Data General

Software Documentation

FORTRAN 5 Programmer's Guide (AOS)

FORTRAN 5
Programmer's Guide
(AOS)

093-000154-02

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. 093-000154

Printed in the United States of America
Revision 02, January 1985

NOTICE

FORTRAN 5
Programmer's Guide
(AOS)
093-000154-02

Revision History:	Effective with:
Original Release - October 1978	
First Revision - January 1981	
Second Revision - January 1985	(FORTRAN 5 Rev. 6.10)

CONTENT UNCHANGED

The content in this revision is unchanged from 093-000154-01. This revision changes only printing and binding details.

Preface

As a programmer fluent in FORTRAN or a similar language and familiar with the Advanced Operating System (AOS), you will find this Programmer's Guide a useful companion to the *FORTRAN 5 Reference Manual* (093-000085).

This manual instructs you in writing your own runtime routines and in using the FORTRAN 5 runtime libraries. We detail various aspects of operating FORTRAN 5 under AOS, error handling, the runtime environment, and the general concepts of multitasking. If you write your own runtime routines, Chapter 5, "The FORTRAN 5 Assembly Language Interface," will be of special interest to you.

We group the runtime routines in chapters by the functions they perform. Equipped with an understanding of the operating instructions described in Chapter 1, "FORTRAN 5 under AOS," you can call the runtime routines detailed in Chapters 7 through 24. At the end of each runtime routine chapter is a sample FORTRAN 5 program that contains calls to one or more of the routines in that chapter.

We have organized the manual as follows:

Chapter 1	Describes how to run FORTRAN 5 under AOS. We include information on how to compile and link a FORTRAN 5 program as well as how to change certain FORTRAN 5 default actions.
Chapter 2	Details error handling. We describe the actions FORTRAN 5 takes when it intercepts an error, your control of these actions, and the actions you take when a routine returns an error code.
Chapter 3	Introduces the runtime environment. We define basic programming terms and describe different resources and how they are allocated at runtime.
Chapter 4	Presents the general concepts of multitasking. We detail tasks and their resources, and memory partitions in a multitask environment.
Chapter 5	Details how to write FORTRAN 5 assembly language runtime routines. We explain runtime stack disciplines, and Data General assembly language fundamentals. We include examples of runtime routine code.
Chapter 6	Introduces the runtime routine chapters. We explain the types of arguments in the routines and detail the organization of Chapters 7 through 24.
Chapters 7-24	The runtime routines.
Appendix A	FORTRAN 5 Runtime Error Parameters (FORTRAN 5 errors and system errors)
Appendix B	Exceptional Condition Codes
Appendix C	Calls To the Runtime Routines (alphabetical, with chapter and page reference)

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\left\{ \begin{array}{l} \text{required}_1 \\ \text{required}_2 \end{array} \right\}$ 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 this 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 carriage return (CR) 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, in examples we use

THIS TYPEFACE TO SHOW YOUR ENTRY↲
THIS TYPEFACE FOR SYSTEM QUERIES AND RESPONSES.

) is the CLI prompt.

Contacting Data General

- If you have comments on this manual, please use the prepaid Remarks Form that appears after the Index. We want to know what you like and dislike about this manual.
- If you need additional manuals, please use the enclosed TIPS order form (USA only) or contact your Data General sales representative.

End of Preface

Contents

Chapter 1 - Using FORTRAN 5 under AOS

Chapter 2 - Error Handling

Chapter 3 - Runtime Environment Fundamentals

Chapter 4 - Multitask Programming in FORTRAN 5

Chapter 5 - FORTRAN 5 Assembly Language Interface

Chapter 6 - About the Runtime Routines

Chapter 7 - Checking for Arithmetic Errors

Chapter 8 - Performing Logical Operations with Integers and Words

Chapter 9 - Managing Logical Disks and Directories

Chapter 10 - Maintaining Files

Chapter 11 - File Input/Output

Chapter 12 - Console Handling

Chapter 13 - Using the System Clock and Calendar

Chapter 14 - Initiating Tasks in a Multitask Environment

Chapter 15 - Changing Task States in a Multitask Environment

Chapter 16 - Obtaining Task-Related Information in a Multitask Environment

Chapter 17 - Intertask Communication

Chapter 18 - Requesting Delayed or Periodic Task Initiation

Chapter 19 - Enabling and Disabling the Multitask Environment

Chapter 20 - Using Overlays

Chapter 21 - User/System Clock Commands

Chapter 22 - Transferring Control Between Programs And Accessing Command Line Information

Chapter 23 - Reporting Errors and Messages

Chapter 24 - Using Extended Memory

Appendix A - FORTRAN 5 Runtime Error Parameters

Appendix B - Exceptional Condition Codes

Appendix C - Calls to the Runtime Routine

Appendix D - Alphabetized List of FORTRAN 5 Statements

Appendix E - Fortran 5 Runtime Databases

Appendix F - CLRE Math Routines

Appendix G - ASCII Table

Appendix H - Entry points for FORTRAN 5 Runtime Environment Routines

Tables

Illustrations

Chapter 1
Using FORTRAN 5 under AOS

FORTRAN 5 supports many of the advanced features of the Advanced Operating System (AOS). You can use FORTRAN 5 to take full advantage of AOS without resorting to the use of assembly language. FORTRAN 5 produces code that rivals assembly language in compactness and speed of execution, but provides the ease of programming and debugging associated with high-level languages.

Shareable Code

FORTRAN 5 produces programs that are fully shareable. With this feature, all users running a FORTRAN 5 program execute the same copy of the program residing in memory. This results in more efficient utilization of memory.

Load-on-call Overlays

FORTRAN 5 supports the Load-on-call overlay features of the AOS Resource Call Facility (see the *AOS Programmer's Manual* (093-000193) for more information on Resource Calls). An overlay is a portion of a program that resides in an overlay file. Using the Load-on-Call facility from your program, you can load the overlay into main memory from the overlay file. Consequently, seldom used subroutines and functions need not take up space in main memory until they are actually called. You decide which routines are memory resident and which routines reside in disk overlays when you link your program. You need not recompile your routines to change this overlay structure.

The Common Language Runtime Environment

Through the AOS CLRE, routines written in different languages can call each other in the same program as long as they don't use conflicting features. FORTRAN 5 routines share a common runtime interface with Data General's AOS PL/1 and DG/L™, languages. These languages also use the same set of mathematical routines (see Appendix F).

Command Line Interpreter (CLI) Macro Files

A CLI macro file contains a group of CLI commands. When you call a CLI macro file by entering its name as a CLI command, the CLI automatically executes all the commands in it. FORTRAN 5 provides two CLI macros for use in compiling and linking programs: F5.CLI and F5LD.CLI. F5.CLI invokes FORTRAN 5 to compile a source program to produce an object file. F5LD.CLI links object programs to produce an executable program file.

You must separately compile each FORTRAN 5 main program, subroutine, and subprogram. Use the F5.CLI command to do this. After you compile your source programs, use F5LD.CLI to build your executable program file. F5LD.CLI invokes the system utility, Link, which names the FORTRAN 5 runtime libraries in proper order.

The following example documents a series of AOS CLI commands that compile, link, and execute a FORTRAN 5 program.

Compile:

```
F5 MAIN)
F5 SUB)
F5 XFUN)
F5 XSUB)
```

Link:

```
F5LD MAIN SUB1 XFUN XSUB)
```

Execute:

```
XEQ MAIN)
```

Compiling a FORTRAN 5 Program Under AOS

To compile a FORTRAN 5 program under AOS, type in the FORTRAN 5 command followed by the pathname of the source file.

The format of the FORTRAN 5 command line is

F5 *[function switches]* inputpathname

where inputpathname is the name of your FORTRAN 5 source file and *function switches* are any combination of the switches in Table 1-1.

Compilation Examples

- F5 MYPROG↲

  Compiles MYPROG.FR , if it exists, or MYPROG. Since there is no /E switch, the system sends all errors to the current @OUTPUT pathname. The compiler produces the object file, MYPROG.OB.

- F5/ER/NOI/L=PROG.LS/CODE PROG↲

  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 listing includes the generated code, but not lines from the INCLUDE statement. The system does not create an error file.

Table 1-1. Function Switches

Switch	Meaning
/B	Produce a brief listing. /B includes the input source program, the storage map, the list of all subprograms called, and the error list. The listing does not include the generated code.
/C	Check the index of the source program. If you specified a listing file, /C sends the source and the error list to it. /C also sends the error list to the error file, if one exists.
/D	Debug. Compile code that allow the long form error traceback routine to output line numbers. This option doesn't provide more information when an error occurs, but does provide a more convenient form. Don't use /D in final versions of programs. See Chapter 2 for more information on this switch.
/E /E=pathname	Output errors to pathname. If you omit =pathname, E suppresses error messages. If you omit /E, the compiler outputs messages to the current CLI output pathname.
/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 /L=pathname	Output listing to pathname. If you omit =pathname, the list pathname is the current CLI LIST pathname. If you omit /L, but use /B, you get a brief listing as output to the current CLI LIST pathname. Do not send the listing output directly to the line printer, @LPT; the line printer prints your output on five separate pieces and might include other users' output.
/N	Do not produce an object file.
/O=pathname	Give the object file this name. If you don't use /O, inputpathname.OB is the object pathname.
/P	Assumes punched card input format. The compiler uses only the first 72 characters of each input line as FORTRAN 5 source code. However, it does send the entire input line to the listing file, if one exists.

(continues)

Table 1-1. Function Switches

Switch	Meaning
/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=A(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 that array.
/X	Compile lines with an X in column one. If you do not use /X, the system treats these lines as commands.
/NOLEF	Don't generate Load Effective Address instructions (LEFs). This switch is useful if you're using I/O instructions in assembly language routines combined with FORTRAN 5 programs. See the *AOS Programmer's Manual* for further information on the LEF mode of program execution.

Linking a FORTRAN 5 Program Under AOS

Use the F5LD command to link your FORTRAN 5 program. This command uses the F5LD.CLI macro. F5LD.CLI invokes the AOS linker, LINK. It also names the FORTRAN 5 libraries in the proper order.

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

1. main FORTRAN 5 program

2. user subprograms

3. support libraries (e.g., Commercial Subroutine package)

The F5LD command line has the form

F5LD *[function switches]* mainprogram *[argument switches]* [subprogram *[argument switches]* ...])

Where

mainprogram	is the name of your FORTRAN 5 main program unit
subprogram	is the name of the FORTRAN 5 subprogram that one of your FORTRAN 5 routines uses.
function switches	represent any combination of the following optional function switches.
argument switches	represent any combination of the following optional argument switches.

The F5LD command may also contain LINK overlay designators described later in this section.

LINK interprets the following switches directly. If you use them in situations where you use XEQ LINK instead of F5LD , apply them to LINK not XEQ .

F5LD LINK Function Switch	**Action**
/ALPHA	Produces a symbol table listing, sorted alphabetically by symbol name.
/E=pathname	Sends error messages to pathname instead of the default output file, @OUTPUT .
/KTOP=n	Specifies the top of the program's address space. n specifies a number of 1024-word pages.
/L=pathname	Outputs the listing to the specified pathname . If you omit =pathname , LINK outputs the listing to the current CLI listfile.

F5LD LINK Function Switch	Action
/NTOP=n	Specifies the top of the program's address space. n specifies a maximum address.
/NUMERIC	Produces a symbol table listing sorted by numeric value.
/MAP	Produces a map listing the size of each object partition.
/MODMAP	Produces a module-by-module map that lists the size of each object partition.
/MODSYM	Produces a module-by-module list of symbols.
/O=pathname	Assigns pathname.PR to the executable program file. If you omit this switch, the program file assumes the name of the first module in the FORTRAN 5 command line with the extension .PR .
/REV=ww[.xx[.yy[.zz]]]	Sets the revision number of the generated program (e.g. /rev=2.37). yy and zz are meaningful only for AOS/VS.
/SYS=RDOS	Generates a .SV program file executable under RDOS AOS.
/SYS=VS16	Generates a program file executable under AOS/VS.
/TASKS=n	Specifies the maximum number of concurrent tasks the program will need for execution.
/a=b	Changes partition attributes. a and b must be one of the following: UC unshared code UD unshared data SD shared code SD shared data When you use this switch, the system treats object modules of type a as though they were of type b.

F5LD Argument Switch	Action
/a=b	See the description of a=b in the preceding section. When you append this switch to an argument filename, it modifies attributes of that module.
/ALIGN=N	When attached to the name of a common block, it causes Link 5 to align that block on a (2**n) word boundary. For example, BLK1/align=10 aligns the common block, BLK1 on a 1024-word boundary.
/SHARED	When attached to the name of a common block, it causes Link to place that common block in the shared data partition.

F5LD LINK Overlay Designators	Action
!*	Indicates the start of a module list you want to place in a single overlay area.
*!	Indicates the end of a module list you want to place in an overlay area.
!	Indicates the divisions between overlays within an overlay area. This argument must appear between !* and *! .

Numeric values for Link switches are decimal by default. You can append a radix specifier (Rn) to a numeric value to change its radix (e.g. /ALIGN=9 and /ALIGN=11R8 have the same meaning).

See the following FORTRAN 5 LINK examples, and the *LINK Reference Manual* (093-000254) for additional information.

The following switches are interpreted by F5LD.CLI . Most are not LINK switches and cannot be abbreviated.

F5LD.CLI Switch	Action
/STRING	Places Link's termination message in [!STRING]
/LONGTRACE	Includes LONGTRACE.OB to produce the most descriptive form of traceback. You must have included /LONG in the F5 compilation command for 1 or more routines.
/QCALLS	Includes F5ASYS.LB. You must supply this switch if the program calls any of the AOS QCALL runtime routines described in the *FORTRAN QCALLS Reference Manual* (093-000239).
/TASKS=n	Includes F5TASK.LB. Do not include this switch unless the program includes two or more tasks.

Linking Examples

- F5LD MYPROG⤶

  Links the main program, MYPROG , and the required FORTRAN 5 runtime routines.

- F5LD/L=NEWPROG.LM/O=NEWPROG/ALPHA PROG⤶

  Creates the executable program file, NEWPROG.PR (/O=NEWPROG) from the object file, PROG.OB . Includes the required FORTRAN 5 runtime routines . Generates a listing file, NEWPROG.LM. , that includes an alphabetically sorted list of symbol names and values (/ALPHA).

Linking a FORTRAN 5 Program That Contains Overlays

If you type

```
F5LD EXAMPLE !* SUB3 ! SUB4 SUB5 *!
```

FORTRAN 5 invokes LINK to build EXAMPLE3.PR and its overlay file, EXAMPLE3.OL . This program includes a single overlay area with two overlays, one containing SUB3 and the other containing SUB4 and SUB5 .

The symbols in this example have the following meanings:

!*	Begin the definition of an overlay area
!	Separate overlays within an overlay area
*!	End the definition of an overlay area

You must separate overlay designators from module names by one or more spaces or tabs. You can append F5LD LINK argument switches to the overlay area start designator (!*). F5LD LINK argument switches apply to all modules within an overlay area.

The command

```
F5LD EXAMPLE4 !* SUB1 SUB2 SUB3 ! SUB4 SUB5 ! SUB6 *!
```

builds EXAMPLE4.PR and EXAMPLE4.OL which contains 3 overlay areas. The first overlay area contains SUB1,SUB2, and SUB3 . The second contains SUB4 and SUB5 . The third contains SUB6 .

Limiting the Amount of Memory Available to the FORTRAN 5 Environment

By default, a single-task FORTRAN 5 program begins execution with only enough unshared memory pages to satisfy the stack requirements of the main program. The runtime environment requests additional unshared pages (up to the 64KW address space limit) as the program requires stack space. The runtime environment reports a stack overflow error if no additional memory is available to the program when a stack overflow occurs (see Chapter 3, *Runtime Environment Fundamentals,* for more information on stack overflows). The program does not release memory acquired during execution until it terminates.

You can override the default memory allocation for both single-task and multitask programs either when you link the program or when you execute it.

To change the amount of memory requested, follow the instructions in DMEM.SR for editing the file. Assemble the file by following the instructions in Chapter 5, and include DMEM.OB in the F5LD command. See the instructions in DMEM.SR for additional information.

You can limit the amount of memory the program can use when you execute it by using the /MEM switch for the PROCESS CLI command. You must use the PROCESS CLI command rather than the EXECUTE command if you use the /MEM switch.

You can also limit the amount of memory the program uses when you link the program. You can do this in two ways:

- Use the LINK /NTOP and /KTOP switches to establish an upper limit on memory use.
- Edit the file DMEM.SR supplied with FORTRAN 5 to specify both the upper limit and the size of the initial request for single-task programs.

You can also use the file DMEM.SR to force single-task programs to request the full available address space for non-dynamic memory allocation.

Unlike single-task programs, the memory usage in multitask programs is not dynamic. Multitask programs use the full amount of memory available for the program's address space throughout the lifetime of the program.

Setting a Maximum Line Length for Output

FORTRAN 5 provides standard default line lengths, but you can define your own defaults. You can also override the standard defaults explicitly, file-by-file. By default, the longest line that you can write to a line-oriented file is 136 characters.

The OPEN statement sets specific line lengths and allows the following two options:

LEN=n specifies the line length

ATT="L" specifies that the file organization is line-oriented rather than record-oriented or stream-oriented

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 in the case of formatted I/O, or the excess spills over to the next line in the case of free-formatted I/O.

See the description of the OPEN statement in the *FORTRAN 5 Reference Manual* for more informaton.

Error Conditions

When a line exceeds the maximum line length, one of the following results occurs, depending on the type of output.

The error message OUTPUT RECORD TOO LONG appears for the following reasons:

- A line of formatted output exeeds the maximum line length.
- A data item other than a Hollerith or string constant spills from one data item to a second, and cannot fit on the new line.

The excess data item spills to a second line when a line of free-formatted output, including Hollerith and string constants, exceeds the maximum line length.

Changing the Default Line Length

If you don't use options to open a file, AOS assumes that it is line-oriented. The file then has the AOS default line length of 136 characters. Use the following method to change the default line length:

1. Edit the file LINESIZE.SR , using the instructions in that file, and change the default value to the one you want.
2. Assemble LINESIZE.SR (see Chapter 5 *FORTRAN 5 Assembly Language Interface*).
3. Include LINESIZE.OB in the F5LD command line when building the program file.

FORTRAN 5 Unit Numbers

All input and output routines in FORTRAN 5 reference files by their unit numbers. FORTRAN 5 manages unit numbers on a per-program basis. If you open a file in one routine, any routine can access it.

A program can access up to 64 files simultaneously, numbered from 0 to 63.

FORTRAN 5 I/O (Input/Output) Preconnections

Under FORTRAN 5, there are conventional I/O statement/unit number and unit number/pathname preconnections.

When you open a file explicitly with the FORTRAN 5 open statement, the system associates the unit number you provide in the statement with the file you specify in the statement. You can, however, specify a unit number in an I/O statement before the system associates it with a particular file. The system then checks a file preconnection table and either

- Opens the file if an entry exists in the table associating that unit number to a pathname, or
- Signals an error if no entry exists in the table for that unit number.

Editing the preconnection source files supplied with FORTRAN 5 lets you specify your own unit number/pathname and device name preconnections. The two preconnection files are DGCPCT.OB and IBMPCT.OB. By default, the system uses the Data General preconnections shown in Table 1-2, DGCPT Preconnections. If you want file preconnections similar to those used in IBM FORTRAN 4, name the file IBMPCT.OB in your F5LD command line. See Tables 1-2 and 1-3 for the DGCPCT.SR and IBMPCT.SR default preconnections.

You can output to unit numbers not explicitly opened using IBMPCT preconnections. If you do this, the system opens a temporary file called *unit number.F5* . (This is not the case when you use Data General Preconnections.)

Statement Preconnections

The statements TYPE, ACCEPT, PUNCH, PRINT and READ don't allow you to explicitly mention unit numbers. See Table 1-4 for the default statement unit numbers.

Changing Default I/O Preconnections

If you want to define your own I/O preconnections, or alter the standard Data General or IBM preconnections, follow this procedure:

1. Create a copy of either DGCPCT.SR or IBMPCT.SR (supplied with FORTRAN 5), depending on which preconnection style you prefer. Use the directions in this new file to edit it.

   You can provide most of the same information in the file preconnection table that you provide in the OPEN statement.

2. Assemble your preconnection file as described in chapter 5, "The FORTRAN 5 Assemble Language Interface".

3. The assembly produces an object file. Include the pathname to this file in your F5LD command line.

Table 1-2. DGCPCT Preconnections

Unit Number	Device Name	Meaning
6	@PLT	Incremental plotter
9	@DATA	Current DATA file
10	@OUTPUT	Current OUTPUT file
11	@INPUT	Current INPUT file
12	@LIST	Current LIST file (has P attribute)
13	@PTR	Paper tape reader
14	@PTP	Paper tape punch

Table 1-3. IBMPCT Preconnections

Unit Number	Device Name	Meaning
5	@DATA	Current DATA file
6	@LIST	Current LIST file
10	@OUTPUT	Current OUTPUT file
11	@INPUT	Current INPUT file

Table 1-4. Statement File Preconnections

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

The Program Development Cycle: A Coding Example

```
)
) XEQ LINEDIT SAMPLE.FR
  ---------------------
Do You want SAMPLE.FR to be Created? YES
                                     ---
?APPEND
 ------
C          This is a sample FORTRAN 5 program to demonstrate the
-----------------------------------------------------------------
C          development cycle of a program.
----------------------------------------
C
-
       TYPE "Hello, World!"
----------------------------
       VAR = SIN(0.5)
----------------------
       TYPE "The SIN of 0.5 is",VAR
------------------------------------
       STOP "That's All, Folks!"
---------------------------------
       END
----------
<ESC>
-----
?LIST ALL
 --------
C          This is a sample FORTRAN 5 program to demonstrate the
C          developement cycle of a program.
C
       TYPE "Hello, World!"
       VAR = SIN(0.5)
       TYPE "The SIN of 0.5 is",VAR
       STOP "That's All, Folks!"
       END
?BYE
 ---

) F5/L=SAMPLE.LIST SAMPLE
  -----------------------
FORTRAN 5 Version 6.10  Tuesday, November 6, 1982 -> SAMPLE.FR <-
   No Compilation Errors

) F5LD SAMPLE
  -----------
LINK REVISION 4.01 ON 11/06/80 AT 09:10:10
SAMPLE.PR CREATED

) XEQ SAMPLE
  ----------
Hello, world!
The SIN of 0.5 is  0.45346E 00
STOP That's All, Folks!

That's All, Folks!
)
```

End of Chapter

Chapter 2
Error Handling

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

FORTRAN 5 acts upon three kinds of errors:

- Fatal errors
- Transparent errors
- Recoverable errors

Fatal errors are errors from which recovery is impossible or undesirable. In this case, FORTRAN 5 outputs a message to the error files and terminates your program. Errors of this type include stack overflow and subscript out-of-bounds.

Transparent errors are errors that FORTRAN 5 reports, though the program continues to execute. You can neither intercept control nor suppress the reporting of the error. FORTRAN 5 reports a transparent error if you supply illegal arguments for intrinsic functions.

Recoverable errors are errors that FORTRAN 5 reports or passes on to you for action. You decide how to handle the situation. When you call a 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 a 1 back in that variable if the routine is completed successfully. If a problem occurred, it will return an error code in that variable. FORTRAN 5 never acts on an error that occurs in such a routine, but always leaves the action up to you. The majority of I/O (Input/Output) errors are recoverable errors.

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.

If FORTRAN 5 statements such as **DELETE**, **RENAME**, or **WAKEUP** detect errors, FORTRAN 5 handles them because it can't pass an error code to you. When the system detects errors in I/O statements that have **ERR=** or **END=** clauses, it transfers control to the statement label you name in the appropriate clause. After the transfer of control, you can determine what error occurred by calling the runtime routine, **GETERR** .

You can change the default actions taken for certain runtime errors. We detail how and when you can make these changes later in this chapter.

Status Variables

A status variable is an integer that receives either a 1 or an error code upon return from a routine. In a runtime routine call that returns a status variable, the status variable is always the last argument.

Never ignore the error code returned in status variables. An error code other than 1 indicates an error. Always check these variables for information about occurring errors.

Call the CHECK routine if you want FORTRAN 5 to check the error code and report an error if one occurred. Use the following format:

```
CALL CHECK(error variable)
```

You pass CHECK the same error variable name you passed to a previous runtime routine call. When CHECK sees a value of 1, indicating no errors, program execution continues. If the value is not 1, CHECK invokes the error reporter, and the program stops.

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. Any error code returned in a status variable is three greater than the actual error code value.

If you don't use CALL CHECK , you should check the error status yourself. If you do this, you can control the error processing. The file in Appendix A, F5ERR.FR, contains FORTRAN 5 error parameters. They define the mnemonics of error conditions that the runtime routines can return in a status variable. Use these parameters to check for specific errors (FORTRAN 5 error parameters have the ISA offset of 3 added to them).

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. Instead of referring to the Appendix, you can incorporate into your program all of F5ERR.FR with the INCLUDE statement.

ERR= and END= Options in FORTRAN 5 Statements

In FORTRAN 5, failure to include ERR= or END= clauses in specific tasking and I/O statements causes termination of your program when errors occur. These clauses specify the following:

ERR=label — label is a statement label number that receives control when FORTRAN 5 detects an error condition during execution of the statement.

END=label — label is a statement label number that receives control when FORTRAN 5 detects an end-of-file condition 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 in all other cases. If END= is not present, ERR= takes control of end-of-file conditions as well.

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

A call to GETERR clears the internally saved error code. This is 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 reporting 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 an internal FORTRAN 5 runtime environment routine. You can choose one of three types of traceback: LONGTRACE, short form (default) traceback, and NOTRACE.

In selecting a form of traceback for error handling, consider the following information. The short form gives you the same information as LONGTRACE, but occupies far less memory. LONGTRACE outputs routine names. It also outputs source line numbers if you compile your routines with the global switch, /LONGTRACE . Both LONGTRACE and short form traceback report memory locations as octal numbers.

Incorporating line numbers with LONGTRACE has both an advantage and a disadvantage. If you use /LONGTRACE , you don't need a code listing of your program's routines to determine where an error occurred. However, using /LONGTRACE slows down your program's execution.

LONGTRACE

Adding the /LONGTRACE switch into your F5LD command line gives you the most readable traceback. The following is an example of LONGTRACE output

```
**ERROR** reported by SQR22?4
Called at offset 26 in program unit SUBR1
Called at offset 13 (Line 2) in program unit .MAIN
Illegal argument for SQRT
```

In this example an attempt to take the square root of a negative number caused an error. SQR22?4 , the double precision square root function (The CLRE name for the FORTRAN 5 DSQRT routine), reported the error. The subroutine SUBR1 called SQR22?4. At offset 26_8 from the start of SUBR1 , the compiler generated a call operation. The main program, .MAIN, called SUBR1 on line 2. Since you compiled .MAIN with the global /LONGTRACE switch, the traceback output includes the source line numbers of the subroutine calls to SUBR1 and DSQRT .

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 report:

```
**ERROR** reported by (50)
Called at 70023+26
Called at 70000+13
Illegal argument for SQRT
```

The short form traceback requires some information from the listing file Link produces when you specify the /NUMERIC function switch. Figure 2-1 is a segment of the Link listing file with the information you need.

With the Link listing, you can determine that the (50) in the first line of the error report is the starting address of the SQR22?4 routine at location 50_8. Similarly, you can determine that location 70023_8 is the starting address of SUBR1 , and 70000_8 is the start of the main program (MAIN) code. Thus, 70023+26 refers to offset 26_8 in SUBR1 .

```
.SSE        000042
.SOV        000043
SQR22?4     000050
.ERET       000052
.RTER       000053
.
.
.MAIN       070000
.SUBR1      070023
.F5         070234
```

Figure 2-1. Link Segment

NOTRACE

In addition to the error handling alternatives, LONGTRACE and short form traceback, you can also choose NOTRACE. NOTRACE produces no traceback output at all, only an error message. Include NOTRACE.OB in your **F5LD** command line to suppress Traceback. NOTRACE saves considerable space in your program. However, because you will receive no indication where an error occurred with this alternative, only use NOTRACE in completely debugged code.

Floating Point Errors

The ECLIPSE® Floating Point Unit (FPU) provides a passive means of detecting floating point errors whenever they occur. FORTRAN 5 uses this mechanism to report four types of floating point errors:

Error	Definition
Floating Point Overflow	While processing a floating point calculation, an exponent overflow occurred. The result is correct except the exponent is 128 too small.
Floating Point Underflow	While processing a floating point calculation, an exponent underflow occurred. The result is correct except the exponent is 128 too large.
Floating Point Division by Zero	While processing a floating point division, the FPU detected a zero divisor. It aborted the division and did not change the operands.
Mantissa Overflow	During a numeric scaling operation or a real to integer conversion, the FPU shifted a significant bit out of the high order end of the mantissa. The significance of the result was lost.

FORTRAN 5 provides a routine, the floating point trap handler, which acts upon these errors. This routine performs a default action for each of the floating point errors. However, you can change its actions to suit your specific needs.

Default Actions

When the ECLIPSE FPU detects an error, the floating point trap handler does two things. First, it determines which instruction caused the error. Second, it takes some action based on which of the four floating point error conditions is set in the ECLIPSE Floating Point Status Register. The following list defines the default actions FORTRAN 5 takes for each error.

Error	FORTRAN 5 Default Action
Overflow	Reports a fatal error and terminates the program.
Underflow	Sets the result of the operation to zero. FORTRAN 5 does not report an error.
Division by zero	Reports a nonfatal (transparent) error and continues program execution. Since the FPU leaves the operands unchanged, the result appears to be the value of the numerator.
Mantissa Overflow	Takes no action and continues execution.

Changing Default Actions

The default actions for floating point errors may not suit your particular application. However, you can override them depending on your needs.

If you don't want any floating point error detection, the file NOTRAP.OB supplied with FORTRAN 5 disables the floating point trap mechanism. When you load NOTRAP.OB with your program, FORTRAN 5 does not provide the floating point trap handler. You will have no floating point error detection. To check for floating point errors, you must call the runtime routines OVERFL and DVDCHK .

If you want floating point traps, but the default actions are not appropriate, you can change them. Do this by editing the assembly language source file, FPTRAP.SR supplied with FORTRAN 5. This source file establishes the severity of the floating point errors and how they affect program execution. Edit FPTRAP.SR using the instructions in the file itself to change the actions in these areas:

- Whether or not the FORTRAN 5 reports an error
- Severity of the generated error (Fatal or Transparent)
- What value FORTRAN 5 places in the erroneous Floating Point accumulator (zero, largest number with the same sign, smallest number with the same sign, or unchanged result)

Once you have made changes in FPTRAP.SR , you assemble the changed file with the macroassembler as described in Chapter 5, "The FORTRAN 5 Assembly Language Interface." Then include FPTRAP.OB in your F5LD command line when you link your programs.

Error Files

You can direct error message output to any number of error files. By default, error message output goes to your process output file, @OUTPUT . You can reassign or specify additional error files by editing the files DGPCT.SR or IBMPCT.SR supplied with FORTRAN 5. Decide which file to edit by reading the section on file preconnections in Chapter 1, "FORTRAN 5 Under AOS."

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

```
EFILE @OUTPUT
```

from the source file. To send errors to a disk pathname, **EFILE1** , you would add

```
EFILE "EFILE1"
```

EFILE could be a disk file pathname or a link to another file. 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 Fundamentals

In this chapter we will describe how the computer executes your FORTRAN 5 programs within a runtime environment. This information will be useful if you want to know how your program actually performs the functions specified in the FORTRAN statements. It also serves as an introduction to Chapter 4, "Multitasking in FORTRAN 5," and Chapter 5, "FORTRAN 5 Assembly Language Interface." In this chapter, we make no assumptions about what you know about the computer.

Terminology

We will begin by discussing some basic terminology used later in this chapter, and in Chapters 4 and 5. You may be familiar with many of these terms, but the definitions given here will clarify their usage in relation to the runtime environment. This selection of terms is not meant to be a complete glossary of computer terminology.

Compilation

A compiler is a program that translates a program written in a high-level computer language, such as FORTRAN 5, into a machine language. The ECLIPSE computer executes the compiled machine language program, known as an executable program, at runtime. This translation process is known as compilation.

The translated output produced by the FORTRAN 5 compiler consists of object modules. Object modules are files with the .OB extension that the compiler creates for each routine in your FORTRAN 5 program; one object module (.OB) for each source routine (.FR).

Executable Program

An executable FORTRAN 5 program consists of the compiled object modules and additional modules supplied by the FORTRAN 5 runtime libraries. The Link utility combines these modules into an executable program.

When you issue the F5LD command, AOS invokes Link. Link performs two functions: it binds all of the object modules together and supplies modules for runtime routines from the runtime libraries (files with the .LB extension). The output of Link is an executable program which you can execute (a file with the .PR extension).

Code

Code refers to the executable machine language instructions that occupy either 1 or 2 16-bit words in main memory. The executable code in your program is one of two classes:

- User code, produced either by the FORTRAN 5 compiler or from your assembly language sources (if you have any).
- Runtime code, supplied by FORTRAN 5 from the runtime libraries.

Almost all of the code in a FORTRAN 5 program is shareable. If several users execute the program at the same time, only one copy of the shared code must exist in main memory for all users of the program. As a result, memory usage is decreased.

Data

Data consists of space for the variables, arrays and constants in your object program. Data is also the temporary storage space required by the code.

There are three different types of data in your program:

- User data which is space for user variables, arrays, and constants.
- Runtime data which is space that the FORTRAN 5 runtime routines require for temporary storage.
- System data which is space that AOS requires for information about your process and its tasks.

Most of the data in your program is unshared. Therefore, each user executing your program at a given time has his own copy of the data. Some of the constants in your program may be shared since the program cannot alter constants. All users of the same program can use the same copy of the constants.

Process

Your executable program, together with a set of system resources is called a process. These resources include main memory, I/O devices, the floating-point unit, and the Central Processing Unit (CPU). A process competes for resources with other processes which exist on the computer as it executes.

Each process consists of one or more tasks.

Tasks and Multitasking

A task is a single flow of control through a program, and is a logically complete unit of program execution. A program having only one task is called a single-task program. While executing, a task uses process resources such as memory and CPU time. A program can have from one to thirty-two tasks.

