRPR	Attempt to read a read-protected file.
ERFOP	File not opened.
ERFIL	Read error.
EROVN	Illegal overlay number.
EROVA	Overlay file is not a contiguous file.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
EXTERNAL OV3
        .
        .
        .
CALL EST (OV3,IER)
CALL CHECK (IER)
```

Notes

You must pair each overlay request with an eventual overlay release or else the area will be reserved indefinitely.

Reference

.TOVLD (Task call)

OVCLOSE
closes an overlay file.

Format

CALL OVCLOSE (ier)

Argument

ier is an integer variable that receives a numeric status code.

Error Conditions

Error codes that may return in ier include:

ERFOP	Attempt to refer to a channel not in use.
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
CALL OVCLOSE (IER)
CALL CHECK (IER)
```

Reference

.CLOSE (System call)

)VEXIT

releases the overlay in which the current subroutine is executing and returns to the caller of the current subroutine. (In a singletask environment OVEXIT returns to the caller of the current subroutine.)

ɒrmat

ALL OVEXIT (overlay name,ier)

.rguments

verlay name is the name of the overlay you want to elease.

er is an integer variable that receives the routine's ɔmpletion status code.

rror Condition

he error code that may return in ier is:

ROVN Invalid overlay number; the overlay area is not occupied by the given user overlay.

xample

CALL OVEXIT (OVNAM,IER)
CALL CHECK (IER)

eference

OVREL (Task call)

OVKILL

kills the calling task and releases the overlay in which the task is executing. (In a singletask environment, this causes program termination because there is only one task to kill.)

Format

CALL OVKILL (overlay name,ier)

Arguments

overlay name is the name of the overlay you want to kill and release.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

EROVN Invalid overlay number.

Example

```
EXTERNAL OSUB1
         .
         .
         .
CALL OVKILL (OSUB1,IER)
CALL CHECK (IER)
```

Reference

.OVKIL (Task call)

OVLOD (Aliases: FOVLY, FOVLD)
loads an overlay.

Format

CALL OVLOD (*[unit number]* ,overlay name,flag,ier)

Arguments

unit number is ignored.

overlay name is the name of the overlay.

flag is an integer set to -1 when the overlay is to be loaded unconditionally.

ier is an integer variable that receives a numeric status code.

Error Conditions

Error codes that may return in ier include:

EREOF	End of file.
ERRPR	Attempt to read a read-protected file.
ERFOP	File not opened.
ERFIL	Read error.
EROVN	Illegal overlay number.
EROVA	Overlay file is not a contiguous file.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
EXTERNAL OV3
        .
        .
        .
CALL OVLOD (OV3,-1,IER)
CALL CHECK (IER)
```

Reference

.OVLOD (System call)

OVOPN
opens an overlay file.

Format

CALL OVOPN (*[unit number,]* filename,ier)

Arguments

unit number is ignored.

filename is an aggregate that contains the name of the overlay file.

ier is an integer variable that receives a numeric error status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
EREOF	End of virtual overlay file.
ERRPR	Attempt to read a read-protected virtual overlay file.
ERDLE	Nonexistent file.
ERUFT	Attempt to use channel which is already in use.
ERMEM	Insufficient memory to load virtual overlays.
ERFIL	File read error of virtual overlay file.
EROVA	Not a contiguous file (virtual overlays only).
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second time-out occurred.
ERENA	No linking allowed (N attribute).
FEOAO	Overlay file already open.

Example

```
CALL OVOPN (200,"PROG.OL",IER)
CALL CHECK (IER)
```

Notes

You must open an overlay file before you can access any overlay within.

Reference

.OVOPN (System call)

OVREL (Aliases: FOVRL, UNEST)

releases an overlay. (In a singletask RDOS environment this performs no action.)

Format

CALL OVREL (overlay name,ier)

Arguments

overlay name is the name of the overlay you want to release.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

EROVN	Invalid overlay number; the overlay area is not occupied by this user overlay.

Example

```
EXTERNAL OVLY1
        .
        .
        .
CALL OVREL (OVLY1,IER)
CALL CHECK (IER)
```

Notes

You cannot issue this command from the overlay you want to release.

Reference

.OVREL (Task call)

End of Chapter

Chapter 16
Delayed or Periodic Execution

The operating system provides a facility for delayed and periodic task initiation. It also maintains a queue of initiation requests; a queue table represents each of these requests.

Each queue table contains three types of information. First, there is information describing the tasks to be initiated: task identifier, task priority, stack size, and the task's initial subroutine. If the subroutine resides in an overlay, the information includes that overlay name and the conditional load flag. A second type information designates times for task initiation: hour, second within the specified hour, interval between initiations, and number of times to initiate a task. The system uses a third type information for its bookkeeping.

Preparations for using delayed and periodic task initiation require you to perform several activities. You must allocate memory for the queue tables. You usually do this in COMMON or STATIC storage. Give an array(s) dimensions sufficiently large for the queue tables.

Since the operating system uses the allocated memory between the table's queuing and dequeuing, do not change the values in the array during those times. If subroutines allocate stack space for queue tables, you must dequeue them before the subroutine returns. When the subroutine returns, the stack space is free.

In the queue table, supply information describing the tasks you want initiated. See Table II-16-1 for how to do this. After setting up the queue table, transfer it to the operating system as a task initiation request.

At the time you designate, the operating system will attempt to initiate the described task. Any errors at this point foil the initiation and the system tries it later.

When a queue table's requests are satisfied, the system dequeues it. Then you have access to the table. You must kill each task when its work is finished. If your program RETURNs from the task's initial subroutine, it automatically kills itself.

You can cancel a set of requests in the queue table. Direct the system to remove the corresponding queue table from its queue. See the next section for the specific call.

Requesting Delayed/Periodic Task Initiation

FORTRAN 5 provides two ways in which you request delayed or periodic task initiation: ISA-style routines and RDOS task calls.

You may use a set of ISA-style routines. Call ASSOCIATE to fill in the task descriptions. Specify task initiation time and transfer the queue table to the operating system by calling START, TRNON, or CYCLE. START requests initiation after a specified delay, while TRNON requests initiation at a specific future time. CYCLE requests periodic initiation with a delay between initiations.

If you must terminate some of the requests before the operating system finishes with them, call CANCL.

When you use the ISA-syle routines, the task's initial subroutine may not reside in an overlay unless the subroutine is a load-on-call routine.

Besides issuing calls to specific ISA routines, you can request delayed and periodic task initiation through RDOS task calls.

Supply the queue table information according to the table below. Completing the table requires you to call FQTASK; this call also transfers the queue table to the operating system. To terminate tasks, call DQTSK as you would call CANCL.

Unlike the ISA-style routines, task calls can handle initial sub- routines which need their overlays loaded explicitly. RDOS task calls can also handle load-on-call overlay subroutines.

Overlay Considerations

If you use overlays, there are several guidelines you must follow when requesting delayed and periodic task initiation.

With the ISA calls, you may use overlays only with the load-on-call mechanism. Before using overlays with either initiation request method, you must call OVOPN. Never let a task be killed before releasing the overlay. Calling OVEXIT most conveniently releases the overlay and returns; it causes automatic termination.

Table II-16-1. Queue Table

INDEX	MNEMONIC	MEANING
1	QPC	Maintained by the system.
2	QNUM	Number of times to initiate a task of this description (-1 to initiate an unlimited number of tasks). If you call TRNON, START, or CYCLE, the routines fill in this value automatically. If you call FQTSK, you must fill in this value.
3	QTOV	Overlay descriptor for the initial subroutine of the task(s) to be initiated (-1 to indicate that the initial subroutine is resident). ASSOCIATE and FQTASK fill in this entry automatically.
4	QSH	Starting hour (-1 to request immediate initiation). START, TRNON, and CYCLE fill in this value for you. If you use FQTASK, you must fill in this value yourself.
5	QSMS	Starting second within the hour (reserved but ignored if QSH equals -1). If QSH isn't equal to -1 and you call FQTSK, you must fill in this value.
6	QPRI	Task ID (left byte) and task priority (right byte). When you call ASSOCIATE, the routine fills in this value automatically. If you call FQTSK, you must fill it in yourself.
7	QRR	Rerun time increment in seconds. If QNUM equals 1 or -1 and you call FQTSK, you must fill in this value.
8	QTLNK	Maintained by the system.
9	QOCH	Overlay channel number (ignored if QTOV equals -1). If you call FQTSK, the routine fills in this entry when necessary.
10	QCOND	Conditional/unconditional overlay load flag (ignored if QTOV equals -1). If the initial subroutine of the task(s) to be initiated resides in an overlay, and if you call FQTSK, fill in this value. (-1 causes unconditional loading of the overlay, and 0 specifies conditional loading).
11	reserved	Reserved but not currently used.
12	Q.EPA	Task entry point address. Maintained by FORTRAN 5.
13	Q.MEM	Task stack (partition) size. When you call ASSOCIATE, the routine fills in the entry automatically. If you call FQTSK, you must fill in this entry.

Premature Termination of a Request

You can prematurely terminate a task initiation request. Specify the request by referring to the task identifier. (If a task has no identifier, ID=0, you can't specify it.) When the queue table leaves the operating system's queue, its information does not change. You can cause the removed queue table to be enqueued again with or without modification.

Never use the queue table before the operating system removes it from the queue.

Timing

Synchronizing the components of delayed and periodic task initiation is important. Generally, a task initiation doesn't necessarily start at the second you requested. It may start later. The starting time associated with a task initiation request is accurate only to one second.

In some routines you can specify time units. The routines, START and CYCLE, convert the time interval to seconds, rounding any fraction up to the next whole second. For example, if you call START and specify any nonzero delay, the task will start at a future second. If you call CYCLE and specify a cycle time of a fraction of a second, the cycle time is converted to one second.

You may request task initiation for future days. If you supply a starting time earlier than the current time, the request designates task initiation for that time the following day. When you specify a starting time later than midnight, the request will appear as a certain day, (hours/24) and hour (hour-(hours/24)*24). Hour= 24*D+H specifies Day (D) and Hour (H).

Three of the values associated with the timing of task initiation, QNUM, QSH, and QRR, are unsigned. Before describing these values we remind you FORTRAN 5 represents the values 32768 through 65534 as -32768 through -2. Minus 1 represents 65535.

If the value for QNUM is -1, an unlimited number of tasks will be initiated. For the value 0, error ERQTS (Error in User Task Queue Table) returns. Otherwise, the values of QNUM are between 1 and 65534.

QSH, the starting hour, will cause the task to be initiated immediately if its value is -1. Otherwise, its values lie between 0 and 65534.

QRR, the rerun time increment based on seconds, has values all lying between 0 and 65535.

The following is an alphabetical list of all runtime routines you will find in this chapter:

ASSOCIATE	Associates a queue table with a task initiation request.
CANCL	Prematurely removes a queued request.
CYCLE	Requests periodic task initiation.
DQTASK	Alias for DQTSK.
DQTSK	Prematurely removes a queued request.
FQTASK	Alias for FQTSK.
FQTSK	Queues a request for delayed or periodic execution.
START	Queues a request for task initiation after a specified delay.
TRNON	Queues a request for task initiation at a specific time.

ASSOCIATE

associates a queue table with a task initiation request.

Format

```
CALL ASSOCIATE (subroutine name,queue table,
  task ID,priority,stack size,ier)
```

Arguments

subroutine name is the name of the task's initial task You must include this name in an EXTERNAL statement.

queue table is a 13-word aggregate.

task ID is an integer that specifies the task's identification number.

priority is an integer that specifies the task's initial priority.

stack size is an integer that specifies the stack size for the task.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Example

```
INTEGER IRAY (13)
EXTERNAL SUB06
      .
      .
      .
CALL ASSOCIATE (SUB06,IRAY,3,1,0,IER)
CALL CHECK (IER)
```

CANCL

prematurely removes a queued request.

Format

```
CALL CANCL (queue table,ier)
```

Arguments

queue table is a 13-word aggregate.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERTID	The queue table you provided contained a task ID of 0, or no such request was found.

Example

```
INTEGER IRAY(13)
      .
      .
      .
CALL CANCL (IRAY,IER)
CALL CHECK (IER)
```

Notes

This call removes the queue table from the queue of requests but doesn't alter the information within the table.

Reference

.DQTSK (Task call)

CYCLE

requests periodic task initiation.

Format

CALL CYCLE (queue table,cycle time,time units,ier)

Arguments

queue table is an aggregate that contains the queue table associated with this task.

cycle time is an integer that specifies the number of time units between initiations.

time units is an integer that has one of the following values:

0	Basic System Units (Real-time clock ticks)
1	Milliseconds
2	Seconds
3	Minutes
4	Hours

ier is an integer variable that receives the routine's completion status code.

Rules

Before using this routine, you must call ASSOCIATE.

Error Conditions

Error codes that may return in ier include:

ERQTS	Illegal information in User Task Queue Table.
FERTC	No real-time clock.
FEITU	Illegal time units code.

Example

```
INTEGER ITIME(17)
EXTERNAL SUB06
        .
        .
        .
CALL ASSOCIATE (SUB06,ITIME,3,1,0,IER)
CALL CYCLE (ITIME,5,0,IER)
CALL CHECK (IER)
```

Reference

.QTSK (Task call)

DQTSK (Alias: DQTASK)

prematurely removes a queued request.

Format

CALL DQTSK (task ID,ier)

Arguments

task ID is an integer that specifies the task's identification number.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERTID	Either you specified a task ID of 0, or no such task ID was found.

Example

```
CALL DQTSK (INUM,IER)
CALL CHECK (IER)
```

Notes

This call removes the queue table from the queue of requests but doesn't alter the information within the table.

Reference

.DQTSK (Task call)

FQTSK (Alias: FQTASK)

queues a request for delayed or periodic task initiation.

Format

CALL FQTSK (*[overlay name,]* subroutine name, queue table,ier)

Arguments

overlay name is the name of the overlay in which the task resides. You must include this name in an EXTERNAL statement.

subroutine name is the name of the task's initial subroutine. You must include this name in an EXTERNAL statement.

queue table is an aggregate that contains the queue table associated with this task.

ier is an integer variable that receives the routine's completion status code.

Rules

You must insert certain information into the queue table before calling this routine (see the notes at the beginning of this chapter).

Error Conditions

Error codes that may return in ier include:

ERQTS	Illegal information in User Task Queue Table.
ERQOV	TOVL not loaded for an overlay queued task.

Example