AOS has the ability to synchronize execution of more than one task at a time. A multitask program consists of multiple, concurrent flows through the program.

During execution of a program, the various tasks compete with each other for the resources of the process. The AOS multitask scheduler controls this competition by allocating resources to the highest priority task that is ready to execute.

For more information about multitasking, see Chapter 4, "Multitasking."

Figure 3-1 shows the operation of a multitask process.

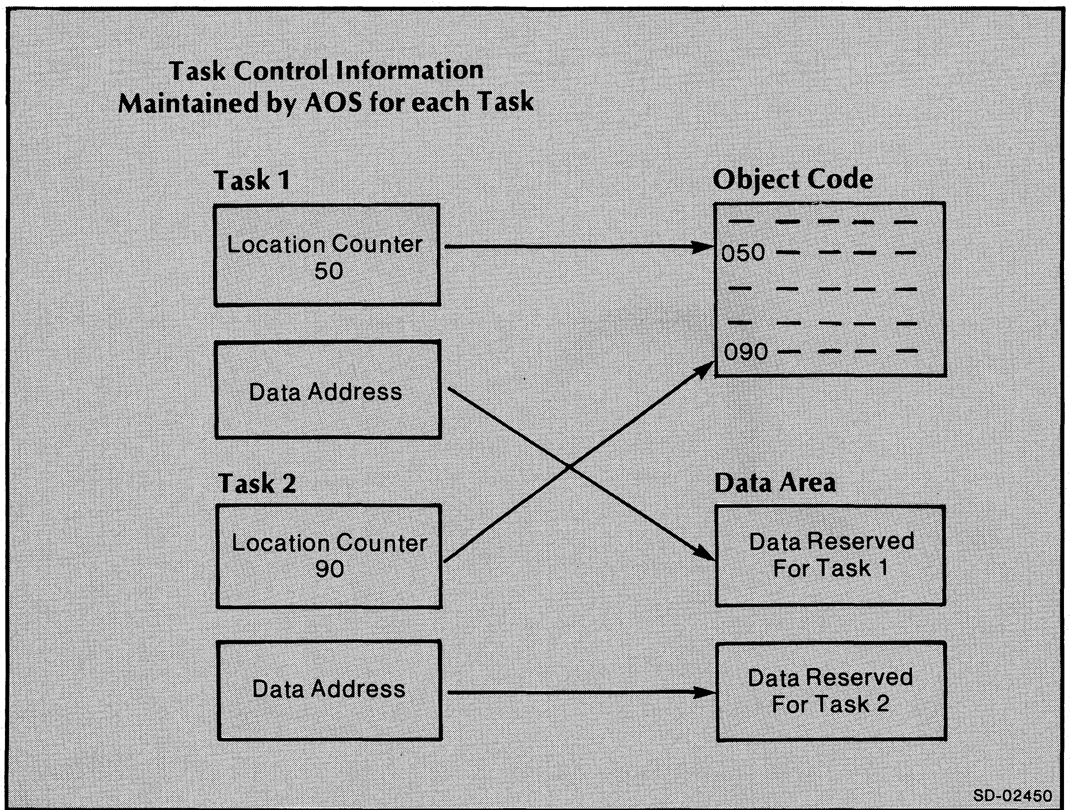

Figure 3-1. Multitasking

Resources

AOS treats the computer as if it is made up of many separate resources available to its users. Resources include the main memory space in which the program will reside, space on disks for the storage of files, and pathways through which the progam can access files. Other resources are physical devices such as magnetic tapes, printers, and card readers. AOS is responsible for allocating the available resources among all users. Your program makes use of some or all of these resources at different times.

We will now detail the different resources in more depth and discuss the way they relate to your program in the runtime environment.

CPU Time

The CPU is the part of the computer that performs logical, control, and arithmetic operations. All functions that a FORTRAN 5 program performs involve use of the CPU, and each machine language instruction specifies an action for the CPU to perform.

AOS manages CPU time as different processes compete for it. Once you start program execution, a process continues running until it either makes an AOS system call or is interrupted by AOS at the end of its allocated time for CPU control. AOS then selects another process for execution. The allocation of CPU time to different processes executing simultaneously is called time-sharing.

Just as processes compete for CPU time, tasks within a process compete for CPU time. AOS examines the priorities of the tasks which are ready to execute, and gives control of the CPU to the highest priority ready task. Each task executes until it suspends itself or until AOS suspends it at at the end of its allocated time for CPU control.

If several tasks have equal priority, then they receive control of the CPU in a round-robin fashion.

Figure 3-2 shows the CPU interacting with various processes in main memory.

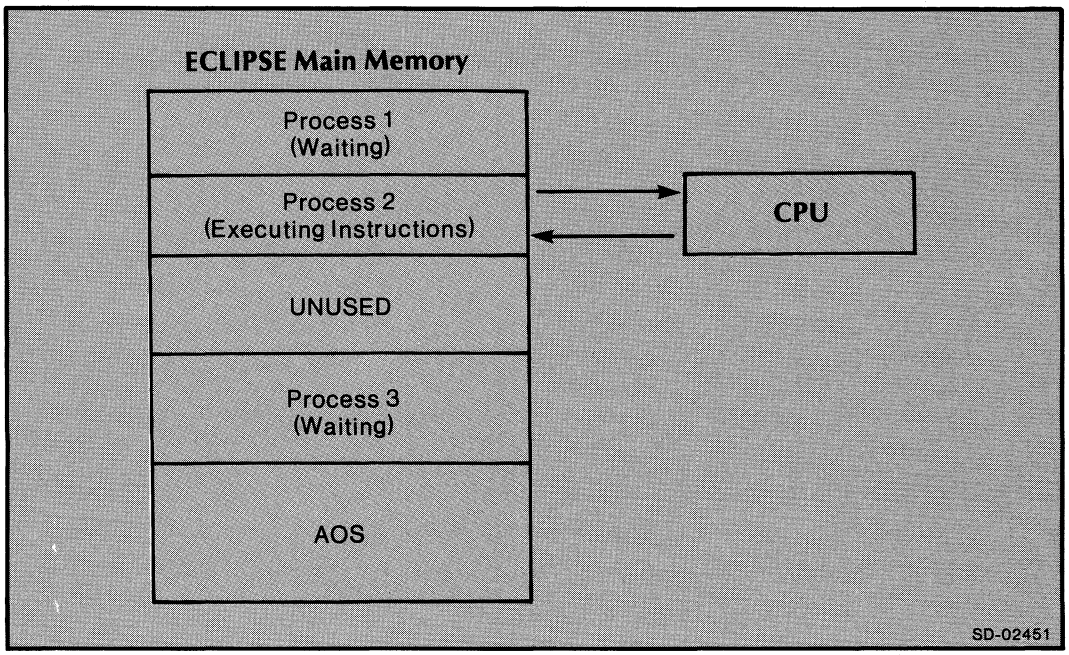

Figure 3-2. The CPU Interacting With Processes in Main Memory

Memory

Before a process can execute, AOS must load the program to be executed into main memory. FORTRAN 5 programs make use of a basic unit of storage in main memory called a 16-bit word. Each word can contain all or part of an ECLIPSE computer instruction or a variable piece of data used in a program.

Each word in memory is uniquely identified by an address. Because FORTRAN 5 treats each 16-bit word as a signed integer, each address in memory must be in the range of 0 to 32,767 (2^{15} -1). Although your ECLIPSE computer may have more main memory available than these 32,768 addressable words, each FORTRAN 5 program is limited to this amount of memory.

The range of addresses possible for a program is called its address space. Within the address space of a FORTRAN 5 program lies portions of its executable code and portions of its data. The ECLIPSE does not execute instructions or access data unless they reside in main memory.

Some portions of your program's code and data can reside on disk. AOS system calls bring these disk resident portions into main memory before the CPU can access them.

Figure 3-3 shows the address space in main memory.

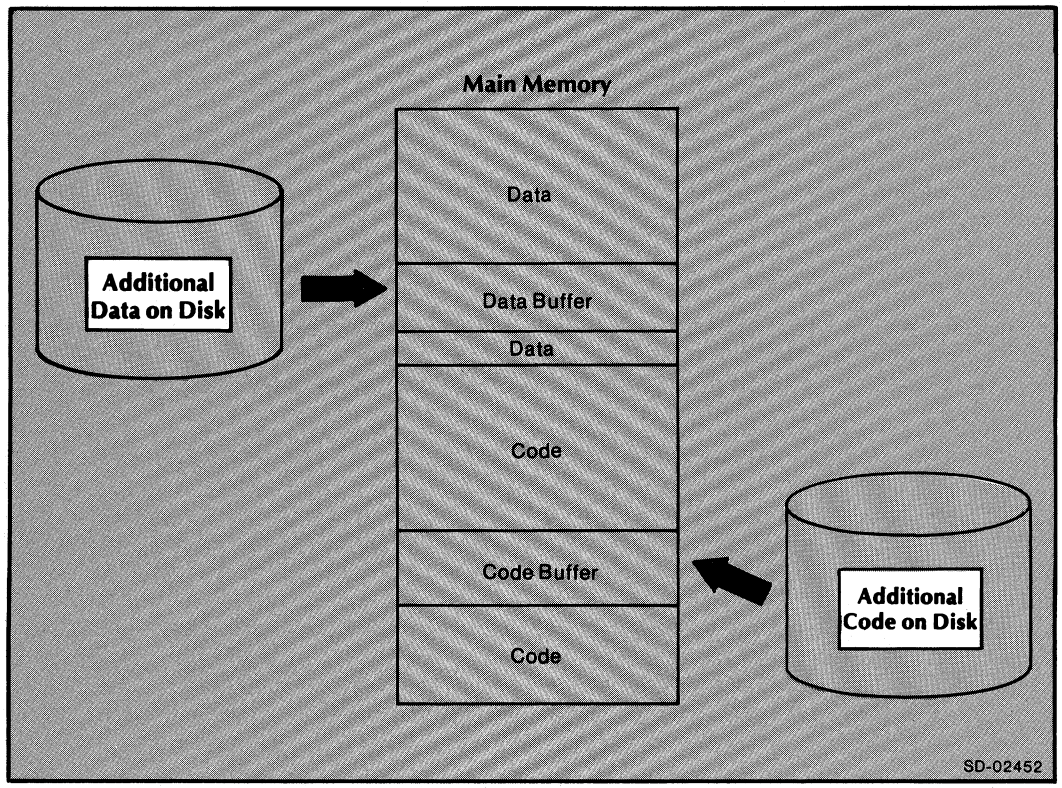

Figure 3-3. Address Space in Main Memory

Input/Output (I/O) Channels

All access to files and devices in AOS must take place along an abstract data path called a channel. Before you can access a file or device, you must open it; ie, AOS must assign a channel number for use when accessing that file and return it to the FORTRAN 5 runtime environment routines. FORTRAN 5 runtime routines refer to a file by its channel number to perform any operations on it. FORTRAN 5 maintains a table that contains the association between AOS channels and FORTRAN 5 unit numbers.

The Runtime Environment

Runtime is the time when the system executes your compiled and linked program. The way the executing program interacts with AOS and the ECLIPSE computer to obtain system resources determines the runtime environment. The runtime code and data that FORTRAN 5 and AOS provide at runtime are a part of the runtime environment.

The Runtime Stack

FORTRAN 5 reserves part of the user data area within each task for an abstract data structure called a runtime stack. During program execution, the program treats it as a last-in, first-out list. It adds items on at the top of the stack (pushes) and removes them in the opposite order from which it added them (pops).

During runtime, the ECLIPSE computer maintains information within main memory about the current top of the runtime stack and its upper limit. ECLIPSE machine language instructions permit the program to push and pop 16-bit words and to examine and alter locations within the stack. If the stack reaches its upper limit, an error called a stack overflow occurs.

Figure 3-4 details the runtime stack.

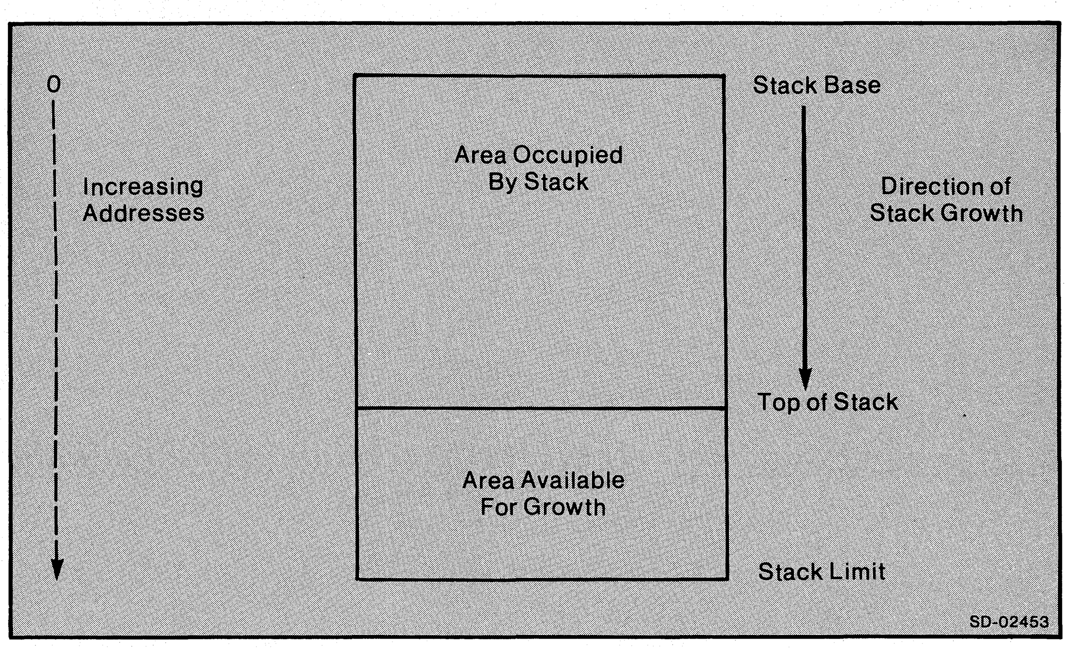

Figure 3-4. The Runtime Stack

FORTRAN 5 maintains copies of all program variables not in COMMON, STATIC, or data-initialized storage on the runtime stack. By referencing these variables, FORTRAN 5 subroutines and functions can call themselves. This technique is called "recursion." Recursive routines are useful for performing some action a variable number of times.

Since each FORTRAN 5 task in a program has its own runtime stack, several tasks can execute code for the same routine at the same time. This technique is called "re-entrancy."

The program also passes the addresses of arguments to subroutine and function calls on top of the runtime stack. The program also saves the current state of the executing routine on top of the stack before a called routine begins execution. This permits the program to restore the caller's state of execution upon return from the called routine.

Runtime routines also make use of the runtime stack.

COMMON and STATIC storage, constants, and some runtime and system data are not maintained on the runtime stack.

Figure 3-5 shows the runtime stack before and after a push operation.

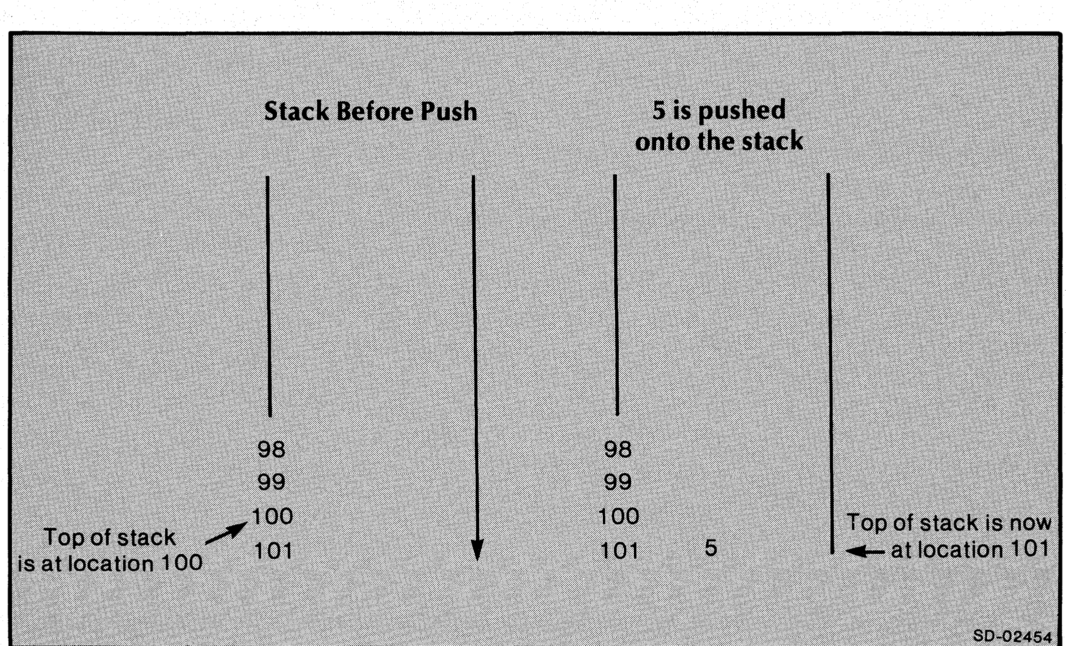

Figure 3-5. A Push Operation

Runtime Memory Allocation

AOS determines the amount of memory it needs for a FORTRAN 5 program by the amount of memory required at three levels of activation: per-process, per-task and per-routine-activation.

Per-process data includes executable code, COMMON blocks and STATIC variables, locations containing pointers to runtime routines, and data maintained by the FORTRAN 5 runtime environment and AOS. Each process has one copy of its per-process data. All tasks active within that process can access this data. Per-process data is also known as process global data.

AOS and FORTRAN 5 maintain per-task data separately for each task in a process. This data includes information on each task's processing state:

- the contents of the CPU registers in which arithmetic operations are performed
- its program counter, which contains the address of the next machine instruction to be executed
- the task's runtime stack and its associated pointers. (partitions and state variables are described later)

Each routine a task executes can access this data. The AOS scheduler and the FORTRAN 5 runtime environment routines coordinate the use of each copy of this data.

In single-task programs AOS maintains only one set of per-task data.

FORTRAN 5 maintains per-routine activation data on the runtime stack. Each routine activation causes FORTRAN 5 to create another copy of that routines local variables and arrays on the stack. The runtime environment allocates this space just before a routine begins execution, and releases it when the routine finishes execution. The data's lifetime is therefore only the length of time that the routine is executing. Space for routine argument addresses, the subroutine return address, and temporary storage for intermediate results from calculations is also maintained on the runtime stack for each activation of a routine.

Stack Partitions

A stack partition is an area of memory that AOS sets aside for a task's per-task data. The stack partition includes a per-task database called the Global Area, and space for the task's runtime stack. FORTRAN 5 allocates a stack partition for a task when the task is initiated, before it begins execution. FORTRAN 5 frees the stack partition when the task terminates. FORTRAN 5 maintains a list of available stack partitions within the runtime environment.

Multitask stack partitions have a fixed allocation which occurs before the program begins executing. Thus, any stack overflow in any task causes a fatal runtime error.

In a single-task program, AOS allocates only enough 1024-word pages of memory for the stack to permit the main program to start execution. If a stack overflow occurs during program execution, the FORTRAN 5 stack overflow handling routine requests enough additional pages of memory from AOS to continue executing the program. Once AOS allocates all addressable memory to the process, FORTRAN 5 reports a fatal runtime error for any additional stack overflows.

File Input/Output (I/O)

At runtime, much work takes place during the execution of a single OPEN, CLOSE, READ, or WRITE statement in FORTRAN 5. When you specify an I/O operation for a certain unit number through an I/O statement, the compiler translates the statement into one or more calls to FORTRAN 5 runtime environment routines. These routines call AOS to carry out the I/O operation.

FORTRAN 5 runtime routines that perform input and output also call AOS to perform I/O operations. All I/O system calls to AOS take place through a process called the ghost in AOS and the agent in AOS/VS. The ghost or agent buffers most data between a file and a user program. This eliminates the necessity of maintaining large data buffers in in the user's address space.

Open

A FORTRAN 5 runtime routine makes an AOS system call to open the file you name and associate a unit number with that file. AOS then returns the number of a channel that is associated with the open file. FORTRAN 5 uses that channel number for all further requests to AOS that refer to that unit number.

Read/Write

If you perform a formatted READ or WRITE statement, FORTRAN 5 runtime environment routines perform any necessary reformatting of the data. During the course of a single READ or WRITE statement, the FORTRAN 5 runtime environment routines create a data area for their use on top of the user's runtime stack. This data base is called an I/O Control Block (IOCB). The runtime environment routines associate this area when the code invoked for the I/O statement completes execution.

Figure 3-6 shows a Read operation in action.

Close

When you close a file, a FORTRAN 5 runtime routine issues an AOS system call to release the channel number. The channel number assigned to that file becomes available for reuse.

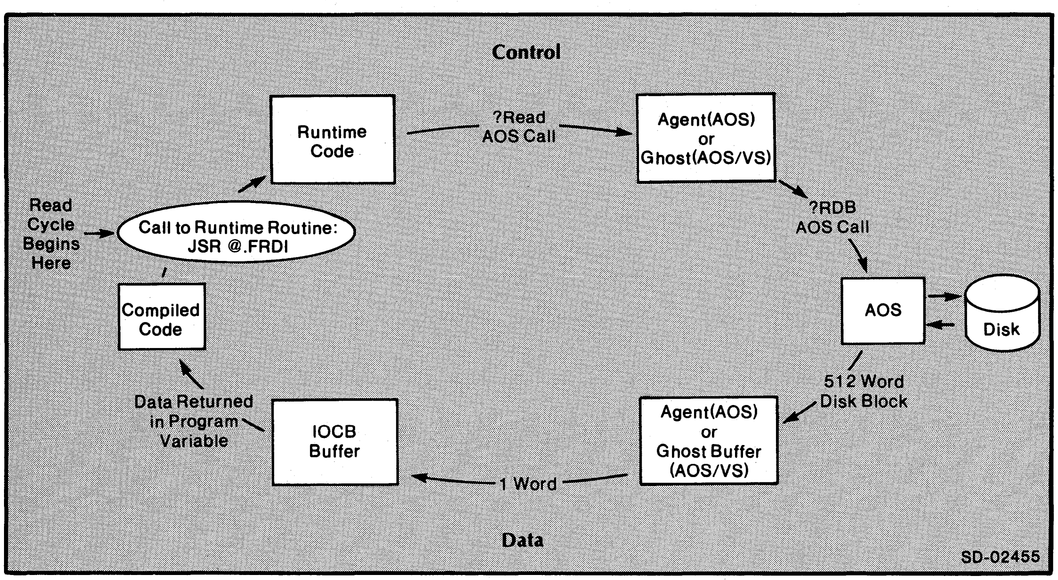

Figure 3-6. I/O Operation

Configuration of Main Memory

In the following section, we describe the layout of main memory for both single-task and multitask programs as depicted in Figure 3-7. We will begin at the smallest address (location 0) and move upward through the address space to the maximum address (location 77777_8).

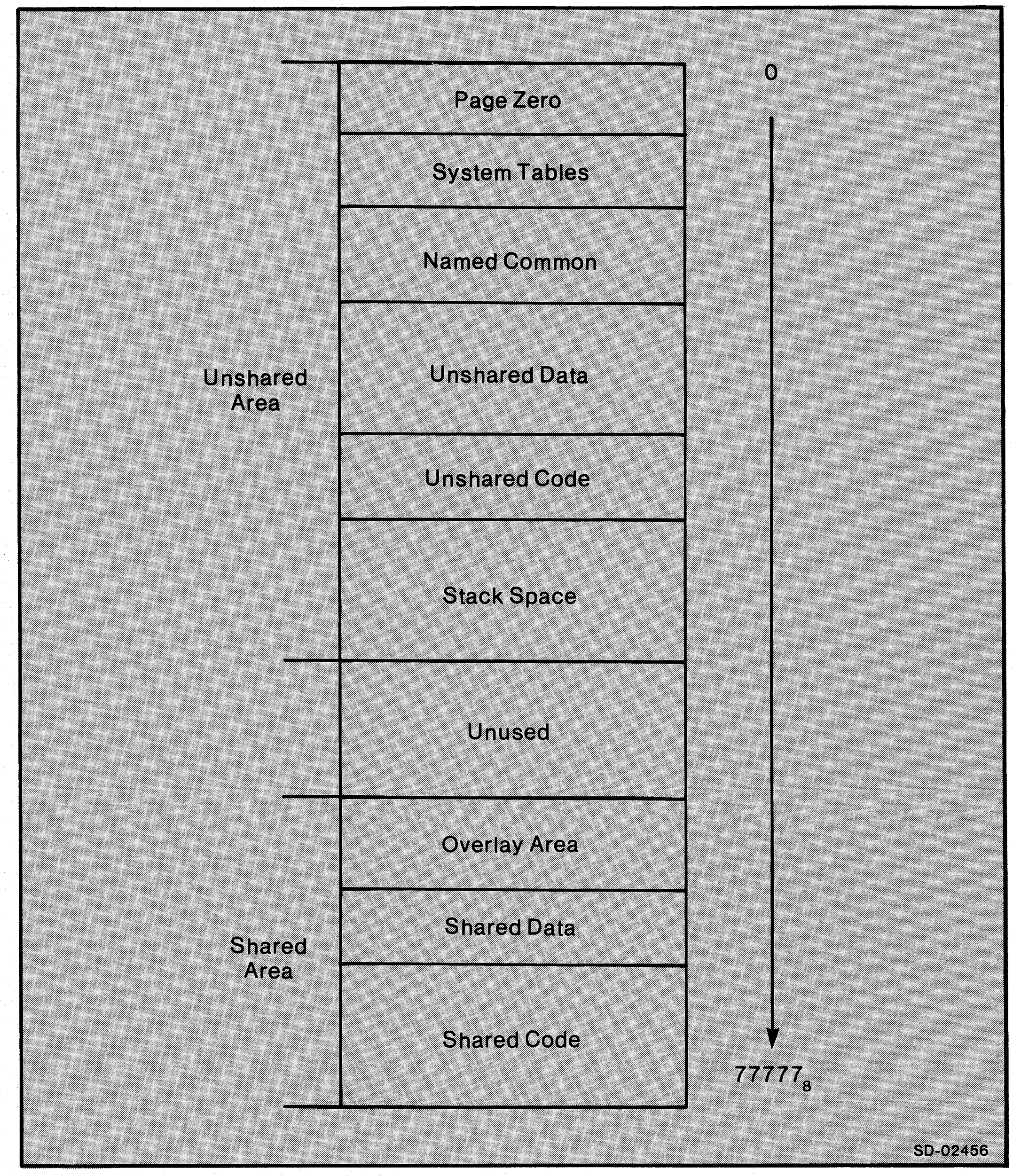

Figure 3-7. Layout of Main Memory

Page Zero

The first 2000_8 locations in the address space are known as *Page Zero* . The object program can access the first 400_8 words of Page Zero using the eight bit offset in a one-word ECLIPSE instruction. These locations are therefore convenient for use as frequently used variables and pointers. AOS and FORTRAN 5 maintain the state variables for the currently executing task and pointers to the FORTRAN 5 runtime environment routines here.

The User Status Table (UST)

Following Page Zero is a set of tables maintained by AOS and the FORTRAN 5 runtime environment routines. The User Status Table (UST) is a per-process data area in which AOS maintains information on the process. This information includes the number of tasks in the process and the location of other system databases.

The Task Control Block (TCB)

The TCB is a per-task data area in which AOS maintains such information about a task as its task identification number (ID), its priority, and locations for maintaining the contents of its accumulators and program counter. In 16-bit AOS, the Task Control Blocks (TCBs) are located above the UST. In AOS/VS, the TCBs are not maintained in the program's address space, but instead reside in the agent.

The Overlay Directory

Next, in programs containing overlays, is an overlay directory in which AOS maintains information on overlay areas within the program. This information includes which overlay is presently being loaded in the program and the number of tasks executing routines in that overlay. For more information about overlays see Chapter 20, "Using Overlays."

TCB Extensions

Following the task control blocks, are a set of per-task data areas called the TCB-extensions. Within each task's TCB extension, FORTRAN 5 and AOS maintain task state information not contained in the TCB itself. In AOS, a pointer to the TCB-extension exists in the TCB. In AOS/VS, FORTRAN 5 maintains a pointer to the TCB extension in the per-task variable, ?USP .

COMMON Blocks

After the TCB extensions is the fixed per-process program data. This includes COMMON blocks, STATIC variables, and DATA-initialized variables.

Unshared Code and Data Partitions

Above these static data areas is the unshared code partition. Located here is any unshared code which exists in the program. By default, the only unshared code in a FORTRAN 5 program is the FORTRAN 5 runtime initializer. This is a routine that establishes the runtime environment data areas. Most of the code space for the initializer becomes a part of the address space available for the runtime stack space.

Following the unshared code partition is the unshared data partition. This area contains the runtime stack partition for each task.

Unallocated Region

In single-task programs, AOS does not allocate the portion of the program's address space between the unshared and shared areas until the program requires the memory addresses in this "no man's land" for growth of the runtime stack. No unallocated region exists in multitask programs.

Overlay Area

Next, in programs containing overlays, is the area into which AOS loads the overlays. For more informaton about overlays, see Chapter 20, "Using Overlays."

Shared Data Partition

Above the unallocated space or the overlay area is the space for the shared data partition. It contains constant per-process data such as literals passed as arguments to subroutines or functions. Link places this passed literal data here so that it's addresses can be passed between routines in different overlays.

Shared Code Partition

Following the shared data partition is the shared code partition. In this area are three types of code:

- compiler-generated user code
- code supplied by AOS for system call interfaces between the program and AOS
- FORTRAN 5 runtime code

Address Space

Link always allocates the shared portions of the program from the top of the address space downwards in memory. It allocates the unshared portions of the program from location zero upwards in memory.

Multitask programs use all 32KW of available address space. This is because the runtime initializer allocates stack partitions before the program begins execution. In single-task programs, the amount of memory used grows with the needs of the program's runtime stack. In the case where the single-task runtime stack does not take up all of the available space, AOS does not allocate the logical addresses between the unshared data area and the shared data area. Thus, the process does not waste memory space that other processes could use.

FORTRAN 5 Runtime Databases

FORTRAN 5 maintains several data areas on either a per-process or per-task basis. These databases are located in the unshared data partition. For more information see Appendix (E), "FORTRAN 5 Runtime Databases."

File Control Tables

The file control table is a per-process database in which FORTRAN 5 maintains information about each FORTRAN I/O unit number. This information includes whether or not that unit number is currently assigned to an open file, and if opened, which AOS channel number is assigned to that unit. AOS also maintains information on the attributes of the unit, such as whether the file is line-oriented or blank-padded.

Task Global Area

At the bottom of each task's stack partition is its task global area. A per-task variable in page zero contains a pointer to the current task's global area. The task global area contains the I/O and task control information.

I/O Control Block (IOCB)

The I/O control block (IOCB) is a per-task database. FORTRAN 5 creates it on the task's stack during the lifetime of a single FORTRAN 5 read or write operation, or during a FORTRAN 5 runtime routine call which performs a read or write. The IOCB contains all necessary data for the I/O operation, including a buffer for line-oriented and record-oriented transfers. Also contained in the IOCB is the AOS system call packet used for the I/O system calls. A word in the task's global area contains a pointer to the current IOCB.

End of Chapter

Chapter 4
Multitask Programming in FORTRAN 5

This chapter presents general concepts of multitasking, whereas Chapters 14 through 19 detail the multitasking routines. Before reading this chapter, read about multitasking in the *FORTRAN 5 Reference Manual*.

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

Tasks and Their Resources

In a multitask environment, tasks share physical resources. FORTRAN 5 and AOS manage these resources together, depending on a particular task's resource requirements. The current state of a task's resources defines the task. For a more detailed description of resources, see "Chapter 3 Runtime Environment Fundamentals," and Chapter 5, "The FORTRAN 5 Assembly Language Interface."

AOS handles the accumulators, carry, unique storage position (USP), the hardware stack, and the program counter. AOS manages these resources through the task control block (TCB) for each task.

For example, AOS allocates an area of memory for each task to store its copy of the accumulator's values in when the task isn't executing. The multitask scheduler saves and restores the task's state.

FORTRAN 5 handles memory partitions, floating point unit, and page zero locations called *task state variables* (.SP, .FP, .SSE .GP, .RP). FORTRAN 5's management of resources utilizes the TCB extension.

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

Example

One task within your program might communicate with a terminal to get requests to examine the data file of an accounting record. Concurrently, another task could access the data file itself through READ statements. A third task could record the request made by executing WRITE statements to a logging file. Although each process actually executes only one instruction in one task at a given time, AOS switches execution control between tasks so rapidly that all tasks seem to be executing simultaneously.

Non-FORTRAN 5 Tasks

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

Tasks written entirely in assembly language may not need any of the FORTRAN 5 resources such as stack partitions. Therefore, you can avoid wasting memory or FORTRAN 5 resources, by defining the partition specification as 100000K in the stack size parameter. This partition specification prevents the task from receiving a TCB extension or a FORTRAN 5 memory partition.

Memory Partitions in a Multitask Environment

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

Changing Default Memory Partitions

Since tasks may require different amounts of memory, the allocation of default size partitions may be inefficient for some programs. You may want to explicitly specify the size of the memory partitions that will be allocated for each task. Your program will then require less memory space at runtime.

You can use the files PARTITION.SR and DPART.SR supplied with FORTRAN 5 to create a partition specification table. The runtime initializer uses this table at runtime to control the allocation of task partitions. By editing DPART.SR in the manner described in PARTITION.SR, and by assembling and linking DPART.OB into your program, you can define the number and size of stack partitions to be allocated. Then, when you initiate a task within your program, you can use the partition specification parameter for the task initiation request. You can select the exact stack size required for the request. The partition specifier is described later in this chapter.

The amount of space each task requires depends on the following:

- The nesting of calls made by the task to subroutines and runtime routines.
- The number and size of local variables and arrays allocated on the stack by each subroutine and runtime routine executed by the task.

The process of "customizing" the stack requirements of each task is a process of trial and error. Therefore, you will probably want to do it for fully debugged programs only, to maximize their efficiency.