```
INTEGER IRAY (13)
EXTERNAL SUBR1
      .
      .
      .
CALL FQTSK (SUBR1,IRAY,IER)
CALL CHECK (IER)
```

Notes

Each time a call to this routine creates and activates a new task, it's your responsibility to eventually kill the task. If the task resides in an overlay, you must release the overlay area.

Reference

.QTSK (Task call)

START

queues a request for task initiation after a specified delay.

Format

CALL START (queue table,delay time,time units,ier)

Arguments

queue table is an aggregate that contains the queue table associated with this task.

delay time is an integer that specifies the number of time units you want to delay before executing this task.

time units is an integer that has one of the following values:

0	Basic System Units (Real-time clock ticks)
1	Milliseconds
2	Seconds
3	Minutes
4	Hours

ier is an integer variable that receives the routine's completion status code.

Rules

Before using this routine, you must call ASSOCIATE.

Error Conditions

Error codes that may return in ier include:

ERQTS	Illegal information in User Task Queue Table.
FERTC	No real-time clock.
FEITU	Illegal time units code.

Example

```
INTEGER ITSK(13)
      .
      .
      .
CALL START (ITSK,10,2,IER)
CALL CHECK (IER)
```

Reference

.QTSK (Task call)

TRNON

queues a request for task initiation at a specified time.

Format

CALL TRNON (queue table,time array,ier)

Arguments

queue table is an aggregate that contains the queue table associated with this task request.

time array is an integer array whose first three elements contain the hours, minutes, and seconds of the task initiation time.

ier is an integer variable that receives the routine's completion status code.

Rules

Before using this routine, you must call ASSOCIATE.

Error Condition

The error code that may return in ier is:

ERQTS	Illegal information in User Task Queue Table.

Example

```
      INTEGER IRAY(13), ITIME(3)
      EXTERNAL SUB06
            .
            .
            .
      CALL ASSOCIATE (SUB06,IRAY,3,1,0,IER)
      CALL CHECK (IER)
C     SET UP ITIME ARRAY
      CALL TRNON (IRAY,ITIME,IER)
      CALL CHECK (IER)
```

Reference

.QTSK	(Task call)

End of Chapter

Chapter 17
Task/Operator Communications in a Multitask Environment

At the system console ($TTI, $TTO), you can pass messages to a task or receive messages from a task when you call one of two runtime routines, TRDOP and TWROP. When you call either of these routines, do not call either RDOPR or WROPR, detailed in Chapter 21. They also read and write from a console but only in a singletask environment.

FORTRAN 5 supports the RDOS task/operator communications facility, OPCOM, with the runtime routines: IOPC and IOPROG. Runtime routine IOPC initializes the OPCOM package. You call runtime routine IOPROG once for each program description you want known to OPCOM. Then the RUN or QUE commands can invoke OPCOM. For information on how the operator uses the OPCOM package, consult the *RDOS Reference Manual,* Chapter 5, *Task/Operator Communications Module (OPCOM).* Remember if you are using OPCOM, do not issue a call to either RDOPR or WROPR.

The following is an alphabetical list of all runtime routines you will find in this chapter:

IOPC	Initiates operator communications.
IOPROG	Adds a program description to the program table array.
TRDOP	Reads a message from the console.
TWROP	Writes a message to the console.

IOPC

prepares for operator communications.

Format

CALL IOPC (program table,frame count,queue area,
queue table count,ier)

Arguments

program table is an aggregate containing program frames. Each frame can contain information about a program to which a RUN or QUE OPCOM command can refer by its program number.

frame count is an integer that gives the number of program descriptions the program table can contain.

queue area is an aggregate that contains queue tables built by OPCOM as they are needed to execute programs as tasks.

queue table count is an integer that gives the number of queue tables the queue area may contain. (The maximum is 255.)

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERNOT No TCB's available.

Example

Refer to the example in IOPROG.

Notes

When the operator issues a RUN or QUE command, OPCOM takes a free queue table from the queue area and sets it up with information from the RUN or QUE command and from the appropriate program frame (the one corresponding to the program number given in the RUN or QUE command). The task-queueing facility of RDOS then initiates the task either once or many times at appropriate intervals.

Use IOPC if you want to run and queue programs. This provides the operator with the full repertoire of OPCOM commands, once appropriate calls to IOPROG have set up the program table.

Allocate a program table array sufficiently large to contain the desired number of program descriptions. Each program description requires 8 words in the program table aggregate. In addition, one extra word is required to hold a table terminator. Thus, if n is the maximum number of programs that are to be described in the program table, the size of the program table aggregate must be at least (8*n+1) words. Do not concern yourself with the detailed layout of the program table.

The queue area aggregate that you allocate is used by OPCOM as a pool of queue tables. Each RUN or QUE command awaiting execution requires one queue table. The length of a queue table is (currently) 13 words. Thus, if m is the maximum number of RUN and QUE commands that will be awaiting execution simultaneously, the size of the queue table aggregate must be (13*m) words. Using OPCOM, you need not concern yourself with the detailed layout of a queue table.

Reference

.IOPC (Task call)

IOPROG

adds a program description to the program table.

Format

CALL IOPROG (subroutine name,program number, task ID, priority, *[stack size,] [overlay name,load flag,]* ier)

Arguments

subroutine name is the initial subroutine the task will execute.

program number is an integer that gives the number by which the RUN or QUE commands can refer to the program.

task ID is an integer specifying the identification number to assign to the task. It's stored as part of the program description. When a program is run or queued, it uses this identification number for the initiated task unless a task with this ID already exists, in which case it must be an ID of 0.

priority is an integer that specifies the initial priority for the task unless the priority argument is specified in the RUN or QUE command. The priority is stored as part of the program description.

stack size is an integer that gives the stack size you want to allocate to the program when it is executed as a task. If you omit this argument, a stack size parameter of 0 is used.

overlay name is the name of the overlay in which the program resides. You must include this name in an EXTERNAL statement in the program calling IOPROG. Include it also in an OVERLAY statement in a subprogram that resides in the overlay. Omit this argument if the program is core-resident.

load flag is an integer which is -1 when the overlay in which the initial subroutine for the program resides is to be loaded unconditionally. Omit this argument if the program is core-resident.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFOP	File not open.
ERMEM	Insufficient memory to execute program.
FEPRI	Illegal task priority.
FETID	Illegal task identifier.

Example

```
INTEGER PTABLE(25),QTABLE(65)
CALL OVOPN ("EXAMP.OL",IER)
CALL CHECK (IER)
CALL IOPC (PTABLE,3,QTABLE,5,IER)
CALL CHECK (IER)
EXTERNAL SUBR1,SUBR2,SUBR3
CALL IOPROG (SUBR1,1,1,5,IER)
CALL CHECK (IER)
CALL IOPROG (SUBR2,2,2,6,OVLY1,0,IER)
CALL CHECK (IER)
CALL IOPROG (SUBR3,3,3,6,OVLY1,0,IER)
CALL CHECK (IER)
```

Notes

Once you call IOPC, the operator may issue commands to OPCOM. However, RUN and QUE commands are not meaningful until you set up the appropriate program descriptions in the program table by using IOPROG. You may only add as many program descriptions as you specified in number of programs in the call to IOPC. If you attempt to add too many, an error message will occur.

If any of the programs to be added to the program table by calls to IOPROG reside in overlays, open the overlay file with a call to OVOPN before making the call to IOPC, or else IOPROG will give an error message. For a program residing in an overlay, it remains your responsibility to release the overlay area when the task is done executing.

TRDOP

reads a message from the console.

Format

```
CALL TRDOP (message,byte count,ier)
```

Arguments

message is an aggregate that receives the ASCII message.

byte count is an integer variable that receives the byte count of the message. This includes the carriage return.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERNOT	No TCB's available.
ERMPR	Address is outside address space (mapped systems only).
EROPM	Operator messages not specified at SYSGEN time.

Example

```
CALL TRDOP (IMESS,ICNT,IER)
CALL CHECK (IER)
```

Notes

A task issuing a call to TRDOP may reside in either the foreground or the background. More than one task within a program may issue an outstanding request for a task message.

Reference

.TRDOP (Task call)

TWROP

writes a message to the console.

Format

```
CALL TWROP (message,flag,ier)
```

Arguments

message is an ASCII string less than 125 characters in length including a carriage return terminator.

flag is -1 if you want the task's identification number printed with the message, and is -1 if you want it suppressed.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERMPR	Address is outside address space (mapped systems only).
EROPM	Operator messages not specified at SYSGEN time.

Example

```
CALL TWROP ((FLAG IS 10),1,IER)
```

Notes

The system outputs two exclamation points and an F (if the calling task is in the foreground) or a B (if the calling task is in the background). Following is the calling task's identification number unless you indicated -1 in flag, followed by the message itself.

Reference

.TWROP (Task call)

End of Chapter

Chapter 18
User/System Clock Commands

Although you may call any of the FORTRAN 5 system clock routines in either a singletask or multitask environment, the routines are of little use in a singletask environment.

The following is an alphabetical list of all runtime routines you will find in this chapter:

FDELAY	Delays a task for a given number of clock ticks.
FDELY	Alias for FDELAY.
GFREQ	Alias for GHRZ.
GHRZ	Obtains the real-time clock frequency.
WAIT	Suspends a task for a specified time.

FDELAY (Alias: FDELY)

delays a task for a given number of clock ticks.

Format

```
CALL FDELAY (ticks)
```

Argument

ticks is an integer that specifies the number of clock ticks you want to delay the calling task.

Error Condition

The error condition that may result is:

FERTC	No real-time clock.

Example

```
CALL FDELAY (10)
```

Notes

An error causes termination of your program.

Reference

.DELAY	(System call)

GHRZ (Alias: GFREQ)

gets the real-time clock frequency.

Format

```
CALL GHRZ (frequency)
```

Argument

frequency is an integer variable that receives one of the following values:

Code	Meaning
0	There is no real-time clock in this system.
1	Frequency is 10 HZ.
2	Frequency is 100 HZ.
3	Frequency is 1000 HZ.
4	Frequency is 60 HZ (line frequency).
5	Frequency is 50 HZ (line frequency).

Error Conditions

No error conditions are currently defined.

Example

```
CALL GHRZ (IFREQ)
```

Reference

.GHRZ	(System call)

WAIT

suspends a task for a specified amount of time.

Format

CALL WAIT (delay count,time units,ier)

Arguments

delay count is an integer that specifies the number of time units you want to suspend the task.

time units is an integer that specifies the delay count unit measurement; it may have one of the following values:

0	Basic System Units (Real-time clock ticks)
1	Milliseconds
2	Seconds
3	Minutes

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

FEITU	Illegal time units code.
FERTC	No real-time clock.

Example

```
CALL WAIT (IPULS,0,IER)
CALL CHECK (IER)
```

Reference

.DELAY	(System call)

End of Chapter

Chapter 19
Enabling and Disabling the Multitask Environment

In a normal multitask environment, the tasks you initiate will compete for CPU control according to their relative priorities. Although you can assign highest priority to one task, it may need an even greater execution privilege. In this situation, you can temporarily disable the multitask environment. This allows the task to continue executing but prohibits all other tasks from gaining CPU control.

The task gains privileged control by calling the subroutine, SINGLETASK. After the call, the task can issue system and task calls as usual with the guarantee that no other task can gain control. The task retains privileged control until it calls subroutine MULTITASK, kills itself, or suspends itself. (A system call class of suspension is the exception.) Then the multitask environment is re-enabled, and all tasks again compete for CPU control.

Don't disable the multitask environment unless you must ensure that a task's sequence of operations are indivisible relative to other tasks' execution. If you need not prevent other tasks from executing but want one task to receive primary consideration for execution, assign that task highest priority. If you must deny other tasks access to a critical resource, such as a sensitive database, but don't need to restrict task execution outside this resource, use the XMT/REC mechanism.

The privileged control a task receives when it calls SINGLETASK is not sole control of the CPU. Input/output interrupts still occur; any interrupt routines you defined will continue executing.

When you call SINGLETASK in a foreground/background operation, the subroutine disables the multitask environment for one ground only. The tasks in the other ground execute as usual.

We describe the following runtime routines in this chapter:

MULTITASK	Enables the multitask environment.
SINGLETASK	Disables the multitask environment.

MULTITASK

re-enables the multitask environment.

Format

CALL MULTITASK

Arguments

None.

Error Conditions

No error conditions are currently defined.

Notes

When a task calls MULTITASK, it relinquishes privileged control of the CPU. Tasks can then compete for CPU control.

SINGLETASK

disables the multitask environment.

Format

CALL SINGLETASK

Arguments

None.

Error Conditions

No error conditions are currently defined.

Notes

This call gives a task privileged control of the CPU.

End of Chapter

Chapter 20
Calling To and Returning From Programs

Your program may require memory beyond your allocated user address space. RDOS facilities allow you to segment programs so they fit within the constraints of available memory. In unmapped systems these facilities are overlays, program swaps, and chaining.

Program swaps permit disk file images to overwrite resident memory or core images. They are save files containing core images of total user address space. The files exist in up to five levels where one level calls another. The Command Line Interpreter (CLI) usually executes at the highest level in the system, level zero.

Any program executing under the operating system can suspend its own execution and invoke another program. The suspended program is stored temporarily on disk. Upon termination of the program swap (execution of the lower level program), operation of the calling program (the higher level program) is resumed where it left off. This operation is called *returning.*

Besides program swaps, *chaining* segments programs to extend the amount of address space available to a program. A calling program can subdivide into separately executable segments. The total program size exceeds the largest possible user address space many times over. Chaining enables one program segment to call for another segment of the whole program. This segment in turn may call for another segment. The entire program exists at the same program level.

For more information on program swapping, returning, and chaining, see the *RDOS Reference Manual,* Chapter 4. Pay particular attention to the list of calls and conditions that a change of programs terminates.

The following is an alphabetical list of all runtime routines you will find in this chapter:

CHAIN	Transfers control to a program at the current program level.
CHECK	Checks the status returned from a routine.
COMARG	Reads one argument string and its switches from a file in COM.CM format.
EBACK	Terminates a program and indicates an error.
ERROR	Outputs a message to the console with error traceback and terminates execution of the current program.
EXIT	Terminates a program with no error.
FCHAN	Transfers control to a program at the current program level.
FSWAP	Transfers control to a program at the next program level.
GETERR	Obtains the ISA status code for the error most recently causing an ERR= or END= branch.
MESSAGE	Outputs a message to the console with error traceback and continues execution of the current program.
SWAP	Transfers control to a program at the next program level.

CHAIN

transfers control to a program at the current program level.

Format

CALL CHAIN (filename,ier)

Arguments

filename is an aggregate containing the name of the program that receives control.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERSV1	File requires "S"ave attribute.
ERDLE	File does not exist.
ERMEM	Attempt to allocate more memory than is available.
ERADR	Illegal starting address.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERUSZ	Too few channels defined at load time or SYSGEN time.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).

Example