When you terminate a FORTRAN 5 task, its stack partition is returned to the pool of available partitions by the runtime environment routines. You can then reallocate the partition for another task.

The size of default partitions depends on three variables:

- The total amount of memory available for partitions
- The amount of memory allocated for fixed-size partitions
- The number of tasks you specify (via Link's /TASKS= switch) when you Link the program.

The runtime initializer first allocates space for any fixed-size partitions you specify through DPART.SR. If you have explicitly requested a specific number of default size partitions in DPART.SR, the runtime initializer allocates only that number of default size partitions. If you do not explicitly specify a number of default size paritions, the runtime initializer will allocate enough default partitions to insure that at least one partition exists for each task. These extra partitions are allocated from memory which remains available after any fixed-size partitions are allocated.

The FORTRAN 5 runtime initializer apportions available memory according to the partition specification table. The section in Chapter 1 on "Limiting the Amount of Memory Available to the FORTRAN 5 Environment" details how you can restrict the amount of memory the runtime initializer treats as available memory.

Allocating Memory Partitions

When you initiate a FORTRAN 5 task, the runtime initializer allocates a partition from the pool of available partitions. (If none is available, then you get an error message.) The runtime initializer assigns a partition according to the task's stack size parameter. The system specifies the stack size in several ways:

- If you initiate the task using the TASK statement then you can designate the stack size with the STK= option. (See the *FORTRAN 5 Reference Manual* for details). If you do not use this option, the runtime initializer allocates a default size partition.
- For the runtime routines ASSOCIATE, FTASK, IOPROG, and ITASK , you can designate the stack size as an argument to the call . The stack size parameter is optional for FTASK, IOPROG, and ITASK . If you do not specify a stack size, the runtime initializer allocates a default stack.
- If you initiate the task from assembly language with the macro call, S?TASK , then the program passes the stack size parameter in an accumulator.
- DPART.SR specifies the stack size for the FORTRAN 5 main program as a parameter. By default, the runtime initializer allocates a default size partition to the main program.

The following 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 the runtime initializer can't find an available partition 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 the program passes to it, plus one additional word if the program is a function of a subprogram
- Five 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

Various classes of task suspensions exist. For information on multiple suspensions, refer to the *AOS Programmer's 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 5
FORTRAN 5 Assembly Language Interface

Assembly language is a direct symbolic representation of the machine code that the ECLIPSE computer executes. Like FORTRAN, assembly language removes the requirement that you program in the binary machine language of the ECLIPSE computer. Like FORTRAN, assembly language permits you to assign symbolic names to variables instead of referencing specific locations in memory. Unlike FORTRAN, each executable statement of assembly language translates into a single machine instruction. The compiler may translate a single executable FORTRAN statement into many machine instructions. Thus, with assembly language, you have very direct control over what you are doing and how it is carried out.

This chapter is for those who want to code their own assembly language runtime routines. It provides a more in-depth view of the runtime environment than Chapter 3, and will help you understand the assembly language code the compiler generates. All the figures in this chapter are intended to depict the general layout of FORTRAN 5 data areas, not their exact format.

You will better understand this chapter if you have some familiarity with ECLIPSE assembly language, but this is not a necessity.

Why Write Assembly Language Routines?

There are three main reasons for writing your own assembly language programs:

- You may want to do something that you can't do directly from FORTRAN 5. An example of this is performing operations on non-FORTRAN data types such as packed-decimal, using ECLIPSE commercial or character instruction sets.

- You may want some part of the program to be as fast as possible for a real-time application such as device interrupt handling.

- You may want to write a runtime routine not available in the FORTRAN 5 runtime libraries.

In general, if you have some portion of a program which FORTRAN 5 cannot do efficiently, you should consider assembly language.

ECLIPSE Architecture Introduction

We will take a moment to describe the ECLIPSE architecture. The FORTRAN 5 compiler and runtime routines use an instruction set which performs operations on the following:

- 16-bit integers

- 32-bit single precision floating point numbers

- 64-bit double precision floating point numbers

All integer arithmetic in the program takes place in 4 16-bit general purpose CPU registers, called accumulators or ACs. Floating point arithmetic takes place in 4 64-bit floating point registers, called floating point accumulators or FPACS. The CPU performs most arithmetic operations by loading the operands from main memory into the appropriate type of registers, performing the operation, and storing the result back into main memory. For a description of the ECLIPSE instruction set, see the Principles of Operation manual for the model of ECLIPSE you use.

The FORTRAN 5 Runtime Stack Discipline

Certain page zero state variables define the stack activities. When the AOS task scheduler gives control to a task, it sets up these state variables in page zero. The contents of these words describe the per-task data area that the executing task will use. When a task is not executing, its values for the state variables are stored in its per-task data area (TCB or task global area).

Both AOS and FORTRAN 5 make use of the state variables. The following is a list of these variables and their functions:

Name	Location	Purpose
.SP	40_8	Stack Pointer; contains a pointer to the location which is the current top of the runtime stack.
.FP	41_8	Frame Pointer; contains a pointer to the current routine activation data on the runtime stack.
.SSE	42_8	Stack Limit or Stack Extent; pointer to the last location which is available for the runtime stack.
.SOV	43_8	Stack Overflow Handler Address; pointer to the stack overflow handling procedure. In the event of a stack overflow, this mechanism acquires more space for the stack from AOS, or reports a runtime error.
.RP	$< 400_8$	Return Pointer; the FORTRAN 5 runtime environment support routines use this as a temporary storage area for return addresses and other information.
.GP	$< 400_8$	Global Pointer; contains a pointer to the task's global area.

In AOS/VS, ?USP (location 16_8 contains a unique storage pointer to a database called the TCB extension.

The Stack Frame

The ECLIPSE computer SAVE instruction creates the stack frame. The stack frame contains the contents of the calling unit's AC0, AC1, and AC2. It also contains the contents of the frame pointer at the time the routine was called. The stack frame contains 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 return address). The stack frame also has space reserved for local storage.

The SAVE instruction is normally the first instruction of a routine. It does the following:

1. Pushes the contents of AC0, AC1, and AC2 onto the stack in order.
2. Pushes the current value of the frame pointer (.FP) onto the stack.
3. Concatenates the carry bit and the rightmost 15 bits of AC3 and pushes them onto the stack. (This saves the value of the return address for the calling routine.) A JSR or EJSR instruction which calls this routine places the return address into AC3.
4. Places the current value of the stack pointer (.SP) in the frame pointer, (.FP).
5. Finally, increments the stack pointer (.SP) by the number of words specified as the argument to the SAVE instruction. (This allocates storage for per-routine activation data.)

Figure 5-1 details the stack frame.

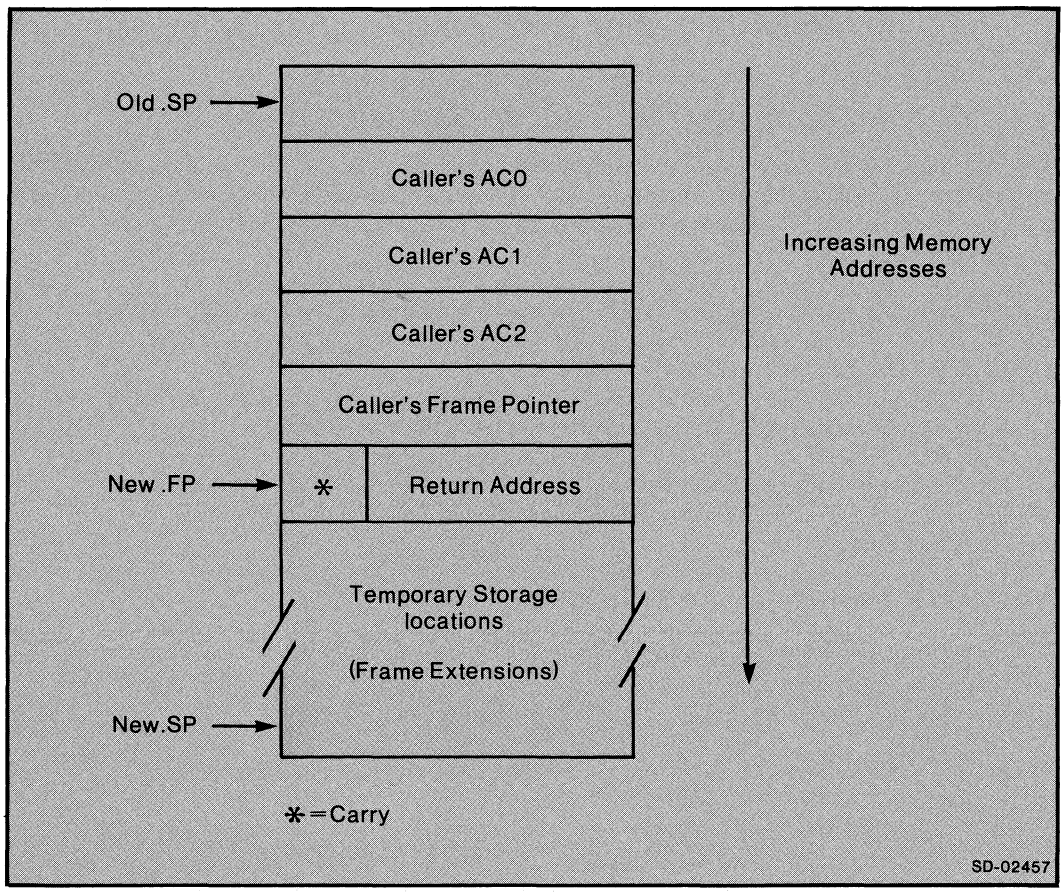

Figure 5-1. The Stack Frame

Using the Stack

The machine instruction set of the ECLIPSE computer contains several instructions for using the stack. They are

PSH Places the contents of one or more accumulators on top of the stack in ascending order.

POP Removes the top words from the stack and places them into one or more accumulators in descending order.

SAVE Creates a new frame on the stack. SAVE places a "return block" on the stack which retains the state of the calling routine's accumulators and program counter for resumption on reactivation.

RTN Removes the last stack frame from the top of the stack. The RTN instruction causes a return to the calling routine by reversing the operations of the SAVE instruction. The contents of the accumulators and the carry bit are restored. The frame pointer and the stack pointer are restored to their previous value, and control returns to the address pushed from AC3.

Subprogram Linkage Conventions

You use a technique known as Call-by-Reference to pass arguments to subprograms. With this method, you pass the arguments to a subprogram by pushing their addresses onto the stack in reverse order (the address of the last argument is pushed first). All addresses are 15-bit word addresses.

This list summarizes the actions FORTRAN 5 requires for a subprogram call:

1. Load the stack pointer (.SP, location 40_8) in AC2.
2. Push the addresses of the arguments onto the stack in reverse order.
3. If calling a function, push the address of the variable to receive the returned value.
4. Call the routine.
5. On return, store AC2 into the Stack Pointer (.SP).

When the called subprogram begins, the end of the argument list is one word after the address passed into AC2. The passed value in AC2 is known as the Stack Marker. Figure 5-2 shows the runtime stack at various stages of a subroutine call.

Stage 1 What the stack looks like before the argument address are pushed.

Stage 2 What the stack looks like after the argument addresses are pushed.

Stage 3 What the stack looks like after the called routine executes the SAVE instruction.

Once you push the argument addresses, you call the subprogram with the AOS resources call, ?RCALL . ?RCALL first places the current value of the program counter, the return address, into AC3. ?RCALL then transfers control to the address you specified as its argument.

The SAVE instruction places the return address on the stack. We recommend that you use the FCALL macro in your assembly language routines to invoke ?RCALL . We describe FCALL later in this chapter.

Stage 3 in Figure 5-2 shows the stack after execution of the SAVE instruction in the called subprogram.

When the called subprogram returns via the RTN instruction, the calling routine resumes execution at the location after the ?RCALL .

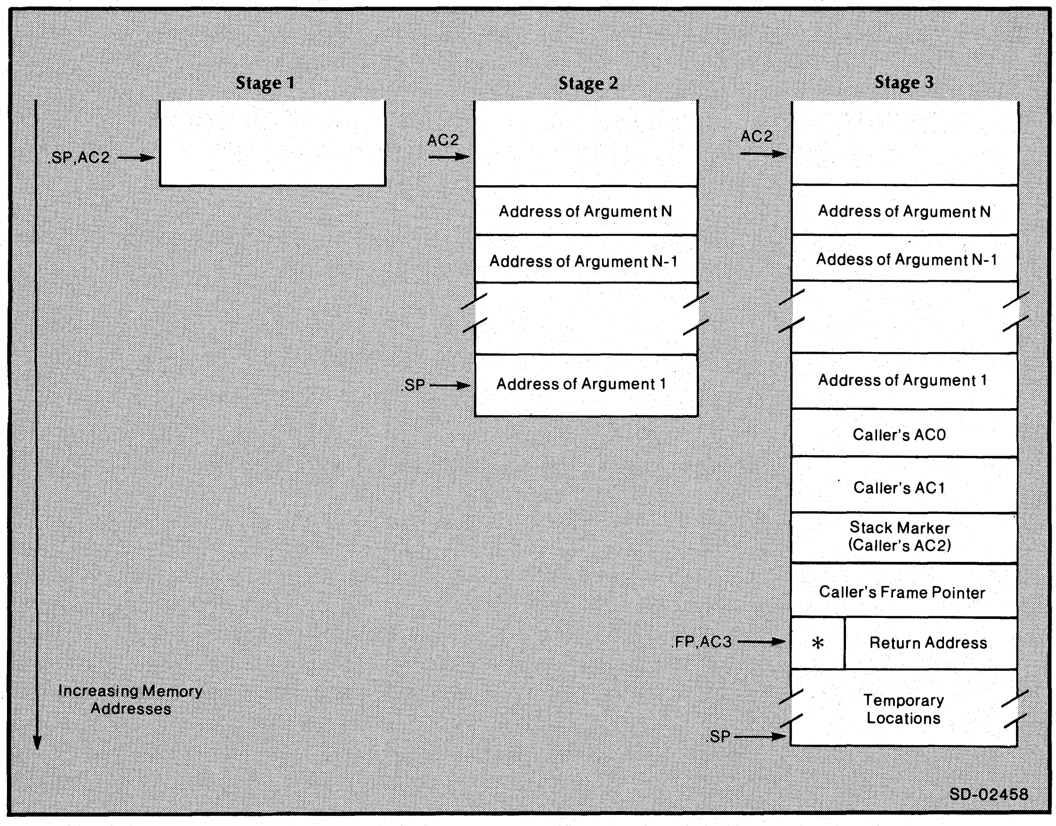

Figure 5-2. The Runtime Stack at Various Stages of a Subroutine Call

Assembling Your Assembly Language Routines

This section describes the actions you must perform to assemble any assembly language source file into an object file (.OB). You can then include the object file in your programs when you link them with the F5LD command. You must follow this procedure when you assemble your own assembly language routines and when you assemble one of the assembly language routines supplied by FORTRAN 5.

First, build a FORTRAN 5 permanent symbol file if one does not already exist on your AOS system. You need to do this only once. You can use a permanent symbol file for all FORTRAN 5 assembly language sources. Only rebuild the permanent symbol file if you begin using a new version of FORTRAN 5. The next section of this chapter, "The Permanent Symbol (.PS) File", describes the permanent symbol file and how to create it.

Once you create a permanent symbol file, you assemble your assembly language source files with the macroassembler (MASM.PR in AOS and MASM16.PR in AOS/VS).

In AOS, the format of the MASM command is

X MASM/8 */L=listpathname/B=objectpathname/PS=pspathname* sourcefilename

In AOS/VS, the format of the MASM command is

X MASM16/8 */L=listpathname/B=objectpathname/PS=pspathname* sourcefilename

The /8 switch directs MASM to generate 8-character symbols rather than the default 5-character symbols. The optional /L= , /B= , and /PS= switches specify the pathnames of the listing file, the object file, and the permanent symbol file respectively. SOURCEFILENAME is the pathname of the source file. If your FORTRAN 5 permanent symbol file has a name other than MASM.PS in AOS and MASM.16.PS in AOS/VS, you must use the /PS= switch to identify it.

You cannot use the MASM.PS supplied in :UTIL or the MASM16.PS supplied by AOS/VS in :UTIL to assemble your FORTRAN 5 assembly language routines.

If you do not specify the /B= switch for the assembly, the assembler names the object sourcefilename.OB if the source file is named sourcefilename.SR .

The Permanent Symbol (.PS) File

If you know what an assembler .PS file is then you can skip this section.

The Permanent Symbol File is a pre-assembled version of assembly language symbols and macros. It is supported by the macroassembler. If you specify the /S switch when executing the assembler, it scans the source files named in the command line and builds a permanent symbol file. By default, the permanent symbol file is called MASM.PS in AOS and MASM16.PS in AOS/VS. From then on, whenever you invoke the assembler the source program being assembled can refer to symbols and macros in the .PS file. The source program treats these symbols as though they are defined in the source file itself. Furthermore, the /S switch permits you to change symbolic values within all source files without having to edit the source files themselves.

By using the MASM.PS file, you can write "Parametric Programs". You can avoid coding absolute values for packet offsets, error code numbers, and system parameters into your source programs by using symbolic values. If you use these symbols and any of their values change, you need not recode your routine; just reassemble it.

The following examples show how to build a FORTRAN 5 MASM.PS.

In AOS:

X MASM/8/S */PS=pathname* [AF5SYM.AS]

In AOS/VS:

X MASM16/8/S */PS=pathname [VF5SYM.AS]*

If you specify the /PS= switch, you can create a .PS file named anything other than MASM.PS; e.g., F5MASM.PS. This permits you to maintain different permanent symbols files for different uses. You can name the desired permanent symbol file via the /PS switch when you assemble your routines.

Files Which Make Up the FORTRAN 5 Permanent Symbol File

In this chapter, we do not attempt to list all of the symbols and macros defined in the FORTRAN 5 MASM.PS. However, we will highlight the most important symbols and macros. You can become familiar with the contents of the permanent symbol file by examining the source files that compose it. These source files are named later in this section.

The Symbols that FORTRAN 5 uses fall into two groups:

- Operating system defined symbols
- FORTRAN 5 defined symbols

The operating system defined symbols include the instruction op-code specifications, system call parameters, and system error codes. EBID.SR and ECID.SR define the instruction op-codes. The system call parameters and system symbols are defined in PARU.SR (in AOS) or PARU.16.SR (in AOS/VS). The system call definitions are in SYSID.SR in AOS and SYSID.16.SR in AOS/VS.

The primary FORTRAN 5 symbol files are F5SYM.SR and FMAC.SR. F5SYM.SR defines the core of FORTRAN 5 symbols and macros. FMAC.SR defines symbols and macros that FORTRAN 5 shares with Data General's FORTRAN IV.

Two additional files, AF5SYM.SR and VF5SYM.SR define whether the permanent symbol file is for AOS or AOS/VS respectively. The remaining files that compose the .PS file contain additional symbols and macros, some of which are described later.

Using the symbols and macros in the .PS file ensures that changes in FORTRAN 5 will not affect your assembly language source files.

In addition to assisting you in creating parametric programs, the MASM.PS file can assist you in creating operating system independent source routines. You can use the majority of the macros defined in MASM.PS in any runtime environment. For those macros and symbols which you cannot use in all environments, the assembler and the FORTRAN 5 MASM.PS define a conditional assembly feature and a set of conditional assembly symbols.

The following symbols are defined as switches for use with the conditional assembly pseudo-ops (.IF, .DO, and .ENDC):

Mnemonic	Target Environment
NSW	Conditional code is for NOVA® computers
ESW	Conditional code is for ECLIPSE computers
MVSW	Conditional code is for ECLIPSE MV/8000 computers
RSW	Conditional code is for RDOS or RTOS
RDSW	Conditional code is for RDOS but not RTOS
RTSW	Conditional code is for RTOS but not RDOS
ASW	Conditional code is for AOS or AOS/VS
AESW	Conditional code is for AOS but not AOS/VS
AVSW	Conditional code is for the AOS/VS but not AOS

An Assembly Language Programming Example

Figure 5-3 is the source code for the FORTRAN 5 runtime routine DIR in F5ISA.LB. We have added line numbers for reference. Following the source code, we explain each line and describe its function.

The source code in Figure 5-3 builds the module for their DIR runtime routine in both AOS and RDOS.

```
1:          ; COPYRIGHT (C) DATA GENERAL COPRORATION 1980.
2:          ; ALL RIGHTS RESERVED.
3:          ; LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATI
4:          ; ON
5:
6:          ; DIR
7:          ;
8:          ; CHANGES THE CURRENT DEFAULT DIRECTORY
9:          ;
10:         ; CALLING SEQUENCE (ISA):
11:         ;
12:         ;        CALL DIR (<DIRECTORY-NAME>, <ERROR>)
13:
14:                  TITLE   DIR
15:
16:         DEFARGS
17:                  DEF     NAME     ;NAME OF NEW DIRECTORY
18:                  DEF     ;IER     ;ISA ERROR RETURN
19:         DEFTEMPS
20:
21:         FEMTRY   DIR
22:
23:                  LDA     1,NAME,3          ;AC1 -> NAME
24:                  MOVZL   1,0               ;AC0 -> NEW DIRECTORY NAME
25:
26:         **.DO ASW
27:                  ?DIR
28:
29:         **.ENDC RSW
30:                  .SYSTM                    ;CALL THE SYSTEM TO
31:                  .DIR                      ;CHANGE THE CURRENT DIRECTORY
32:         **[RSW]
33:
34:                  ISA.ERR                   ;ISA ERROR RETURN
35:                  ISA.NORM                  ;ISA NORMAL RETURN
36:
37:
38:                  END
```

Figure 5-3. Source for DIR

Lines 1-12

Lines 1-12 are comments that contain a copyright, the name of the routine, and the calling sequence.

Line 14

Line 14 is a macro invocation (TITLE) that builds a descriptive line in the listing. For example

```
.TITLE DIR  ;AOS ECLIPSE   FORTRAN 5
```

You can use this information to be sure that the version of the routine you are assembling (RDOS,AOS,AOS/VS and ECLIPSE , NOVA) is correct, based on symbols in the .PS file. TITLE also initializes some variables which other macros need.

Lines 16-18

Line 16 (DEFARGS) is a macro invocation that begins the definition of arguments the calling routine will pass to this routine.

The calling routine will define symbolic names for the stack offsets of the addresses of the two arguments. The stack offset is where the calling routine will place the address of NAME at runtime. The code can then reference that argument without a particular offset in the instructions. Thus, you can write LDA 0,@NAME,3 rather than LDA 0,@-5,3 . Your code will always refer to the proper frame pointer offset, even if that offset changes at some future time.

The DEF macro assigns the frame pointer offset to the symbolic name passed as an argument (NAME).

The DEF in line 17 defines NAME as the symbol for the first argument passed. DEF in line 18 acts as a place holder for the second argument.

This is the ISA error return variable, usually called IER . Note that by placing a semicolon before IER in line 18, we reserve a location on the stack for IER without explicitly defining a symbol called IER . Thus, we prevent conflicts between the name IER and other symbols that begin with IER .

Line 19

Line 19 invokes the macro DEFTMPS which terminates the end of argument definition. It also begins the definitions of temporary locations which this routine can use; although, in this case, it doesn't use any. However, DEFTMPS must appear even if the routine uses no temporary locations. If we did need temporary locations, we would use the DEF macro just as we did after DEFARGS .

Line 21

Line 21 invokes the FENTRY macro. FENTRY defines an entry point for DIR and saves the correct amount of words in the assembled source program. The number of words saved depends on the number of temporary locations DFTMPS reserves and whether the routine calls any other FORTRAN-type variables (in this case 0). TITLE, DEFARGS, DEFTMPS, DEF and FENTRY are defined in FMAC.SR.

Line 23

Line 23 is the first line of real code. Here we load the address of the first argument, NAME , into AC0. The comment uses the notation -> to indicate that AC0 contains the word address of (points at) NAME . Had we included the indirection symbol (@) in this line, the value of NAME , rather than its address, would be loaded.

Line 24

Line 24 converts the 15-bit word pointer to NAME into a 16-bit pointer (byte pointer). The system call we are about to do requires this action. The MOVZL 0,1 instruction moves AC0 to AC1, shifting left one bit to create the byte pointer. The notation => in the comment indicates that AC1 now contains the byte pointer to NAME .

Line 26

Line 26 (**.DO ASW) makes use of two features of the macroassembler. The .DO ASW means that the assembler will assemble the following code only if the symbol ASW has a non zero value. Thus, if we build the AOS version of the routine, the symbol ASW (AOS switch) is a one, and the system assembles the following lines up to .ENDC . The ** on the line causes the system to suppress the listing of this line.

Line 27

Line 27 is an AOS system call macro (?DIR). It causes the system to change the working directory to the directory whose name is passed in AC1 as the byte pointer.

Line 29

The .ENDC RSW in line 29 signals the end of the .DO condition in line 26. The symbol RSW causes the system to skip assembly until it finds a bracketed RSW (line 32). This convention (.DO, .ENDC label , and [label]) provides an IF-THEN-ELSE functionality for the assembler. The code between .DO ASW is included in the AOS version only. The code between .ENDC RSW and [RSW] (lines 30-31) is included in the RDOS version only.

Lines 30 and 31

Lines 30 and 31 cause the system to generate the RDOS .SYSTM .DIR system call for the RDOS versions of this routine. .DIR in RDOS works like the ?DIR system call in AOS.

Lines 34 and 35

Lines 34 and 35 use the ISA.NORM and ISA.ERR macros to put either a one or an error code into the ISA error variable. Because a system call skips the next sequential word if it is successful, you want IER set to one if the next word is skipped by the system call. If the next word is not skipped, an error has occurred and you want IER set to the appropriate error code. The ISA.ERR macro invokes a routine at runtime to put the error code passed back from the system call (in AOS) into IER . The ISA.NORM macro places a one into IER . Both ISA.NORM and ISA.ERR cause the program to execute a return instruction at runtime which returns control to the calling routine.

Line 38

Line 38 (END) is a “clean-up” macro which completes the assembly.

Notes

TITLE, DEFARGS, DEF, DEFTMPS and FENTRY are defined in FMAC.SR.

Calling Other Routines

Your assembly language routines can call other FORTRAN 5 convention routines via the FCALL macro. You supply the name of the routine to be called as an argument to FCALL, as in the following example:

```
FCALL SUBR
```

The routine name (SUBR) is declared external by FCALL. FCALL invokes the AOS resource manager via ?RCALL . FCALL is defined in F5SYM.SR.

About S?ATTR

The macros S?ATTR, TITLE, DEFARGS, DEF, DEFTMPS, and FENTRY are the standard means of setting up an assembly language subroutine for a FORTRAN 5 program. S?ATTR is the only macro not mentioned in the example above. S?ATTR is a macro defined in F5SYM.SR which signals that the routine you are writing is going to call another FORTRAN 5 routine. The calling sequence for S?ATTR is

```
S?ATTR FCALL
```

or

```
S?ATTR RCALL
```

A S?ATTR FCALL indicates that the routine will call another FORTRAN 5 routine. This indication is necessary to reserve extra space for a bookkeeping area in the SAVE generated by FENTRY . An S?ATTR RCALL indicates that the routine will use resource calls directly in AOS and not via the FCALL macro. You must set the RCALL attribute to reserve two words for the ?RCALL manager in the SAVE that FENTRY generates. You need not specify the RCALL attribute if you use the S?ATTR FCALL . S?ATTR must appear after TITLE and before DEFTMPS . S?ATTR is defined in F5SYM.SR.

Writing Routines That Have a Variable Number of Arguments

You can write assembly language routines that have optional arguments. The A?CNT macro enables you to count the number of arguments actually passed to a routine. The format of the call is

```
A?CNT AC
```

where AC is 0 for AC0 or 1 for AC1. The AC receives the count of the number of arguments passed to the routine. Before you can call A?CNT , AC3 must contain the frame pointer, and AC2 must contain the stack marker. This arrangement should exist immediately following the SAVE performed by FENTRY .

Several additional macros for dealing with optional arguments are defined in SMARK.SR, which is included in the .PS file.

Initiating Tasks From Assembly Language

You can initiate tasks from assembly language by using FORTRAN 5 supplied macros. This section describes several of these macros which are defined in F5SYM.SR.

S?TASK

S?TASK initiates a task.

You can use the S?TASK macro to create either a FORTRAN 5 task or a non-FORTRAN 5 task. FORTRAN 5 tasks utilize resources that FORTRAN 5 manages, such as the runtime stack. The calling sequence for S?TASK is described below:

AC0	Left Byte = Task ID Right byte = Task Priority
AC1	Task start address
AC2	Partition size parameter
Error Return	(error code in AC0)
Normal return	(AC0, AC1, AC2 and Carry preserved, AC3 = Frame Pointer)

S?QTSK

S?QTSK Requests delayed or periodic initiation of a task.

You can use S?QTSK to invoke a queued task for execution at some future time. The database the system requires for a queued task is an aggregate of length Q.LEN. It is called a queue table. The queue table consists of an AOS ?TASK packet followed by several words of data for use by the FORTRAN 5 runtime environment routines. The offsets within the queue table are defined in F5SYM.SR and have names beginning with "Q.". The format of the S?QTSK macro invocation is:

AC2	Address of queue table of length Q.LEN
Error return	Error code in AC0.
Normal return	AC0, AC1, AC2, and carry preserved, (AC3 = Frame Pointer).

A?TASK

A?TASK initiates a task and passes information in a queue table rather than in the ACs.

The A?TASK macro gives you more control over the task initiation than that provided by S?TASK.

Offset Q.MEM in the queue table, passed to S?QTSK or A?TASK should contain a partition size specifier.

The partition size specifier indicates whether or not you should allocate a partition for a task, and if so, its size. All tasks which contain FORTRAN 5 compiled code must have a stack partition. See Chapter 4 for a description of partition specifiers.

Accessing COMMON, STATIC, and Data-Initialized Storage

You can refer to a FORTRAN 5 named COMMON block with an external symbol (.EXTN pseudo-op) using the name of the named COMMON block. You address all variables and arrays in a common block relative to the start of the COMMON block.

Because FORTRAN 5 permits data initialization of unlabelled COMMON, do not allocate unlabelled COMMON via the .COMM pseudo-op. FORTRAN 5 treats unlabelled COMMON storage like labelled COMMON and gives it the block name ".BLAN". You can access unlabelled COMMON by declaring an external reference (.EXTN pseudo-op) for ".BLAN".

The FORTRAN 5 compiler generates STATIC data and data-initialized storage as if it were a named COMMON block. The name of this Static data block is formed from the first seven characters of the routine name with a "." appended. Thus, the system would define the static storage for a routine called "SUB1" as ".ENT SUB1.".

RT.ERR

Use RT.ERR to invoke the error reporter to report an AOS error. AOS errors have mnemonics that begin with the letters ER. Errors reported by RT.ERR cause the system to terminate the program.

The two forms of calls to RT.ERR are

Call	Action
RT.ERR	Invoke the FORTRAN 5 error reporter. Code passed in AC0. All errors are fatal.
RT.ERR code	Loads error code code into AC0, then invokes the FORTRAN 5 error reporter. All errors are fatal.

The first form, without the argument, generates a single word of code that the program can skip. The second form may not be skipped.

F5.ERR

Invoke F5.ERR to report FORTRAN 5 errors. FORTRAN 5 errors have mnemonics that begin with F? or F. . Table 5-2 describes the information passed in AC0.

Table 5-1. AC0 Format For F5.ERR

Bit Field	Value	Meaning
0-1	0	You don't care about the fatality of error.
&	1	Generate a transparent error.
&	2	Generate a recoverable error.
&	3	Generate a program fatal error.
2-15	Code	FORTRAN 5 error to be reported.

As with RPT.ERR, assume that the runtime environment preserves none of the ACs if F5.ERR returns to your program. See Chapter 2 for a description of error classes (transparent, recoverable, and fatal).

To access FORTRAN 5 error codes, you must declare them with the assembler's .EXTN pseudo-op because they are defined as external symbols. The mnemonics beginning with F? include both the error code in bits 2-15 and a default value in the fatality field. The mnemonics beginning with F. include the error code only.

You can find the FORTRAN 5 error code mnemonics in Appendix A. Each of the FORTRAN 5 error codes listed in F5ERR.FR begin with the letters "FE". By appending the remaining three letters of the name to either F? or F. , you get the names of the appropriate error symbols.

For example, the FORTRAN 5 error, "Illegal Input Number" is given the name "FEINM" in F5ERR.FR. The symbol for "Illegal Input Number" that contains both the code and a default fatality field is F?INM. The code alone, without the fatality field, is represented by the symbol F.INM.

If you invoke the F5.ERR with an F. symbol instead of an F? symbol, you indicate that you don't care about the severity of the generated error because fatality field of an F. symbol is zero.

Do not try to report any errors except FORTRAN 5 errors using F5.ERR. If you do, the results are unpredictable; the error reported tries to interpret the 16-bit code as a 2-bit fatality field and a 14-bit FORTRAN 5 error code.

The two types of F5.ERR invocations are

Call	**Meaning**
F5.ERR	Invoke the FORTRAN 5 error reporter. AC0 has previously been loaded with a FORTRAN 5 F? error code.
F5.ERR code	Load the FORTRAN 5 error code code into AC0 and invoke the FORTRAN 5 error reporter. The F5.ERR macro declares code as external.

The first form of the call, without arguments, is guaranteed to generate a single word that the program can skip over.

Calling FORTRAN 5 Built-in and Math Routines

You can call the routine entry points for the FORTRAN 5 built-in routines in Appendix F through the use of the BCALL macro. The calling sequence for these routines is also described in Appendix F.

The naming conventions and calling sequences for mathematic built-in functions is described in Appendix F.

.FIOPREP and .IUNIT

With the routines .FIOPREP and .IUNIT , you can perform I/O using FORTRAN unit numbers. Both routines convert a FORTRAN unit number into an AOS channel number. .IUNIT also attempts to open the unit if a preconnection for that unit exists. You must declare .FIOPREP and .IUNIT external with the assembler's .EXTD pseudo-op.

.IUNIT (COMMON)

Converts a unit number to an operating system channel number, and performs a pre-connected open if necessary.

Input: AC0 = FORTRAN 5 unit number

Called: JSR @.IUNIT

Output: AC2 = operating system channel number
AC3 = frame pointer
AC0, AC1 unchanged
Carry may be destroyed.