```
CALL CHAIN ("FIL.SV",IER)
CALL CHECK (IER)
```

Reference

.EXEC (System call)

CHECK

checks the status code returned from a routine.

Format

CALL CHECK (error)

Argument

error is an integer variable that has received a routine's completion status code returned as ier by a prior call to an ISA-style routine.

Error Conditions

If the error status code is 1, return is made with no error reported; otherwise, the error condition is reported and the program is terminated.

Example

```
CALL CHECK (IER)
```

COMARG

reads one argument string and its switches from a file in COM.CM format.

Format

CALL COMARG (unit number,string, *[switches,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of the file in COM.CM format.

string is an aggregate that receives a string of ASCII characters.

switches is an aggregate that receives two words of switch information.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Illegal command for device.
EREOF	End of file.
ERRPR	Attempt to read a read-protected file.
ERFOP	Attempt to refer to a file not opened.
ERLLI	Exceeded line limit (132 nonterminator characters).
ERPAR	Parity error.
ERFIL	File read error.
ERRD	Attempt to read into system area.
ERDIO	File accessible by direct block I/O only.
ERSIM	Simultaneous reads on same QTY line.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERMCA	Attempt to perform an MCA read on a channel where reading is currently in progress.
ERCLO	QTY/MCA input terminated by channel close.
FEIFN	Illegal unit number.

Example

Assume you entered MYPROG/C JOE/D HARRY to start execution of the current program MYPROG. The following statements show a possible application of calls to COMARG:

```
OPEN 0,"COM.CM"
      .
      .
CALL COMARG (0,ITEXT,ISWITCH,IER)
      .
      .
CALL COMARG (0,ITEXT,ISWITCH,IER)
      .
      .
CALL COMARG (0,ITEXT,ISWITCH,IER)
```

In the first call to COMARG, ITEST receives the ASCII string MYPROG, and ISWITCH receives the value of switch C. In the second call, ITEXT receives the ASCII string JOE and ISWITCH receives the value of switch D. In the third call, ITEXT receives the ASCII string HARRY. Note that the three calls to COMARG have identical formats.

Reference

See Appendix C of the *RDOS Command Line Interpreter Reference Manual* for more information on the command file, COM.CM.

EBACK

terminates a program and indicates an error.

Format

CALL EBACK (error)

Argument

error is an integer that specifies a routine's completion status code; typically, it is a code returned in ier by a prior call to an ISA-style subroutine.

Error Conditions

No error conditions are currently defined.

Example

```
CALL DFILW ("FILE20",IER)
IF (IER .NE. 1) CALL EBACK (IER)
CALL CHECK (IER)
```

Notes

Upon execution of a call to EBACK, unconditional return is made to the next higher level program. If the next higher level program is the CLI, one of the following occurs:

1. If the error status code is an ISA-style error code or a FORTRAN 5 error code, the appropriate message is displayed, taken from the parameter table F5ERR.FR (see Appendix A of this document).

2. If a null error occurs (ERNUL), the CLI does not report a message.

3. If the error EREXQ is returned, the CLI takes its next command from disk file CLI.CM or FCLI.CM.

4. If you specify one of your own error status codes, i.e., if the code is not recognized by the CLI, UNKNOWN ERROR CODE *n* is displayed, where *n* is your error status code in octal.

Reference

.ERTN (System call)

ERROR

outputs a message to the console with error traceback and terminates execution of the current program.

Format

CALL ERROR (error message)

Argument

error message is an aggregate that contains your message.

Error Conditions

No error conditions are currently defined.

Example

```
CALL ERROR ("FATAL ERROR FROM PHASE 2A")
```

Notes

Upon execution of a call to ERROR, error traceback is performed. Control then passes to the console where USER ERROR: is displayed, followed by the error message you specified. A core image of the program is then saved in BREAK.SV. Return is made to the next higher level program.

Reference

.ERTN (System call)

EXIT

terminates a program with no error.

Format

CALL EXIT

Arguments

None.

Error Conditions

No error conditions are currently defined.

Reference

.RTN (System call)

FCHAN

transfers control to a program at the current program level.

Format

CALL FCHAN (filename)

Arguments

filename is an aggregate that contains the name of the program to which you want to transfer control.

Error Conditions

The error conditions that may result include:

ERFNM	Illegal filename.
ERSV1	File requires "S"ave attribute.
ERDLE	File does not exist.
ERMEM	Attempt to allocate more memory than is available.
ERADR	Illegal starting address.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERUSZ	Too few channels defined at load time or SYSGEN time.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).

Example

```
CALL FCHAN ("KEEPR.SV")
```

Notes

The error conditions cause termination of the calling program.

Reference

.EXEC (System call)

FSWAP

transfers control to a program at the next (lower) program level.

Format

CALL FSWAP (filename)

Argument

filename is an aggregate specifying the name of the program that receives control.

Error Conditions

The error conditions that may result are:

ERFNM	Illegal filename.
ERSV1	File requires "S"ave attribute.
ERDLE	File does not exist.
ERCM3	Trying to push too many levels.
ERMEM	Attempt to allocate more memory than is available.
ERADR	Illegal starting address.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERUSZ	Too few channels defined at load time or SYSGEN time.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).

Example

```
CALL FSWAP ("FILE.SV")
```

Notes

When calling this routine in a multitask environment, invoke the SINGLETASK routine first to freeze the environment. To reinstate the multitask environment, invoke MULTITASK on return from the swap.

Any error causes termination of the calling program.

Reference

.EXEC (System call)

GETERR

obtains the ISA status code for the error most recently causing the ERR= or END= branch to be taken.

Format

CALL GETERR (var)

Argument

var is an integer receiving the status code. This code may be passed to CHECK for reporting the error.

Error Conditions

No error conditions are currently defined.

Example

```
      READ FREE(11,ERR=100)X,Y,Z
      .
      .
      .
100   CALL GETERR(I)
      IF(I.EQ.ERSPC) GOTO 200
```

Notes

GETERR resets the internal error value. This is your only option for resetting it. GETERR returns a value of 1 if no error has occurred since the last call to it.

Since the initializer sets the error value to 1, the first call to GETERR returns the value 1 if no error has occurred.

MESSAGE

outputs a message to the console with error traceback and continues execution of the current program.

Format

CALL MESSAGE (error message)

Argument

error message is an aggregate that contains your message.

Error Conditions

No error conditions are currently defined.

Example

```
CALL MESSAGE ("NONFATAL ERROR #17")
```

Notes

Upon execution of a call to MESSAGE, error traceback is performed. Control then transfers to the console where USER MESSAGE is displayed, followed by the error message you specified. Return is made to the program that invoked the MESSAGE routine.

SWAP

transfers control to a program at the next (lower) program level.

Format

CALL SWAP (filename,ier)

Arguments

filename is an aggregate containing the name of the program that receives control.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERSV1	File requires "S"ave attribute.
ERDLE	File does not exist.
ERMEM	Attempt to allocate more memory than is available.
ERADR	Illegal starting address.
ERDNM	Device code exceeds 77 octal (returned by .DEBL).
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERUSZ	Too few channels defined at load time or SYSGEN time.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
ERENA	No linking allowed (N attribute).

Example

```
CALL SWAP ("FILE.SV", IER)
CALL CHECK (IER)
```

Notes

When calling this routine in a multitask environment, invoke the SINGLETASK routine first to freeze the environment. To reinstate the multitask environment, invoke MULTITASK on return from the swap.

Reference

.EXEC (System call)

End of Chapter

Chapter 21
Using Extended Memory

FORTRAN 5 provides you with a method for using more than 32K words of memory. It interfaces with the window mapping facility. If you are unfamiliar with this facility, refer to the *RDOS Reference Manual.*

The FORTRAN 5 interface consists of nine routines. Five of the routines correspond to the operating system primitives and enable you to use the window mapping facility provided by RDOS. Following are the five routines: VMEM, returns the available amount of extended memory; MAPDF, defines the window map; REMAP, remaps one or more of the window blocks to extended memory blocks; ERDB, reads specified block(s) from a disk file into extended memory; and EWRB, writes a series of blocks from extended memory to a specified file.

Of the four remaining routines, VLOAD copies the contents of a disk file into extended memory, whereas VDUMP copies all of extended memory to a specified disk file. In these two routines, you need only specify the unit number. If the system does not have sufficient space to accomplish its job, it returns to you the number of 256-word blocks that were read or written successfully. VFETCH and VSTASH enable you to treat extended memory in a manner similar to a FORTRAN array. With the initializing MAPDF call, you select an element size for this array (default-size is one word). Then you address extended memory merely by specifying the index of the element desired. For example, to set up a large double precision array in extended memory, set the element size to 4, and then address four-word chunks of extended memory (treated by the program as double precision numbers) with the indexes 1, 2, 3, ... etc. With this scheme, all extended memory appears as one large, singly-subscripted array.

Using Extended Memory in a Multitask Environment

A single program-- a single ground-- defines only one window map. Several tasks may share the same window map. The FORTRAN 5 interfaces to the window mapping facility do the following:

- allow multiple tasks to concurrently access the same database in extended memory; and
- enable you to implement a scheme in which each of several tasks has its own window map. The interfaces provide you with access to the system window mapping primitives for implementation.

Several tasks may concurrently access extended memory through the VFETCH and VSTASH calls. These calls remap window block 0 to the appropriate map block, then transfer a block of words. While one task accesses extended memory through window block 0, another task activity is suspended through the .DRSCH/.ERSCH task calls. (Refer to the *RDOS Reference Manual,* Chapter 5, *Task Calls.)* This means that a task's access of extended memory is protected from other tasks' accesses, if the blocks of words accessed all lie in the same 1024-word, extended memory block. If the block of words you want transferred spans two or more extended memory blocks, other tasks may gain control and access extended memory. This occurs when a given task must perform another remapping operation to bring the next portion of the desired block of extended memory into the window.

When the task with control of the VFETCH/VSTASH code completes each (possibly partial) transfer, it unlocks the code. The highest priority task awaiting the code, if any, then locks the code and proceeds with its next remap and transfer. Thus, all tasks which call VFETCH and VSTASH use window block 0 as a *scratch* window block. Your program must not remap window block 0 on its own if it is using VFETCH or VSTASH.

You can create a separate window map for each of several tasks by issuing a call to REMAP. In this case, allocate a window of *n* blocks in the call to MAPDF, where *n* is the number of tasks accessing extended memory through their own window blocks. Then assign one of these window blocks to each task for its exclusive use; each task performs its own REMAP calls. (The remapping code is not locked into the multitask case. Note also that there is no protection if two or more tasks access the same block in the map through their separate window blocks, which is entirely possible and permissible.)

You may find it useful to implement a combination of the above two techniques. A task could then use the VFETCH/VSTASH facility as well as having its own window block. In this case, you should define the window size as $n+1$ blocks; VFETCH and VSTASH would use block 0, and blocks 1 through *n* would be reserved as the individual window blocks for the *n* different tasks. In a situation like this, all tasks may not need a window block of their own. Thus you can reduce the window size significantly.

The following is an alphabetical list of runtime routines you will find in this chapter:

CVF	COMPLEX form for VFETCH.
DCVF	DOUBLE PRECISION complex form for VFETCH.
DVF	DOUBLE PRECISION form for VFETCH.
ERDB	Reads a series of blocks from a contiguous or random disk file into extended memory.
ERDBLK	Alias for ERDB.
EWRB	Writes a series of blocks from extended memory to a contiguous or random disk file.
EWRBLK	Alias for EWRB.
IVF	INTEGER form for VFETCH.
MAPDF	Defines a window map or redefines the permanent element size.
REMAP	Alters the mapping of window blocks to extended memory blocks.
VDUMP	Copies all of extended memory to a disk file.
VF	REAL form for VFETCH.
VFETCH	Fetches one or more elements from extended memory.
VLOAD	Initializes all extended memory to the contents of a disk file.
VMEM	Determines the amount of extended memory available to a program.
VS	Alias for VSTASH.
VSTASH	Copies one or more elements into extended memory.

ERDB (Alias: ERDBLK)

reads a series of 256-word blocks from a contiguous or random disk file into extended memory.

Format

CALL ERDB (unit number,disk block,mem block, block count *[partial count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of a contiguous or random disk file.

disk block is an integer that specifies the initial disk block number you want to read. The first block is 0.

mem block is an integer that specifies the initial extended memory or quarter-block (a 256-word portion of a full 1024-word memory block) number into which data is read.

block count is an integer that specifies the number of disk blocks you want to transfer (the maximum is 255).

partial count is an optional integer variable that receives the number of quarter-blocks transferred successfully in the event of an end-of-file condition.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICM	Illegal command.
ERICD	Illegal command for device.
ERSV1	Not a randomly or contiguously organized file.
EREOF	End of file.
ERRPR	File is read protected.
ERFOP	File not open.
ERFIL	File data error.
EROVA	File not accessible by direct block I/O.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.
FEBLN	Illegal extended memory block number.
FEBLC	Illegal block count.

Example

```
CALL ERDB (3,6,0,4,ICNT,IER)
CALL CHECK (IER)
```

This call reads disk blocks 6, 7, 8, and 9 from unit 3 into the first 1024-word block of extended memory. If the disk file is shorter than 10 disk blocks, ICNT will receive the number of blocks that were successfully read.

Reference

.ERDB (System call)

EWRB (Alias: EWRBLK)

writes a series of blocks from extended memory to a contiguous or random disk file.

Format

CALL EWRB (unit number,disk block,mem block, block count *[partial count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number of a contiguous or random disk file.

disk block is an integer that specifies the initial disk block number to which data is written. The first block is 0.

mem block is an integer that specifies the initial extended memory or quarter-block (a 256-word portion of a full 1024-word memory block) number from which data is written.

block count is an integer that specifies the number of disk blocks of data you want to transfer (the maximum is 255).

partial count is an optional integer variable that receives the number of quarter-blocks transferred successfully in the event of an end-of-file condition or when disk file space is exhausted.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICM	Illegal command (unmapped systems only).
ERICD	Illegal command for device.
ERSV1	Not a randomly or contiguously organized file.
EREOF	End of file.
ERWPR	File is write protected.
ERFOP	File not open.
ERSPC	Disk file space exhausted.
EROVA	File not accessible by direct I/O.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.
FEBLN	Illegal extended memory block number.
FEBLC	Illegal block count.

Example

```
CALL EWRB (1,6,3,5,ICNT,IER)
CALL CHECK (IER)
```

This call writes the last quarter of the first 1024-word extended memory block and the entire second memory block to relative disk blocks 6 through 10 on FORTRAN 5 unit 1. If the disk file is contiguous and shorter than 11 blocks in length, or if disk file space is exhausted in the course of writing, ICNT will receive the number of disk blocks successfully written. (If the disk file is a random file of length less than 11 blocks, the system will attempt to extend the length of the file.)

Reference

.EWRB (System call)

MAPDF

defines a window map or redefines the permanent element size.

Formats

CALL MAPDF (count,window array,window size, *[element size,]* ier)

CALL MAPDF (element size,ier)

Arguments

count is an integer that specifies the total number of blocks of memory you want to allocate for window mapping use, including blocks already in the window and additional extended memory blocks.

window array is an aggregate in your program through which references to extended memory are made.

window size is an integer that specifies the size of the window in 1024-word blocks.

element size is an integer that specifies the size of an element in words. (If omitted, the element size is 1.)

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICM	Illegal command (if given in an unmapped environment).
ERMEM	Attempt to define too large a map.
FEW1K	Window aggregate does not begin on a 1K-word boundary.

Examples

1.
```
INTEGER WINDOW (2048)

CALL MAPDF (7,WINDOW,2,4,IER)
CALL CHECK (IER)
```

The above call sets up a window map using a total of 7 blocks of memory. Since the aggregate WINDOW is 2K (2048) words in size, the number of additional extended memory blocks to be allocated is 5. You must set up the array, WINDOW, in common and load it such that it begins on a 1K boundary. (See the example later on for how to do this with the RLDR /N switch.) The element size is set to 4, meaning that extended memory will be accessed in multiples of 4 words at a time.

2.
```
CALL VMEM (N,IER)
CALL CHECK (IER)

CALL MAPDF (N+1,IWIND,1,IER)
CALL CHECK (IER)
```

The above two calls allocate all available extended memory for use in window mapping. The window, IWIND, consists of a single 1K-word block. The total number of blocks participating in the window mapping will be n + 1. The element size is set to 1 by default.

3.
```
      .
      .
C           SET UP WINDOW MAPPING
C           ;DEFINE ELEMENT SIZE TO BE 3
CALL MAPDF (7,WINDOW,1,3,IER)
      .
      .
            ;ACCESS ELEMENTS OF SIZE 3
CALL MAPDF (5,IER)

      .
      .
            ;ACCESS ELEMENTS OF SIZE 5
CALL MAPODF (3,IER)
      .
      .
            ;ACCESS ELEMENTS OF SIZE 3
```

Notes

Only one window and map can exist within a program. Since the window is block-aligned, the window array you define must start at the beginning of a block.

Note that there are two forms of this call. You may use the first form only once in a program; you cannot change the values that it established, except for the element size, which you may change by using the second form of the MAPDF call.

Reference

.MAPDF (System call)

REMAP

alters the mapping of window blocks to extended memory blocks.

Formats

CALL REMAP (starting window block,
starting map block, *[number of blocks,]* ier)

CALL REMAP (block number,ier)

Arguments

starting window block is an integer that specifies the number of the starting block in the window you want to map.

starting map block is an integer that specifies the number of the starting block in the map to which blocks in the window will be mapped.

number of blocks is an integer that specifies the number of consecutively numbered blocks you want to remap; if you omit this argument, one block is remapped.

block number is an integer that specifies the block number in the map to which window block 0 should be mapped.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERADR	Illegal REMAP parameters.
FEBLN	Block number in window or map exceeds 255.

Examples

1.
```
CALL REMAP (3,IER)
CALL CHECK (IER)
```

Map block 0 in the window to block 3 in the map.

2.
```
CALL REMAP (O,3,IER)
CALL CHECK (IER)
```

Same as Example 1.

3.
```
CALL REMAP (2,0,3,IER)
CALL CHECK (IER)
```

Map blocks 2, 3, and 4 in the window to blocks 0, 1, and 2 in the map, respectively.

Notes

Note that there are two forms of this call. The first form remaps any number of consecutively numbered window blocks to consecutively numbered map blocks. The second form of the call is intended mainly for windows of a single block, block 0. However, it may also remap block 0 of a multiple-block window.

Reference

.REMAP (Task call)

VDUMP

copies all extended memory to a disk file.

Format

CALL VDUMP (unit number, *[block count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number on which the disk file was opened.

block count is an integer variable that receives the number of disk blocks successfully written if the write operation cannot complete due to an end-of-file condition (for a contiguous file) or if disk file space is exhausted.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNO	Illegal channel number.
ERICM	Illegal command (unmapped system).
ERICD	Illegal command for device.
ERSV1	Not a randomly or contiguously organized file.
EREOF	End of file.
ERWPR	File is write-protected.
ERFOP	File not open.
ERSPC	Disk file space exhausted.
EROVA	File not accessible by direct block I/O.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
CALL VDUMP (2,ICNT,IER)
CALL CHECK (IER)
```

This call dumps all extended memory currently defined to the disk file opened on FORTRAN 5 unit 2. (The previous contents of the disk file are lost.) If disk file space is exhausted during the write operation, or if the disk file is contiguous and smaller in size than the total size of defined extended memory, then the number of 256-word disk blocks that were successfully written are returned in ICNT.

Reference

.EWRB (System call)

VFETCH

copies one or more elements from extended memory.

Format

CALL VFETCH (data area,index *[,elements [,size]]*)

Arguments

data area is an aggregate that defines the area into which data from extended memory is read.

index is an integer that specifies the index of the element you want to fetch, or the index of the first of a number of consecutive elements you want to fetch. The first element is 1.

elements is an integer that specifies the number of elements you want to fetch (if omitted, one element is fetched).

size is an integer that specifies the element size for the current transfer (if omitted, the permanent element size given by the most recent call to MAPDF is used).

Error Condition

The error code that may result is:

FEEOB	Extended memory reference out of bounds.

Examples

All of the following examples assume that the element size has been set to 4 by a prior MAPDF call, and that the variable DX and the array DY are declared double precision.

1. CALL VFETCH (DX,1200)

This call fetches into variable DX the 1200th 4-word element of extended memory (that is, the words at offsets 4796 through 4799).

2. CALL VFETCH (DY(12),2500,3)

This call fetches DY(12), DY(13), and DY(14) from the 2500th, 2501st, and 2502nd 4-word elements of extended memory.

3. CALL VFETCH (I,4797,1,1)

This call fetches the 4797th word (at offset 4796) of extended memory into the integer variable I. *size* temporarily overrides the permanent element size of 4. (Note that you must give *elements,* even though it is 1, because you gave *size.*)

```
4.      DOUBLE PRECISION DVF
        .
        .
        DX = DVF(4500)
        DY(64) = DVF(5875)
```

This example shows how you can use the alternate entry points of VFETCH to make VFETCH act as a function. The first of the two assignment statements fetches the 4500th 4-word element in extended memory into variable DX. The second statement fetches the 5875th 4-word element of extended memory into double precision array element DY(64).

```
5.      DOUBLE PRECISION DVF
        C     INTEGER IVF          (IMPLICITLY TYPED)

        C     STATEMENT FUNCTIONS TO SIMULATE
              ARRAYS IN EXTENDED MEMORY

        C     IVA LOOKS LIKE A 200 X 20 INTEGER
              ARRAY (FOR READING)

              IVA (I,J) = IVF (20*(J-1)+I,1,1)

        C     DVA LOOKS LIKE A 3000-ELEMENT
              DOUBLE PRECISION
        C     ARRAY WHICH OCCUPIES EXTENDED
              MEMORY FOLLOWING THE
        C     4000-WORD INTEGER "ARRAY" IVA
              DVA (I) = DVF (1000+I)
              .
              .
              CALL MAPDF (N,IWIND,4,IER)
              .
              .
              DX = DVA (4500)
              DY (64) = DVA (875)

              J = IVA (K,L) + IVA (L,K)
              .
              .
```

As illustrated above, you can use statement functions to make array references in extended memory appear syntactically identical to ordinary subscripted array references. Essentially, VFETCH treats extended memory as a vector (i.e., linearly subscripted). Therefore, all that is necessary to provide a syntax for multiple subscripting is a statement function which performs the mapping of a multiply-subscripted indexing function into a linear subscript. A prime advantage of this syntax for VFETCH is that you can have several implicit calls to VFETCH in a single line (as shown above in the assignment to J).

Note that there is no equivalent syntax for VSTASH. In the definition of statement function IVA, you must pass an element size of 1 explicitly, because the permanent element size has already been defined to be 4 in these examples. For the same reason, no element size need be passed in calls to VFETCH under the guise of DVF.

Notes

The action of VFETCH is to read *elements* * *size* words from extended memory, beginning at offset (index-1 * *size*, into the FORTRAN 5 aggregate data area.

Alternate entry points IVF, VF, DVF, CVF, and DCVF allow you to use VFETCH as functions of several data types. You can use IVF when VFETCH is to act as an integer function (when a single word is to be fetched from extended memory), and use VF in the real case (when fetching 2-word elements), etc. However, all entry points to VFETCH (including VFETCH itself) are functionally identical. See examples 4 and 5 for use of the more readable, more array-like syntax for VFETCH. Note that you must declare DVF, CVF, and DCVF DOUBLE PRECISION, COMPLEX, or DOUBLE PRECISION COMPLEX (respectively), if used.

VLOAD

initializes all of extended memory to the contents of a disk file.

Format

CALL VLOAD (unit number, *[block count,]* ier)

Arguments

unit number is an integer that specifies the FORTRAN 5 unit number on which the disk file was opened.

block count is an integer variable that receives the number of disk blocks successfully read if the read operation cannot be completed due to an end-of-file condition.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICM	Illegal command.
ERICD	Illegal command for device.
ERSV1	Not a randomly or contiguously organized file.
EREOF	End of file.
ERRPR	File is read-protected.
ERFOP	File not open.
ERFIL	File data error.
EROVA	File not accessible by direct block I/O.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.
FEIFN	Illegal unit number.

Example

```
CALL VLOAD (2,ICNT,IER)
CALL CHECK (IER)
```

This call loads all extended memory currently defined with the contents of the disk file opened on FORTRAN 5 unit 2.

If the disk file is smaller than the amount of extended memory currently defined, then the number of 256-word disk blocks that were read successfully is returned in ICNT.

Reference

.ERDB (System call)

VMEM

determines the amount of extended memory available to a program.

Format

CALL VMEM (count,ier)

Arguments

count is an integer variable that receives the number of free 1024-word blocks of extended memory.

ier is an integer variable that receives the routine's completion status code.

Error Condition

The error code that may return in ier is:

ERICM	Illegal command (if given in an unmapped environment).

Example

```
CALL VMEM (I,IER)
CALL CHECK (IER)
```

Reference

.VMEM (System call)

VSTASH (Alias: VS)

copies one or more elements into extended memory.

Format

CALL VSTASH (data area,index *[,elements [,size]]*)

Arguments

data area is an aggregate that defines the area from which data is written into extended memory.

index is an integer that specifies the index of the position in extended memory where the element is to be copied, or the index of the position in extended memory where the first of a number of consecutive elements is to be copied.

elements is an integer that specifies the number of elements you want to copy (if omitted, one element is copied).

size is an integer that specifies the element size for the current transfer (if omitted, the permanent element size given by the most recent call to MAPDF is used).

Error Condition

The error code that may return in ier is:

FEEOB	Extended memory reference out of bounds.

Examples

Usage of VSTASH is identical to that of VFETCH as described in examples 1, 2, and 3. Note that you may not use VSTASH or VS as a function, as you may with VFETCH and its alternate entry points. (See the last two examples, 4 and 5, under VFETCH.)

Notes

VSTASH transfers *elements* * *size* words from the FORTRAN 5 aggregate data area to extended memory, beginning at offset (index-1) **size*.

End of Chapter

Chapter 22
Foreground/Background Programming

An RDOS multiprogram environment allows you to run two programs at once: a foreground program and a background program. These programs exist independently, and each one has its own task scheduler. These two programs can have equal priority, or you may give the foreground program a higher priority. If the foreground has higher priority, it does not pass control to the background program until no ready task exists in the foreground.

Systems with the Memory Management and Protection Unit (MMPU) provide an absolute hardware protection to separate foreground and background programs. When you bootstrap an RDOS system in a mapped environment, the CLI is brought into execution in the background. At this point there is no foreground program loaded, so all available memory is allocated to the background. After you issue the CLI command, SMEM, to allocate some memory to the foreground, you may load and execute save files in the foreground in one of two ways. Either issue the CLI command, EXFG, or design the running background program to call the FORTRAN 5 runtime routine, EXFG.

In an unmapped environment, you separate the foreground and background by the same procedure. However, hardware does not protect the address space of the two programs in an unmapped system; software partitions differentiate the programs. For example, a system failure results when a background program returns to a higher level background program that requires memory occupied by the foreground program.

Checkpointing temporarily interrupts one background program, so a background program with higher priority can execute. You can use checkpointing by issuing a call to the FORTRAN 5 runtime routine, EXBG, within a running foreground program.

To determine whether a program is in the foreground or the background, use the FORTRAN 5 runtime routine, GROUND.

Foreground and background programs can communicate via a Multiprocessor Communications Adapter (MCA) line. Alternatively, foreground and background programs may each define a communications area within their program areas to transmit messages to the other via a FORTRAN 5 runtime routine. To set up a communications area, use the runtime routine, ICMN; to read messages from and write messages to this area, use the runtime routines, RDCMN and WRCMN, respectively. To determine whether or not a program is running in the foreground, issue a call to the runtime routine, FGND.

The runtime routines, RDOPR and WROPR, allow you to read messages from and write messages to the console. However, keep in mind that foreground and background programs cannot read from or write to the same output device simultaneously. When reading, there is no way to divert a stream of data to two different programs; when writing, data is spooled to the device by the first user to issue the request.