Any errors detected are fatal.

.FIOPREP

Obtains an AOS channel number corresponding to a FORTRAN 5 unit number.

Input: AC1 = FORTRAN 5 unit number

Called: JSR @.FIOPREP
error return
good return

Good Return Output: AC1 = FORTRAN 5 unit number
AC2 = AOS channel number
AC3 = frame pointer
AC0 and carry are unchanged.

Error Return Output: AC0 = error code
Other ACs destroyed

.FIOPREP skips the instruction after the JSR if no error occurs.

End of Chapter

Chapter 6
About the Runtime Routines

A FORTRAN 5 runtime routine is either an assembly language function or subroutine. You call a runtime routine in a source program and it is executed at runtime. Runtime routines have been previously assembled and are combined with the object program when you link your program with the F5LD command. You call a runtime routine the same way you call your own subroutines.

FORTRAN 5 provides several libraries of runtime routines which we detail in the following chapters. You can call these routines or write your own additional runtime routines. For information on how to write your own routines, see Chapter 3, "Runtime Environment Fundamentals," and Chapter 5, "FORTRAN 5 Assembly Language Interface."

You also have direct access to many of the AOS assembly language system calls through a set of runtime library routines called QCALLS. These FORTRAN 5 subroutines permit you to make use of operating system functions previously available only through assembly language. Because the QCALL routines interface directly with AOS, they are very efficient. This means increased execution speed for your programs. Programs which use the QCALLs will run under AOS and AOS/VS. For additional information on the QCALLs, see the *FORTRAN QCALLS Reference Manual* (093-000239).

We group the runtime routines into chapters by the functions they perform. Where appropriate, an introduction precedes the routines' descriptions. We also direct you to supplementary references and appendixes when necessary. Each chapter begins with an alphabetical listing of the calls and ends with a page long example of the routines used in the chapter.

We refer to the format you use to call the routine as the calling sequence. The calling sequence specifies the order in which you must enter the arguments.

We will now explain the format of the runtime routine chapters.

Arguments

An argument is either a source or destination of data that the runtime routine uses. The value of an argument can mean several things to the routine:

- It can indicate what actions the routine should perform.
- It can provide the data with which to perform these actions.
- It can tell the routine to return information about other arguments in the same call.
- It can also provide data and tell the routine to return information about other arguments in the same call.

There are two classes of arguments: typed arguments and non-specific aggregates. You must pass to the subroutine the class of argument the subroutine expects.

Typed Arguments

A typed argument is always a variable of a certain FORTRAN data type. The data types you use most frequently with FORTRAN 5 runtime routines are integers and real numbers. We indicate that a routine must have a specific data type when the routine expects one. (The *FORTRAN 5 Reference Manual* describes FORTRAN data types more fully.)

Aggregates

An argument of no specified data type is an aggregate. You use an aggregate in a situation where the runtime routine doesn't care what data type it receives. A runtime routine treats an aggregate merely as a sequence of contiguous words or bytes.

When a runtime routine returns information in an aggregate, you must provide an aggregate large enough to contain all returned data. We indicate the size of the aggregate the routine expects in the routine's description.

If we specify that you must provide a string of ASCII characters in an aggregate, you must ensure that a null (0) byte terminates the string.

If you specify a quoted string as an aggregate argument input value, the FORTRAN 5 compiler ensures the string is terminated by a null. For example, if you call the DFILW runtime routine to delete a file, you pass the name ofthe file to be deleted as either

```
CALL DFILW (file-to-be-deleted,IERRCODE)
```

or

```
CALL DFILW (namearray,IERRCODE)
```

In the first case, the runtime environment deletes the file file-to-be-deleted . The FORTRAN 5 compiler ensures that the string file-to-be-deleted is terminated by a null (0) byte.

In the second case, you have previously placed the ASCII characters of the file name into the array namearray . You must ensure that a null (0) byte terminates the file name in namearray .

IER

The last argument for many of the routines is IER . IER is an integer status variable that receives a numeric status code. The ISA (Instrument Society of America) defines these codes as follows:

Code	Value
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 the file, F5ERR.FR, into your program with the INCLUDE statement.

Do not omit IER from an argument list, if it is specified. If you do, the results will be unpredictable. You need not use the status variable IER . You can use any integer variable name to receive the error code.

Error Conditions

We list possible error conditions for each runtime routine. We also refer you to an appendix of error conditions when this is appropriate.

The routines can receive exceptional condition codes from some system and task calls. We list these codes by category under the error conditions section, when applicable. For example, when you create a directory with CDIR , the error codes that may occur are File System codes. Appendix B describes each of these codes in detail.

For details on FORTRAN 5's error handling, see Chapter 2, "Error Handling."

Examples

We use each routine and its arguments in a FORTRAN 5 situation. If a routine has no arguments, we may omit the example.

Notes and Rules

"Notes" contains information about a routine's purpose, and any aliases for the routine's name.

"Rules" is a section reserved for items essential to calling the routine.

References

In the "Reference" section, we may name one or more system calls. The AOS system calls have names beginning with a question mark; for example ?READ . The description of these calls in the *AOS Programmer's Manual* (093-000120) help describe the specifics of the routine's functionality.

Coding Example

At the end of each runtime routine chapter is a sample program that uses one or more of the runtime routines in that chapter. Because this example is a complete program, you may find it more comprehensive thn the example given with the individual runtime routine descriptions.

Intrinsic Functions

The FORTRAN 5 mathematical functions are described in the *FORTRAN 5 Reference Manual* .

End of Chapter

Chapter 7
Checking for Arithmetic Errors

You can substitute the routines in this chapter for the floating point trap mechanism when you want an explicit check for floating point errors. The floating point trap mechanism provides only a passive check. See Chapter 2, "Error Handling," for an explanation of the floating point trap mechanism and the types of errors the ECLIPSE floating point unit generates.

The Routines In This Chapter

DVDCHK	Checks for a prior floating point division-by-zero
OVERFL	Checks for a prior floating point underflow or overflow.

DVDCHK
Checks for a prior floating point division-by-zero.

Format
CALL DVDCHK (code)

Argument

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

- 1 if division-by-zero occurred.
- 2 if division-by-zero did not occur.

Error Conditions
No error conditions are currently defined.

Notes
Each call to DVDCHK resets the division-by-zero status bit in the floating point status register. Thus, every call to DVDCHK reports on division-by-zero occurrences since the previous call to DVDCHK , or since the start of the program for the first call.

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

OVERFL

Checks for a prior floating point underflow or overflow.

Format

CALL OVERFL (code)

Argument

code — 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 did not.

If both overflow and underflow occurred, the routine signals overflow (code 1).

Error Conditions

No error conditions are currently defined.

Notes

Each call to OVERFL resets the overflow and underflow status bit in the floating point status register. Thus, every call to OVERFL reports on overflow-underflow occurrences since the previous call to OVERFL , or since the start of the program for the first call.

Example

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

Coding Example

```
C       This program demonstrates the use of OVERFL and DVDCHK
C       to make explicit checks for floating point errors.  You
C       can perform either a multiply or a divide, and
C       can specify the two operands.  After the operation is
C       performed, the codes returned by OVERFL and DVDCHK are
C       used to report any floating point errors that occured.

        REAL VAR1, VAR2, RES     ;2 operands and a result
        INTEGER IOVCODE, IDVCODE, IOPER

1       ACCEPT "Enter 1 to multiply, 2 to divide or 3 to STOP: ", IOPER

        IF ((IOPER.GE.1).AND.(IOPER.LE.3)) GO TO 2
        TYPE "Number must be 1, 2, or 3"
        GO TO 1

2       IF (IOPER.EQ.3) STOP "You Stopped Me"

        ACCEPT "Enter first operand: ",VAR1,"Enter second operand: ",
     +      VAR2

        IF (IOPER.EQ.2) GO TO 20

C       Multiplication

10      RES = VAR1 * VAR2

        GO TO 30

C       Division

20      RES = VAR1 / VAR2

C       Call routines DVDCHK and OVERFL  and use the returned
C       information to report on errors that occured.

30      CALL DVDCHK (IDVCODE)    ; Returns floating point error code

        CALL OVERFL (IOVCODE)    ; Return floating point error code

        IF ((IOVCODE.EQ.2).AND.(IDVCODE.EQ.2))
     +         TYPE "No floating point error occurred"

        IF (IOVCODE.EQ.1) TYPE "Floating point overflow occurred"

        IF (IOVCODE.EQ.3) TYPE "Floating point underflow occurred"

        IF (IDVCODE.EQ.1) TYPE "Division by zero occurred"

        GO TO 1

        END
```

End of Chapter

Chapter 8
Performing Logical Operations with Integers and Words

Each runtime routine in this chapter permits access to and manipulation of the bits of integer variables. The routines treat integers as unsigned 16-bit aggregates. With the exception of ISET and ICLR , each routine is a function that returns an integer value to the routine that invoked it.

The functions IAND , IOR , IXOR and NOT perform the logical operations AND, inclusive OR, exclusive OR, and complement, respectively. They perform these functions on a bit-by-bit basis between two integers. The function NOT performs the logical complement on each bit of an integer.

Table 8-1 summarizes the values returned in a given bit of the function result for each pair of corresponding bits in the two arguments.

Table 8-1. Values Returned for Argument Bits

Function	Arg. 1 Bit	Arg. 2 Bit	Result
IAND	0	1	0
	1	1	1
	0	0	0
	1	0	0
IOR	0	1	1
	1	1	1
	0	0	0
	1	0	1
IXOR	0	1	1
	1	1	0
	0	0	0
	1	0	1
NOT	0	-	1
	1	-	0

For more information on logical operations see the *FORTRAN 5 Reference Manual* .

The bit numbering scheme in the runtime routines ICLR, ISET, and ITEST is as follows:

0 for the least significant bit (rightmost)

15 for the most significant bit (leftmost)

The Instrument Society of America (ISA) mandates this scheme.

The Routines in This Chapter

IAND	Produces the bit-by-bit logical AND of two integers.
ICLR	Sets a bit in a word to 0.
IOR	Produces the bit-by-bit logical inclusive OR of two integers.
ISET	Sets a bit in a word to 1.
ISHIFT	Shifts the bits in an integer.
ITEST	Tests a bit in a word for 1 or 0.
IXOR	Produces the bit-by-bit logical exclusive OR of two integers.
NOT	Produces the bit-by-bit logical complement of an integer.

IAND

Produces the bit-by-bit logical AND of two integers.

Format

IAND (int1,int2)

Arguments

int1	the first integer operand.
int2	the second integer operand.

Error Conditions

No error conditions are currently defined.

Examples

Example 1.

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

Example 2.

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

ICLR

Sets a bit in a word to 0.

Format

CALL ICLR (word,bit)

Arguments

word	a one-word aggregate that contains the bit you want to clear (set to 0).
bit	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.

Notes

If you issue a call to ICLR with a bit that is outside the legal range, then the routine does not perform the operation. In this case, you do not receive an error status code or error message.

Example

```
C       SET BIT 3 OF K TO ZERO
        CALL ICLR(K,3)
```

IOR

Produces the bit-by-bit logical inclusive OR of two integers.

Format

IOR (int1,int2)

Arguments

int1	the first integer operand.
int2	the second integer operand.

Error Conditions

No error conditions are currently defined.

Examples

Example 1.

```
C       ASSIGNS TO K THE BIT-WISE LOGICAL
C       OR OF J AND IDEF
        K = IOR(J,IDEF)
```

Example 2.

```
C       COMPARES THE LOGICAL OR OF M
C       AND J TO ZERO
        IF (IOR(M,J) .NE.1) GO TO 100
```

ISET
Sets a bit in a word to 1.

Format

CALL ISET (word,bit)

Arguments

word	a one-word aggregate that contains the word whose bit you want to set.
bit	an integer that specifies the position of the bit you want set to one; bits are numbered from 0, the rightmost, to 15, the leftmost.

Error Conditions

No error conditions are currently defined.

Notes

If you issue a call to ISET with a bit that is outside the legal range, then the operation is not performed. You do not receive an error message.

Example

```
CALL ISET(1,3)
```

ISHIFT
Shifts the bits in an integer.

Format

ISHIFT (integer,count)

Arguments

integer — the integer you want to shift.

count — an integer that specifies the number and the direction of the bits you want to shift.

Error Conditions

No error conditions are currently defined.

Notes

If count is an integer n , the system responds in these ways when n has the following values:

n=0 no shift

n>0 shift left n bits, bringing in zeros from the right (logical shift).

n<0 shift right n bits, bringing in bits from the left that were shifted out of the right (circular shift).

To perform a right logical shift, use (15-n) for n . n can be greater than 15; the effect is the same as mod(n,15) .

Alias is ISHFT

Examples

Example 1.

```
C       ASSIGN TO INEW THE VALUE OF IOLD
C       SHIFTED RIGHT 5 BITS
        INEW = ISHFT(IOLD,-5)
```

Example 2.

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

ITEST

Tests a bit in a word for 1 or 0.

Format

ITEST (word,bit)

Arguments

word	the integer you want to test.
bit	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.

Notes

This function returns a 1 if the specified bit is set, and a zero if the bit is cleared.

If you issue a call to ITEST with a bit that is outside the legal range, you receive the result associated with the rightmost bit (bit 0).

If you declare ITEST as LOGICAL , you receive these results:

.TRUE.	if the bit is set (1)
.FALSE.	if the bit is clear (0)

Examples

Example 1.

```
C       CHECKS THE K'TH BIT OF I FOR 1 OR 0
        J = ITEST(I,K)
```

Example 2.

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

IXOR

Produces the bit-by-bit logical exclusive OR of two integers.

Format

IXOR (int1,int2)

Arguments

int1	the first integer operand.
int2	the second integer operand.

Error Conditions

No error conditions are currently defined.

Notes

Alias is IEOR .

Examples

Example 1.

```
      IOTHREE = IXOR(IONE,ITWO)
```

Example 2.

```
C     CHECK TO SEE IF ALL BITS OF I AND J DIFFER
      IF (IXOR(I,J) .EQ.1) GO TO 200
```

NOT

Produces the bit-by-bit logical complement of an integer.

Format

NOT (integer)

Argument

integer	the integer you want to complement.

Error Conditions

No error conditions are currently defined.

Examples

Example 1.

```
      ICAN = NOT(IMAY)
```

Example 2.

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

Coding Example

```
C       This program demonstates the use of the integer logical
C       operation functions.  You enter a decimal
C       integer, and this program outputs the number in
C       binary, octal and hexadecimal.

        INTEGER IBIN(16)        ;16 binary digits
        INTEGER IOCT(6)         ;6 octal digits
        INTEGER IHEX(4)         ;4 hexadecimal digits

        INTEGER IDIGITS(8)      ;ASCII characters for digits

        DATA IDIGITS/"01","23","45","67","89","AB","CD","EF"/

1       ACCEPT "Enter an optionally-signed decimal integer: ",INUM

C       Expand INUM into 16 binary digits in IBIN.  The ITEST function
C       returns the 1/0 value of each bit in INUM.
C       Simultaneously, the program makes a copy of INUM in ICOPY by
C       using ISET and ICLR.

        DO 10 I = 1,16
           IBITPOS = 16-I                      ;Bit position from 15 to 0
           IBIN(I) = ITEST(INUM,IBITPOS)
           IF (IBIN(I).EQ.0) CALL ICLR(ICOPY,IBITPOS)
           IF (IBIN(I).EQ.1) CALL ISET(ICOPY,IBITPOS)
10      CONTINUE

C       ICOPY should now contain a bit-by-bit copy of INUM.
C       ICOPY is now unpacked into 6 octal digits in IOCT
C       through the use of ISHIFT and IAND.  Digits are
C       extracted from right to left, so the loop goes down
C       instead of up.


        DO 20 I = 6,1,-1
           IOCT(I) = IAND(ICOPY,000007K) ;Mask out rightmost 3 bits
           ICOPY = ISHIFT(ICOPY,-3)      ;Shift ICOPY right 3 bits
20      CONTINUE

C       Since the leftmost octal digit in INUM (and ICOPY) should
C       reflect only the leftmost bit in INUM (since 3 does not
C       go into 16 evenly), we replace IOCT(1) by its rightmost
C       bit only by using the IAND function.

        IOCT(1) = IAND(IOCT(1),000001K) ;Extract rightmost bit

C       The following code mimics the above octal digit extraction
C       for the hexadecimal conversion using the original INUM.

        DO 30 I=4,1,-1          ;4 digits from right to left
           IHEX(I) = IAND(INUM,0000017K) ;Extract rightmost 4 bits
           INUM =ISHIFT(INUM,-4)         ;Shift INUM 4 bits right
30      CONTINUE

C       Now write out each representation

        WRITE (11,1102)
1102    FORMAT("Binary Representation: ",Z)

        DO 40 I=1,16
          WRITE (11,1105) BYTE(IDIGITS,IBIN(I)+1)
```

```
40      CONTINUE

        TYPE

        WRITE (11,1103)
1103    FORMAT("Octal Representation: ",Z)

        DO 50 I=1,6
           WRITE (11,1105) BYTE(IDIGITS,IOCT(I)+1)
50      CONTINUE

        TYPE

        WRITE (11,1104)
1104    FORMAT("Hexadecimal Representation: ",Z)

        DO 60 I=1,4
           WRITE (11,1105) BYTE(IDIGITS,IHEX(I)+1)
60      CONTINUE

1105    FORMAT(R1,Z)

        TYPE
70      ACCEPT "Enter 1 to continue or 0 to STOP: ",ICHOICE
        IF (ICHOICE.EQ.1) GO TO 1
        IF (ICHOICE.EQ.0) STOP
        GO TO 70

        END
```

End of Chapter

Chapter 9
Managing Logical Disks and Directories

For the routines in this chapter, you will need to specify the size of aggregates containing pathnames and logical disk names. We define size parameters in QSYM.FR to assist you in doing this: the symbol QMXL gives the maximum pathname length in bytes.

The Routines In This Chapter

CDIR	Creates a directory.
CPART	Creates a control point directory.
DIR	Changes the working directory.
GDIR	Obtains the current working directory name.
INIT	Initializes a logical disk.
RELEASE	Releases a logical disk.

CDIR

Creates a directory.

Format

CALL CDIR (directory name,IER)

Arguments

directory name	an aggregate that contains the pathname of the directory you want to create.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.

Example

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

Reference

?CREATE (System call)

CPART
Creates a control point directory.

Format

CALL CPART (CPD name,size,IER)

Arguments

CPD name	an aggregate that contains the pathname of the control point directory you want to create.
size	an integer that contains the maximum number of 256-word disk blocks you want to allocate to the control point directory.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER are

File System codes.

Example

```
C      CREATE CPD HAVING A MAXIMUM
C      SIZE OF 80 BLOCKS
       CALL CPART ("NEWCPD",80,IER)
       CALL CHECK (IER)
```

Reference

?CREATE (System call)

DIR
Changes the working directory.

Format

CALL DIR (directory name,IER)

Arguments

directory name	an aggregate that contains the name of the new working directory.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.

Example

```
C       MAKE USMAN THE WORKING DIRECTORY
        CALL DIR ("USMAN",IER)
        CALL CHECK (IER)
```

Reference

?DIR (System call)

GDIR
Obtains the working directory name.

Format

CALL GDIR (directory name,IER)

Arguments

directory name	an aggregate that receives the working directory name.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in IER are

File System codes.

Example

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

Reference

?GNAME (System call)

INIT

Initializes a logical disk.

Format

CALL INIT (LD name,IER)

Arguments

LD name	an aggregate that contains the name of a logical disk device.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.
Initialization and Release codes.

Notes

An initialized logical disk remains initialized until you release it by calling RELEASE .

Example

```
CALL INIT ("2DPF2 ",IER)
CALL CHECK (IER)
```

Reference

?INIT (System Call)

RELEASE

Releases a logical disk.

Format

CALL RELEASE (LD name,IER)

Arguments

LD name	an aggregate that contains the name of a logical disk.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return IER are

File System codes.
Initialization and Release codes.

Notes

Alias is RLSE .

Example

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

Reference

?RELEASE (System call)

Coding Example

```
C       This program demonstrates the use of the directory
C       management routines.
C

        INCLUDE "QSYM.FR"         ;Include AOS constants

C       NAME1, NAME2, and NAME3 will be used as buffers for pathnames

        INTEGER NAME1(QMXPL), NAME2(QMXPL), NAME3(2)

        DATA NAME3/"DO","C<0>"/ ;Filename "DOC" with null terminator

        CALL GDIR(NAME1, IER)     ;Find out which directory we are in
        CALL CHECK(IER)

        WRITE (11,1101) NAME1(1)          ;Write out the name
1101    FORMAT("The current directory is:  ",S80)

        TYPE "Enter the Name of a directory to be created here"
        READ (10,1001) NAME2(1)           ;Get a directory name
1001    FORMAT(S80)

        CALL CDIR(NAME2, IER)             ;Create the new directory
        CALL CHECK(IER)

C       Call DIR to enter the new directory

        CALL DIR(NAME2, IER)              ;Change directories
        CALL CHECK(IER)

C       Call CPART to create a control point directory of 20000 blocks.
C       The name of the CPD will be "BACKUP.CPD"

        CALL CPART("BACKUP.CPD", IER)     ;Create the CPD
        CALL CHECK(IER)

C       Return to the original directory
        CALL DIR(NAME1, IER)              ;Back up 2 levels
        CALL CHECK(IER)

C       Create a subdirectory called "DOC" using data initialized name
        CALL CDIR(NAME3, IER)             ;Create "DOC"
        CALL CHECK(IER)

        END
```

End of Chapter

Chapter 10
Maintaining Files

For the routines in this chapter, you must specify the size of pathnames. QSYM.FY defines size parameters to assist you in doing this: the symbol QMXPL gives the maximum pathname length in bytes.

The Routines In This Chapter

CFILW	Creates a disk file.
CHSTS	Obtains the current directory status for an opened unit.
DFILW	Deletes an unopened disk file.
FDELETE	Deletes an unopened disk file.
FRENAME	Renames an unopened disk file.
LINK	Creates a link entry in the current directory.
RENAME	Renames an unopened disk file.
UNLINK	Deletes a link entry.

CFILW

Creates a disk file.

Format

CALL CFILW (pathname,file type, *[size,]* IER)

Arguments

pathname	an aggregate that contains the pathname of the disk file you want to create.
file type	an integer with either of these two values which indicate the following: 2 noncontiguous file 3 contiguous file
size	an integer that specifies the number of 256-word disk blocks you want to allocate. This argument is meaningful only for contiguous files; it is ignored for noncontiguous files.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.

Notes

Alias is CFIL .

Example

```
CALL CFILW ("FILNM.DC",2,IER)
CALL CHECK (IER)
```

Reference

?CREATE (System call)

CHSTS
Obtains the status for an opened unit.

Format
CALL CHSTS (unit number,status,IER)

Arguments
unit number	an integer that specifies the FORTRAN 5 unit number.
status	an aggregate that receives QSLTH words of channel status information.
IER	an integer variable that receives the routine's completion status code.

Rules
You must have read access to the parent directory of the file when you call CHSTS.

Error Conditions
The error codes that may return in IER are

File System codes.

Notes
F5SYM.FR contains the current value of QSLTH.

For a description of the returned information, see the description of the ?FSTAT call in the *AOS Programmer's Reference Manual*.

Example
```
INTEGER ISTAT(22)
.
.
CALL CHSTS (3,ISTAT,IER)
CALL CHECK (IER)
```

Reference
?FSTAT (System call)

DFILW

Deletes an unopened disk file.

Format

CALL DFILW (pathname,IER)

Arguments

pathname — an aggregate that contains the pathname of the file you want to delete.

IER — an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER are

File System codes.

Notes

Use DFILW rather than FDELETE if you do not want an error condition to cause program termination.

Alias is DFIL .

Example

```
CALL DFILW ("TEST.PR",IER)
```

Reference

?DELETE (System call)

FDELETE

Deletes an unopened disk file.

Format

CALL FDELETE (pathname)

Argument

pathname   an aggregate that contains the pathname.

Error Conditions

The error conditions that may result are

File System codes.

Notes

The error conditions that may result from using FDELETE cause program termination. Use DFILW rather than FDELETE if you don't want program termination.

Example

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

Reference

?DELETE (System call)

FRENAME
Renames a file.

Format

CALL FRENAME (old pathname,new filename)

Arguments

old pathname	an aggregate that contains the current pathname to the disk file you want to rename.
new filename	an aggregate that contains the new filename of the disk file.

Rules

new filename must be a simple filename, not a pathname.

Error Conditions

The error conditions that may result are

File System codes.

Notes

The error conditions that may result from using FRENAME cause program termination. Use RENAME if you don't want an error to terminate the program.

Example

```
CALL FRENAME (":UDD:DOC:DC","CHAP4.DC")
```

LINK

Creates a link entry.

Format

CALL LINK (pathname1,pathname2,IER)

Arguments

pathname1	an aggregate that contains the pathname of the link entry you want to create.
pathname2	an aggregate that contains the pathname of the file onto which you want to link.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.

Notes

If the last filename in pathname2 is a link, then the system will not resolve the link.

Example

```
CALL LINK ("PROG1.PR","PROG2.PR",IER)
CALL CHECK (IER)
```

Reference

?CREATE (System Call)

RENAME
Renames a file.

Format

CALL RENAME (old pathname,new filename,IER)

Arguments

old pathname	an aggregate that contains the new pathname of the disk file.
new filename	an aggregate that contains the new filename of the disk file.
IER	an integer variable that receives the routine's completion status code.

Rules

new filename must be a simple filename, not a pathname.

Error Conditions

The error codes that may return in IER are

File System codes.

Example

```
CALL RENAME (":UDD:DOC:DC","CHAP4.DC",IER)
CALL CHECK (IER)
```

Reference

?RENAME (System call)

UNLINK

Deletes a link entry.

Format

CALL UNLINK (pathname,IER)

Arguments

pathname — an aggregate that contains the pathname of the link entry you want to delete.

IER — an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

ERDID — Attempt to delete a directory containing entries of one or more inferior directories.

ERDIU — Attempt to delete the working directory.

ERIFT — Attempt to delete a permanent file.

File System codes

Example

```
CALL UNLINK ("PROG1.PR",IER)
CALL CHECK (IER)
```

Reference

?DELETE (System call)

Coding Example

```
C       This program demonstrates the file maintainence routines.
C       Files are deleted, created, renamed, and linked together.
C

        INCLUDE "QSYM.FR"          ;Include the definitions of AOS
                                   ;constants

C       NAME1 will be used as a buffer for file names.

        INTEGER NAME1(QMXPL)

C       ISTAT will be used as a buffer for information returned
C       by CHSTS

        INTEGER ISTAT(QSLTH)

C       The following code creates a file whose name is supplied
C       by the user, and creates 2 links "FILE1" AND "FILE2" to
C       the created file.

        TYPE "Enter the pathname of a directory to be created:"
        READ (11,1101) NAME1(1)
1101    FORMAT(S257)

        CALL DFILW(NAME1, IER)          ;Attempt to delete file
                                        ;before creation. Errors
                                        ;are ignored
        CALL DFILW("FILE1", IER)
        CALL DFILW("FILE2", IER)

        CALL CFILW(NAME1, 2, IER)       ;Create non-contiguous file.
        CALL CHECK(IER)                 ;Any error will be fatal.

C       Create the 2 links

        CALL LINK("FILE1", NAME1, IER)  ;Link FILE1 to the new file
        CALL CHECK(IER)
        CALL LINK("FILE2", NAME1, IER)  ;Link FILE2 to the new file
        CALL CHECK(IER)

C       Rename "SOURCEFILE.FR" to "SOURCEFILE.BU"

        CALL RENAME("SOURCEFILE.FR", "SOURCEFILE.BU", IER)
        ; Any errors are ignored

C
C       This part of the program reports the size in bytes of the
C       file "MASTER.FR" via the CHSTS call.

        OPEN 2, "MASTER.FR"
        CALL CHSTS(2, ISTAT, IER)       ;Status if file in array ISTAT
        CALL CHECK(IER)

C       See the description of the ?FSTAT AOS system call in the
C       AOS Programmer's Manual for a description of the layout of
C       information returned by CHSTS

C       Use the information in word offset QSSTS (Defined in QSYM.FR)
C       to report on whether or not MASTER.FR has the permanence
C       attribute set or not.

        IF ( ITEST(ISTAT(QSSTS), QFPRM) .EQ. 0) GO TO 30
             TYPE "MASTER.FR is a Permanent File"
        GO TO 40
30           TYPE "MASTER.FR is not a Permanent File"
40      STOP

        END
```

End of Chapter

Chapter 11
File Input/Output

Several options are available to open, close, read or write files.

Opening Files

You must associate a FORTRAN 5 unit number with a file by opening it in order to read or write to it. You can open a file by calling one of the following runtime routines: **OPEN**, **FOPEN**, or **APPEND** . The **OPEN** statement in FORTRAN 5 also opens a file. (See Appendix E, "FORTRAN 5 Language Statements".)

Files opened with these routines do not respond to ANSI carriage control characters in the first character of output lines. To use ANSI carriage control, you must either use the **OPEN** statement with ATT="P" or the preconnected opening. (See the *FORTRAN 5 Reference Manual* and Chapter 2 of this manual for more information on the preconnected opening.)

Closing Files

You disassociate a FORTRAN 5 unit number from a file by closing the file. Close a file by calling the runtime routines **CLOSE** or **FCLOSE** . The **RESET** routine closes all open files. You can also use the **CLOSE** statement in FORTRAN 5 to close the file. (See Appendix D, "FORTRAN 5 Language Statements".)

Reading and Writing Files

After you open a file, you can read from and write to it. There are four 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. See Table 11-1.

Table 11-1. Read and Write Modes

Mode	Data type	Runtime Routines
Line Mode	An ASCII character string terminated by either a carriage return, form feed, or null character.	RDLIN WRLIN
Sequential Mode	Unedited. Data is transmitted exactly as read or written from a file or a device.	REDSEQ WRSEQ
Record Mode	Fixed length records within a disk file that are accessed randomly by record number.	READRW WRITRW
Direct Block Mode	Data exists in 256-word blocks in a file. Only entire blocks of disk space can be read or written.	RDBLK WRBLK

For more information on I/O in AOS, refer to the *AOS Programmer's Manual* (093-000120).

The Routines In This Chapter

APPEND	Opens a file for appended output.
BACKSPACE	Backspaces a file to the previous logical record.
CHRST	Restores the position of a file saved by CHSAV.
CHSAV	Saves the program's current position within a file.
CLOSE	Closes a file.
FCLOSE	Closes a file.
FOPEN	Opens a file.
FSEEK	Positions a file to a given logical record.
OPEN	Opens a file.
RDBLK	Reads a series of 256-word blocks from a file.
RDLIN	Reads a line from a file.
RDSEQ	Reads a series of bytes from a file.
READRW	Reads a series of logical records from a file.
RESET	Closes all open files.
REWIND	Positions a file at its beginning.
WRBLK	Writes a series of 256-word blocks to a file.
WRITRW	Writes a series of logical records to a file.
WRLIN	Writes a line to a file.
WRSEQ	Writes a series of bytes to a file.

APPEND

Opens a file for appended output.

Format

CALL APPEND (unit number,pathname,mode, *[record size,]* IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you open.
pathname	an aggregate that contains the name of the file.
mode	is unconditionally ignored. It is included to make the argument list identical to that of the OPEN routine.
record size	an integer that specifies the size of a record in bytes.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.
System Call Codes.
Channel-Related codes.

Notes

APPEND positions you at the end of a file. Any WRITING occurs after the existing data in the file.

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

- APPEND allows you to open and extend a file that already exists.
- The system ignores the mode argument in the call to APPEND because you open the file specifically for appending.

Example

```
CALL APPEND(13,"LOGFL",IDUMMY,IER)
CALL CHECK(IER)
```

Reference

?OPEN (System call)

BACKSPACE

Backspaces a file to the previous record.

Format

CALL BACKSPACE (unit number)

Arguments

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

Rules

You must have specified a record length for a file to backspace it.

Error Conditions

The error conditions that may result are

File System codes.

Notes

Alias is FBCKSP .

Example

```
CALL BACKSPACE(3)
```

Reference

?SPOS (System call)

CHRST

Restores the position of a file saved by CHSAV.

Format

CALL CHRST (unit number,position)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number.
position	a 2-word aggregate that contains the 32-bit position of the file when saved by a call to CHSAV .

Error Conditions

The error conditions that may result are

File System codes.

Notes

You can use CHRST to set a file position to a user specified 32-bit file position; you need not call CHSAV before calling CHRST .

Example

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

Reference

?SPOS (System call)

CHSAV

Saves the program's current position within a file.

Format

CALL CHSAV (unit number,position)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number.
position	a 2-word aggregate that receives the current position of the specific file.

Error Conditions

The error conditions that may result are

File System codes.

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	an integer that specifies the FORTRAN 5 unit number of the file you want closed.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Notes

You can prepare this file for printing (ATT = "P") when you open it with the FORTRAN 5 OPEN statement. If you do, FORTRAN 5 appends a NEW LINE and a carriage return to the last line of the file before closing it.

Example

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

Reference

?CLOSE (System call)

FCLOSE
Closes a file.

Format

CALL FCLOSE (unit number)

Argument

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

Error Conditions

The error conditions that may result are

File System codes.
System Call codes.
Channel-Related codes.

Notes

You can prepare this file for printing (ATT="P") when you open it with the FORTRAN 5 OPEN statement. If you do, FORTRAN 5 appends a NEW LINE and a carriage return to the last line of the file before closing it.

Call CLOSE if you don't want a fatal error to terminate program execution.

Example

```
CALL FCLOSE (2)
```

Reference

?CLOSE (System call)

FOPEN
Opens a file.

Format

CALL FOPEN (unit number,pathname)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want opened.
pathname	an aggregate that contains the name of the file.

Error Conditions

The error conditions that may result are

File System codes.
System Call codes.
Channel-Related codes.

Notes

If the named file doesn't exist, then the system creates it.

Example

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

Reference

?OPEN (System call)

FSEEK

Positions a file to a given logical record.

Format

CALL FSEEK (unit number,record number)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want positioned.
record number	an integer that specifies the FORTRAN 5 unit number where you want the system to position you.

Rules

You must specify a record length when you open the file if you use FSEEK for that file.

Error Conditions

The error conditions that may result are

File System codes.
System Call codes.
Channel-Related codes.

Notes

Record numbering begins with record 1. You must open the file before you can position it with FSEEK .

Example

```
C       POSITIONS UNIT 6 BEFORE
C       RECORD # IREC
        CALL FSEEK (6,IREC)
```

OPEN

Opens a file.

Format

CALL OPEN (unit number,pathname,mode, *[record size,]* IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want to open.
pathname	an aggregate that contains the name of the file.
mode	an integer with one of the following values: 0 append 1 read only 2 shared read and write access 3 exclusive read and write access 4 exclusive read and write access
record size	an integer that specifies the size of a record in bytes
IER	an integer variable that receives the routine's completion status code

Error Conditions

The error codes that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Notes

If you OPEN a file in exclusive mode, no other user can access that file.

If the file does not exist, the system will not create it. However, the system will return the error code ERFDE (file name does not exist). You can then call CFIL to create the file, or use the FORTRAN 5 OPEN statement without an ERR= option to create the file.

Examples

Example 1.

```
C       OPEN FOR READING ONLY
        CALL OPEN (6,"MYFILE",1,IER)
        CALL CHECK(IER)
```

Example 2.

```
C       OPEN FOR APPENDING WITH
C       20-BYTE RECORDS
        CALL OPEN (13,"OUTPUT",0,20,IER)
        CALL CHECK(IER)
```

Reference

?OPEN (System call)

RDBLK

Reads a series of 256-word blocks from a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want to read.
starting block	an integer that specifies the block at which to start reading (the first block is numbered 0).
data array	an aggregate that receives the data read.
count	an integer that specifies the total number of 256-word blocks you want to read.
returned count	an integer variable that receives the total number of blocks successfully read only when an end-of-file condition is set. Otherwise this argument is not set.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Example

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

Reference

?READ (System call)

RDLIN
Reads a line from a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number from which you want to read data.
data array	an aggregate that receives the line read.
returned count	an integer variable that receives the number of bytes read (including the terminator).
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Notes

You should generally make data array 136 bytes long to accommodate the longest possible line unless you specify a different record length when you open the file.

RDLIN reads characters up to a data sensitive delimiter. These delimiters are NEW LINE, Form Feed, Carriage Return, and Null.

Using RDLIN to read from a terminal causes typed characters to echo on your terminal. RDSEQ does not echo typed characters.

Example

```
      INTEGER ARAYD (69)
      .
      .
      CALL RDLIN (5,ARAYD,ICNT,IER)
C     SIGNAL ANY ERROR
      CALL CHECK (IER)
C     BRANCH IF TERMINATOR IS THE ONLY
C     CHARACTER
      IF (ICNT .EQ.1) GO TO 15
```

Reference

?READ (System call)

RDSEQ

Reads a sequence of bytes from a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number from which you want to read data.
data array	an aggregate that receives the data read.
count	an integer that specifies the number of bytes you want to read.
returned count	an integer variable that receives the partial read count in bytes only when the system encounters an end-of-file condition. Otherwise this argument is not set.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Notes

RDSEQ does not echo characters read from terminals.

Example

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

Reference

?READ (System call)

READRW

Reads a series of logical records from a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number from which you want to read data.
starting record	an integer that specifies the number of the first record you want to read (the first record of a FORTRAN 5 file is numbered 1).
data array	an aggregate that receives the records that AOS reads.
count	an integer that specifies the number of records you want to read.
returned count	an integer variable that receives the number of bytes in a record that are read only when the system encounters an end-of-file condition (otherwise this argument is not set).
IER	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 when you opened it.

Error Conditions

The error codes that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Notes

Aliases are READR and RDRW .

Example

```
        INTEGER PDQARY (400)
        .
        .
C       FOUR-WORD RECORDS
        OPEN 14,"INPUTDATA",LEN=8
        .
        .
C       READ RECORDS 100 TO 104
        CALL READRW (14,100,PDQARY,5,ICNT,IER)
        CALL CHECK (IER)
```

Reference

?READ (System call)

RESET

Closes all open files.

Format

CALL RESET

Arguments

None

Error Conditions

The error conditions that may result are

File System codes.
System Call codes.
Channel-Related codes.

Notes

When you call this routine in a multitask environment, invoke the SINGLETASK routine first. This disables task rescheduling to insure that no I/O is in progress during the call. Afterwards, reinstate the multitask environment with a call to MULTITASK .

Reference

?CLOSE (System call)

REWIND
Positions a file at its beginning.

Format
CALL REWIND (unit number)

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

Error Conditions
The error conditions which may result are

File System codes.

Notes
Alias is FRWND .

Example
```
CALL REWIND(INPCH)
```

Reference
?SPOS (System call)

WRBLK

Writes a series of 256-word blocks to a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want to write.
starting block	an integer that specifies the first block you want to write (the first block is numbered 0).
data array	an aggregate that contains the data you want to write.
count	an integer that specifies the number of blocks you want to write.
returned count	an integer variable that receives the number of blocks successfully written when the system exhausts space in the file's directory.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Example

```
      INTEGER AREA9 (1024)
      .
      .
      OPEN 9,"FILE9"
      .
      .
C     WRITE BLOCKS #4 THRU #6 FROM
C     AREA9 ONTO UNIT9
      CALL WRBLK (9,4,AREA9,3,IER)
      CALL CHECK (IER)
```

Reference

?WRITE (System call)

WRITRW

Writes a series of logical records to a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number to which you want to write data.
starting record	an integer that specifies the number of the first record to which you want to write.
data array	an aggregate that contains the records you want to write.
count	an integer that specifies the number of records you want to write.
IER	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

The error codes that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Notes

Alliases are WRTRW and WRITR.

Example

```
C       8-BYTE BUFFER
        COMPLEX MESAREA
        .
        .
C       FOUR-BYTE RECORDS
        CALL OPEN (7,"OUTDATA",3,4,IER)
        .
        .
C       WRITE RECORDS 14 AND 15
        CALL WRITRW (7,14,MESAREA,2,IER)
        CALL CHECK (IER)
```

Reference

?WRITE (System call)

WRLIN
Writes a line to a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number to which you want to write data.
data array	an aggregate that contains the line you want to write.
returned record	an integer variable that receives the total number of bytes the call wrote if an error occurs.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System Codes.
System Call codes.
Channel-Related codes.

Notes

This routine transfers characters up to and including a data sensitive delimiter. These delimiters are form feed, NEW LINE, carriage return, and null.

Example

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

Reference

?WRITE (System call)

WRSEQ

Writes a sequence of bytes to a file.

Format

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

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number to which you want to write data.
data array	an aggregate that contains the data you want to write.
count	an integer that specifies the number of bytes you want to write.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

File System codes.
System Call codes.
Channel-Related codes.

Notes

The system writes exactly the number of bytes you specified in count to unit number.

Example

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

Reference

?WRITE (System call)

Coding Example

```
C         This program demonstrates the use of runtime routines
C         to perform file input and output.
C         The program opens the file "RAWDATA" for input.  You can then
C         enter a record number on the file to be
C         read or written.
C
C

C         The array IBUFF will contain a line read or written

          INTEGER IBUFF(41)

C         Open @OUTPUT for appending

          CALL APPEND(2, "@OUTPUT", 0, IER)

C         Open @DATA for reading as a record-oriented file of
C         80 byte records

          CALL OPEN(3, "RAWDATA", 2, 80, IER)
          CALL CHECK(IER)

          CALL OPEN(4, "@INPUT", 2, IER)
          CALL CHECK (IER)


1         ACCEPT "Enter a record Number", IREC

C         A Record number less than zero will terminate the program
          IF (IREC.LT.0) GO TO 100

5         WRITE (11,1101)
1101      FORMAT ("Enter R to read or W to write: ",Z)

          READ (10,1001) ICHOICE
1001      FORMAT (A1)

          IF (ICHOICE.EQ."R ") GO TO 10
          IF (ICHOICE.EQ."W ") GO TO 20
          TYPE "Please type R or W"
          GO TO 5

C         User wants to read record IREC
10        CALL READRW (3, IREC, IBUFF, 1, IER)
          IF (IER.EQ.1) GO TO 15

C         An error code was returned.  Report it
          TYPE "ERROR CODE RETURNED:", IER
          GO TO 1

C         Write out IBUFF if no error
15        CALL WRLIN (2, IBUFF, IER)
          CALL CHECK(IER)                  ;Any error is fatal

          GO TO 1                          ;Get another record number
```

```
C       User wants to write to record IREC
C       First, read the line to be written
20      TYPE "Enter the line to be written to the file:"
        CALL RDLIN(4, IBUFF, IER)

        CALL WRITRW(3, IREC, IBUFF, 1, IER)

        GO TO 1          ;Process the next request

C       Close files and stop
100     CALL CLOSE(3, IER)
        CALL CHECK(IER)
        CALL CLOSE(2, IER)
        CALL CHECK(IER)

        STOP "You Stopped Me"
        END
```

End of Chapter

Chapter 12
Console Handling

When AOS starts up your program, it enables console interrupts. You can disable them with with the routine ODIS and enable them with the routine OEBL .

The Routines In This Chapter

GCIN	Obtains the input console name.
GCOUT	Obtains the output console name.
ODIS	Disables console interrupts.
OEBL	Enables console interrupts.

GCIN
Obtains the input console name.

Format

CALL GCIN (pathname,IER)

Arguments

pathname	a 5-word aggregate that receives the input console name (@CONSOLE) .
IER	an integer variable that receives a routine's completion status code.

Error Conditions

No error conditions are currently defined.

Example

```
INTEGER CONM (5)
.
.
CALL GCIN (CONM,IER)
CALL CHECK (IER)
```

GCOUT
Obtains the output file.

Format

CALL GCOUT (pathname,IER)

Arguments

pathname	a 5-word aggregate that receives the output console name (@CONSOLE).
IER	an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Example

```
INTEGER OUTCON(5)
.
.
CALL GCOUT (OUTCON,IER)
CALL CHECK (IER)
```

ODIS
Disables console interrupts.

Format

CALL ODIS (IER)

Arguments

IER	an integer variable that receives the routine's completion code.

Error Conditions

No error conditions are currently defined.

Example

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

Notes

After you call ODIS, you must call OEBL before you can perform a console interrupt.

Reference

?ODIS (System call)

OEBL

Enables console interrupts.

Format

CALL OEBL (IER)

Argument

IER	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)
```

Notes

This call does not supply an interrupt-processing capability.

Reference

?OEBL (System call)

Coding Example

```
C          This program demonstrates a call to GCOUT.
C
           INTEGER ICON(5)          ;Array which recieves the console name

           CALL GCOUT(ICON, IER)    ;Get the console name
           CALL CHECK(IER)


C          Open the console file
           OPEN 1,ICON

           WRITE FREE (1) "I am talking to the Console"

           CLOSE 1

           STOP "End of this"
```

End of Chapter

Chapter 13
Using the System Clock and Calendar

The FORTRAN 5 runtime routines in this chapter obtain the time of day, set the time, obtain the current date, and set the date. You can call one of two routines to perform each of these functions: an ISA formatted routine with an argument that is a three-word integer array, or a routine with three separate integer arguments. For example, to set the time you can 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.

To set the date or time, the caller must be running as the operator process (PID 2) or one of it's brothers.

The Routines 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,IER)

Argument

date — a three-word aggregate which receives the integer values for the month, day, and year in words 1,2, and 3 respectively.

IER — an integer variable that receives a routine's completion status code.

Error Conditions

No error conditions are currently defined.

Notes

The year is returned as a four-digit integer.

Example

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

Reference

?GDAY (System call)

FGDAY
Obtains the current date.

Format

CALL FGDAY (month,day,year)

Arguments

month	an integer variable that receives the current month by number (1 through 12).
day	an integer variable that receives the current day by number (1 through 31).
year	an integer variable that receives the current year since 1900 by number (0 through 199).

Error Conditions

No error conditions are currently defined.

Notes

The year is returned as a two-digit number.

Example

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

Reference

?GDAY (System call)

FGTIME
Obtains the current time.

Format
CALL FGTIME (hour,minute,second)

Arguments
hour — an integer variable that receives the current hour (0 through 23).

minute — an integer variable that receives the current minute (0 through 59).

second — an integer variable that receives the current second (0 through 59).

Error Conditions
No error conditions are currently defined.

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

Reference
?GTOD (System call)

FSDAY

Sets the current date.

Format

CALL FSDAY (month,day,year)

Arguments

month — an integer that specifies the month by number (1 through 12).

day — an integer that specifies the day by number (1 through 31).

year — an integer that specifies the year since 1900 by number (0 through 199).

Rules

Only the operator process or one of its brothers can call FSDAY .

Error Conditions

The error conditions that can result are

ERTIM — Illegal month, day, or year.

ERPRV — Caller not privileged for this action.

Notes

Using FSDAY you only need to specify the year since 1900, whereas with SDATE you must specify all four digits.

Example

```
C       SET THE DATE TO OCTOBER 17, 2001
        CALL FSDAY (10,17,101)
```

Reference

?SDAY (System call)

FSTIME
Sets the time.

Format

CALL FSTIME (hour,minute,second)

Arguments

hour	an integer that specifies the hour (0 through 23).
minute	an integer that specifies the minute (0 through 59).
second	an integer that specifies the second (0 through 59).

Rules

Only the operator process or one of its brothers can call FSTIME.

Error Conditions

ERTIM	Illegal time of day.
ERPRV	Caller not privileged for this action.

Example

```
C       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	a 3-word aggregate that contains new values for the date as month, day, year in words 1,2, and 3 respectively.
IER	an integer variable that receives the routine's completion status code.

Rules

Only the operator process or one of its brothers can call SDATE .

Error Conditions

ERTIM	Illegal day, month, year.
ERPRV	Caller not privileged for this action.

Notes

You specify the year as a 4-digit number.

Example

```
          DIMENSION IDATE(3)
          .
          .
          IDATE(1)=7
          IDATE(2)=4
          IDATE(3)=76
C         SET THE DATE TO
C         JULY 4, 1976
          CALL SDATE (IDATE,IER)
          CALL CHECK (IER)
```

Reference

?SDAY (System call)

STIME

Sets the time.

Format

CALL STIME (time,IER)

Arguments

time — a 3-word aggregate whose elements contain the time based on a 24-hour clock as hours, minutes, and seconds in words 1,2, and 3 respectively.

IER — an integer variable that receives the routine's completion status variable.

Rules

Only the operator process or one of its brothers can call STIME .

Error Conditions

The error conditions that can return in IER are

ERTIM — Illegal time of day.

ERPRV — Caller not privileged for this action.

Example

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

Reference

?STOD (System call)

TIME

Obtains the current time.

Format

CALL TIME (time,IER)

Arguments

time — a 3-word aggregate whose elements receive the current time of day based on a 24-hour clock in hours, minutes, seconds in words 1, 2, and 3, respectively.

IER — an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Example

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

Reference

?GTOD (System call)

Coding Example

```
C       This program prints out the current date and time.
C

        CALL FGDAY (IMONTH, IDAY, IYEAR)
        CALL FGTIME (IHOUR, IMIN, ISEC)

        TYPE "The Date is: ",IMONTH,"/",IDAY,"/",MOD(IYEAR,100)

        TYPE "The Time is: ",IHOUR,":",IMIN,":",ISEC

        END
```

End of Chapter

Chapter 14
Initiating Tasks in a Multitask Environment

When you initiate a task with one of these calls, you can assign it an optional identification number (task ID) and an optional priority.

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

Tasks can have equal priorities. In this case each task receives CPU control in equal amounts, in a round robin fashion.

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

Both FTASK and ITASK include a value for partition specifier in their arguments. A partition specifier tells the system how large a stack to allocate for the task. The parameter values for partition are as follows:

0 or 1	a default size stack (the system divides available memory into partitions numerically equal to the number of tasks requiring default size partitions).
2	the smallest available stack.
3	the largest available stack.
>3	a stack of this exact size (you must have specified a partition of this exact size at LINK time using the methods described in Chapter 4).
100000K	no stack at all (can't be a FORTRAN 5 task).

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.

The Routines In This Chapter

FTASK	Initiates a task.
ITASK	Initiates a task.

FTASK
Initiates a task.

Format

CALL FTASK (subroutine,error return label,priority, *[partition specifier]*)

Arguments

subroutine	the initial subroutine the task will execute (you must declare **subroutine** as external).
error return label	a statement label to which the system transfers control if an error occurs during task initiation.
priority	an integer that specifies the initial priority for the task.
partition specifier	an integer that specifies the stack partition size. for the task. If you omit partition specifier, the system uses a partition specifier value of 0 (the default partition size).

Rules

The subroutine name must appear in a FORTRAN 5 EXTERNAL declaration.

You must specify the error return label with a dollar sign; for example, **$100** .

Error Conditions

The error codes that may result are

Task codes.

Example

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

Reference

?TASK (Task call)

ITASK
Initiates a task.

Format

CALL ITASK (subroutine,task ID,priority, *[partition specifier]* ,IER)

Arguments

subroutine	the initial subroutine the task executes (you must declare **subroutine** as external).
task ID	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	an integer that specifies the initial priority for the task.
partition specifier	an optional integer argument that specifies the stack partition size for the task. If you omit partition specifier, the system uses a partition specifier of 0 (default partition size).
IER	an integer variable that receives the routine's completion status code.

Rules

The subroutine name must appear in a FORTRAN 5 EXTERNAL declaration.

Error Conditions

The error codes that may return in IER are

FEPNA	Requested partition unavailable.
FEPRI	Illegal task priority.
FESTK	Illegal stack size.

Task codes.

Example

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

Reference

?TASK (Task call)

Coding Example

```
C       This program consists of a main program and two subroutines.
C       The main program initiates two tasks, each of which
C       writes a simple message.

        EXTERNAL TSK1, TSK2      ;The names of the tasks *MUST* be
                                 ;declared external when using
                                 ;ITASK and FTASK

        CALL FTASK (TSK1, $100, 2)      ;Initiate TSK1 at priority 2
        CALL WAIT(1,2,IER)              ;Wait 1 second
        CALL CHECK(IER)

        CALL ITASK (TSK2, 0, 2, IER)    ;Initiate TSK2 at priority 2
        CALL CHECK(IER)

        CALL WAIT(3,2,IER)              ;Wait 3 seconds
        CALL CHECK(IER)

        STOP "All Done!"

C       Control will be transferred here by FTASK if an error
C       occurs.

100     STOP "An error occurred during FTASK"

        END

-----------------------------------------------------------------

C       This is the first task
C
        SUBROUTINE TSK1

        TYPE "Task 1 is alive!"
        RETURN

        END
-----------------------------------------------------------------

C       This is the second task
C
        SUBROUTINE TSK2

        TYPE "Task 2 is alive!"
        RETURN

        END
```

End of Chapter

Chapter 15
Changing Task States in a Multitask Environment

You can use the runtime routines in this chapter to suspend, ready and terminate tasks. If you call one of the runtime routines that suspends a task, you can call a corresponding routine to ready the task and cancel the suspension. Corresponding routines are listed in the "Notes" section when appropriate.

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

There are several methods you can use to terminate tasks. For example, by executing a RETURN statement in the task's initial subroutine, the task will terminate itself. Another way in which a task can terminate itself is through the KILL routine. With the routines TIDK, AKILL, and the KILL and DESTROY statements, a task can terminate itself or other tasks.

The Routines In This Chapter

AKILL	Kills all tasks of a given priority.
ARDY	Readies all tasks of a given priority.
ASUSP	Suspends all tasks of a given priority.
KILL	Kills the calling task.
PRI	Changes the priority of the calling task.
SUSP	Suspends the calling task.
TIDK	Kills the task specified by an ID number.
TIDP	Changes the priority of the task specified by an ID number.
TIDR	Readies the task specified by an ID number.
TIDS	Suspends the task specified by an ID number.

AKILL

Kills all tasks of a given priority.

Format

CALL AKILL (priority)

Arguments

priority	an integer that specifies a task priority number.

Error Conditions

The error conditions that may result are

Task codes.

Example

```
C       KILL ALL TASKS HAVING PRIORITY 4
        CALL AKILL(4)
```

Reference

?PRKILL (Task call)

ARDY

Readies all tasks of a given priority.

Format

CALL ARDY (priority)

Arguments

priority	an integer that specifies a task priority number.

Error Conditions

The error conditions that may return are

Task codes.

Notes

ARDY can ready a task suspended by ASUSP .

Example

```
INTEGER PR
PR=1
CALL ARDY (PR)
```

Reference

?PRRDY (Task call)

ASUSP

Suspends all tasks of a given priority.

Format

CALL ASUSP (priority)

Argument

priority	an integer that specifies a task priority.

Error Conditions

The error conditions that may result are

Task codes.

Notes

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

Example

```
CALL ASUSP (2)
```

Reference

?PRSUS (Task call)

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

Error Conditions

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

?SUS (Task call)

TIDK

Kills the task specified by an ID number.

Format

CALL TIDK (task ID,IER)

Arguments

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

IER — an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER are

FETID — You specified an illegal ID to a task.

Task codes.

Notes

Aliases are TIDKILL, ABORT and DESTROY .

Example

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

Reference

?IDKIL (Task call)

TIDP

Changes the priority of a task specified by an ID number.

Format

CALL TIDP (task ID,priority,IER)

Arguments

task ID	an integer that specifies the ID number of the task receiving the new priority.
priority	an integer that specifies the new priority of the task.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

FEPRI	You specified an illegal task priority.
FETID	You designated an illegal ID for a task.

Task codes .

Notes

Aliases are TIDPRI, CHNGE and CHPRI .

Example

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

Reference

?IDPRI (Task call)

TIDR

Readies a task specified by an ID number.

Format

CALL TIDR (task ID,IER)

Arguments

task ID	an integer that specifies the ID number of the task you want to ready.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

FETID	You designated an illegal ID for a task.

Task codes.

Notes

You can use TIDR to ready a task suspended by TIDS .

Aliases are RELSE and TIDRDY .

Example

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

Reference

?IDRDY (Task call)

TIDS

Suspends a task specified by an ID number.

Format

CALL TIDS (task ID, IER)

Arguments

task ID	an integer that specifies the ID number of the task you want to suspend.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

FETID	You designated an illegal ID for a task.

Task codes.

Notes

You can use TIDR to ready a task suspended by TIDS .

Aliases are TIDSUSP and HOLD .

Example

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

Reference

?IDSUS (Task call)

Coding Example

```
C        This program initiates 5 tasks at different priorities,
C        then uses various runtime routines to suspend and kill
C        them.
C

C        Initiate the 5 tasks using FORTRAN 5's TASK statement,
C        and passing the task ID to each task as an argument.

         TASK SUB(1), ID=1, PRI=1
C                Task 2 and Task 3 have priority 2
         TASK SUB(2), ID=2, PRI=2
         TASK SUB(3), ID=3, PRI=2
C                Task 4 and Task 5 have priority 3
         TASK SUB(4), ID=4, PRI=3
         TASK SUB(5), ID=5, PRI=3

         CALL WAIT(1,2,IER)      ;Wait 1 second for everything to
                                 ;start up
         CALL CHECK(IER)

         CALL ASUSP(3)           ;Suspend tasks at priority 3
                                 ;(Task 4 and Task 5)

         TYPE "Tasks 4 and 5 suspended"

         CALL AKILL(2)           ;Terminate tasks at priority 2
                                 ;(Task 2 and Task 3)

         TYPE "Tasks 2 and 3 killed"

         CALL TIDR(4, IER)       ;Ready task 4
         CALL CHECK(IER)

         TYPE "Task 4 readied"

         CALL TIDK(5, IER)       ;Kill task 5
         CALL CHECK(IER)

         TYPE "Task 5 killed"

         CALL AKILL(3)           ;Kill all tasks having Priority 3
         TYPE "Task 4 killed"

         CALL AKILL(1)           ;Kill all tasks having Priority 1
         TYPE "Task 1 killed"

         TYPE "All done"

         STOP
         END

---------------------------------------------------------------

C        This routine is the module used by all five tasks invoked by
C        the main program.
C        This code will continue execution until it's task is killed
C        by the main program.
C

         SUBROUTINE SUB(ID)

1        TYPE "TASK ",ID," IS RUNNING"
         CALL WAIT(500,1,IER)            ;WAIT 500 MILLISECONDS
         CALL CHECK(IER)
         GO TO 1

         END
```

End of Chapter

Chapter 16
Obtaining Task-Related Information in a Multitask Environment

The Routines In This Chapter

GETEV	Obtains a task's event number.
GETPRI	Obtains a task's priority.
MYEV	Obtains the calling task's event number.
MYID	Obtains the calling task's ID number.
MYPRI	Obtains the calling task's priority.

GETEV
Obtains a task's event number.

Format

CALL GETEV (task ID,event,IER)

Arguments

task ID	an integer that specifies the task's ID number.
event	an integer variable that receives the event number associated with a task.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

FEEVT Illegal event usage (task is not a FORTRAN 5 task).

Task codes .

Notes

A zero event represents a lack of event association.

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	an integer that specifies the task's ID number.
priority	an integer variable that receives the task's priority.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

Task codes.

Example

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

MYEV

Obtains the calling task's event number.

Format

CALL MYEV (event)

Argument

event	an integer variable that receives the task's event number.

Error Conditions

No error conditions are currently defined.

Notes

A zero event represents a lack of event association.

Example

```
CALL MYEV (J)
```

MYID

Obtains the calling task's ID number.

Format

CALL MYID (task ID)

Argument

task ID	an integer variable that receives the task's ID number.

Error Conditions

No error conditions are currently defined.

Notes

Zero represents a lack of ID.

Example

```
CALL MYID (I)
```

MYPRI

Obtains the calling task's priority.

Format

CALL MYPRI (priority)

Arguments

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

Error Conditions

No error conditions are currently defined.

Example

```
CALL MYPRI (I)
```

Coding Example

```
C         This routine reports various task state information
C         about itself and a child task that it creates.
C

C         Anticipate a wakeup message from the task we are about
C         to create.

          ANTICIPATE 1

C         Initiate another task

          TASK SUB(1), ID=1, PRI=2

C         Give task 1 enough time to execute the ANTICIPATE statement

          CALL WAIT (1, 2, IER)
          CALL CHECK(IER)

C         Report the task's priority

          CALL GETPRI(1, IPRI, IER)
          CALL CHECK(IER)

          TYPE "The priority of the other task is ",IPRI

C         Report the task's event number

          CALL GETEV(1, IEV, IER)
          CALL CHECK(IER)

          TYPE "The event number of the other task is ",IEV

C         Wake up the other task

          WAKEUP IEV

C         Now report some information about ourselves

          CALL MYEV(IEV)
          CALL MYPRI(IPRI)
          CALL MYID(ID)

          TYPE "My event number is ", IEV
          TYPE "My priority is ", IPRI
          TYPE "My task ID is ", ID

C         Wait for a wakeup message from the other task, which will
C         then kill itself, leaving only this task.

          WAIT 1

C         By returning, we kill the last task and terminate this program

          STOP
          END
```

```
C       This subroutine forms the second task.  It waits for a
C       wakeup message from the main task, then wakes up the main task.
C

        SUBROUTINE SUB(ID)

        ANTICIPATE 5              ;We will do a WAIT 5 and we don't
                                  ;want to miss our WAKEUP message.

        TYPE "Task ",ID," is alive!"

        WAIT 5                    ;Suspend ourselves until the main
                                  ;task wakes us up.

        TYPE "Task ",ID," is awake again!"

        WAKEUP 1                  ;Wake up the main task

        RETURN
        END
```

End of Chapter

Chapter 17
Intertask Communication

In a multitask program, tasks can send and receive one-word messages. If several tasks attempt to receive the same message, only the highest priority task receives the message.

The system suspends a task that is receiving a message. Because of this suspension, you can use the routines in this chapter to synchronize tasks.

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

The Routines In This Chapter

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

REC

Receives a one-word message from another task.

Format

CALL REC (mailbox,message)

Arguments

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

message — an integer variable that receives the message.

Rules

Declare the mailbox in COMMON storage.

Never directly change or examine a mailbox.

Error Conditions

No error conditions are currently defined.

Notes

REC suspends the calling task until it receives the message.

Example

```
COMMON/BOX/ISLOT
CALL REC (ISLOT,IMSG)
```

Reference

?REC (Task call)

XMT

Transmits a one-word message to another task.

Format

CALL XMT (mailbox,message,error label)

Arguments

mailbox	a one-word aggregate through which the message passes.
message	a nonzero integer value you want to transmit to another task.
error label	a statement label to which the system transfers control when an error occurs.

Rules

Declare the mailbox in COMMON storage.

Never directly change or examine a mailbox.

You must specify statement label with a dollar sign; for example, $100 .

Error Conditions

The error conditions that may result are

Task codes.

Notes

The difference between XMT and XMTW is as follows:

XMT deposits a message.

XMTW deposits a message, and suspends the program until the task receives the transmitted message.

Example

```
      COMMON/MAIL/ITSK(20)
C     TRANSMIT THE MESSAGE IN IMSG
C     THROUGH THE MAILBOX ITSK(10)
      CALL XMT (ITSK(10),IMSG,$100)
```

Reference

?XMT (Task call)

XMTW

Transmits a one-word message to another task and waits for the task to receive it.

Format

CALL XMTW(mailbox,message,error label)

Arguments

mailbox	a one-word aggregate through which the message passes.
message	a nonzero integer value you want to transmit to another task.
error label	a statement label to which the system transfers control when an error occurs.

Rules

Declare the mailbox in COMMON storage.

Never directly change or examine a mailbox.

You must specify error label with a dollar sign; for example, $100 .

Error Conditions

The error conditions that may result are

Task codes.

Notes

The difference between XMT and XMTW is the following:

XMT deposits a message.

XMTW deposits a message and suspends the program until the task receives the transmitted message.

Example

```
COMMON/MBOX/ISLOT
CALL XMTW (ISLOT,IABC,$1050)
```

Reference

?XMTW (Task call)

Coding Example

```
C       This program demonstrates intertask communications.  The
C       routine reads characters from the input file using
C       1-character binary reads.  The characters are transmitted
C       to the other task, which writes them out.  When the routine has
C       read 25 characters, it terminates.
C
        COMMON/BOX/MAIL             ;1 word message slot

        ICOUNT = 0                  ;Count of characters read

C       Open the channel for RDSEQ

        CALL OPEN(11,"@OUTPUT",3,IER)
        CALL CHECK(IER)

C       Anticipate the wakeup message from the task we are
C       about to initiate.

        ANTICIPATE 1

C       Initiate the other task at the same priority as us, but
C       with a task I.D. of 1.

        TASK WRITER, ID=1

C       Wait for the other task to begin, then anticipate another
C       WAKEUP message.

        WAIT 1
        ANTICIPATE 1

C       Once we are awakened by another task, we enter the following
C       loop which reads 25 characters and transmits them to the
C       other task.

        DO 10 I=1,25

                CALL RDSEQ(11,ICHAR,1,IER)      ;Read a character w/out echo
                CALL CHECK(IER)

                CALL XMTW(MAIL,ICHAR,$100)      ;Transmit the read character
                                                ;waiting to continue until it
                                                ;is received by another task.

10      CONTINUE

C       Kill the other task

        CALL TIDK(1,IER)
        CALL CHECK(IER)

        STOP "End of Program"           ;Normal termination

C       The program reaches the following code if an error occurs
C       during XMTW.
100     STOP    "An Error Occurred in XMTW"

        END
```

```
C       This routine writes out characters received from the other


C       task.  It sends an initial WAKEUP message to the other
C       task to indicate that it is up and running.

        SUBROUTINE WRITER

        COMMON/BOX/MAIL          ;Mail slot for messages

C       Tell the main task that we are ready.

        WAKEUP 1

C       The following loop waits for a character from the main
C       task, then writes it on the output file.

10      CALL REC(MAIL,ICHAR)     ;Wait for a message

        CALL WRSEQ(11,ICHAR,1,IER)      ;Write the character to the output file
        CALL CHECK(IER)
        GO TO 10         ;Continue until we are killed.

        END
```

End of Chapter

Chapter 18
Requesting Delayed or Periodic Task Initiation

Normally, the system activates a task as soon as you initiate it. However, AOS provides a mechanism that initiates a task at a time you designate. The routines in this chapter enable you to create and queue a task initiation request that AOS will execute at some future time.

Queue Tables

The initiation request you make takes the form of a queue table. In the queue table, you supply information describing the tasks you want to initiate. After setting up the queue table, transfer it to the operating system as a task initiation request, via the ASSOCIATE routine.

Before using delayed or periodic task initiation, you must allocate an array in COMMON or STATIC storage large enough for the queue tables.

Each queue table contains three types of information:

- Descriptive information about the tasks you want to initiate. This includes the task identifier, task priority, stack size, and the task's initial subroutine.
- Information that designates times for task initiation: hour, second within the specified hour, interval between initiations, and number of times to initiate a task.
- Information that the system uses for its own bookkeeping.

See Table 18-1 for a detailed description of the queue table.

When you request delayed task initiation, AOS places the request on a queue for later processing. AOS removes the request from the queue for one of the following reasons:

- All requested task initiations have been performed.
- A call to CANCL removes the request from the queue.

AOS uses the memory allocated in COMMON or STATIC storage between queuing and dequeuing the tables. Because of this, you can not change the values in the array between the time you queue a table and the time you remove it from the queue. If subroutines allocate stack space for queue tables, you must remove them from the queue before the subroutine returns. When the subroutine returns, the stack space is free.

Task Initiation

Call ASSOCIATE to fill in a table with 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 it at a specific future time. CYCLE requests periodic initiation with a delay between initiations.

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.

Task Completion

When a queue table's requests are satisfied, the system dequeues it and you then have access to the table. You must terminate each task when its work is finished. If your program returns from the task's initial subroutine, it automatically terminates itself.

Premature Termination of a Request

If you must terminate one or more requests before the operating system finishes with them, call CANCL . Tell AOS which request(s) you want to terminate by referring to the queue table for that request.

When the queue table leaves the operating system's queue, its information does not change. You can queue the removed queue table with or without modification.

Never use the queue table before the operating system removes it from the queue.

Timing

Generally, a task initiation doesn't 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 routine converts the cycle time to one second.