If you are not familiar with RDOS foreground/background programming, Chapter 6 in the *RDOS System Reference Manual* gives information that is essential for using the FORTRAN 5 foreground/background runtime routines efficiently.

The following is an alphabetical list of all runtime routines you will find in this chapter:

EXBG	Checkpoints a mapped background program.
EXFG	Loads and executes a program in the foreground.
FGND	Determines whether or not a foreground program is running.
GROUND	Determines if the current program is in the foreground or the background.
ICMN	Defines a message area for interground communication.
RDCMN	Readies a message from the other ground's communication area.
RDOPR	Reads a message from the console.
WHERE	Alias for GROUND.
WRCMN	Writes a message into the other ground's communication area.
WROPR	Writes a message to the console.

EXBG

checkpoints a mapped background program; i.e., suspends one background program temporarily in order to execute another background program.

Format

```
CALL EXBG (filename,priority,ier)
```

Arguments

filename is an aggregate that contains the name of the background program you want to execute.

priority has one of the following values:

0	Gives the checkpoint program the same priority as the old background program.
1	Gives the checkpoint program the same priority as the current foreground program.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERICM	Attempt to checkpoint in an unmapped system.
ERSV1	File requires "S"ave attribute.
ERDLE	File does not exist.
ERUFT	Not enough channels.
ERCM3	Trying to checkpoint a new background program.
ERMEM	Attempt to allocate more memory than is available.
ERDSN	Directory specifier unknown.
ERLDE	Link depth exceeded.
ERDNI	Directory not initialized.
ERUSZ	No room for UFT's.
ERMPR	Address is outside address space (mapped systems only).
ERNTE	Program to be checkpointed is not checkpointable, or attempt to create two outstanding checkpoints.
ERDTO	Ten second disk time-out occurred.

Example

```
CALL EXBG ("PROG10.SV",0,IER)
CALL CHECK (IER)
```

Notes

Only a foreground program may issue the checkpoint call and it must be in a mapped environment.

Execution of a checkpointed program is resumed when either a CTRL-A or CTRL-C is detected, or when the new background program terminates. If a keyboard interrupt is detected, the message CP INT is issued on $TTO. When the checkpointed program is restored, the message CP RTN is issued on $TTO.

Reference

.EXBG (System call)

EXFG

loads and executes a program in the foreground.

Format

CALL EXFG (filename,priority,ier)

Arguments

filename is an aggregate that contains the name of the program you want to load and execute.

priority is one of the following:

0	Gives the foreground program a higher priority than the background program.
1	Gives the foreground program the same priority as the background program.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERSV1	File requires "S"ave attribute.
ERDLE	File does not exist.
ERUFT	Not enough channels.
ERMEM	Attempt to allocate more memory than is available.
ERADR	Illegal starting address.
ERDSN	Directory specifier unknown.
ERDNI	Directory not initialized.
ERFGE	Foreground already exists.
ERUSZ	No room for UFT's.
ERMPR	Address is outside address space (mapped systems only).
ERDTO	Ten second disk time-out occurred.

Example

```
CALL EXFG ("FGPROG.SV",0,IER)
CALL CHECK (IER)
```

Notes

You must issue a call to EXFG from a currently running background program.

Reference

.EXFG (System call)

FGND

determines whether or not a foreground program is running; determines the level at which a program is running.

Format

CALL FGND (code, *[level]*)

Arguments

code is an integer variable that receives one of the following:

0	A foreground program is not running.
1	A foreground program is running.

level is an integer variable that receives one of the following:

Code	Meaning
1	Background level 0
2	Background level 1
3	Background level 2
4	Background level 3
5	Background level 4
6	Foreground level 0
7	Foreground level 1
8	Foreground level 2
9	Foreground level 3
10	Foreground level 4

Error Conditions

No error conditions are currently defined.

Example

```
CALL FGND (ICODE,ILEVEL)
```

Reference

.FGND (System call)

GROUND (Alias: WHERE)

determines whether the current program is in the foreground or in the background.

Format

CALL GROUND (code)

Argument

code is an integer variable that receives one of the following:

0 Current program is in background.
1 Current program is in foreground.

Error Conditions

No error conditions are currently defined.

Example

```
CALL GROUND (I)
```

ICMN

defines an area for interground communication.

Format

CALL ICMN (communications area,length,ier)

Arguments

communications area is an aggregate that defines an area used for sending messages to or receiving messages from a program in the other ground.

length is an integer that specifies the size of the communications area in words (no greater than 256 decimal).

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERCMS	Communications area exceeds the program size or attempt to overwrite the system.
ERMPR	Address is outside address space (mapped systems only).

Example

```
DIMENSION MESSAG(200)
        .
        .
CALL ICMN (MESSAG,200,IER)
CALL CHECK (IER)
```

Reference

.ICMN (System call)

RDCMN

reads a message from the other ground's communication area.

Format

```
CALL RDCMN (message area,offset,count,ier)
```

Arguments

message area is an aggregate that receives the message.

offset is an integer that specifies the word offset within the other ground's communications area where the message begins.

count is an integer that specifies the number of words you want to read.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERCMS	The size of the requested message exceeds the communications area size.
ERCUS	No communications area is defined in the other program.
ERMPR	Address is outside address space (mapped systems only).

Example

```
DIMENSION MESARY(200)
      .
      .
CALL RDCMN (MESARY,100,100,IER)
CALL CHECK (IER)
```

Reads the second half of the message.

Notes

The length of the message cannot exceed 256 (decimal) words.

Reference

.RDCM (System call)

RDOPR

reads a message from the console.

Format

```
CALL RDOPR (message area,count,ier)
```

Arguments

message area is an aggregate that receives the message.

count is an integer variable that receives the total number of bytes read (including the terminator) if the normal return is taken.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICM	Operator messages not specified.
ERMPR	Address is outside address space (mapped systems only).

Example

```
DIMENSION MESAGE(67)
      .
      .
CALL RDOPR (MESAGE,ICNT,IER)
```

Notes

You may call RDOPR from either the foreground or the background. Only one message may be outstanding at any one moment in the foreground and one in the background.

The first character in the message must be a CTRL-E character (echoed as an exclamation point); if no system call to read an operator message was issued, a bell is sounded when CTRL-E is depressed. The second character must be either F or B to indicate whether the foreground or the background program is to receive the message. Any other character entered will sound the console bell and no further text will be accepted until you type F or B. The message then follows; the last character must be a carriage return, form feed, or null. The total message length (including the terminator) can be up to 132 characters.

To cancel a message, a RUBOUT must be depressed in column 2 instead of F or B.

Do not issue a call to RDOPR if you are using OPCOM or the task message runtime routines TWROP and TRDOP (see Chapter 15).

Reference

.RDOP (System call)

WRCMN

writes a message into the other ground's communication area.

Format

CALL WRCMN (message,offset,count,ier)

Arguments

message is an aggregate that contains the message.

offset is an integer that specifies the word offset within the other ground's communications area where the message begins.

count is an integer that specifies the number of words you want to write.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERCMS	Message too large for communications area.
ERCUS	No communications area is defined in the other program.
ERMPR	Address is outside address space (mapped systems only).

Example

```
DIMENSION MESSAG(100)
      .
      .
CALL WRCMN (MESSAG,20,10,IER)
CALL CHECK (IER)
```

Notes

The length of the message cannot exceed 256 (decimal) words.

Reference

.WRCM (System call)

WROPR

writes a message to the console.

Format

CALL WROPR (message,ier)

Arguments

message is an aggregate that contains the message you want to write.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICM	Operator messages not specified.
ERMPR	Address is outside address space (mapped systems only).

Example

```
INTEGER STRNG(67)
      .
      .
CALL WROPR (STRNG,IER)
```

Notes

The message may exist in either the foreground or the background. Only one outstanding write command may exist in the foreground and one in the background.

The message is displayed as two exclamation points, followed by F or B to indicate whether the origin of the message is in the foreground or the background respectively, followed by the message and message terminator.

Do not issue a call to WROPR if you are using OPCOM or the task message runtime routines TWROP and TRDOP (see Chapter 15).

Reference

.WROP (System call)

End of Chapter

Chapter 23
Multiple Processor Routines

The following is an alphabetical list of the runtime routines you will find in this chapter:

BOOT	Bootstraps a new system, partition, or device.
GMCA	Obtains the CPU unit number of the multiprocessing communications adaptors (MCA) unit with the primary device code.
GMCA1	Obtains the CPU unit number of the MCA unit with the secondary device unit.

BOOT

bootstraps a new system, partition, or device.

Format

```
CALL BOOT (unit name,ier)
```

Arguments

unit name is an aggregate that contains the name of the system, partition, or device.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERDLE	Unit name does not exist.
ERRTN	File RESTART.SV does not exist yet; data switches were all set for restarting without operator intervention.
ERDSN	Unknown specifier.
ERMPR	Address is outside address space (mapped system only).
ERDTO	Ten second disk time-out occurred.
ERSFA	Spool file is active.

Example

```
CALL BOOT ("DP0",IER)
CALL CHECK (IER)
```

Notes

This call closes all open files in the currently executing system (both foreground and background), releases all directories and resets all system I/O.

Reference

.BOOT	(System call)

GMCA

gets the MCA unit number of the primary MCA on the CPU executing the program.

Format

CALL GMCA (unit number,ier)

Arguments

unit number is an integer variable that receives the MCA unit number.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Improper device code.
ERDNM	Device not in system (i.e., no MCA was specified at SYSGEN time in this operating system).

Example

```
CALL GMCA (IUNIT,IER)
CALL CHECK (IER)
```

Notes

There may be two MCAs in a given system. A call to GMCA receives the code of the first MCA, and a call to GMCA1 receives the code of the second MCA.

Reference

.GMCA (System call)

GMCA1

gets the MCA unit number of the secondary MCA on the CPU executing the program.

Format

CALL GMCA1 (unit number,ier)

Arguments

unit number is an integer variable that receives the number of the secondary CPU unit.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERICD	Improper device code input to system call.
ERDNM	Device not in system (i.e., no MCA was specified at SYSGEN time in this operating system).

Example

```
CALL GMCA1 (IUNIT2,IER)
CALL CHECK (IER)
```

Reference

.GMCA (System call)

End of Chapter

Chapter 24
Controlling Spooling

SPOOLing, Simultaneous Peripheral Operation On-Line, is implemented for the following devices: $LPT, $LPT1, $PTP, $PTP1, $TTO, $TTO1, $TTP, and $TTP1.

The following is an alphabetical list of all runtime routines you will find in this chapter:

SPEBL	Enables spooling for a specified device.
SPDIS	Disables spooling for a specified device.
SPKILL	Kills a currently existing spooling operation.

SPEBL
enables spooling for a specified device.

Format

```
CALL SPEBL (device name,ier)
```

Arguments

device name is an aggregate that contains the name of a spoolable device.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERICD	Illegal command for device.
ERDNM	Device not in system.
ERMPR	Address is outside address space (mapped systems only).

Example

```
CALL SPEBL ("$LPT",IER)
CALL CHECK (IER)
```

Reference

.SPEA (System call)

SPDIS
disables spooling for a specified device.

Format

```
CALL SPDIS (device name,ier)
```

Arguments

device name is an aggregate that contains the name of a spoolable device.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERICD	Illegal command for device.
ERDNM	Device not in system.
ERMPR	Address is outside address space (mapped systems only).

Example

```
CALL SPDIS ("$LPT1",IER)
CALL CHECK (IER)
```

Reference

.SPDA (System call)

SPKILL

kills a currently existing spooling operation.

Format

```
CALL SPKILL (device name,ier)
```

Arguments

device name is an aggregate that contains the name of a spoolable device.

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERFNM	Illegal filename.
ERICD	Illegal command for device.
ERDNM	Device not in system.
ERMPR	Address is outside address space (mapped systems only).

Example

```
CALL SPKILL ("$TTO",IER)
CALL CHECK (IER)
```

Reference

.SPKL (System call)

End of Chapter

Chapter 25
User Devices and Interrupts

You can issue calls to set up user devices and interrupt service routines in either a singletask or multitask environment, and from either the foreground or the background.

The following is an alphabetical list of all runtime routines you will find in this chapter:

DUCLK	Defines a user clock.
FINRV	Removes a user interrupt device introduced by FINTD.
FINTD	Introduces a device to generate user-processed interrupts.
RUCLK	Removes a previously defined user clock.

DUCLK
defines a user clock.

Format

CALL DUCLK (frequency,service routine,ier)

Arguments

frequency is an integer that specifies the number of system clock ticks that elapse between user clock interrupts.

service routine is a symbol that names either a page zero location which contains a pointer to the service routine, or the address of the routine itself. (This symbol must be declared EXTERNAL in the program calling DUCLK.)

ier is an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in ier include:

ERIBS	A user clock already exists.
ERMPR	Address is outside address space (mapped systems only).

Example

```
EXTERNAL IROUT
      .
      .
      .
CALL DUCLK (IFREQ,IROUT,IER)
CALL CHECK (IER)
```

Notes

The service routine must be written in assembly language and may not use the FORTRAN 5 stack or state variables and the floating point unit.

Reference

.DUCLK (System call)

FINRV

removes a user interrupt device introduced by FINTD.

Format

CALL FINRV (device)

Argument

device is an integer that gives the code of the user device to remove.

Error Condition

The error condition that may result is:

ERDNM	Illegal device code (greater than 77 octal) or attempt to remove a SYSGENed device.

Example

```
CALL FINRV (66)
```

Reference

.IRMV (System call)

FINTD

introduces a device to generate user-processed interrupts.

Format

CALL FINTD (device,table)

Arguments

device is an integer that specifies the device code of the user device you want to introduce.

table is an aggregate that contains the device control table.

Error Conditions

Error conditions that may result are:

ERDNM	Illegal device code (greater that 77 octal).
ERIBS	Interrupt device code in use or ten user devices already outstanding.
ERDCH	Insufficient room in data channel map.
ERMPR	Address is outside address space (mapped systems only).

Example

```
CALL FINTD (9,ITBLE)
```

Reference

.IDEF (System call)

RUCLK
removes a previously defined user clock.

Format

CALL RUCLK (ier)

Argument

ier is an integer variable that receives the routine's completion status code.

Error Condition

An error code that may return in ier is:

ERIBS	No user clock is defined.