You can 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 on the current day, the request will appear as a certain day (hours/24), and hour (hour-(hour/24)*24). Hours=24*D+H specifies Day(D) and Hour(H).

Three values associated with time initiation, QDCC, QDSH, and QDCI are unsigned. Before describing these values we remind you that FORTRAN 5 represents the values 32768 through 65534 as -32768 through -2. Minus 1 represents 65535.

If the value of QDCC is -1, the system initiates an unlimited number of tasks. If the value is 0, the the system returns the error ERQTS (error in user task table). Otherwise, the values of QDCC are between 1 and 65534.

QDSH represents the starting hour. If it has a value of -1, the system initiates the task immediately. Otherwise, its values lie between 0 and 65534.

QDCI represents the rerun time based in seconds. Its values lie between 0 and 65335.

Table 18-1. Queue Table

Index	Mnemonic	Meaning
1	QDTLNK	LINK maintained by the system
2	QDPRI	Task priority. When you call ASSOCIATE, the routine fills this value in automatically.
3	QDID	Task ID. When you call ASSOCIATE, the routine fills this value in automatically.
4	QDPC	Maintained by the system
5-11	Reserved	Reserved
12	QDSH	Starting hour (-1 to request immediate initiation). START, TRNON, and CYCLE fill in this value for you.
13	QDSMS	Starting second within the hour (reserved, but ignored if QSH=-1).
14	QDCC	Number of times to initiate a task of this description (-1 to initiate an unlimited number of tasks). When you call TRNON, START, or CYCLE, the routines fill this value automatically.
15	QDCI	Creation time increment in seconds. If QNUM equals 1 or -1 and if you call FQTSK, you must fill in this value.
16	Reserved	Reserved
17	Q.MEM	Task partition descriptor. When you call ASSOCIATE, the routine fills the entry in automatically.
18	QTLEN	Reserved

The Routines In This Chapter

ASSOCIATE	Associates a queue table with a task initiation request.
CANCL	Prematurely removes a queued request.
CYCLE	Requests periodic task initiation.
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	the name of the task's initial routine. You must include this name in an EXTERNAL statement.
queue table	an aggregate 17 words long.
priority	an integer that specifies the task's initial priority.
stack size	an integer that specifies the stack size for the task.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Example

```
.
.
INTEGER IRAY(17)
EXTERNAL SUB06
.
.
.
CALL ASSOCIATE(SUB06,IRAY,3,0,IER)
CALL CHECK(IER)
```

CANCL

Prematurely removes a queued request.

Format

CALL CANCL (queue table,IER)

Arguments

queue table	an aggregate that contains the queue table associated with the request you want to cancel.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

Task codes.

Notes

This call removes the queue table from the queue of requests. However, it does not alter the information within the table.

Example

```
CALL CANCL(IRAY,IER)
CALL CHECK(IER)
```

Reference

?DQTSK (Task call)

CYCLE

Requests periodic task initiation.

Format

CALL CYCLE(queue table,cycle time,time units,IER)

Arguments

queue table	an aggregate that contains the queue table associated with this task.
cycle time	an integer that specifies the number of time units between initiations.
time units	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	an integer variable that receives the routine's completion status code.

Rules

Before using this routine, you must call ASSOCIATE .

Error Conditions

The error codes that may return in IER are

FERTC	No real time clock.
FEITU	Illegal time units code

Task codes.

Notes

The smallest effective resolution for periodic task initiations is one second.

Example

```
.
.
INTEGER ITIME(17)
EXTERNAL SUB06
.
.
.
CALL ASSOCIATE(SUB06,ITIME,3,1,0,IER)
CALL CYCLE(ITIME,5,0,IER)
CALL CHECK(IER)
```

Reference

?TASK (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	an aggregate that contains the queue table associated with that task.
delay time	an integer that specifies the number of time units you want to delay before executing this task.
time units	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	an integer variable that receives the routine's completion status code.

Rules

Before using this routine, you must call ASSOCIATE .

Error Conditions

The error codes that may return in IER are:

FERTC	No real time clock.
FEITU	Illegal time units code.

Task Codes.

Notes

The smallest effective resolution for periodic task initiations is one second.

Example

```
INTEGER IRAY (17)
.
.
CALL START(IRAY,10,2,IER)
CALL CHECK(IER)
```

Reference

?TASK (Task call)

TRNON

Queues a request for task initiation at a specific time.

Format

CALL TRNON(queue table,time array,IER)

Arguments

queue table	an aggregate that contains the queue table associated with this task request.
time array	an integer array whose first three elements contain the hours, minutes, and seconds of the task initiation time.
IER	an integer variable that receives the routine's completion status code.

Rules

Before using this routine, you must call ASSOCIATE .

Error Conditions

The error codes that may return in IER include

Task codes.

Example

```
      INTEGER IRAY(17),ITIME(3)
      EXERNAL 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

?TASK (task call)

Coding Example

```
C         This program demonstates the queued tasking mechanism
C         by initiating a queued task that will run at 5 second
C         intervals to report the time of day.
C

          STATIC ITABLE (18)       ;18 word qtable in static storage

C         Declare the name of the task as external

          EXTERNAL SUB

C         Call ASSOCIATE to indicate the relationship between the
C         Qtable and the task we are about to create

          CALL ASSOCIATE(SUB, ITABLE, 1, 1, IER)
          CALL CHECK(IER)

C         Call CYCLE to specify the time between task initiations

          CALL CYCLE(ITABLE, 5, 2, IER)    ;5 seconds per initiation
          CALL CHECK(IER)

C         Wait for 2 minutes, while the time reports are made

          CALL WAIT(2,3,IER)
          CALL CHECK(IER)

C         Cancel the queued task request, then wait for the cancellation

          CALL CANCL(ITABLE, IER)
          CALL CHECK(IER)

          CALL WAIT(5,2,IER)
          CALL CHECK(IER)

          STOP "All Done"
          END


C         This subroutine is the time reporting task.  It calls
C         FGTIME to get the current time, then writes the time
C         to the output file.  This task kills itself by executing
C         a RETURN statement.
C
          SUBROUTINE SUB

          CALL FGTIME(IHOUR,IMIN,ISEC)

          TYPE "The time is ",IHOUR,":",IMIN,":",ISEC
          RETURN
          END
```

End of Chapter

Chapter 19
Enabling and Disabling the Multitask Environment

In a normal multitask environment, the tasks you initiate compete for CPU control according to their relative priorities. However, in certain situations you may want a task to execute without competing for CPU control. To accomplish this, temporarily disable the multitask environment. The privileged task has exclusive control of the CPU until it relinquishes that privilege.

Disabling the Multitask Environment

Disable the multitask environment only when it is mandatory that other tasks not interrupt the privileged task.

SINGLETASK disables the multitask environment. After a task issues this call, it gains full CPU control. No other task can compete for CPU control until the privileged task calls MULTITASK .

When a task gains CPU control through SINGLETASK , interrupts are still enabled.

Enabling the Multitask Environment

MULTITASK re-enables the multitask environment. All tasks can again compete for CPU control.

Other Options

The use of SINGLETASK is not appropriate in all situations where a task needs exclusive CPU control. The following are two such situations, and the actions you can take:

- You want a task to receive primary CPU control, but need not prevent the execution of other tasks. In this case, simply give that task highest priority.
- You must deny other tasks access to a critical resource such as a sensitive database. However, you don't need to restrict task execution outside this resource. In this case, you can use the XMT/REC mechanism to "lock" the critical piece of code.

The 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.

Coding Example

```
C       This program demonstrates the disabling and enabling of
C       the multitask environment.  It creates a task that        h
C       increments a variable in common once every 1/4 second.
C       This task disables multitasking to examine and reset this
C       counter once every 5 seconds.  This routine terminates
C       the program after 10 iterations.
C
        COMMON /TICKS/ ICOUNT            ;Common variables are
                                         ;initialized to zero

        ANTICIPATE 4                     ;Prepare for WAKEUP from
                                         ;other task

C       Initiate the other task

        TASK SUB, ID=1

        WAIT 4                           ;Wait for other task to
                                         ;start up

C       The following loop waits 5 seconds, then disables scheduling
C       in order to read the counter ICOUNT.  It then resets the
C       counter to zero and re-enables scheduling.

        DO 10 I=1,10

            CALL WAIT(5,2,IER)           ;Wait 5 seconds
            CALL CHECK(IER)

            CALL SINGLETASK              ;Disable rescheduling

            TYPE "The counter is ",ICOUNT

            ICOUNT = 0                   ;Reset counter

            CALL MULTITASK               ;Restart the world

10      CONTINUE

        CALL TIDK(1, IER)                ;Kill the other task
        CALL CHECK(IER)

        STOP "End of Program"

        END
```

```
C        This subroutine increments the variable ICOUNT once
C        every 250 milliseconds.  Before it begins, it wakes
C        up the main task to indicate that it is up and running.

         SUBROUTINE SUB

         COMMON /TICKS/ ICOUNT

C        Wake up the main task

         WAKEUP 4

C        The following loop will be repeated until this task
C        is killed by the main task.

1        CALL WAIT(250, 1, IER)  ;Wait 250 milliseconds
         CALL CHECK(IER)

         ICOUNT = ICOUNT + 1     ;Bump count

         GO TO 1                 ;Loop ad infinitum

         END
```

End of Chapter

Chapter 20
Using Overlays

An overlay consists of one or more subroutines or functions stored in a disk file and used by a memory resident program. An overlay file is a disk file containing overlays. An overlay area is that portion of main memory that the program uses to load an overlay for execution of one of the overlay routines. Figure 20-1 shows how the overlay area relates to main memory.

FORTRAN 5 supports the AOS Load-On-Call facility of the AOS resource call (?RCALL) mechanism. See the *AOS Programmer's Guide* (093-000154) for more information on the Load-On-Call facility.

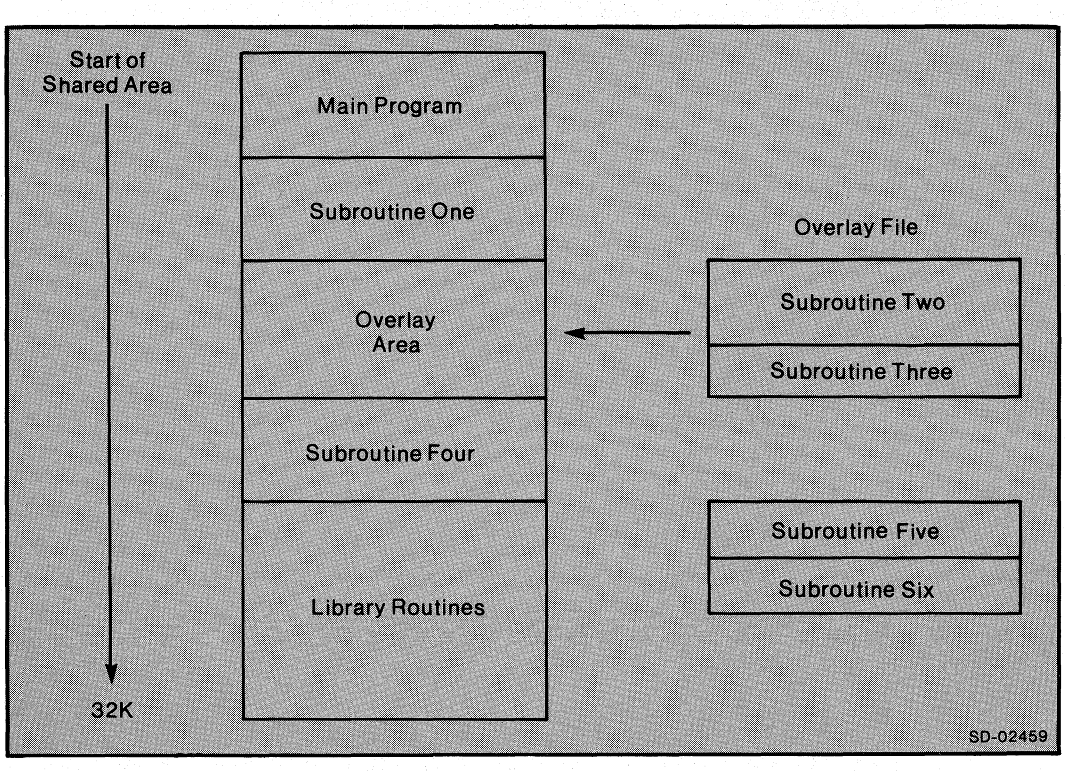

Figure 20-1. Overlays

Explicit Overlay Management

If you use the Load-On-Call facility you need not use the routines in this chapter to explicitly call an overlay routine. However, you may want to explicitly call overlay routines for the following reasons:

- Compatability with FORTRAN IV
- Running your program using RDOS FORTRAN 5
- Explicit control of the overlay because it is a non-FORTRAN 5 routine or because it contains data

Loading Overlays

You can request that AOS load overlays conditionally or unconditionally.

A conditional request loads the overlay area only if it is not already resident. Use a conditional request when you do not want to reinitialize the overlay when you recall it.

An unconditional request causes AOS to load the overlay even if it is already present. This reinitializes any data in the overlay.

Releasing Overlays

You must release an overlay when you finish with it so the main program can reuse the overlay area. Until you release it, the system assumes that an overlay routine has not completed execution. When you attempt to load in a new overlay under these circumstances, the loading routine suspends indefinitely as it waits for the overlay area to be released.

Many routines in this chapter require that you specify an overlay name. You declare an overlay name by an OVERLAY statement in exactly one subroutine or function in the overlay. Declare the name EXTERNAL in all other routines using it.

In this example, the user declared the overlay OFRED as EXTERNAL :

```
EXTERNAL OFRED
.
CALL OVLOAD (OFRED,-1,IER)
CALL CHECK (IER)
CALL FRED (Y)
.
END
```

The subroutine declares the overlay name:

```
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. FORTRAN 5 allows a *unit number* as an optional argument because FORTRAN IV versions of the routines require the information.

The Routines In This Chapter

EST	Loads an overlay unconditionally.
OVCLOSE	Closes an overlay file.
OVEXIT	Releases an overlay and returns to the overlay routine caller.
OVKILL	Kills a task and releases the overlay in which it is currently executing.
OVLOD	Loads an overlay.
OVOPN	Opens an overlay file.
OVREL	Releases an overlay.

EST

Unconditionally loads an overlay.

Format

CALL EST ([*unit number,]* overlay name,IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want to load. AOS ignores this argument.
overlay name	the name of the overlay (you must declare **overlay name** as external).
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in **IER** are

EROVN	Illegal overlay number.
ERADR	Illegal overlay size.

File System codes .

Notes

You must pair each overlay request with an eventual overlay release. Otherwise, AOS will reserve the overlay area indefinitely.

Example

```
EXTERNAL OV3
.
.
.
CALL EST (OV3,IER)
CALL CHECK (IER)
```

Reference

?OVLOD (System call)

OVCLOSE

Closes an overlay file.

Format

CALL OVCLOSE (IER)

Argument

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

Notes

Although OVCLOSE has no meaning in AOS, we include the routine for RDOS compatability.

OVEXIT

Releases an overlay and returns to the overlay routine caller.

Format

CALL OVEXIT (overlay name,IER)

Arguments

overlay name	the name of the overlay you want to release (you must declare overlay name as external).
IER	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 the given user overlay.

Notes

OVEXIT releases the overlay in which the current subroutine is executing. It then returns to the caller of the current subroutine.

Example

```
CALL OVEXIT (OVNAM,IER)
CALL CHECK (IER)
```

Reference

?OVREL (System call)

OVKILL
Kills the calling task and releases its overlay.

Format

CALL OVKILL (overlay name,IER)

Arguments

overlay name	the name of the overlay you want to kill and release (you must declare **overlay name** as external).
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

EROVN Invalid overlay number.

File System codes.

Notes

OVKILL kills the calling task and releases the overlay in which the task is executing. In a single-task environment, this causes program termination because there is only one task to kill.

Example

```
EXTERNAL OSUB1
.
.
.
CALL OVKILL (OSUB1,IER)
CALL CHECK (IER)
```

Reference

?OVKIL (System call)

OVLOD
Loads an overlay.

Format

CALL OVLOD (*[unit number,]* overlay name,flag,IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want to load. AOS ignores this argument.
overlay	the name of the overlay (you must declare **overlay** as external).
flag	an integer set to -1 when you want to load the overlay unconditionally.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

ERADR	Illegal overlay size.
EROVN	Illegal overlay number.

File System codes .

Notes

You must pair each overlay request with an eventual overlay release. Otherwise, AOS will reserve the overlay area indefinitely.

Aliases are FOVLY and FOVLD .

Example

```
EXTERNAL OV3
.
.
.
CALL OVLOD (OV3,-3,IER)
CALL CHECK (IER)
```

Reference

?OVLOD (System call)

OVOPN

Opens an overlay file.

Format

CALL OVOPN (*[unit number,]* pathname,IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of the file you want to open. AOS ignores this argument.
pathname	an aggregate that contains the name of the overlay file.
IER	an integer variable that receives the routine's completion status code.

Notes

Although OVOPN has no meaning in AOS, we include it for RDOS compatibility.

OVREL
Releases an overlay.

Format

CALL OVREL (overlay name,IER)

Arguments

overlay name	the name of the overlay you want to release (you must declare overlay name as external).
IER	an integer variable that receives a routine's completion status code.

Error Conditions

The error code that may return in IER is

EROVN	Invalid overlay number; the overlay area is not occupied by this user overlay.

Notes

You cannot issue this command from a routine within the overlay you want to release.

Aliases are FOVRL and UNEST .

Example

```
EXTERNAL OVLY1
.
.
.
CALL OVREL (OVLY1,IER)
CALL CHECK (IER)
```

Reference

OVREL (System call)

Coding Example

```
C       This program demonstrates the use of primitive overlay
C       management.  It calls two subroutines that will reside in
C       different overlays.
C
C       The F5LD command for this program would look like:
C
C       F5LD MAIN !* SUB1 ! SUB2 *!
C

        EXTERNAL OVERLAY1, OVERLAY2       ;The names of the two overlays
                                          ;must be declared external

        CALL OVLOD (OVERLAY1, -1, IER)    ;Unconditionally load OVERLAY1
        CALL CHECK(IER)

        CALL SUB1                         ;Call the subroutine in
                                          ;OVERLAY1

        CALL OVREL(OVERLAY1, IER)         ;Release the overlay area
        CALL CHECK(IER)

        CALL OVLOD (OVERLAY2, -1, IER)    ;Unconditionally load OVERLAY2
        CALL CHECK(IER)

        CALL SUB2                         ;Call the subroutine in
                                          ;OVERLAY2

        CALL OVREL(OVERLAY2, IER)         ;Release the overlay area
        CALL CHECK(IER)

        STOP "All Done"
        END
----------------------------------------------------------------

C       This subroutine resides in the first overlay (OVERLAY1).
C       It writes a message and returns.
C
        SUBROUTINE SUB1

        OVERLAY OVERLAY1         ;Declare the name of this overlay

        TYPE "Subroutine 1 has been called"

        RETURN

        END
----------------------------------------------------------------

C       This subroutine resides in the second overlay (OVERLAY2).
C       It writes a message and returns.
C
        SUBROUTINE SUB2

        OVERLAY OVERLAY2         ;Declare the name of the overlay

        TYPE "This is subroutine 2 "

        RETURN
        END
```

End of Chapter

Chapter 21
User/System Clock Commands

The *clock ticks* we name in this chapter refer to the interrupts generated by your system's real time clock.

The Routines In This Chapter

FDELAY	Delays a task a given number of clock ticks.
GHRZ	Obtains the real time clock frequency.
WAIT	Suspends a task for a specified time.

FDELAY
Delays a task for a given number of clock ticks.

Format
CALL FDELAY (ticks)

Argument

ticks	an integer that specifies the number of clock ticks you want to delay the calling task.

Error Conditions
No error conditions are currently defined.

Notes
An error causes termination of your program.

If you specify a number of ticks that are not a multiple of the real time clock period, then the system rounds off the delay interval to the succeeding multiple.

Alias is FDELY .

Example
```
CALL FDELAY (10)
```

Reference
?DELAY (System call)

GHRZ

Gets the real time clock frequency.

Format

CALL GHRZ (frequency,IER)

Argument

frequency	an integer variable that receives one of the following values:
	0 Frequency is 60 HZ (A.C. line frequency)
	1 Frequency is 10 HZ
	2 Frequency is 100 HZ
	3 Frequency is 1000 HZ
	4 Frequency is 50 HZ (A.C. line frequency)
IER	an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Notes

Alias is GFREQ .

Example

```
CALL GHRZ (IFREQ,IER)
CALL CHECK (IER)
```

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 — an integer that specifies the number of time units you want to suspend a task.

time units — an integer that specifies the delay count unit measurement. It can have one of the following values:

0	Real time clock ticks
1	Milliseconds
2	Seconds
3	Minutes

IER — an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Example

```
CALL WAIT (IPULS,0,IER)
CALL CHECK (IER)
```

Reference

?DELAY (System call)

Coding Example

```
C         This program demonstrates FDELAY, GHRZ and WAIT.
C

C         Report the real-time clock frequency to the user.

          CALL GHRZ(IFREQ, IER)           ;Determine the RTC frequency
          CALL CHECK(IER)

          TYPE "Your real-time clock has a frequency of ", IFREQ

C         Wait 30000 of these real-time clock ticks

          CALL WAIT(30000, 0, IER)
          CALL CHECK(IER)

          TYPE "30000 ticks later..."

C         Wait another 30000 ticks

          CALL FDELAY (30000)

          STOP "End of ticking"
          END
```

End of Chapter

Chapter 22
Transferring Control Between Programs And Accessing Command Line Information

AOS swapping and chaining facilities allow you to segment large programs and execute the segments separately.

Through swapping and chaining you can treat several programs as if they were one large program. However, because the system treats each segment as a separate part, it does not maintain data when it swaps from one segment to another and back again.

Swapping

Program swapping permits a program file to temporarily replace an executing program.

The following is a swapping sequence:

1. The executing program suspends its own execution to invoke another program.
2. AOS temporarily stores the suspended program on disk.
3. The new program is invoked.
4. The new program completes its execution.
5. AOS returns the original program and resumes its execution.

Chaining

Program chaining permits a calling program to be subdivided into separate, executable segments. Each segment calls the next one sequentially. The chained program cannot return to its caller. The following is a chaining sequence:

1. The executing program suspends its own execution to invoke another segment.
2. The new segment is invoked.
3. The new segment completes its execution.

The new segment can, in turn, chain a new segment. Chains go forward but do not return.

Accessing Command Line Information

When you invoke a program from the CLI, that program can determine the switches and arguments supplied when the program was invoked. The COMINIT routine initiates this mechanism. COMARG is then called repeatedly to access successive command arguments. The program name itself is argument zero. The example given for the COMARG routine in this chapter demonstrates this process.

COMARG returns the end-of-file code in the ier argument when no additional arguments exist. COMTERM is then called to terminate the sequence. Since these routines also exist in RDOS, this mechanism provides a mechanism which is independent of the operating system.

The Routines In This Chapter

CHAIN	Transfers control to another program.
COMARG	Reads one argument string and its switches from the command line.
COMINIT	Initializes argument processing for COMARG .
COMTERM	Terminates COMARG argument processing.
FCHAN	Transfers control to another program.
FSWAP	Temporarily transfers control to another program.
SWAP	Temporarily transfers control to another program.

CHAIN

Transfers control to another program.

Format

CALL CHAIN(pathname,IER)

Arguments

pathname	an aggregate containing the pathname of the program to be executed.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error codes that return in IER are

ERMEM	Attempt to allocate more memory than is available.
ERNSW	Insufficient amount of swap file space for the the system to maintain the new program file on disk.

File System codes .

Example

```
CALL CHAIN ("PROG2.PR",IER)
CALL CHECK (IER)
```

Reference

?CHAIN (System call)

COMARG

Reads one argument string and its switches from the command line.

Format

CALL COMARG(unit number,string, *[switches,]* IER)

Arguments

unit number	an integer that specifies a FORTRAN 5 unit number (AOS ignores this argument).
string	an aggregate that receives the ASCII text of a command argument.
switches	a 2-word aggregate that receives 26 bits of switch information.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER are

File System Codes.
System Call Codes.
Channel-Related Codes.

Notes

COMARG is equivalent to the RDOS COMARG routine.

The system ignores the unit number argument.

The system reports an end-of-file after it reads the last argument.

The aggregate *switches* receives a 26-bit map of single character switches. The 16 bits of the first word represent the A through P switches. The leftmost 10 bits of the second word represent the Q through Z switches. Other bits are turned off.

AOS supports only single character switches to provide common functionality with the RDOS version of this routine.

Example

Assume you entered

X MYPROG/C JOE/D/F HARRY

to start execution of the current program, MYPROG . The following statements show a possible application of calls to COMARG .

```
        CALL COMINIT(0,IER)
        .
C       READ ARGUMENT 0
        CALL COMARG(0,ITEXT,ISWITCH,IER)
        IF(IER.EQ.EREOF) GO TO 10
        CALL CHECK(IER)
        .
        .
C       READ ARGUMENT 1
        CALL COMARG(0,ITEXT,ISWITCH,IER)
        IF(IER.EQ.EREOF) GO TO 10
        CALL CHECK(IER)
        .
        .
C       READ ARGUMENT 2
        CALL COMARG(0,ITEXT,ISWITCH,IER)
        IF(IER.EQ.EREOF) GO TO 10
        CALL CHECK(IER)
        .
        .
10      CALL COMTERM(0,IER)
        CALL CHECK(IER)
```

In the first call to COMARG , ITEST receives the ASCII string MYPROG (the CLI removes the X). ISWITCH receives the bit flags corresponding to the switch, /C . In the second call, ITEXT receives the ASCII string JOE . ISWITCH receives the switch bits for /D and /F . Note that the three calls to COMARG have identical formats.

References

?GTMES (System call)

QGTMES (Runtime routine) QGTMES provides full access to the command line.

COMINIT

Initializes argument processing for COMARG.

Format

CALL COMINIT(unit number,IER)

Arguments

unit number	an integer that specifies a FORTRAN 5 unit number (AOS ignores this argument).
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER include

File System codes.
System Call codes.
Channel-Related codes.

Notes

The system ignores the argument unit number .

COMARG uses an internal counter set by COMINIT .

Example

```
CALL COMINIT(0,IER)
CALL CHECK(IER)
.
.
CALL COMARG(0,IAR(1),IER)
CALL CHECK(IER)
```

COMTERM

Terminates COMARG argument processing.

Format

CALL COMTERM(unit number,IER)

Arguments

unit number — an integer that specifies a FORTRAN 5 unit number (AOS ignores this argument).

IER — an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Notes

The system ignores the argument unit number .

COMTERM invalidates the internal counter used by COMARG .

Example

```
CALL COMTERM(0,IER)
CALL CHECK (IER)
```

FCHAN

Transfers control to another program.

Format

CALL FCHAN(pathname)

Argument

pathname	an aggregate that contains the pathname of a program to which you want to transfer control.

Error Conditions

The error conditions that may result are

ERNSW	Insufficient amount of swap file space for the system to maintain the new program file on disk.

File System codes .

Notes

The error conditions terminate the program.

Example

```
CALL FCHAN ("PROG2.PR")
```

Reference

?CHAIN (System call)

FSWAP
Temporarily transfers control to another program.

Format
CALL FSWAP("PROG3.PR")

Argument
pathname	an aggregate specifying the pathname of the program that receives control.

Error Conditions
The error conditions that may result are

File System codes.

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.

Example
```
CALL FSWAP("PROG.30")
```

Reference
?PROC (System call)

SWAP

Temporarily transfers control to another program.

Format

CALL SWAP(pathname,IER)

Arguments

pathname	an aggregate containing the pathname of the program that receives control.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

ERNSW	Insufficient amount of swap file space for the system to write the new program file to disk.

File System codes .

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.

Example

```
CALL SWAP("PROG.PR",IER)
CALL CHECK (IER)
```

Reference

?PROC (System call)

Coding Example

```
C        This program will write out each of the arguments on
C        the command line that invoked this program, and
C        report on the existence of any single character switches
C        on any argument.
C        It will then SWAP to the CLI which will return back to
C        this program when the BYE command is given.
C
C        This program would be executed by a command like:
C
C        XEQ MAIN/A/B/C/D ARG1/E ARG2 ARG3/X/Y/Z
C

C        The following loop processes each of the command line arguments
C        until an END OF FILE error code is returned (EREOF).

         INCLUDE "F5ERR.FR"      ;Define the value of EREOF

         INTEGER ICOMMAND(136/2+1)        ;Buffer for command arguments
         INTEGER ISWITCHES(2)             ;2 words of switch info
         INTEGER ICHAR(13)                ;Upper case alphabet

         DATA ICHAR/"AB","CD","EF","GH","IJ","KL","MN","OP","QR","ST",
     +         "UV","WX","YZ"/

1        CALL COMARG(0, ICOMMAND, ISWITCHES, IER)
         IF (IER.EQ.1) GO TO 10  ;Continue if no error

C        If an error occurred, check to see if it is EREOF
C        (End of File). If so, continue on to the next part of
C        the program.

         IF (IER.EQ.EREOF) GO TO 30       ;Go to the next part
         CALL CHECK(IER)                  ;otherwise report the error


 10      WRITE (11,1101) ICOMMAND(1)
1101     FORMAT(S80)

C        Check each of the bits in the switches array to see if
C        any single character switches were supplied.

         DO 20 I=1,2             ;For each word in SWITCHES
           DO 20 J=15,0,-1       ;For each bit from LEFT to RIGHT
             IF (ITEST(ISWITCHES(I),J).EQ.0) GO TO 20

C        If the bit is set, report the letter of the corresponding
C        switch
         M = BYTE(ICHAR,(16*(I-1))+(16-J))

         WRITE(11,1202) M
1202     FORMAT("The /", A2," switch was set.")
20       CONTINUE

         GO TO 1         ;Repeat for each command argument
```

```
C
C       In this second part of the program, the AOS CLI is
C       invoked via SWAP.  Typing BYE to the CLI will return
C       to this program which will write a termination message.

30      CALL SWAP (":CLI.PR", IER)
        CALL CHECK(IER)

C       After the CLI returns, we produce a termination message

        TYPE "Welcome back from the AOS CLI!"

        STOP
        END
```

End of Chapter

Chapter 23
Reporting Errors and Messages

In this chapter we refer to ISA error codes. See the section on status variables in Chapter 2, "Error Handling" and the section on IER in Chapter 6, "About the Runtime Routines" for more information on ISA error codes.

The Routines In This Chapter

CHECK	Checks the status returned from a runtime routine.
EBACK	Terminates a program and indicates an error.
ERROR	Outputs a runtime error message and terminates the program.
EXIT	Terminates the program with no error.
GETERR	Determines the cause of the last END= or ERR= branch.
MESSAGE	Outputs a message to the error files and continues program execution.

CHECK
Checks the status returned from a runtime routine.

Format

CALL CHECK(error)

Argument

error	an integer variable that has received a routine's completion status code returned in IER by a prior call to a runtime routine

Error Conditions

If the error status code is 1, the routine returns to the calling program and reports no errors. In all other cases, the routine invokes the error reporter which reports the error conditions with a traceback and terminates the program.

Example

```
CALL CHECK(IER)
```

EBACK

Terminates a program and indicates an error.

Format

CALL EBACK(error)

Argument

error   an integer that specifies an error code

Upon execution of a call to EBACK , the routine makes an unconditional return to the father process. If that program is the CLI, one of the following occurs:

- If the error status code is an AOS error code or a FORTRAN 5 error code, the system displays the appropriate error message. This message is taken from the system error file, :ERMES.

- If you specify one of your own error status codes, the CLI does not recognize it. The system then displays UNKNOWN ERROR CODE *n* , where *n* is the argument passed to EBACK.

Error Conditions

No error conditions are currently defined.

Example

```
CALL DFILW("FILE20",IER)
IF (IER.NE.1) CALL EBACK (IER)
```

Reference

?RETURN (System call)

ERROR

Outputs a runtime error message and terminates the program.

Format

CALL ERROR (error message)

Argument

error message   an aggregate that contains your message

Error Conditions

No error conditions are currently defined.

Notes

When you call ERROR , the FORTRAN 5 error reporter performs an error traceback and sends your message to the error files. The program terminates.

Example

```
CALL ERROR ("FATAL ERROR FROM PHASE 2A")
```

Reference

?RETURN (System call)

EXIT

Terminate a program with no error.

Format

CALL EXIT

Arguments

None

Error Conditions

No error conditions are currently defined.

Notes

The aliases for EXIT are BACK and FBACK .

Reference

?RETURN (System call)

GETERR

Determine the cause of the last END= or ERR= branch.

Format

CALL GETERR(error)

Argument

error	an integer receiving the error code from the error causing the last ERR= or END= branch in a FORTRAN 5 I/O statement

Error Conditions

No error conditions can occur.

Notes

GETERR returns a code specifying the error that caused the program to take an ERR= or END= branch in an I/O or task statement. You can pass this code to CHECK to report the error.

You can also use GETERR after an alternate return from those routines which have an alternate return argument; e.g., FTASK .

GETERR resets the internal error value. This is your only method of resetting it. GETERR returns a value of 1 if no error has occurred since the last call to it.

Example

```
      READ FREE(11,ERR=100)X,Y,Z
      .
      .
      .
100   CALL GETERR(I)
      IF (I.EQ.ERSPC)GOTO 200
```

MESSAGE

Outputs a message to the error files and continues program execution.

Format

CALL MESSAGE(error message)

Argument

error message   an aggregate that contains your message

Error Conditions

No error conditions are currently defined.

Notes

When you execute a call to MESSAGE , the FORTRAN 5 error reporter performs an error traceback and sends your message to the error files. The routine then returns to the calling program.