Example

```
CALL RUCLK (IER)
CALL CHECK (IER)
```

Reference

.RUCLK (System call)

End of Chapter

Appendix

Appendix A
F5ERR.FR - FORTRAN 5
Runtime Error Parameters

System Errors

```
PARAMETER

ERFNO=   3     ,;(SYSTEM)
         ;ILLEGAL CHANNEL NUMBER

ERFNM=   4     ,;(SYSTEM)
         ;ILLEGAL FILENAME

ERICM=   5     ,;(SYSTEM)
         ;ILLEGAL SYSTEM COMMAND

ERICD=   6     ,;(SYSTEM)
         ;ILLEGAL COMMAND FOR DEVICE

ERSV1=   7     ,;(SYSTEM)
         ;NOT A SAVE FILE

ERWR0=   8     ,;(SYSTEM)
         ;ATTEMPT TO WRITE AN EXISTENT FILE

EREOF=   9     ,;(SYSTEM)
         ;END OF FILE

ERRPR=   10    ,;(SYSTEM)
         ;FILE READ PROTECTED

ERWPR=   11    ,;(SYSTEM)
         ;FILE WRITE PROTECTED

ERCRE=   12    ,;(SYSTEM)
         ;FILE ALREADY EXISTS

ERDLE=   13    ,;(SYSTEM)
         ;FILE DOES NOT EXIST

ERDE1=   14    ,;(SYSTEM)
         ;PERMANENT FILE

ERCHA=   15    ,;(SYSTEM)
         ;FILE ATTRIBUTE PROTECTED

ERFOP=   16    ,;(SYSTEM)
         ;FILE NOT OPEN

ERFUE=   17    ,;(SYSTEM)
         ;FATAL UTILITY ERROR

EREXQ=   18    ,;(SYSTEM)
         ;EXECUTE CLI.CM (NO ERROR)

ERNUL=   19    ;(SYSTEM)
         ;INVISIBLE ERROR CODE

ERUFT=   20    ,;(SYSTEM)
         ;CHANNEL ALREADY IN USE

ERLLI=   21    ,;(SYSTEM)
         ;LINE TOO LONG

ERRTN=   22    ,;(SYSTEM)
         ;ATTEMPT TO RESTORE A
         NON-EXISTENT IMAGE

ERPAR=   23    ,;(SYSTEM)
         ;PARITY ERROR

ERCM3=   24    ,;(SYSTEM)
         ;PUSH DEPTH EXCEEDED

ERMEM=   25    ,;(SYSTEM)
         ;INSUFFICIENT MEMORY TO EXECUTE
         PROGRAM

ERSPC=   26    ,;(SYSTEM)
         ;FILE SPACE EXHAUSTED

ERFIL=   27    ,;(SYSTEM)
         ;FILE DATA ERROR

ERSEL=   28    ,;(SYSTEM)
         ;UNIT IMPROPERLY SELECTED

ERADR=   29    ,;(SYSTEM)
         ;NO STARTING ADDRESS

ERRD=    30    ,;(SYSTEM)
         ;ATTEMPT TO READ INTO SYSTEM
         SPACE

ERDIO=   31    ,;(SYSTEM)
         ;DIRECT I/O ACCESS ONLY

ERDIR=   32    ,;(SYSTEM)
         ;FILES MUST EXIST IN THE SAME
         DIRECTORY
```

```
ERDNM=  33    ,;(SYSTEM)
        ;DEVICE NOT IN SYSTEM

EROVN=  34    ,;(SYSTEM)
        ;ILLEGAL OVERLAY NUMBER

EROVA=  35    ,;(SYSTEM)
        ;NO DIRECT I/O

ERTIM=  36    ,;(SYSTEM)
        ;INVALID TIME OR DATE

ERNOT=  37    ;(SYSTEM)
        ;OUT OF TCB'S

ERXMT=  38    ,;(SYSTEM)
        ;SIGNAL TO BUSY ADDRESS

ERSQF=  39    ,;(SYSTEM)
        ;SQUASH FILE ERROR

ERIBS=  40    ,;(SYSTEM)
        ;DEVICE ALREADY IN SYSTEM

ERICB=  41    ,;(SYSTEM)
        ;INSUFFICIENT CONTIGUOUS BLOCKS

ERSIM=  42    ,;(SYSTEM)
        ;QTY ERROR

ERQTS=  43    ,;(SYSTEM)
        ;ERROR IN USER TASK QUEUE TABLE

ERNMD=  44    ,;(SYSTEM)
        ;NO MORE DCB'S

ERIDS=  45    ,;(SYSTEM)
        ;ILLEGAL DIRECTORY NAME

ERDSN=  46    ,;(SYSTEM)
        ;NO SUCH DIRECTORY

ERD2S=  47    ,;(SYSTEM)
        ;DIRECTORY SIZE INSUFFICIENT

ERDDE=  48    ,;(SYSTEM)
        ;DIRECTORY DEPTH EXCEEDED

ERDIU=  49    ,;(SYSTEM)
        ;DIRECTORY IN USE

ERLDE=  50    ,;(SYSTEM)
        ;LINK DEPTH EXCEEDED

ERFIU=  51    ,;(SYSTEM)
        ;FILE IN USE

ERTID=  52    ,;(SYSTEM)
        ;TASK ID ERROR

ERCMS=  53    ,;(SYSTEM)
        ;COMMON SIZE ERROR

ERCUS=  54    ,;(SYSTEM)
        ;COMMON USAGE ERROR

ERSCP=  55    ;(SYSTEM)
        ;FILE POSITION ERROR

ERDCH=  56    ,;(SYSTEM)
        ;INSUFFICIENT ROOM IN DATA CHANNEL
        MAP

ERDNI=  57    ,;(SYSTEM)
        ;DIRECTORY NOT INITIALIZED

ERNDD=  58    ,;(SYSTEM)
        ;NO DEFAULT DEVICE

ERFGE=  59    ,;(SYSTEM)
        ;FOREGROUND ALREADY RUNNING

ERMPT=  60    ,;(SYSTEM)
        ;ILLEGAL PARTITION VALUE

EROPD=  61    ,;(SYSTEM)
        ;DIRECTORY SHARED

ERUSZ=  62    ,;(SYSTEM)
        ;NO ROOM FOR UFTS

ERMPR=  63    ,;(SYSTEM)
        ;ADDRESS ERROR IN .SYST ARGUMENT

ERNLE=  64    ,;(SYSTEM)
        ;NOT A LINK ENTRY

ERNTE=  65    ,;(SYSTEM)
        ;CANNOT CHECKPOINT CURRENT BG

ERSDE=  66    ,;(SYSTEM)
        ;SYS.DR ERROR

ERMDE=  67    ,;(SYSTEM)
        ;MAP.DR ERROR

ERDTO=  68    ,;(SYSTEM)
        ;DEVICE TIME OUT

ERENA=  69    ,;(SYSTEM)
        ;LINK ACCESS NOT ALLOWED

ERMCA=  70    ,;(SYSTEM)
        ;MCA REQUEST OUTSTANDING

ERSRR=  71    ,;(SYSTEM)
        ;TRANSMISSION TERMINATED BY
        RECEIVER
```

```
ERSDL=   72   ,;(SYSTEM)
         ;SYSTEM DEADLOCK

ERCLO=   73   ;(SYSTEM)
         ;CHANNEL CLOSED BY ANOTHER TASK

ERSFA=   74   ,;(SYSTEM)
         ;SPOOL FILES ACTIVE

ERABT=   75   ,;(SYSTEM)
         ;TASK NOT FOUND FOR ABORT

ERDOP=   76   ,;(SYSTEM)
         ;DEVICE PREVIOUSLY OPENED

EROVF=   77   ,;(SYSTEM)
         ;SYSTEM STACK OVERFLOW

ERNMC=   78   ,;(SYSTEM)
         ;NO MCA RECEIVE REQUEST
         OUTSTANDING

ERNIR=   79   ,;(SYSTEM)
         ;ATTEMPT TO RELEASE AN OPEN
         DEVICE

ERXMZ=   80   ,;(SYSTEM)
         ;A ZERO .XMT OR .IXMT MESSAGE

ERCANT=  81   ,;(SYSTEM)
         ;YOU CAN'T DO THAT

ERQOV=   82   ,;(SYSTEM)
         ;TOVLD NOT LOADED FOR QUEUED
         OVERLAY TASKS

EROPM=   83   ,;(SYSTEM)
         ;OPERATOR MESSAGES NOT SYSGENED

ERFMT=   84   ,;(SYSTEM)
         ;DISK FORMAT ERROR

ERBAD=   85   ,;(SYSTEM)
         ;INVALID BAD BLOCK TABLE

ERBSPC=  86   ,;(SYSTEM)
         ;INSUFFICIENT SPACE IN BAD BLOCK
         POOL (CORE)

ERZCB=   87   ,;(SYSTEM)
         ;ATTEMPT TO CREATE A ZERO LENGTH
         CONTIGUOUS FILE

ERNSE=   88   ,;(SYSTEM)
         ;PROGRAM NOT SWAPPABLE

ERBLT=   89   ;(SYSTEM)
         ;BLANK TAPE

FESOV=   3076 ,;(FATAL)
         ;STACK OVERFLOW

FEDAT=   3077 ,;(FATAL)
         ;INSUFFICIENT ARGUMENTS FOR DATA
         INITIALIZATION

FESBS=   3078 ,;(FATAL)
         ;SUBSCRIPT OUT OF BOUNDS

FEFMT=   3079 ,;(RECOVERABLE)
         ;ILLEGAL FORMAT ITEM

FEINM=   3080 ,;(RECOVERABLE)
         ;ILLEGAL INPUT NUMBER

FERCL=   3081 ,;(RECOVERABLE)
         ;OUTPUT RECORD TOO LONG

FERCS=   3082 ,;(RECOVERABLE)
         ;INPUT RECORD TOO SHORT

FEIFN=   3083 ,;(RECOVERABLE)
         ;ILLEGAL UNIT NUMBER

FEATT=   3084 ,;(RECOVERABLE)
         ;INVALID OR INCONSISTENT FILE
         ATTRIBUTE

FESEK=   3085 ,;(RECOVERABLE)
         ;RECORD FILE REQUIRED FOR SEEK

FESTK=   3086 ,;(RECOVERABLE)
         ;ILLEGAL STACK SIZE

FEEVT=   3087 ,;(RECOVERABLE)
         ;ILLEGAL EVENT USAGE

FESQR=   3088 ,;(TRANSPARENT)
         ;ILLEGAL ARGUMENT FOR SQRT

FEEXP=   3089 ,;(TRANSPARENT)
         ;ILLEGAL ARGUMENT FOR EXP

FELOG=   3090 ,;(TRANSPARENT)
         ;ILLEGAL ARGUMENT FOR LOG

FEASC=   3091 ,;(TRANSPARENT)
         ;ILLEGAL ARGUMENT FOR ASIN OR
         ACOS

FEATN=   3092 ,;(TRANSPARENT)
         ;ILLEGAL ARGUMENT FOR ATAN2

FEPWR=   3093 ;(TRANSPARENT)
         ;ILLEGAL EXPONENTIATION

FEINT=   3094 ,;(TRANSPARENT)
         ;INTEGER OVERFLOW ON CONVERSION
```

FERTN= 3095 ,;(TRANSPARENT)
;INVALID RETURN

FEFNU= 3096 ,;(RECOVERABLE)
;UNIT NUMBER IN USE

FEMOP= 3097 ,;(RECOVERABLE)
;ILLEGAL MODE FOR OPEN

FERCR= 3098 ,;(RECOVERABLE)
;RECORD COUNT REQUIRED FOR CONTIGUOUS FILE CREATE-ON-OPEN

FEEOB= 3099 ,;(FATAL)
;EXTENDED MEMORY REFERENCE OUT OF BOUNDS

FEW1K= 3100 ,;(RECOVERABLE)
;WINDOW AGGREGATE DOES NOT BEGIN ON 1024-WORD BOUNDARY

FEBLN= 3101 ,;(RECOVERABLE)
;ILLEGAL BLOCK NUMBER

FEBLC= 3102 ,;(RECOVERABLE)
;ILLEGAL BLOCK COUNT

FENPC= 3013 ,;(RECOVERABLE)
;NO FILE PRECONNECTED TO UNIT NUMBER

FERLN= 3104 ,;(RECOVERABLE)
;ILLEGAL VALUE FOR RECORD LENGTH IN LEN= SPECIFIER

FELEF= 3105 ,;(FATAL)
;INCONSISTENT SPECIFICATION FOR LEF MODE

FEONO= 3106 ,;(RECOVERABLE)
;OVERLAY FILE NOT OPEN

FEOAO= 3107 ,;(RECOVERABLE)
;OVERLAY FILE ALREADY OPEN

FETID= 3108 ,;(RECOVERABLE)
;ILLEGAL TASK IDENTIFIER

FEPRI= 3109 ,;(RECOVERABLE)
;ILLEGAL TASK PRIORITY

FEEVN= 3110 ,;(RECOVERABLE)
;ILLEGAL EVENT NUMBER

FEPNA= 3111 ;(RECOVERABLE)
;REQUESTED PARTITION NOT AVAILABLE

FEITU= 3112 ,;(RECOVERABLE)
;ILLEGAL TIME UNITS CODE

FERTC= 3113 ,;(RECOVERABLE)
;NO REAL TIME CLOCK

FETMQ= 3114 ,;(RECOVERABLE)
;TOO MANY QUEUE BLOCKS SPECIFIED

FEFPU= 3115 ,;(RECOVERABLE)
;FLOATING POINT HARDWARE IS NOT PRESENT

FEMDV= 3116 ,;(RECOVERABLE)
;MULTIPLY/DIVIDE HARDWARE IS NOT PRESENT

FEMEM= 3117 ,;(RECOVERABLE)
;INSUFFICIENT MEMORY FOR FORTRAN 5 PROGRAM

FEIOP= 3118 ,;(RECOVERABLE)
;DID NOT ALLOW FOR IOPROG IN IOPC CALL

FEPTO= 3119 ;(RECOVERABLE)
;PROGRAM TABLE OVERFLOW

End of Appendix

Index

ABORT II-12-1, II-12-6
.ABORT II-12-3
abort task II-12-3
activation, routine I-3-1
  per-routine I-3-2
aggregates II-1-1
AKILL II-12-1, II-12-2
.AKILL II-12-2
allocation
  memory, foreground/background II-22-1
  memory, runtime I-3-1
  partition II-10-2
  user address space II-20-1
APPEND II-6-1, II-6-2
.APPEND II-6-2
ARDY II-12-1, II-12-2
.ARDY II-12-2
arithmetic errors, checking II-2-1
array II-21-1
  singly-subscripted II-21-1
assembly language routines I-1-2
  assembling I-1-3
  CLI commands I-1-3
  called from FORTRAN 5 I-3-5
  macros, examples I-3-6, I-3-7
  calling a FORTRAN 5 routine I-3-8
ASSOCIATE II-10-2, II-16-1, II-16-3, II-16-4
ASUSP II-12-1, II-12-3
.ASUSP II-12-3
attributes, file
  change II-5-2
  obtain II-5-5

background II-7-1, II-22-1
BACKSPACE II-6-1, II-6-3
bit
  clear II-3-2
  set II-3-3
  test II-3-4
blocks
  COMMON I-3-1, II-15-1
  I/O control I-3-2
  read II-21-3
  task control I-3-2
  task control block extensions I-3-2
  write II-21-4
BOOT II-23-1
.BOOT II-23-1
bootstrap
  BOOT II-23-1
calling to/returning from programs II-20-1
CANCL II-16-1, II-16-3, II-16-4
.CCONT II-5-1
CDIR II-4-1
CFIL II-5-1
CFILW II-5-1
CHAIN II-20-1, II-20-2
chaining II-20-1
channel number II-5-4
CHATR II-5-1, II-5-2
.CHATR II-5-2
CHECK II-20-1, II-20-2
checking, arithmetic errors II-2-1
checkpointing II-22-1, II-22-2
CHLAT II-5-1, II-5-2
.CHLAT II-5-2
CHNGE II-12-1, II-12-6
CHPRI II-12-1, II-12-6
CHRST II-6-1, II-6-3
CHSAV II-6-1, II-6-4
CHSTS II-5-1, II-5-3
.CHSTS II-5-3
clock
  real-time II-18-1
  user II-25-1
  user/system II-18-1
clock and calendar, system II-9-1
CLOSE II-6-1, II-6-4
.CLOSE II-6-4, II-6-5, II-6-12, II-15-2
close
  overlay file II-15-2
code, executable I-3-2
COMARG II-20-1, II-20-3
COMMON
  blocks I-3-1, II-15-1
  Storage II-16-1
Command Line Interpreter II-20-1, II-20-3
communication
  interground II-22-4
communication, intertask II-14-1
communications, task/operator II-17-1
compiling, FORTRAN 5 program under RDOS
  command line I-1-1
  examples I-1-2
  switches, global and local I-1-1, I-1-2
conditions, error II-1-2
configuration, main memory I-3-2
console input/output II-7-1

console
  Read/write messages II-17-4
  Task/operator communications II-17-1
console interrupts II-7-3
control
  transfer II-20-2, II-20-7
controlling spooling II-24-1
conventions
  documentation iii
  ISA II-1-1
  linkage, runtime routines I-3-5
copy
  extended memory II-21-7
  elements II-21-7, II-21-10
.CPAR II-4-2
CPART II-4-1, II-4-2
.CRAND II-5-1
create
  Disk file II-5-1
  secondary partition II-4-2
  subdirectory II-4-1
.CREATE II-5-1
CVF II-21-2, II-21-8
CYCLE II-16-1, II-16-3, II-16-5

data
  per-process I-3-1
  per-task I-3-2
DATE II-9-1
DATSW II-8-1
DCVF II-21-2, II-21-8
DEF I-3-5
DEFARGS I-3-5
.DELET II-5-3
DFIL II-5-1, II-5-3
DFILW II-5-1, II-5-3
DEFTMPS I-3-6
.DELAY II-18-1, II-18-2
delayed or periodic execution II-16-1
  initiation, termination request II-16-3
  synchronization II-16-3
DESTROY II-12-1, II-12-3
device
  initialization II-4-4
  release II-4-5
devices, directories managing II-4-1
DIR II-4-1, II-4-2
.DIR II-4-2
direct block mode II-6-1
directories and devices, managing II-4-1
directory
  initialization II-4-4
  release II-4-5
  resolution file II-5-7
directory devices, multiple II-4-1
disk file, create II-5-1
DPART.SR II-10-2
DQTASK II-16-2, II-16-5
DQTSK II-16-1, II-16-3, II-16-5
.DQTSK II-16-4, II-16-5
.DRSCH
DUCLK II-25-1
.DUCLK II-25-1
DVDCHK II-2-1
DVF II-21-2, II-21-8

EBACK II-20-1, II-20-4
enable, disable multitask environment II-19-1
END I-3-6
end zone I-3-4, I-3-3, II-10-1
environment
  multitask, changing task states II-12-1
  unmapped II-22-1
  multiprogram II-22-1
.EOPEN II-6-7
EQUIV II-4-1, II-4-3
.EQUIV II-4-3
ERDB II-21-1, II-21-2, II-21-3
.ERDB II-21-3, II-21-9
ERDBLK II-21-2, II-21-3
ERR=, END= I-2-1, I-2-2, I-3-2, II-20-6
ERROR II-20-1, II-20-4
error conditions II-1-2
error handling I-2-1
error(s)
  arithmetic II-2-1
  codes I-2-1
  conditions II-1-2
  EBACK II-20-4
  ERROR II-20-4
  files I-2-3
  GETERR II-20-6
  handling I-2-1
  messages I-2-3
  processing I-2-1
  reporter I-2-1
event mechanism II-10-1
EWRBLK II-21-2, II-21-4
.ERSCH
.ERTN II-20-4
EST II-15-1, II-15-2
EWRB II-21-1, II-21-2, II-21-4
.EWRB II-21-4, II-21-7
EXBG II-22-1, II-22-2
.EXBG II-22-2
.EXEC II-20-2, II-20-5, II-20-7
executable code I-3-2
execution, delayed or periodic II-16-1
  initiation, termination request II-16-3
  synchronization II-16-3
EXFG II-22-1, II-22-3
.EXFG II-22-3

EXIT II-20-1, II-20-5
extended memory II-21-1
  multitask environment II-21-1
  copy II-21-7
  initialize II-21-9
  available II-21-9
event
  mechanism II-10-1
  number, wakeup II-12-4
  number, obtain II-13-1, II-13-2

F5SYM.SR I-3-5, II-10-1
FBCKSP II-6-3
FCHAN II-20-1, II-20-5
FCLOSE II-6-1, II-6-5
FDELAY II-18-1
FDELETE II-5-1, II-5-4
FDELY II-18-1
FENTRY I-3-6
FGDAY II-9-1, II-9-2
FGND II-22-1, II-22-3
.FGND II-22-3
FGTIME II-9-1, II-9-2
file(s)
  attributes, change II-5-2
  close II-6-4, II-6-5, II-6-12
  delete II-5-3
  disk, create II-5-1
  error I-2-3
  I/O II-6-1
  maintaining II-5-1
  open II-6-5, II-6-7
  overlay II-15-1
  overlay, close II-15-2
  overlay, open II-15-4
  parameter I-3-5
  save II-20-1
  status II-5-3
FINRV II-25-1, II-25-2
FINTD II-25-1, II-25-2
.FIOPREP II-5-1, II-5-4
FMAC.SR I-3-5
FOPEN II-6-1, II-6-5
foreground II-7-1, II-22-1
foreground/background programming II-22-1
  interground communication II-22-4
  load program II-22-3
FORTRAN 5
  using FORTRAN 5 under RDOS I-1-1 to I-1-6
  compiler I-1-1
  program, load I-1-1
  compiling program under RDOS I-1-1 to I-1-2
  programming, multitask II-10-1
FOVLD II-15-1, II-15-4
FOVLY II-15-1, II-15-4
FOVRL II-15-1, II-15-5
.FP I-3-2, II-10-1
  frame pointer I-3-4
  stack, singletask and multitask I-3-3
FRWND II-6-1, II-6-12
FSWAP II-20-1, II-20-6
FQTASK II-16-1, II-16-3, II-16-6
FQTSK II-16-3, II-16-6
frame pointer I-3-4
FRENAME II-5-1, II-5-5
FRET I-3-6
FSDAY II-9-3
FSTIME II-9-1, II-9-3
FTASK II-10-2, II-11-1

GCHAR II-7-1
.GCHAR II-7-1, II-7-4
GCIN II-7-1, II-7-2
.GCIN II-7-2
GCOUT II-7-1, II-7-2
.GCOUT II-7-2
.GDAY II-9-1, II-9-2
GDIR II-4-1, II-4-3
.GDIR II-4-3
GETERR II-20-1, II-20-6
GETEV II-13-1
GETPRI II-13-1
GFREQ II-18-1
GHRZ II-18-1
.GHRZ II-18-1
global specifier II-4-3
GMCA II-23-1, II-23-2
.GMCA II-23-2
GMCA1 II-23-1, II-23-2
.GP I-3-2, II-10-1
  stack, singletask and multitask I-3-3
.GPOS II-6-3
GROUND II-22-1, II-22-4
GSYS II-4-1, II-4-4
.GSYS II-4-4
GTATR II-5-1, II-5-5
.GTATR II-5-5
.GTOD II-9-2

HOLD II-12-1, II-12-7

IAND II-3-1
ICLR II-3-1,
ICMN II-22-1, II-22-4
.ICMN II-22-4
ID, task II-11-1
.IDEF II-25-2
IDST II-13-1, II-13-2
.IDST II-13-2
IEOR II-3-4
ier II-1-1
INIT II-4-1, II-4-4
.INIT II-4-4
initialization:directory, partition, device II-4-4
initialize
  extended memory II-21-9
integer(s) II-1-1, II-3-1
  OR II-3-2, II-3-4
  Shift II-3-3

intrepreter, command line II-20-1
interrupts, console II-7-3
intertask communication II-14-1
I/O
  console II-7-1
  control block I-3-2
  file II-6-1
  FORTRAN 5 preconnections I-1-5
IOCB I-3-2, II-10-1
IOPC II-17-1, II-17-2, II-17-3
.IOPC II-17-2
IOPROG II-10-2, II-17-1, II-17-2, II-17-3
IOR II-3-1, II-3-2
.IRMV II-25-2
ISA conventions II-1-1
  status code II-20-6
  style routines II-16-1
ISET II-3-1, II-3-2
ISHFT II-3-3
ISHIFT II-3-3
ITASK II-10-2, II-11-1, II-11-2
ITEST II-3-3
IVF II-21-2
.IWAKE II-12-1, II-12-4
IXOR II-3-4

KILL II-12-1, II-12-4
kill task II-12-2, II-12-4

line mode II-6-1
LINK II-5-1, II-5-6
.LINK II-5-6
linkage conventions I-3-5
load-on-call mechanism I-1-6, II-16-1
loading
  command lines I-1-3
  FORTRAN 5 Programs I-1-3
  overlay II-15-2
  program, foreground II-22-3
logical operations II-3-1
LONGTRACE I-2-2
  multitasking I-2-3

MAC.SV, macroassembler I-1-3
macroassembler, MAC.SV I-1-3
macros
  FORTRAN 5 environment I-3-5
  FORTRAN 5 provisions I-3-5
  PARTITION II-10-2
  S?TASK II-10-3
mag-tape direct mode II-6-1
maintaining files II-5-1
map
  window, define II-21-5
mapping
  window II-21-1, II-21-6
marking the stack I-3-10
main memory, configuration I-3-2
managing directories and devices II-4-1
MAPDF II-21-1, II-21-2, II-21-5, II-21-7
.MAPDF II-21-5
MCA II-22-1
  number, obtain II-23-2
MDIR II-4-1, II-4-5
.MDIR II-4-5 mechanism
  event II-10-1
  load-on-call I-1-6, II-16-1
  overlay I-1-6
  XMT/REC II-19-1
memory
  allocate II-16-1
  extended II-21-1
  foreground/background programming II-22-1
  FORTRAN 5 environment I-1-3
  main, configuration I-3-2
  partitions, multitasking II-10-1
  runtime, allocation I-3-1
message
  intertask communication II-14-1, II-14-2
MESSAGE II-20-1, II-20-7
messages
  console II-17-4, II-22-5, II-22-6
  foreground/background II-22-5
messages, errors
  text, numbers I-2-3
  error files I-2-3
  traceback II-20-4
  MESSAGE II-20-7
MMPU II-22-1
modes
  direct block II-6-1
  line II-6-1
  mag-tape direct II-6-1
  random record II-6-1
  sequential II-6-1
module, task/operator communications II-17-1
MTDIO II-6-1, II-6-6
.MTDIO II-6-6
MTOPD II-6-1, II-6-6
.MTOPD
multiple directory devices II-4-1
Multiprocessor Communications Adaptor II-22-1
multiple processor routines II-23-1
multiprogram environment II-22-1
MULTITASK II-19-1, II-19-2
multitask environment
  changing task states II-12-1
  enable/disable II-19-1
  memory partitions II-10-1
  MULTITASK II-19-1
  obtaining task-related information II-13-1
  runtime memory configuration I-3-3
  task/operator communications II-17-1
  user devices and interrupts II-25-1
multitask scheduler II-10-1
MYEV II-13-2
MYID II-13-3
MYPRI II-13-3