Example

```
CALL MESSAGE (''NONFATAL ERROR #17'')
```

Coding Example

```
C       With this program, you enter an error code and receive
C       the corresponding error message by using CALL CHECK.
C       GETERR, ERROR and MESSAGE are also used.
C

        WRITE (11,1101,ERR=9000)
1101    FORMAT("Enter an ISA error code number in decimal: ",Z)

        READ FREE (10,ERR=9010) IERCD

        CALL CHECK (IERCD)      ;Report the error if IERCD<>1

C       If control returns, the value entered was 1.  Tell the
C       user that fact by calling MESSAGE.  CALL EXIT is then
C       done to terminate the program.  A call to ERROR is not
C       done, because we do not wish to report this via the CLI
C       as an error, only an informative message.

        CALL MESSAGE ("The error code 1 indicates that no error
     + occured")

        CALL EXIT       ;Terminate the program

C       Control should not return

C       The following statements are reached via ERR= branches in
C       the above READ and WRITE.
C       In both cases, CALL GETERR is used to determine the
C       cause of the error, then CALL ERROR is used to report the
C       errors

9000    CALL GETERR(IERCD)      ;Return the cause of the ERR= branch

        CALL ERROR(IER)         ;Report the error. Should not return

9010    CALL GETERR(IERCD)      ;Return the cause of the ERR= branch

        CALL ERROR(IER)         ;Report the error. Should not return

        END
```

End of Chapter

Chapter 24
Using Extended Memory

FORTRAN 5 provides you with an explicit method for accessing larger amounts of data than could fit in your program's address space. This mechanism is called Extended Memory Mapping, and utilizes the AOS shared page (?SPAGE) mechanism described in the *AOS Programmer's Manual*.

Extended Memory Mapping permits you to define a *window* in your program, usually an array in named COMMON. Through the window, you access pages of extended memory which AOS maintains outside your address space. Extended memory can be up to 255 1024-word blocks long.

Use the MAPDF routine to define the size and location of this window. Use the REMAP routine to place an area of extended memory into the window for access by the program. Other routines described in this chapter permit you to initialize the contents of extended memory from a file, or load and dump selected portions of extended memory to a file.

Defining the Window Size

Within your FORTRAN 5 program, you must align the window in memory on a 1024-word boundary. To do this, place the array in named COMMON and make it the first array in the common block. AOS can then move 1024-word pages of memory via the memory management hardware (MAP) of the ECLIPSE . The 1024-word boundary corresponds to a physical memory page boundary.

Aligning the Common Block

Use LINK to align the common block. Include the name of the common block at the end of the F5LD command line, and append the /SHARE and /ALIGN=10 switches to it. LINK aligns the window and permits use of the extended memory routines. Failure to align a window produces the error, *Window Aggregate Does Not Begin On 1024-Word Boundary*. See Chapter 1 for more information about LINK .

Using Extended Memory in a Multitask Environment

A single program defines only one window map. In a multitask environment, several tasks can share the same window map. The FORTRAN 5 interfaces to the extended memory facility do the following:

- allow multiple tasks to concurrently access the same database in extended memory
- enable you to implement a scheme in which each of several tasks has its own window map

Several tasks can concurrently access extended memory through the VFETCH and VSTASH calls. These calls remap window block 0 to the appropriate extended memory block, and then transfer a block of words.

While one task accesses extended memory through window block 0, the system suspends another task's activity through the ?DRSCH/?ERSCH system calls (refer to the *AOS Programmer's Manual*). The system can then protect a task's access of extended memory from other tasks' access, if the blocks of words accessed all lie in the same 1024-word extended memory block.

You may have a situation where a given task must perform a second remapping operation. This would bring the next portion of the desired block of extended memory into the window. When the block of words you want to transfer spans two or more extended memory blocks, other tasks can gain control and access extended memory.

Transfer Completion

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 uses VFETCH or VSTASH .

The VFETCH and VSTASH routines perform the following sequence of actions in a multitask environment:

1. Disable rescheduling to lock out other tasks.
2. Remap the desired extended memory page into window block 0.
3. Perform the transfer of data.
4. Unlock the window, re-enabling task scheduling.

If the transfer involves more than one extended memory block, VFETCH and VSTASH will repeat steps one through four until the transfer is complete.

Separate Window Maps

In a multitask environment, you can create a separate window for each of several tasks. First, 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. Next assign a window block to each task for its exclusive use (each task performs its own REMAP calls).

Notes:

- The system does not lock the remapping code in the multitask environment.
- The system does not protect an extended memory block if two or more tasks access it through their separate windows. Unprotected memory blocks are possible and permissable.

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. If you do this, define the window size as $n+1$ blocks. VFETCH and VSTASH use block 0, and the program can reserve blocks 1 through $n+1$ as the individual window blocks for the *n* different tasks.

Virtual Data Files

The routines in this chapter make use of a temporary virtual data disk file. In this file, AOS maintains any extended memory pages that can no longer reside in main memory.

Before you call MAPDF for the first time, call VOPEN to specify the name of the virtual data file. If you don't, MAPDF opens a default virtual data file named ?pid.VIRTUAL.DATA.TEMP where pid is the program's process ID.

You can close an open virtual data file with VCLOSE , then open a different file with VOPEN, if you wish.

The calls in this chapter exist for both AOS and RDOS, permitting you to write programs that run under either system.

Interprocess Communication Through Shared Data

Two or more processes can open the same virtual data file with VOPEN and access the same extended memory data. This mechanism provides an efficient means of interprocess communication and shared data access. However, it does not provide any data-locking features.

The Routines 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 disk file into extended memory
EWRB	Writes a series of blocks from extended memory to a disk file.
IVF	INTEGER form for VFETCH .
MAPDF	Defines a window map or redefines the default 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 of extended memory using the contents of a disk file.
VOPEN	Opens a virtual data file.
VMEM	Determines the amount of extended memory available to a program.
VCLOSE	Closes a virtual data file.
VSTASH	Copies one or more elements into extended memory.

ERDB

Reads a series of blocks from a disk file into extended memory.

Format

CALL ERDB(unit number,disk block,memory block,blockcount, *[partial count,]* IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of a disk file.
disk block	an integer that specifies the initial disk block number you want to read. The first block is block 0.
memory block	an integer that specifies the initial extended memory quarter block (a 256-word portion of a full 1024-word memory block) number into which the system reads data.
block count	an integer that specifies the number of disk blocks you want to transfer (maximum=255).
partial count	an optional integer variable that receives the number of quarter blocks transferred successfully in the event of an end-of-file condition.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

The error conditions that may return in IER are

FEIFN	Illegal unit number.
FEBLN	Illegal extended memory block number.
FEBLC	Illegal block count.

File system codes.
Memory codes.
Miscellaneous codes.

Example

```
CALL ERDB(3,6,0,4,ICNT,IER)
CALL CHECK(IER)
```

References

?SPAGE (System call)
?READ (System call)

EWRB

Writes a series of blocks from extended memory to a disk file.

Format

CALL EWRB (unit number,disk block,memory block,block count, *[partial count,]* IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number of a disk file.
disk block	an integer that specifies the initial disk block to which the routine writes data.
memory block	an integer that specifies the initial extended memory quarter block (a 256-word portion of a full 1024-word memory block) number from which the routine writes data. ERDB rounds this number up to the next multiple of 4.
block count	an integer that specifies the number of disk blocks of data you want to transfer (maximum = 255).
partial count	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	an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in IER are

FEIFN	Illegal unit number.
FEBLN	Illegal extended memory block number.
FEBLC	Illegal block count.

File system codes.
Memory codes.
Miscellaneous codes.

Notes

Alias is EWRBLK .

Example

```
C      WRITE QUARTER BLOCKS 3-7 TO DISK BLOCKS 6-10
       CALL EWRB(1,6,3,5,ICNT,IRT)
       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 EWRB exhausts disk file space in the course of writing, INCT receives the number of disk blocks successfully written.

References

?READ (System call)
?FLUSH (System call)

MAPDF

Defines a window map or redefines the default element size.

Formats

CALL MAPDF(count,window array,window size, *[element size,]* IER)

CALL MAPDF(element size,IER)

Arguments

count	an integer that specifies the total number of extended memory blocks you want to use. This number should include blocks in the window, plus any additional extended memory blocks.
window array	an aggregate in your program through which you make references to extended memory. You must allocate window array on a 1024-word boundary.
window size	an integer that specifies the size of the window in 1024-word blocks.
element size	an integer that specifies the default size of an element in words for use by VFETCH and VSTASH (if omitted, the element size is 1).
IER	an integer variable that receives the routine's completion status code.

Rules

Before you call MAPDF for the first time, call VOPEN to specify the name of the temporary virtual data file. If you don't, MAPDF opens a default virtual data file named ?pid.VIRTUAL.DATA.TEMP where pid is the program's process ID.

You must align the the common block that contains the window on a 1024-word boundary. See the introduction to this chapter for more information on how to align the common block.

Error Conditions

Error codes that may return in IER are

FEW1K   Window aggregate does not begin on a 1024-word boundary.

File system codes.
Memory codes.
Miscellaneous codes.

Notes

Only one window can exist within a program.

Two forms exist for this call. Note that you can use the first form only once in a program. You cannot change the values it establishes except for the element size. You can change this with the second form of the MAPDF call.

MAPDF uses the variable you supply as count only as a value for use with VLOAD and VDUMP . The maximum value for count should be 255 plus the size of the window in blocks.

Examples

Example 1.

```
      INTEGER WINDOW(2048)
      CALL MAPDF(7,WINDOW,2,4,IER)
      CALL CHECK(IER)
```

In this example, the call sets up a window map using a total of 7 blocks of memory. Since the aggregate **WINDOW** is 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. Then load it so that it begins on a 1K boundary. (You can do this with the **LINK/ALIGN** switch). The element size is set to 4, meaning that the routine will access extended memory in multiples of 4 words at a time.

Example 2.

```
      CALL VMEM (N,IER)
      CALL CHECK(IER)
      .
      .
      CALL MAPDF(N+1,IWIND,1,IER)
      CALL CHECK(IER)
```

In this example, the 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 which participate in the window mapping is n+1. The routine sets the element size to 1 by default.

Example 3.

```
      .
      .
C     SET UP WINDOW MAPPING
C     DEFINE ELEMENT SIZE AS 3
      CALL MAPDF(7,WINDOW,1,3,IER)
      .
      .
C     ACCESS ELEMENTS OF SIZE 3
      CALL MAPDF(5,IER)
      .
      .
C     ACCESS ELEMENTS OF SIZE 5
      CALL MAPDF(3,IER)
```

REMAP

Alters the mapping of window blocks to extended memory blocks.

Formats

CALL REMAP(starting window block,starting extended memory block, *[number of blocks,]* IER)

CALL REMAP(block number,IER)

Arguments

starting window block	an integer that specifies the number of the starting block in the window you want to map (window blocks start at 0).
starting extended memory block	an integer that specifies the number of the starting block in extended memory to which blocks in the window will be mapped (must be between 0 and 255).
number of blocks	an integer that specifies the number of blocks you want to remap. If you omit this argument, the cali remaps 1 block.
block number	an integer that specifies the block number in extended memory to which window block 0 should be mapped.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in IER are

FEBLN	Block number in window or map exceeds 255.
FEVOP	Virtual data file not open.

File system codes.
Memory codes.

Notes

Two forms exist for this call. Note that the first form remaps any number of consecutively numbered window blocks to consecutively numbered extended memory blocks. The second form is intended mainly for windows of a single block, block 0. However, it can also remap block 0 of a multiple block window.

Examples

Example 1.

```
C       MAP BLOCK 0 IN THE WINDOW
C       TO BLOCK 3 IN EXTENDED MEMORY
        CALL REMAP(3,IER)
        CALL CHECK(IER)
```

Example 2. (Performs the same action as Example 1 using the second format.)

```
        CALL REMAP(0,3,IER)
        CALL CHECK(IER)
```

Example 3.

```
C       MAP BLOCKS 2, 3, AND 4 IN THE WINDOW TO
C       BLOCKS 0, 1, AND 2 IN EXTENDED MEMORY, RESPECTIVELY
        CALL REMAP(2,0,3,IER)
        CALL CHECK(IER)
```

Reference

?SPAGE (System call)

VCLOSE

Closes a virtual data file.

Format

CALL VCLOSE(IER)

Argument

IER — An integer variable that receives the routine's completion status code.

Error Conditions

The error codes that may return in IER are

Channel related codes.
File system codes.

Notes

VCLOSE closes the current virtual data file that was previously opened by VOPEN or MAPDF. If no data file was previously open, the error message ERFNO , "Channel Not Open" is returned in IER .

VCLOSE does not flush any modified shared pages before closing the file. Thus, the contents of the file will not reflect the contents of extended memory at the time of the call to VCLOSE. Use VDUMP to obtain a disk file copy of extended memory.

VCLOSE does not delete the virtual data file. You must delete this file yourself.

Example

```
CALL VCLOSE(IER)
CALL CHECK(IER)
```

Reference

?SCLOSE (System call)

VDUMP

Copies all of extended memory to a disk file.

Format

CALL VDUMP(unit number, *[block count,]* IER)

Arguments

unit number	an integer that specifies the FORTRAN 5 unit number on which you opened the disk file.
block count	an integer variable that receives the number of disk blocks successfully written if the write operation cannot complete. The reasons it may not complete are an end-of-file condition or if disk space is exhausted on the virtual data file.
IER	an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in IER are

FEIFN	Illegal unit number.

File system codes.
Memory codes.

Notes

VDUMP uses the number of extended memory blocks supplied in the count argument of the MAPDF call to determine the number of blocks to dump.

VDUMP may exhaust the disk file space in the current directory during the write operation. If this happens, the number of successfully written 256-word disk blocks is returned in ICNT .

Example

```
CALL VDUMP (2,ICNT,IER)
CALL CHECK (IER)
```

This call dumps all of extended memory to the disk file opened on FORTRAN 5 unit 2. (You lose the previous contents of the disk file.)

References

?SPAGE (System call)
?WRITE (System call)

VFETCH

Copies one or more elements from extended memory.

Format

CALL VFETCH(data area,index, *[elements,[size]]*)
IVF(index)
VF(index)
DVF(index)
CVF(index)
DCVF(index)

Arguments

data area	an aggregate that defines the area into which VFETCH reads data from extended memory.
index	an integer that specifies the index of the first element.
elements	an integer that specifies the number of elements you want to fetch (if omitted, one element is fetched).
size	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 Conditions

The error codes that may result are

FEEOB   Extended memory reference out of bounds.

File system codes.

Notes

VFETCH reads *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 acts as an integer function if you want to fetch a single word from extended memory. Use VF in the case of real numbers, when fetching 2-word elements, etc.

All entry points to VFETCH , including VFETCH itself, are functionally identical. See Examples 4 and 5 for use of the more readable, array-like syntax for VFETCH . Note that you must declare DVF, CVF, and DCVF DOUBLE PRECISION, COMPLEX, or DOUBLE PRECISION COMPLEX (respectively), if used.

Examples

All examples assume the following:

You set the element size to 4 by a prior MAPDF call.

You declared the variable DX and the array DY as double precision.

Example 1.

```
C       FETCH THE 1200TH 4-WORD ELEMENT INTO VARIABLE
C       DX (THE WORDS AT OFFSETS 4796 THROUGH 4799)
        CALL VFETCH(DX,1200)
```

Example 2.

```
C       FETCH DY(12),DY(13), AND DY(14) FROM
C       THE 2500TH, 2501ST, AND 2502ND
C       4-WORD ELEMENTS OF EXTENDED MEMORY
        CALL VFETCH(DY(12),2500,3)
```

Example 3.

```
C       FETCH THE 4797TH WORD (AT OFFSET 4797)
C       OF EXTENDED MEMORY INTO THE INTEGER VARIABLE I
        CALL VFETCH (I,4797,1,1)
```

In Example 3, *size* temporarily overrides the permanent element size of 4, specified with MAPDF . Note that you must give *elements* , even though it is 1, because you gave *size* .

Example 4.

```
        DOUBLE PRECISION DVF
        .
        .
C       FETCH THE 4500TH 4-WORD ELEMENT
C       IN EXTENDED MEMORY INTO VARIABLE DX
        DX=DVF(4500)
C       FETCH THE 5875TH 4-WORD ELEMENT OF EXTENDED
C       MEMORY INTO DOUBLE PRECISION ARRAY ELEMENT DY(64)
        DY(64)=DVF(5875)
```

Example 4 shows how you can use the alternate entry points of VFETCH to make VFETCH act as a function.

Example 5.

```
      DOUBLE PRECISION DVF
C     INTEGER IVF (IMPLICITLY TYPED)
      .
      .
C     STATEMENT FUNCTIONS TO SIMULATE
C     ARRAYS IN EXTENDED MEMORY
      .
      .
C     IVA LOOKS LIKE A 200X20 INTEGER
C     ARRAY (FOR READING)
      IVA(I,J)=IVF(20*(J-1)+I,1,1)
      .
      .
C     DVA LOOKS LIKE A 3000-ELEMENT DOUBLE
C     PRECISION ARRAY WHICH OCCUPIES EXTENDED
C     MEMORY FOLLOWING THE 4000-WORD INTEGER
C     "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)
```

In Example 4, VFETCH treats extended memory as a vector; i.e., linearly subscripted. Therefore, to provide a syntax for multiple subscripting, you need to include only a statement function. The statement function performs the mapping of a multiply subscripted indexing function into a linear subscript. An advantage of this syntax for VFETCH is that you can have several implicit calls to VFETCH in a single line. Example 4 illustrates this 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. This is because we have already defined the permanent element size as 4 in these examples. For the same reason, you need not pass an element size in calls to VFETCH under the guise of DVF .

VLOAD

Initializes all of extended memory using the contents of a disk file.

Format

CALL VLOAD (unit number, *[block count,]* IER)

Arguments

unit number — an integer that specifies the FORTRAN 5 unit number on which you opened the disk file.

block count — an integer variable that receives the number of disk blocks successfully read if the system cannot complete the read operation due to an end-of-file condition.

IER — an integer variable that receives the routine's completion status code.

Error Conditions

Error codes that may return in IER are

FEIFN — Illegal unit number.

File system codes.
Memory codes.

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 blocks that were read successfully returned in ICNT .

References

?SPAGE (System call)
?READ (System call)

VMEM

Determines the amount of extended memory available to a program.

Format

CALL VMEM (count,IER)

Arguments

count	an integer variable that receives the number of free 1024-word blocks of extended memory (always returns 255 in AOS).
IER	an integer variable that receives the routine's completion status code.

Error Conditions

No error conditions are currently defined.

Notes

This routine exists in AOS to provide compatibility with RDOS. The number of blocks returned in **count** is always 255.

Example

```
CALL VMEM(I,IER)
CALL CHECK(IER)
```

VOPEN

Opens a virtual data file.

Format

CALL VOPEN(pathname,IER)

Arguments

pathname	An aggregate that contains the pathname of the disk file you want to use to buffer virtual data.
IER	An integer variable that receives the routine's completion status code.

Rules

If you do not call VOPEN before you call MAPDF , MAPDF opens a temporary virtual data file with the default name ?pid.VIRTUAL.DATA.TMP where pid is the process ID of the program.

Error Conditions

The error conditions that may return in IER are

Channel related codes.
File system codes.

Notes

VOPEN allows you to specify the name of the virtual data file that the other routines in this chapter will use. If the file does not exist, VOPEN creates it. If the file already exists, its file element size must be 4.

Example

```
CALL VOPEN(":UDD:LYNNE:VIRTUAL:?TEMPFILE.TMP",IER)
CALL CHECK(IER)
```

Reference

?SOPEN (System call)

VSTASH

Copies one or more elements into extended memory.

Format

CALL VSTASH (data area,index, *[elements,[size]]*)

Arguments

data area	an aggregate that defines the area from which VSTASH writes data into extended memory.
index	an integer that specifies the index of the element in extended memory where VSTASH copies the first of a number of consecutive elements.
elements	an integer that specifies the number of elements you want to copy (if omitted, one element is copied).
size	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 Conditions

The error codes that may return in IER are

FEEOB Extended memory reference out of bounds.

File system codes.
Memory codes.

Notes

VSTASH transfers *elements* * *size* words from the FORTRAN 5 aggregate data area to extended memory, beginning at offset (index-1) * *size* .

Alias is VS .

Examples

Usage of VSTASH is identical to that of VFETCH as described in Examples 1, 2, and 3. Note that you cannot use VSTASH or VS as a function, as you can with VFETCH and its alternate entry points. (See the last two Examples, 4 and 5, under VFETCH .)

Coding Example

```
C         This program demonstrates the extended memory manipulation
C         routines, and performs identically in RDOS and AOS.
C         A 1024 word array in COMMON (IWINDOW) is used as the
C         buffer in the user's address space.  Through this buffer, the
C         program accesses extended memory via VFETCH and VSTASH.
C
C         Enter an index into extended memory and a number of integers
C         you want to stash via VSTASH  beginning at that index.  You
C         then enter the numbers you want to stash.  VSTASH stashes
C         them, then re-reads them using VFETCH.
C         The numbers are then written out.  The index itself is
C         also stashed and fetched  from the first location in
C         extended memory.  Entering 0 as the index terminates the
C         program.

C         The window common block (MAPS) must be aligned on a 1024
C         word boundry and placed in the shared data area.  To do
C         so, include the common block name at the end of the F5LD
C         command line as follows (note that MAIN is the name of
C         this routine) :
C
C                 F5LD MAIN MAPS/SHARED/ALIGN=9
C
          COMMON/MAPS/IWINDOW(1024)          ;Declare the 1Kw window

          DIMENSION IARRY(20), IOUT(20)      ;Buffers for I/O

          CALL VMEM(MEM,IER)          ;Determine the number of extended
          CALL CHECK(IER)             ;memory blocks for RDOS compatibility

C         The following call to mapdf sets up the window, declares
C         that (MEM+1) is the largest number of extended memory blocks
C         which can be VDUMP'ed / VLOAD'ed, declares the size of the
C         window to be 1 (KW), and sets the default number of words to
C         be VFETCH'ed / VSTASH'ed to be 1.  In AOS, this routine also
C         opens the virtual data file which will contain the extended
C         memory data.

          CALL MAPDF((MEM+1),IWINDOW,1,1,IER)
          CALL CHECK(IER)

1         ACCEPT "Enter the index into extended memory: ",INDEX
          IF (INDEX.EQ.0) STOP "You Stopped Me"

2         ACCEPT "Enter the number of integers to be transferred: ",NINT
          IF (NINT.LE.20) GO TO 3          ;Insure not too many

          TYPE "There is a maximum of 20 integers to be entered"
          GO TO 2

3         DO 100 I = 1,NINT
             ACCEPT "Enter integer ",(+I),": ",IARRY(I)
100       CONTINUE

C         The following VSTASH stores the index into the first location
C         in extended memory.
C
          CALL VSTASH(INDEX,1,1)

C         The following VFETCH should return the index value as NINDEX.
C         (NINDEX should equal INDEX).
```

```
C
        CALL VFETCH(NINDEX,1,1)

        TYPE "The index you entered was ",NINDEX

C       The following VSTASH writes the integers you entered into
C       extended memory, starting at the index you entered
C
        CALL VSTASH(IARRY,INDEX,NINT)

C       The integers are then re-read via VFETCH into a different
C       array.
C
        CALL VFETCH(IOUT,NINDEX,NINT)

C       Write out the integers for verification

        TYPE "Here are the numbers you entered: "
        TYPE (IOUT(I),I=1,NINT)