.NDI I-3-2, II-10-1
  stack, singletask and multitask I-3-3
.ND2 I-3-2, II-10-1
  stack, singletask and multitask I-3-3
.NMAX II-10-2
NOT II-3-5
NOTRACE I-2-3
obtaining task-related information, multitask environment II-13-1
ODIS II-7-1, II-7-3
.ODIS II-7-3
OEBL II-7-1, II-7-3
.OEBL II-7-3
OPCOM II-17-1, II-17-2, II-17-3, II-22-5
OPEN II-6-1, II-6-7
.OPEN II-6-5, II-6-7
operations, logical II-3-1
ordinary overlays II-15-1
output
  maximum line length I-1-4
OVCLOSE II-15-1, II-15-2
OVERFL II-2-1
overflow, stack I-3-5
overlay
  file II-15-1
  file, close II-15-2
  file, open II-15-4
  FORTRAN 5 facilities I-1-6
  load II-15-2, II-15-4
  load-on-call II-15-1
  load-on-call mechanism I-1-6
  management system II-15-1
  mechanism I-1-6
  ordinary II-15-1
  release II-15-3, II-15-5
  segment programs II-20-1
  statement II-15-1
  virtual II-15-1
OVEXIT II-15-1, II-15-3
OVLOD II-15-1, II-15-4
.OVLOD II-15-4
OVOPN II-15-1, II-15-4
OVREL II-15-1, II-15-5
.OVREL II-15-3, II-15-5
OVKILL II-15-1, II-15-3
.OVKILL II-15-3
page zero I-3-2, I-3-3, I-3-4, II-10-1
parameter files
  F5SYM.SR I-3-5
  FMAC.SR I-3-5
partition
  bootstrap II-23-1
  initialization II-4-4
  multitask environment II-10-1, II-10-2
  release II-4-5
PARTITION II-10-2
partitions, memory II-10-1
  DPART.SR II-10-2
  PARTITION II-10-2
  PARTITION.SR II-10-2
  specification table II-10-2
PCHAR II-7-1, II-7-4
.PCHAR II-7-4
periodic, or delayed execution II-16-1
  initiation, termination request II-16-3
  synchronization II-16-3
per-routine-activation I-3-2
per-task data I-3-1
pointer
  stack I-3-4
  frame I-3-4
priority, task II-11-1
  foreground/background programming II-22-1
preconnections
default file (illustration) I-1-5
FORTRAN 5 I/O I-1-5
IBM-emulated (illustration) I-1-6
PRI II-12-1, II-12-5
.PRI II-12-5
process I-3-1
  per-process data I-3-1
program table II-17-2, II-17-3
PSH
  instruction I-3-5
  macro I-3-5

Q.EPA II-16-2
QCOND II-16-2
Q.MEM II-16-2
QNUM II-16-2, II-16-3
QOCH II-16-2
QPC II-16-2
QPRI II-16-2
QRR II-16-2, II-16-3
QSH II-16-2, II-16-3
QSMS II-16-2
QTLNK II-16-2
QTOV II-16-2
.QTSK II-16-5, II-16-6, II-16-7
QUE II-17-1, II-17-2, II-17-3
queue
  initiation requests II-16-1, II-16-6
  table II-16-1, II-16-2, II-16-3, II-16-4

random record mode II-6-1
RCHAR II-7-1, II-7-4
.RDB II-6-8
RDBLK II-6-1, II-6-8
.RDCM II-22-5
RDCMN II-22-1, II-22-5
.RDL II-6-9
RDLIN II-6-1, II-6-9
.RDOP II-22-5
RDOPR II-17-1, II-22-5
RDOPR II-22-1, II-22-5
RDOS
  Reference Manual II-17-1, II-20-1, II-21-1
RDRW II-6-1, II-6-11
.RDS II-6-10, II-6-11
RDSEQ II-6-1, II-6-10
RDSW II-8-1
.RDSW II-8-1
read
  blocks, extended memory II-21-3
reading console switches II-8-1
READR II-6-1, II-6-11
READRW II-6-1, II-6-11
ready task II-12-2, II-12-7
REC II-14-1
.REC II-14-1
records
  read II-6-11
  write II-6-14
recursive I-3-1
re-entrant I-3-1
RELEASE II-4-1, II-4-5
release
  overlay II-15-3
RELSE II-12-1, II-12-7
remap
  extended memory II-21-2
REMAP II-21-1, II-21-2, II-21-6
.REMAP II-21-6
RENAME II-5-1, II-5-6
.RENAME II-5-5, II-5-6
requests
  delayed/periodic task initiation II-16-1, II-16-3
  queue initiation II-16-1
  queued, remove II-16-4
RESET II-6-1, II-6-12
resolution file
  directory information II-5-7
resources, task II-10-1
RETURN II-16-1, II-16-3
returning II-20-1
REWIND II-6-1, II-6-12
RLSE II-4-1, II-4-5
.RLSE II-4-5
routine activation I-3-1
  re-entrant I-3-1
  recursive I-3-1
  per-routine I-3-2
.ROPEN II-6-7
routine, assembly language I-1-2, I-3-8
routines, multiple processor II-23-1
routine, runtime II-1-1
.RP I-3-2, II-10-1
  stack, singletask and multitask I-3-3
RSTAT II-5-1, II-5-7
.RSTAT II-5-7
RTN
  instruction I-3-5
  macro I-3-5
.RTN II-20-5
RUCLK II-25-1, II-25-3
.RUCLK II-25-3
RUN II-17-1, II-17-2, II-17-3
runtime
  environment I-3-1
  initialization time II-10-1
  initializer II-10-2
  memory allocation I-3-1
  stack area I-3-4
  stack discipline I-3-4
runtime environment I-3-1
  multitasking, memory partition II-10-1
runtime memory allocation I-3-1
foreground/background programming II-22-1
partition I-10-2
stack I-10-3
runtime routine II-1-1
runtime stack area I-3-4
runtime stack discipline I-3-4

SAVE
  instruction I-3-5
  macro I-3-5
  operation II-10-3
Save files II-20-1
scheduler
  multitask II-10-1
  task II-22-1
SDATE II-9-1, II-9-4
.SDAY II-9-3, II-9-4
sequential mode II-6-1
short form traceback I-2-3
SINGLETASK II-19-1, II-19-2
singletask environment
  runtime memory configuration I-3-3
  SINGLETASK II-19-1, II-19-2
  user devices and interrupts II-25-1
  using overlays II-15-1
SMEM II-22-1
.SP I-3-2, II-10-1
  stack, singletask and multitask I-3-3
  stack pointer I-3-4
.SPDA II-24-1
SPDIS II-24-1
.SPEA II-24-1
SPEBL II-24-1
SPKILL II-24-1
.SPKL II-24-2

SPOOLing
enable II-24-1
disable II-24-1
kill II-24-2
spooling, controlling II-24-1
.SPOS II-6-3, II-6-12
.SSE I-3-2, II-10-1
stack, singletask and multitask I-3-3
stack limit I-3-4
stack I-3-2
area, runtime I-3-4
discipline, runtime I-3-4
limit I-3-4
multitask environment II-10-1, II-10-2, II-10-3
pointer I-3-4
runtime, various stages of call (illus.) I-3-9
size II-16-1
size, parameter II-10-2, II-10-3
space II-16-1
state variables I-3-2
stack limit I-3-4
stack, marking I-3-10
stack overflow I-3-5
stack pointer I-3-4
START II-16-1, II-16-3, II-16-6
S?TASK II-11-3
example II-11-3
STAT II-5-1, II-5-7
.STAT II-5-7
statement
OVERLAY II-15-1
EXTERNAL II-15-1
state variables I-3-2, II-10-1
.SP I-3-2
.FP I-3-2
.SSE I-3-2
.RP I-3-2
.GP I-3-2
.ND1 I-3-2
.ND2 I-3-2 stack, singletask and multitask I-3-3
STATIC
storage II-15-1, II-16-1
variables I-3-1, II-15-1
status, user table I-3-2
STIME II-9-1, II-9-4
.STOD II-9-3, II-9-4
STTSK II-13-1, II-13-2
subdirectory
create II-4-1
SUSP II-12-1, II-12-5
.SUSP II-12-5
suspend task II-12-3, II-12-5, II-12-7
suspensions, task II-10-3
SWAP II-20-1, II-20-7
swaps
programs II-20-1
switches
global I-1-1 to I-1-2
local I-1-2
console, reading II-8-1
synchronization
delayed/periodic task initiation II-16-3
system
overlay management II-15-1
system clock and calendar II-9-1

tables
operating system I-3-2
partition specification II-10-2
program II-17-2
queue II-16-1, II-16-2(illustration), II-16-3, II-17-2
User Status I-3-2
task I-3-1, II-10-1
abort II-12-3
control block I-3-2
control block extension I-3-2
delay II-18-1
event number, obtain II-13-1, II-13-2
extended memory II-21-1, II-21-2
ID II-11-1, II-13-3
identifier II-16-1
initial subroutine II-16-1
initiation II-16-7
initiation, delayed or periodic II-16-1, II-16-6
initiation, multitasking II-11-1, II-11-2, II-11-3
initiation, synchronization II-16-2
kill II-12-2, II-12-4, II-12-6, II-15-3
per-task data I-3-2
priority II-11-1, II-12-5, II-12-6, II-16-1
priority, obtain II-13-1, II-13-3
ready II-12-2, II-12-7
related information, multitasking II-13-1
resources I-10-1
states, multitasking II-12-1
status, obtain II-13-2
suspend II-12-3, II-12-5, II-12-7, II-18-2
.TASK II-11-1, II-11-2, II-11-3
TCB I-3-2, II-10-1, II-10-2
termination
program II-20-4, II-20-5
TIDK II-12-1, II-12-6
.TIDK II-12-6
TIDKILL II-12-1, II-12-6
TIDP II-12-1, II-12-6
.TIDP II-12-6
TIDPRI II-12-1, II-12-6
TIDR II-12-1, II-12-7
.TIDR II-12-7
TIDRDY II-12-1, II-12-7
TIDS II-12-1, II-12-7
.TIDS II-12-7
TIDSUSP II-12-1, II-12-7
TIME II-9-5
timing
delayed/periodic task initiation II-16-3

TITLE I-3-5
.TOVLD II-15-2
traceback
  error II-20-4, II-20-7
  error handling mechanism I-2-2
  LONGTRACE I-2-2
  NOTRACE I-2-2
  short form I-2-2
transfer
  control II-20-5, II-20-7
TRDOP II-17-1, II-17-4
.TRDOP II-17-4
TRNON II-16-1, II-16-3, II-16-7
TWROP II-17-1, II-17-4
.TWROP II-17-4
.ULNK II-5-8
UNEST II-15-1, II-15-5
Unit
  Memory Management and Protection II-22-1
unit numbers I-1-5
  correspond channel numbers II-5-4
  overlays II-15-1
UNLINK II-5-1, II-5-8
.UPDAT II-5-3
.UPDAT II-5-8
UPDATE II-5-1, II-5-8
user devices and interrupts II-25-1
user/system clock
  commands II-18-1
User Status Table I-3-2
UST I-3-2

variables
  STATIC I-3-1
  state I-3-2
  page zero state I-3-4
  task state II-10-1
VDUMP II-21-1, II-21-2, II-21-7
VF II-21-2, II-21-8
VFETCH II-21-1, II-21-2, II-21-7, II-21-8
virtual overlays II-15-1
VLOAD II-21-1, II-21-2, II-21-9
VMEM II-21-1, II-21-2, II-21-9
.VMEM II-21-9
VS II-21-2, II-21-10
VSTASH II-21-1, II-21-2, II-21-10

WAIT II-18-1, II-18-2
wakeup, event number II-12-4
WCHAR II-7-1, II-7-5
.WHCAR II-7-5
WHERE II-22-1, II-22-4
window
  mapping II-21-1
  map II-21-2
  block II-21-2, II-21-6
  map, define II-21-5
words II-3-1
  stack, multitasking II-10-3
.WRB II-6-13
WRBLK II-6-1, II-6-13
.WRCM II-22-6
WRCMN II-22-1, II-22-6
writes
  blocks II-21-4
WRITR II-6-14
WRITRW II-6-1, II-6-14
.WRL II-6-15
WRLIN II-6-1, II-6-15
.WROP II-22-6
WROPR II-17-1
WROPR II-22-1, II-22-6
.WRS II-6-14, II-6-15
WRSEQ II-6-1, II-6-15

XMT II-14-2
.XMT II-14-2
XMT/REC mechanism II-19-1
XMTW II-14-2
.XMTW II-14-2

DataGeneral

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA®4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		
Other (Specify) _____		

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II
- ☐ PL/1
- ☐ APL
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

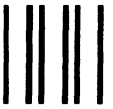
DataGeneral

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

FOLD FOLD

TAPE TAPE

FOLD FOLD



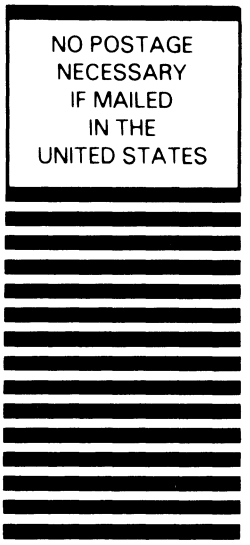
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

DataGeneral

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581



DataGeneral

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

SHIP TO: (if different)
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No._____ Expiration Date_____

☐ Purchase Order Number:_____

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other _____
_____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order _____ Phone _____ Extension _____

Mail Orders to:

Data General Corporation
Attn: Educational Services/TIPS F019
4400 Computer Drive
Westboro, MA 01580
Tel. (617) 366-8911 ext. 4032

_____
Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

_____
Title

_____
DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

CUT ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

ta General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following ms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof ich is accepted by DGC.

PRICES
Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

PAYMENT
Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

SHIPMENT
Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

TERM
Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

CUSTOMER CERTIFICATION
Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

DATA AND PROPRIETARY RIGHTS
Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

DISCLAIMER OF WARRANTY
DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

LIMITATIONS OF LIABILITY
IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

GENERAL
A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

DataGeneral

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   TIPS
   Educational Services – M.S. F019
   Data General Corporation
   4400 Computer Drive
   Westboro, MA 01580
8. We'll take care of the rest!

educational services

User Documentation Remarks Form

Your Name _____ Your Title _____

Company _____

Street _____

City _____ State _____ Zip _____

We wrote this book for you, and we made certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve the manual. Please take a few minutes to respond. Thank you.

Manual Title _____ Manual No. _____

Who are you? ☐EDP Manager ☐Analyst/Programmer ☐Other _____
☐Senior Systems Analyst ☐Operator _____

What programming language(s) do you use? _____

How do you use this manual? *(List in order: 1 = Primary Use)* _____

___ Introduction to the product ___ Tutorial Text ___ Other
___ Reference ___ Operating Guide _____

About the manual:	Yes	Somewhat	No
Is it easy to read?	☐	☐	☐
Is it easy to understand?	☐	☐	☐
Are the topics logically organized?	☐	☐	☐
Is the technical information accurate?	☐	☐	☐
Can you easily find what you want?	☐	☐	☐
Does it tell you everything you need to know	☐	☐	☐
Do the illustrations help you?	☐	☐	☐

If you have any comments on the software itself, please contact Data General Systems Engineering.
If you wish to order manuals, use the enclosed TIPS Order Form (USA only).

Remarks:

Date

NO POSTA
NECESSA
IF MAILE
IN THE
UNITED ST/

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

POSTAGE WILL BE PAID BY ADDRESSEE

DataGeneral

User Documentation, M.S. E-111
4400 Computer Drive
Westborough, Massachusetts 01581

Data General Corporation, Westboro, MA 01580

093-000227-01