        GOTO 1          ;Keep going until 0 index entered

        END
```

End of Chapter

Appendix A
FORTRAN 5 Runtime Error Parameters

FORTRAN 5 Errors				
Parameter	**Decimal Value**	**Octal Value**	**Default Action**	**Message**
FESOV	003076	6004	FATAL	Stack overflow
FEDAT	003077	6005	FATAL	Insufficient arguments for data initialization
FESBS	003078	6006	FATAL	Subscript out of bounds
FEFMT	003079	6007	RECOVERABLE	Illegal format item
FEINM	003080	6010	RECOVERABLE	Illegal input number
FERCL	003081	6011	RECOVERABLE	Output record too long
FERCS	003082	6012	RECOVERABLE	Input record too short
FEIFN	003083	6013	RECOVERABLE	Illegal unit number
FEATT	003084	6014	RECOVERABLE	Invalid or inconsistent file attribute
FESEK	003085	6015	RECOVERABLE	Record file required for seek
FESTK	003086	6016	RECOVERABLE	Illegal stack size
FEEVT	003087	6017	RECOVERABLE	Illegal event usage
FESQR	003088	6020	TRANSPARENT	Illegal argument for SQRT
FEEXP	003089	6021	TRANSPARENT	Illegal argument for EXP
FELOG	003090	6022	TRANSPARENT	Illegal argument for LOG
FEASC	003091	6023	TRANSPARENT	Illegal argument for ASIN or ACOS
FEATN	003092	6024	TRANSPARENT	Illegal argument for ATAN2

(continues)

FORTRAN 5 Errors				
Parameter	**Decimal Value**	**Octal Value**	**Default Action**	**Message**
FEPWR	003093	6025	TRANSPARENT	Illegal exponentiation
FEINT	003094	6026	TRANSPARENT	Integer overflow on conversion
FERTN	003095	6027	TRANSPARENT	Invalid return
FEFNU	003096	6030	RECOVERABLE	Unit number in use
FEMOP	003097	6031	RECOVERABLE	Illegal mode for OPEN
FERCR	00398	6032	RECOVERABLE	Record count required for contiguous file create-on-OPEN
FEEOB	003099	6033	FATAL	Extended memory reference out of bounds
FEW1K	003100	6034	RECOVERABLE	Window aggregate does not begin on 1024-word boundary
FEBLN	003101	6035	RECOVERABLE	Illegal block number
FEBLC	003102	6036	RECOVERABLE	Illegal block count
FENPC	003103	6037	RECOVERABLE	No file preconnected to a unit number
FERLN	003104	6040	RECOVERABLE	Illegal value for record length in LEN= specifier
FELEF	003105	6041	RECOVERABLE	Inconsistent specification for LEF mode
FEONO	003106	6042	RECOVERABLE	Overlay file not open
FEOAO	003107	6043	RECOVERABLE	Overlay file already open
FETID	003108	6044	RECOVERABLE	Illegal task identifier
FEPRI	003109	6045	RECOVERABLE	Illegal task priority
FEEVN	003110	6046	RECOVERABLE	Illegal event number
FEPNA	003111	6047	RECOVERABLE	Requested partition not available
FEITU	003112	6050	RECOVERABLE	Illegal time units code

(continued)

FORTRAN 5 Errors				
Parameter	**Decimal Value**	**Octal Value**	**Default Action**	**Message**
FERTC	003113	6051	RECOVERABLE	No real time clock
FETMQ	003114	6052	RECOVERABLE	Too many queue blocks specified
FEFPU	003115	6053	RECOVERABLE	Floating point hardware not present
FEMDV	003116	6054	RECOVERABLE	Multiply/Divide hardware not present
FEMEM	003117	6055	RECOVERABLE	Insufficient memory for FORTRAN 5 program
FEIOP	003118	6056	RECOVERABLE	Did not allow for IOPROG in IOPC call
FEPTO	003119	6057	RECOVERABLE	Program table overflow
FETIL	003120	6060	RECOVERABLE	Time interval too large
FEIRN	003121	6061	RECOVERABLE	Illegal record number
FEIFV	003122	6062	RECOVERABLE	Illegal flag value
FEFPT	003123	6063	FATAL	Floating point status not valid
FEOVF	003124	6064	FATAL	Floating point overflow
FEUNF	003125	6065	TRANSPARENT	Floating point underflow
FEDVZ	003126	6066	TRANSPARENT	Floating point division by zero
FEMOF	003127	6067	TRANSPARENT	Floating point mantissa overflow
FEZER	003128	6070	FATAL	Infinite loop at location 0
FENTH	003129	6071	FATAL	No floating pt trap handler loaded
FEWNA	003130	6072	FATAL	Wrong number of arguments supplied
FEUSR	003131	6073	FATAL	User exit
FEVOP	003132	6074	FATAL	Virtual data file not open

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERICM	000004	00004	ILLEGAL SYSTEM COMMAND
ERFNO	000005	000005	CHANNEL NOT OPEN
EROPR	000006	000005	CHANNEL ALREADY OPEN
ERSAL	000007	000007	SHARED I/O REQ NOT MAP SLOT ALIGNED
ERMEM	000008	000010	INSUFFICIENT MEMORY AVAILABLE
ERADR	000009	000011	ILLEGAL STARTING ADDRESS
EROVN	000010	000012	ILLEGAL OVERLAY NUMBER
ERIM	000011	000013	ILLEGAL TIME ARGUMENT
ERNOT	000012	000014	NO TASK CONTROL BLOCK AVAILABLE
ERXMT	000013	000015	SIGNAL TO ADDRESS ALREADY IN USE
ERQTS	000014	000016	ERROR IN QTASK REQUEST
ERTID	000015	000017	TASK I.D. ERROR
ERDCH	000016	000020	DATA CHANNEL MAP FULL
ERMPR	000017	000021	SYSTEM CALL PARAMETER ADDRESS ERROR
ERABT	000018	000022	TASK NOT FOUND FOR ABORT
ERIRB	000019	000023	INSUFFICIENT ROOM IN BUFFER
ERSPC	000020	000024	FILE SPACE EXHAUSTED
ERSFT	000021	000025	USER STACK FAULT
ERDDE	000022	000026	DIRECTORY DOES NOT EXIST
ERIFC	000023	000027	ILLEGAL FILENAME CHARACTER
ERFDE	000024	000030	FILE DOES NOT EXIST
ERNAE	000025	000031	FILE NAME ALREADY EXISTS
ERNAD	000026	000032	NON-DIRECTORY ARGUMENTS IN PATHNAME

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
EREOF	000027	000033	END OF FILE
ERDID	000028	000034	DIRECTORY DELETE ERROR
ERWAD	000029	000035	WRITE ACCESS DENIED
ERRAD	000030	000036	READ ACCESS DENIED
ERAWD	000031	000037	APPEND AND/OR WRITE ACCESS DENIED
ERNMC	000032	000040	NO CHANNELS AVAILABLE
ERSRL	000033	000041	RELEASE OF NON-ACTIVE SHARED SLOT
ERPRP	000034	000042	ILLEGAL PRIORITY
ERBMX	000035	000043	ILLEGAL MAX SIZE ON PROCESS CREATE
ERPTY	000036	000044	ILLEGAL PROCESS TYPE
ERCON	000037	000045	CONSOLE DEVICE SPECIFICATION ERROR
ERNSW	000038	000046	SWAP FILE SPACE EXHAUSTED
ERIBS	000039	000047	DEVICE ALREADY IN SYSTEM
ERDNM	000040	000050	ILLEGAL DEVICE CODE
ERSHP	000041	000051	ERROR ON SHARED SET
ERRMP	000042	000052	ERROR ON REMAP CALL
ERGSG	000043	000053	ILLEGAL GHOST GATE CALL
ERPRN	000044	000054	NUMBER OF PROCESSES EXCEEDS 64
ERNEF	00045	000055	IPC MESSAGE EXCEEDS BUFFER LENGTH
ERIVP	000046	000056	INVALID PORT NUMBER
ERNMS	000047	000057	NO MATCHING SEND
ERNOR	000048	000060	NO OUTSTANDING RECEIVE
ERIOP	000049	000061	ILLEGAL ORIGIN PORT
ERIDP	000050	000062	ILLEGAL DESTINATION PORT

(continued)

System Errors			
Parameter	Decimal Value	Octal Value	Message
ERSEN	000051	000063	INVALID SHARED LIBRARY REFERENCE
ERIRL	000052	000064	ILLEGAL RECORD LENGTH SPECIFIED (=0)
ERARC	000053	000065	ATTEMPT TO RELEASE CONSOLE DEVICE
ERDAI	000054	000066	DEVICE ALREADY IN USE
ERARU	000055	000067	ATTEMPT TO RELEASE UNASSIGNED DEVICE
ERACU	000056	000070	ATTEMPT TO CLOSE UNOPEN CHANNEL/DEVICE
ERITC	000057	000071	I/O TERMINATED
ERLTL	000058	000072	LINE TOO LONG
ERPAR	000059	000073	PARITY ERROR
EREXC	000060	000074	RESIDENT PROC TRIED TO PUSH (.EXEC)
ERNDR	000061	000075	NOT A DIRECTORY
ERNSA	000062	000076	SHARED I/O REQUEST NOT TO SHARED AREA
ERSNM	000063	000077	ATTEMPT TO CREATE > MAX # OF SONS
ERFIL	000064	000100	FILE READ ERROR
ERDTO	000065	000101	DEVICE TIMEOUT
ERIOT	000066	000102	WRONG TYPE I/O FOR OPEN TYPE
ERFTL	000067	000103	FILENAME TOO LONG
ERBOF	000068	000104	POSITIONING BEFORE BEGINNING OF FILE
ERPRV	0000069	000105	CALLER NOT PRIVILEGED FOR THIS ACTION
ERSIM	000070	000106	SIMULTANEOUS REQUESTS ON SAME CHANNEL
ERIFT	000071	000107	ILLEGAL FILE TYPE
ERNRD	000072	000110	INSUFFICIENT ROOM IN DIRECTORY

(continued)

System Errors			
Parameter	Decimal Value	Octal Value	Message
ERILO	000073	000111	ILLEGAL OPEN
ERPRH	000074	000112	ATTEMPT TO ACCESS PROC NOT IN HIERARCHY
ERBLR	000075	000113	ATTEMPT TO BLOCK UNBLOCKABLE PROC
ERPRE	000076	000114	INVALID SYSTEM CALL PARAMETER
ERGES	000077	000115	ATTEMPT TO START MULTIPLE GHOSTS
ERCIU	000078	000116	CHANNEL IN USE
ERICB	000079	000117	INSUFFICIENT CONTIGUOUS DISK BLOCKS
ERSTO	000080	000120	STACK OVERFLOW
ERIBM	000081	000121	INCONSISTENT BIT MAP DATA
ERBSZ	000082	000122	ILLEGAL BLOCK SIZE FOR DEVICE
ERXMZ	000083	000123	ATTEMPT TO XMT ILLEGAL MESSAGE
ERPUF	000084	000124	PHYSICAL UNIT FAILURE
ERPWL	000085	000125	PHYSICAL WRITE LOCK
ERUOL	000086	000126	PHYSICAL UNIT OFFLINE
ERIOO	000087	000127	ILLEGAL OPEN OPTION FOR FILE TYPE
ERNDV	000088	000130	TOO MANY OR TOO FEW DEVICE NAMES
ERMIS	000089	000131	DISK AND FILE SYS REV #'S DON'T MATCH
ERIDD	000090	000132	INCONSISTENT DIB DATA
ERILD	000091	000133	INCONSISTENT LD
ERIDU	000092	000134	INCOMPLETE LD
ERIDT	000093	000135	ILLEGAL DEVICE NAME TYPE
ERPDF	000094	000136	ERROR IN PROCESS UST DEFINITION
ERVIU	000095	000137	LD IN USE, CANNOT RELEASE

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERSRE	000096	000140	SEARCH LIST RESOLUTION ERROR
ERCGF	000097	000141	CAN'T GET IPC DATA FROM FATHER
ERILB	000098	000142	ILLEGAL LIBRARY NUMBER GIVEN
ERRFM	000099	000143	ILLEGAL RECORD FORMAT
ERARG	000100	000144	TOO MANY OR TOO FEW ARGUMENTS TOO PMGR
ERIGM	000101	000145	ILLEGAL ?GTMES PARAMETERS
ERICL	000102	000146	ILLEGAL CLI MESSAGE
ERMRD	000103	000147	MESSAGE RECEIVE DISABLED
ERNAC	000104	000150	NOT A CONSOLE DEVICE
ERMIL	000105	000151	ATTEMPT TO EXCEED MAX INDEX LEVEL
ERICN	000106	000152	ILLEGAL CHANNEL
ERNRR	000107	000153	NO RECEIVER WAITING
ERSRR	000108	000154	SHORT RECEIVE REQUEST
ERTIN	000109	000155	TRANSMITTER INOPERATIVE
ERUNM	000110	000156	ILLEGAL USER NAME
ERLIN	000111	000157	ILLEGAL LINK #
ERDPE	000112	000160	DISK POSITIONING ERROR
ERTXT	000113	000161	MSG TEXT LONGER THAN SPEC'D
ERSTR	000114	000162	SHORT TRANSMISSION
ERHIS	000115	000163	ERROR ON HISTOGRAM INIT/DELETE
ERIRV	000116	000164	ILLEGAL RETRY VALUE
ERASS	000117	000165	ASSIGN ERROR - ALREADY YOUR DEVICE
ERPET	000118	000166	MAG TAPE REQ PAST LOGICAL END OF TAPE
ERSTS	000119	000167	STACK TOO SMALL (?TASK)

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERTMT	000120	000170	TOO MANY TASKS REQUESTED (?TASK)
ERSOC	000121	000171	SPOOLER OPEN RETRY COUNT EXCEEDED
ERACL	000122	000172	ILLEGAL ACL
ERWPB	000123	000173	?STMAP BUFFER INVALID OR WRITE PROTECTED
ERINP	000124	000174	IPC FILE NOT OPENED BY ANOTHER PROC
ERFPU	000125	000175	FPU HARDWARE NOT INSTALLED
ERPNM	000126	000176	ILLEGAL PROCESS NAME
ERPNU	000127	000177	PROCESS NAME ALREADY IN USE
ERDCT	000128	000200	DISCONNECT ERROR (MODEM CONTROLLED)
ERIPR	000129	000201	NONBLOCKING PROC REQUEST ERROR
ERSNI	000130	000202	SYSTEM NOT INSTALLED
ERLVL	000131	000203	MAX DIRECTORY TREE DEPTH EXCEEDED
ERROO	000132	000204	RELEASING OUT-OF-USE OVERLAY
ERRDL	000133	000205	RESOURCE DEADLOCK
EREO1	000134	000206	FILE IS OPEN, CAN'T EXCLUSIVE OPEN
EREO2	000135	000207	FILE IS EXCLUSIVE OPEN, CAN'T OPEN
ERIPD	000136	000210	INIT PRIVILEGE DENIED
ERMIM	000137	000211	MULTIPLE ?IMSG CALLS TO SAME DCT
ERLNK	000138	000212	ILLEGAL LINK
ERIDF	000139	000213	ILLEGAL DUMP FORMAT
ERXNA	000140	000224	EXEC NOT AVAILABLE (MOUNT, ETC.)
ERXUF	000141	000225	EXEC REQUEST FUNCTION UNKNOWN

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERESO	000142	000225	ONLY EXEC'S SONS CAN DO THAT
ERRBO	000143	000226	REFUSED BY OPERATOR
ERWMT	000144	000227	VOLUME NOT MOUNTED
ERISV	000145	000230	ILLEGAL SWITCH VALUE (>65K DECIMAL)
ERIFN	000146	000231	INPUT FILE DOES NOT EXIST
EROFN	000147	000232	OUTPUT FILE DOES NOT EXIST
ERLFN	000148	000233	LIST FILE DOES NOT EXIST
ERDFN	000149	000234	DATA FILE DOES NOT EXIST
ERGFE	000150	000235	RECURSIVE GENERIC FILE OPEN FAILURE
ERNMW	000151	000236	NO MESSAGE WAITING
ERNUD	000152	000237	USER DATA AREA DOES NOT EXIST
ERDVC	000153	000240	ILLEGAL DEVICE TYPE FROM AOSGEN
ERRST	000154	000241	AOS RESTART OF SYSTEM CALL
ERFUR	000155	000242	PROBABLY FATAL HARDWARE RUNTIME ERROR
ERCFT	000156	000243	USER COMMERCIAL STACK FAULT
ERFFT	000157	000244	USER FLOATING POINT STACK FAULT
ERUAE	000158	000245	USER DATA AREA ALREADY EXISTS
ERISO	000159	000246	ILLEGAL SCREEN-EDIT REQUEST (PMGR)
ERCPD	000162	000251	CONTROL POINT DIRECTORY MAX SIZE EXCEEDED
ERNSD	000163	000252	SYS OR BOOT DISK NOT PART OF MASTER LD

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERUSY	000164	000253	UNIVERSAL SYSTEM, YOU CAN'T DO THAT
EREAD	000165	000254	EXECUTE ACCESS DENIED
ERFIX	000166	000255	CAN'T INIT LD, RUN FIXUP ON IT
ERFAD	000167	000256	FILE ACCESS DENIED
ERDAD	000168	000257	DIRECTORY ACCESS DENIED
ERIAD	000169	000260	ATTEMPT TO DEFINE > 1 SPECIAL PROC
ERIND	000170	000261	NO SPECIAL PROCESS IS DEFINED
ERPRO	000171	000262	ATTEMPT TO ISSUE MCA REQUEST WITH
ERDIO	000172	000263	ATTEMPT TO ISSUE MCA DIRECT I/O WITH
ERLTK	000173	000264	LAST TASK WAS KILLED
ERLRF	000174	000265	RESOURCE LOAD OR RELEASE FAILURE
ERNNL	000175	000266	ZERO LENGTH FILENAME SPECIFIED
ERBOV	000176	000267	BUFFER OVERFLOW
ERNAK	000177	000270	TRANSMISSION FAILURE (NAK) COUNT
ERTOF	000178	000271	TRANSMISSION FAILURE (TIMEOUTS)
ERDIS	000179	000272	DISCONNECT OCCURRED ON SYNC LINE
EREOT	000180	000273	EOT CHARACTER RECEIVED
EROTH	000181	000274	POSSIBLE LOST DATA ON HASP
ERDCU	000182	000275	DCU INOPERATIVE (CAN'T BE INITIALIZED)
ERCNV	000183	000276	CONVERSATIONAL REPLY RECEIVED
EREPL	000184	000277	END OF POLLING LIST REACHED

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERIRT	000185	000300	ILLEGAL RELATIVE TERMINAL NUMBER
ERRVI	000186	000301	RVI RESPONSE RECEIVED
ERLIN	000187	000302	ILLEGAL LINE NUMBER
ERPLS	000188	000303	NOT ENOUGH SPACE FOR POLL LISTS
ERCTN	000189	000304	CONTENTION SITUATION WHILE BIDDING
ERSEQ	000190	000305	OUT-OF-SEQUENCE GEN ENTRY DURING SINIT
ERNSL	000191	000306	ATTEMPT TO ENABLE NON-SYNC LINE
ERIMM	000192	000307	NOT ENOUGH MEMORY FOR POLL/SELECT LIST
EREPE	000193	000310	LINE ALREADY ENABLED ON ?SEBL CALL
ERDSL	000194	000311	LINE ALREADY DISABLED ON ?SDBL CALL
ERLNA	000195	000312	I/O REQUEST FOR DISABLED LINE
ERLIS	000196	000313	LINE IN SESSION ON ?SSND INITIAL CALL
ERSCS	000197	000314	?SSND CONTINUE WITHOUT LINE IN SESSION
ERBCT	000198	000315	SEND BYTE COUNT EXCEEDS SYSTEM BUFFER
ERBNK	000199	000316	BID ERROR (TOO MANY NAKS)
ERWAB	000200	000317	WABT RECEIVED (HASP LINE ONLY)
ERBPE	000201	000320	USER BUFFER BYTE POINTER INVALID
ERBRT	000202	000321	RETRY COUNT EXCEEDED
ERETX	000203	000322	'ETX' CODE RECEIVED
ERISE	000204	000323	INPUT STATUS ERROR (FORMAT)
ERFCT	000205	000324	FAILURE TO CONNECT

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERUNI	000206	000325	UNINTERPRETABLE RESPONSE RECEIVED
ERENQ	000207	000326	ENQ RECEIVED AFTER TIME-OUT
ERCRC	000208	000327	CRC CHECK
ERINE	000209	000330	INITIALIZATION PARAMETER ERROR
ERTRF	000210	000331	TRANSMITTER FAILURE ERROR
ERLNM	000211	000332	LINE NOT MULTIPOINT
ERNCS	000212	000333	NOT A CONTROL STATION
ERNPL	000213	000334	POLLING LIST NOT DEFINED
ERITF	000214	000335	INCOMPATIBLE LPB TAB FORMAT
ERPRM	000215	000336	CANNOT DELETE PERMANENT FILE
ERSCA	000216	000337	SYSTEM CALL ABORT
ERCAD	000217	000340	EXTENDED CONTEXT ALREADY DEFINED
ERLAB	000218	000341	UNREADABLE TAPE LABEL
ERVOL	000219	000342	INCORRECT LABELED TAPE VOLUME MOUNTED
ERFSI	000220	000343	INCORRECT LABELED TAPE FILE SET
ERSEC	000221	000344	INCORRECT LABELED TAPE FILE SECTION NUMBER
ERGEN	000222	000345	INCORRECT LABELED TAPE FILE GENERATION NUMBER
ERVER	000223	000346	INCORRECT LABELED TAPE FILE VERSION NUMBER
ERNOA	000224	000347	NO OPERATOR AVAILABLE
ERREV	000225	000350	UNKNOWN LABELED TAPE LABEL REVISION
ERCAI	000226	000351	EXTENDED CONTEXT ALREADY INITIALIZED

(continued)

System Errors			
Parameter	Decimal Value	Octal Value	Message
ERCNI	000227	000352	EXTENDED CONEXT NOT INITIALIZED
ERCND	000228	000353	EXTENDED CONTEXT NOT DEFINED
ERMRL	000228	000354	MEMORY RELEASE ERROR
ERITP	000229	000355	TRANSLATION (?READ/?WRITE) ERROR
ERNAG	000230	000356	NO SUCH ARGUMENT - ?GTMES
ERNCF	000231	000357	NOT IN CLI FORMAT - ?GTMES
ERBIF	000232	000360	ILLEGAL BIAS FACTOR
ERTLM	000233	000361	CPU TIME LIMIT EXCEEDED
ERSMX	000234	000362	ERROR IN SETTING MAX CPU LIMIT
ERSMX	000235	000363	ERROR IN MAX CPU LIMIT
ERNM4	000236	000364	ELEMENT SIZE NOT A MULTIPLE OF 4
ERWAK	000237	000365	WACK RESPONSE RECEIVED (SYNC LINE)
ERNAS	000238	000366	PROCESS IS NOT A SERVER
ERCDE	000239	000367	CONNECTION DOES NOT EXIST
ERCTF	000240	000370	CONNECTION TABLE FULL
ERDIU	000241	000371	DIRECTORY IN USE - CANNOT DELETE
ERSHG	000242	000372	ATTEMPT TO GROW GHOST SHARED I/O FILE
ERNIN	000243	000373	ILLEGAL DIRECTORY SPECIFICATION
ERNNA	000244	000374	NETWORK NOT AVAILABLE
ERHAE	000245	000375	HOST ALREADY EXISTS
ERHID	000246	000376	ILLEGAL HOST SPECIFICATION
ERHNE	000247	000377	HOST DOES NOT EXIST

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERCAH	000248	000400	CAN'T RENAME HOSTS
EREMB	000249	000401	EMPTY MAILBOX ON ?RECNU
ERRRR	000250	000402	REMOTE RESOURCE REFERENCE MADE
ERCMH	000251	000402	ATTEMPT TO CREATE MULTIPLE LOCAL HOSTS
ERNAI	000252	000403	NOT AWAITING ?IWKUP
ERIRP	000253	000404	ILLEGAL REMOTE ?PROC PARAMETERS
ERIHN	000254	000405	ILLEGAL HOST NAME
ERNFC	000255	000406	NOT PROPER FOR A VIRTUAL CIRCUIT
ERWSZ	000256	000407	HDLC - INVALID WINDOW SIZE
ERFSZ	000257	000410	INVALID FRAME SIZE
ERSDA	000258	000411	SEND ACTIVE
ERCTY	000259	000412	INVALID CALL TYPE
ERDSC	000260	000413	REMOTE IS DISCONNECTING
ERRIE	000261	000414	LOCAL RECEIVED INVALID RESPONSE
ERRCE	000262	00415	LOCAL RECEIVED CMDR
ERCSE	000263	000416	LOCAL IS IN "CAN'T" SEND
ERLDC	000264	000417	LOCAL IS DISCONNECTING
ERRES	000265	000420	LOCAL WAS RESET
ERBFO	000266	000421	BUFFER OVERFLOW
ERRCA	000267	000422	RECEIVE ACTIVE
ERINF	000268	000423	INITIALIZATION FAILED
ERINC	000269	000424	LOCAL RECEIVED INVALID COMMAND
ERNHL	000270	000425	NON-HDLC ENABLE ATTEMPTED
ERKAD	000271	000426	INTERRUPT WAIT TASK ALREADY DEFINED

(continued)

System Errors			
Parameter	**Decimal Value**	**Octal Value**	**Message**
ERDSE	000272	000427	MAP SLOT ERROR
ERGBE	000273	000430	GET BUFFER ERROR
ERDIE	000274	000431	SYNC DCU INOPERATIVE
ERFOE	000275	000432	ERROR OPENING SLDCU.PR
ERFRE	000276	000433	ERROR READING SLDCU.PR
ERFCE	000277	000435	ERROR CLOSING SLDCU.PR
ERGME	000278	000436	ERROR GETTING MEMORY
ERUNK	000279	000437	UNKNOWN ERROR
ERCBK	000280	000440	CONNECTION HAS BEEN BROKEN
ERNDC	000281	000441	ATTEMPTED HDLC CALL WITH NO DCU200
ERCCS	000282	000442	CANNOT CONNECT TO SELF
ERVNC	000283	000443	NO CONNECTION
ERCDN	000284	000444	CONTROLLER DOES NOT SUPPORT THIS DENSITIY MODE
ERITD	000285	000445	INDECIPHERBLE TAPE DENSITY
ERFTM	000286	000446	FILE/TAPE MISMATCH

(concluded)

End of Appendix

Appendix B
Exceptional Condition Codes

This appendix categorizes and describes the exceptional condition codes (from ERICM through ERVSY) you receive in AC0 when any system call takes an exception return.

Certain exception codes (such as ERMPR, "system call parameter address error") are listed in several categories. Codes are listed alphabetically in each category. The categories are listed in the following order:

Channel-Related
File System
Initialization and Release
IPC
Memory
Miscellaneous
Process
System Call
Task
User Device
Synchronous Line

Exceptional conditions are defined parametrically in the user parameter file, PARU.SR. The system provides an error message file named ERMES which contains a textual description of each error code; you can read the description associated with an error code by issuing system call ?ERMSG or CLI command MES.

Channel-Related Codes

Mnemonic	Description
ERACU	Attempt to close an unopened channel.
ERCIU	Channel in use. Attempt to close a channel with shared pages in use. You must first release the shared pages; only then can you close the channel. In ?GNFN, another system call is outstanding on this channel. You can receive this error from ?GNFN only in a multitask program.
ERFNO	The channel you used in this call is not currently open.
ERICN	Attempt to use a channel number outside the legal range, 0 through 77 octal.
ERNDR	The channel specified to ?GNFN is not opened on a directory.
ERNMC	No free channels. A process cannot have more than 64 channels open at one time.

(continues)

File System Codes

Mnemonic	Description
ERACL	You specified an illegal ACL in an ?SACL call.
ERACN	File entry has no Access Control List (returned by ?GACL).
ERAWD	You attempted to ?CREATE a file without having write-access to the directory which was to contain the file.
ERBOF	You tried to set the file pointer before the beginning of the file (?IRNH/?IRNL of I/O packet set to too large a negative value).
ERBSZ	Attempted read or write of a block with an odd number of characters.
ERCPD	You issued a call which requested more disk space than is available in the control point directory.
ERDAD	Directory access is denied to you.
ERDCT	Modem was disconnected before the completion of a ?READ or ?WRITE.
ERDDE	Directory does not exist.
ERDTO	Device timed out.
EREAD	Execute access is denied to you.
EREOF	End of file. In ?GNFN, this means that there are no more directory entries.
EREO1	You attempted to open a file exclusively (?IEXO in the ?OPEN packet), yet the file was already open.
EREO2	You attempted to open a file which was already opened exclusively (?IEXO in the ?OPEN packet).
ERFAD	File access is denied to you.
ERFDE	Filename does not exist. A filename in a pathname was not found, or the user has no access to that entry. Alternatively, a process's console port number was requested, and the process has no console.
ERFIL	File read error.
ERFTL	You used a filename which was too long; 31 characters is the maximum length. In ?GNFN, your template's length exceeded the 63-character maximum.
ERICB	Not enough contiguous blocks to allocate a disk file element.
ERIFC	Illegal filename character. Legal filename characters are limited to the following: A through Z, a through z, 0 through 9, period (.), dollar sign ($), question mark (?), and underscore (SHIFT O). Illegal template characters in ?GNFN.
ERIFT	Illegal file type. You tried to create a file with an unknown system file type.
ERILB	Illegal library number. Perhaps you deleted or altered the symbol table file (.ST) associated with your program, or you did not make available one or more shared libraries which were required by your program. Ensure that required shared libraries are either present in your working directory or are listed in your process's search list.

(continued)

File System Codes (continued)

Mnemonic	Description
ERILN	Illegal MCA link number. Link number is outside the range 1-15 for a transmitter, or 0-15 for a receiver.
ERILO	You attempted to ?SOPEN a file whose element size is not a multiple of 4.
ERIOT	Illegal type of I/O, e.g., ?RDB/?WRB to a character device.
ERIRB	You supplied a user buffer as a call parameter, and the buffer was too small.
ERIRL	Illegal record length in variable record header. A nondigit was found in the 4-byte length field of a variable record header.
ERIRV	You specified an illegal retry value in ?PRNL to a ?WRB call for MCA I/O.
ERITC	I/O has been terminated by a ?CLOSE call.
ERITF	Incompatible tab format. The data channel line printer has received an unexpected tab character. The I/O request is aborted.
ERLNK	Attempt to create a link whose length exceeds 256 bytes.
ERLTL	Line too long. On a data sensitive read or write, the maximum line length was exceeded before a terminator was detected.
ERLVL	You attempted to create a directory at a tree depth which exceeds the system maximum.
ERMIL	Attempt to exceed the maximum index level, or file exceeds its maximum permissible size.
ERMPR	System call parameter address error.
ERNAD	Used a nondirectory argument in a pathname. All filenames in a pathname, except the last filename, must be filenames of directories. For magnetic tape or MCA units, the last filename may have the form *unit:n,* where *unit* is a magnetic tape or MCA unit, and *n* is a decimal number. For labeled mag tape, the last two entries must be *:volid:filid.*
ERNAE	Filename already exists. You attempted to create a directory entry with a name that is already in use.
ERNRD	Insufficient room in directory. Directories can be a maximum of 2^{11} blocks long.
ERNRR	MCA transmitter timeout because no ready receiver was found.
ERNSA	You requested shared I/O into a nonshared area. Alternatively, memory addresses and/or I/O size are not entirely within the current shared area.
EROPR	You attempted to open a channel that is already open.
ERPAR	Parity error.
ERPET	You tried to read beyond a double tape mark, the logical end of tape.
ERPRM	Permanent file delete error. The file is permanent and cannot be deleted.
ERPUF	Physical unit failure.

(continued)

File System Codes (continued)

Mnemonic	Description
ERPWL	Physical write lock. Write enable ring is missing from a magnetic tape reel and writing was attempted.
ERRAD	Read access is denied to you.
ERRFM	Illegal or unspecified record format.
ERSAL	You issued a shared I/O request, and either the memory address to be used in the transfer does not begin on a 2K-byte boundary or the number of blocks in the transfer is not a multiple of 4.
ERSIM	Simultaneous requests have been made on the same channel; another task has an active request on this channel.
ERSPC	Disk file space is exhausted, or an end-of-tape mark was detected on a write to magnetic tape.
ERSRE	Search list resolution error. Alternatively, some other file system error was received when the system attempted to resolve a directory name in the pathname.
ERSRR	MCA transmission was not completely received because the receiver requested less than the full transmission.
ERSTR	MCA transmission was shorter than requested by the receiver.
ERTIN	MCA transmitter failure detected upon an attempted read.
ERUOL	Physical unit is offline.
ERVIU	LD is in use; the attempted release cannot be performed.
ERWAD	Write access is denied to you.

Initialization and Release Codes

Mnemonic	Description
ERARC	Attempt to release the console device.
ERARG	Internal system I/O error.
ERARU	Attempt to desassign an unassigned device.
ERDAI	Device is already in use. You attempted to assign or open a device that was assigned to another process.
ERDVC	Illegal device type. AOSGEN information is inconsistent. Either device information was entered incorrectly, or it was modified since SYSGEN.
ERFIX	LD needs to be fixed (use the FIXUP utility).
ERIBS	The device you attempted to initialize is already initialized.
ERIDD	The system found inconsistent data in a system database called a Disk Information Block (DIB); your LD cannot be initialized.
ERIDT	You attempted to initialize a spooled device (e.g., @LPT).

(continued)

Initialization and Release Codes (continued)

Mnemonic	Description
ERIDU	The set of disks you tried to initialize do not form the complete LD which was specified to the DFMTR utility.
ERILD	The set of disks you tried to initialize belong to two or more different LDs.
ERMIS	The disk revision number and file system revision number do not match.
ERIPD	Initialization privilege denied. You tried to initialize an LD, but you do not have owner access to its root directory and you were not in superuser mode.
ERVIU	You tried to release an LD that is in use.
ERVNI	LD is not initialized, so it cannot be released.

IPC Codes

Mnemonic	Description
ERIDP	Illegal destination port number.
ERIOP	Illegal origin port number.
ERIVP	Invalid port number. Either the number is outside the legal range, or it is not assigned.
ERMPR	System call address error (see also System Call Codes).
ERNEF	IPC message was longer than the buffer that was to receive it.
ERNMS	No matching send request in IPC spool file, and the receiver did not specify ?IFBNK, i.e., that it should be suspended if no message was ready.
ERNOR	No outstanding receive request, and the sender does not want the message to be spooled.

Memory Codes

Mnemonic	Description
ERADR	Illegal starting address. The program file's starting address does not lie within its address space.
ERMEM	Insufficient amount of memory available. Possibly you have attempted to exceed the maximum amount of core memory which was allotted to your process when it was created.
ERNSW	Out of swap file space. Contiguous disk space for this image cannot be allocated.
EROVN	Illegal overlay number; the overlay area is not currently occupied by the specified overlay; or, the overlay number could not be found in the overlay directory.
ERRDL	Resource deadlock. (For further information, see the general procedure call ?KCALL.)

(continued)

Memory Codes (continued)

Mnemonic	Description
ERROO	Attempt to release an overlay (by ?OVREL, ?OVEX, or ?OVKIL) that was not in use.
ERSEN	The external reference you specified in a shared routine call did not exist in the system tables found in user space (you may have overwritten the area below ?USTART inadvertently).
ERSHP	Error upon setting a shared partition. The pages you specified are already in an unshared area, or are otherwise illogical.
ERSRL	Attempt to release a shared page via ?RPAGE when that page is not in use.

Miscellaneous Codes

Mnemonic	Description
ERASS	You tried to assign a device which was already assigned.
ERBSZ	Illegal block size for device.
ERCGF	Indeterminate internal system error upon either an attempted ?PROC or ?OPEN of a generic filename. If upon a ?PROC, the new process will not be created. If upon an ?OPEN, a fatal system error is indicated, perform a memory dump for system analysis.
ERDID	You tried to delete a directory containing entries of one or more inferior directories, or you tried to delete the working directory ("=").
ERDIF	Illegal dump format, a fatal system error, was detected during the initial system load. Try cleaning the tape heads and repeat the load procedure. If this fails to help, you probably have a bad system tape.
ERDIO	Attempt to issue MCA direct I/O request while device queue contains an entry.
ERESO	You attempted to issue ?EXEC from a process which was not created by EXEC.
ERFUR	Fatal user runtime error. This error indicates an internal system error detected by a routine from URT.LB. Contact your local Data General representative.
ERGES	Internal system error; process is terminated.
ERHIS	Illogical histogram packet, or attempt to start a second histogram when a first already exists.
ERIBM	Inconsistent data in block allocation map; a fatal system error. This error can occur when disk blocks are allocated or deallocated.
ERICL	Message targeted by ?GTMES has an illegal format.
ERIGM	You used an illegal parameter in a ?GTMES call.
ERISV	You attempted to pass a switch whose value exceeds 2^{16} -1.
ERLRF	Resource load or release failure.

(continued)

Miscellaneous Codes (continued)

Mnemonic	Description
ERMPR	System call parameter address error.
ERMRD	Attempt to issue ?SEND to a console which has ?CNRM set in its characteristics.
ERNAC	Attempt to issue ?SEND to a nonconsole device.
ERNSD	In response to "Specify Master Logical Disk" upon a program load, you specified a logical disk which did not contain the system disk. Specify the proper logical disk.
ERPDF	System detected an error in one or more words in the User Status Table.
ERPRO	Attempt to issue an MCO I/O request while direct MCO I/O is in progress.
ERPRV	You are not privileged to perform this action or issue this call.
ERRBO	The operator refused your ?EXEC request.
ERRMP	Internal system error; process is terminated.
ERRST	This error code is used only by the system and you should never receive it. If you do, contact your local Data General representative.
ERSOC	Internal system error; process is terminated.
ERSTO	System stack overflow (an internal system error).
ERTIM	Attempt to set the system clock to an illegal time, or the system calendar to an illegal date.
ERTXT	The actual error message length exceeds the one which was requested. This code is returned only by ?ERMSG.
ERVSY	There are several operations which you cannot perform when using the universal system (supplied on the system tape or diskette). These are described in *How to Load and Generate Your AOS System.* You attempted to perform one of these illegal operations.
ERWMT	You requested the dismounting of an already dismounted tape reel or disk.
ERXMT	Signal to address already in use. You attempted to transmit a message to a nonzero mailbox.
ERXMZ	Attempt to issue ?XMIT with an invalid message. Message must be nonzero.
ERXNA	EXEC module is not present in the system, yet you issued ?EXEC.
ERXUF	You requested an unknown function in the ?EXEC parameter packet, offset ?XRFNC.

(continued)

Process Codes

Mnemonic	Description
ERBLR	Attempt to block a resident process (?BLKPR).
ERBMX	Attempt to create a process with an illegal maximum size. The size of the created process cannot exceed the size of the caller's process.
ERCON	Console device specification error. Either the named device is not a console device, or it is a device which is currently in use by another process.
ERDFN	Upon a ?PROC, the generic DATA file you specified does not exist.
EREXC	A resident process attempted to issue ?PROC and block on its son.
ERGFE	You specified one or more generic files circularly. For example, you specified OUTPUT to be equal to LIST, and LIST equal to INPUT. Or, you specified a generic file to be set to itself (e.g., DATA to DATA).
ERIFN	Upon a ?PROC, the generic INPUT file you specified does not exist.
ERIPR	Illegal ?PROC parameter. Packet defaulted the IN, OUT, DATA or LIST generic filenames, but specified that the father was not to block its son.
EROFN	Upon a ?PROC, the generic OUTPUT file you specified does not exist.
ERLFN	Upon a ?PROC, the generic LIST file you specified does not exist.
ERMPR	System call parameter address error.
ERPDF	The system detected an error in a program's User Status Table.
ERPNM	Illegal process name (e.g., too long or uses illegal characters).
ERPNU	A process name specified in a ?PROC call is in use by another process.
ERPRH	Attempt to access a process which is not in the tree.
ERPRN	Attempt to create a process when the maximum, 64, already exist.
ERPRP	Illegal process priority. You attempted to specify a process priority greater than your own, and you were not privileged to do so.
ERPTY	Illegal process type. You tried to change a target process's type to one which is different from your own (via ?CTYPE) or you tried to create a process (?PROC) of a type different from your own, when you lack privilege ?PVTY.
ERSNM	You tried to create more processes than you are entitled to create (see ?PPCR in the ?PROC parameter packet).
ERUNM	Attempt to assign a username, other than that of the calling process, and caller lacks privilege ?PVUI.

(concluded)

End of Appendix

Appendix C
Calls to the Runtime Routines

Call	Chapter	Page	Call	Chapter	Page
AKILL	15	2	DIR	9	4
APPEND	11	3	DVDCHK	7	1
ARDY	15	3	DVF	24	11
ASSOCIATE	18	4	EBACK	23	2
ASUSP	15	4	ERDB	24	4
BACKSPACE	11	4	ERROR	23	3
CANCL	18	5	EST	20	3
CDIR	9	2	EWRB	24	5
CFILW	10	2	EXIT	23	3
CHAIN	22	3	FCHAN	22	8
CHECK	23	1	FCLOSE	11	8
CHRST	11	5	FDELAY	21	1
CHSAV	11	6	FDELETE	10	5
CHSTS	10	3	FGDAY	13	3
CLOSE	11	7	FGTIME	13	4
COMARG	22	4	FOPEN	11	9
COMINIT	22	6	FRENAME	10	6
COMTERM	22	7	FSDAY	13	5
CPART	9	3	FSEEK	11	10
CVF	24	11	FSTIME	13	6
CYCLE	18	6	FSWAP	22	9
DATE	13	2	FTASK	14	2
DCVF	24	11	GCIN	12	1
DFILW	10	4	GCOUT	12	2

(continues)

Call	Chapter	Page	Call	Chapter	Page
GDIR	9	5	OVCLOSE	20	4
GETERR	23	4	OVERFL	7	2
GETEV	16	1	OVEXIT	20	5
GETPRI	16	2	OVKILL	20	6
GHRZ	21	2	OVLOD	20	7
IAND	8	2	OVOPN	20	8
ICLR	8	3	OVREL	20	9
INIT	9	6	PRI	15	5
IOR	8	4	RDBLK	11	12
ISET	8	5	RDLIN	11	13
ISHIFT	8	6	RDSEQ	11	14
ITASK	14	3	READRW	11	15
ITEST	8	7	REC	17	2
IVF	24	11	RELEASE	9	7
IXOR	8	8	REMAP	24	8
KILL	15	4	RENAME	10	8
LINK	10	7	RESET	11	16
MAPDF	24	6	REWIND	11	17
MESSAGE	23	5	SDATE	13	7
MULTITASK	19	2	SINGLETASK	19	2
MYEV	16	2	START	18	7
MYID	16	3	STIME	13	8
MYPRI	16	3	SUSP	15	5
NOT	8	9	SWAP	22	10
ODIS	12	2	TIDK	15	6
OEBL	12	3	TIDP	15	7
OPEN	11	11	TIDR	15	8

(continued)

Call	Chapter	Page	Call	Chapter	Page
TIDS	15	9	VOPEN	24	17
TIME	13	9	VSTASH	24	18
TRNON	18	8	WAIT	21	3
UNLINK	10	9	WRBLK	11	18
VCLOSE	24	10	WRITRW	11	19
VDUMP	24	11	WRLIN	11	20
VF	24	11	WRSEQ	11	21
VFETCH	24	11	XMT	17	3
VLOAD	24	15	XMTW	17	4
VMEM	24	16			

(concluded)

End of Appendix

Appendix D
Alphabetized List of FORTRAN 5 Statements

Statement	Function
ACCEPT	Allows input/output of data from input console upon prompt.
ANTICIPATE	Associates an event with a task in order to register a WAKEUP on the event that occurs before the WAIT or SUSPEND task.
ASSIGN	Associates a statement label with an integer variable.
Assignment, arithmetic	Assigns the value of an expression to a specified entity.
Assignment, logical	Assigns the value of an expression to a specified logical entity.
BACKSPACE	Backspaces a file's record pointer and positions it to the beginning of the previous record.
BLOCK DATA	Assigns values to variables and arrays in both named and blank COMMON blocks.
CALL	Invokes a subroutine, transferring control from one program unit to another.
CLOSE	Closes an opened file.
COMMON	Allocates an area of data storage accessible to multiple program units, and names the variables and arrays which will reside in this area.
COMPILER	Permits you to specify the compile-time options STATIC, FREE, and DOUBLE PRECISION.
COMPLEX	Specifies a symbolic name to have the data type COMPLEX.
CONTINUE	Provides a place for a label.
DATA	Defines initial values for variables and array elements.
DECODE	Performs data transfers according to a format specification.
DELETE	Deletes a file from disk.
DIMENSION	Names arrays and specifies their dimensions.
DO	Executes a group of statements one or more times.
DOUBLE PRECISION	Specifies a symbolic name to have the data type DOUBLE PRECISION.
DOUBLE PRECISION COMPLEX	Specifies a symbolic name to have the data type DOUBLE PRECISION COMPLEX.

(continues)

Statement	Function
ENCODE	Performs data transfers strictly between variables or arrays internal to your program according to a format specification.
END	Marks the end of a program unit.
ENDFILE	Closes an opened file.
EQUIVALENCE	Associates two or more entities in the same storage area.
EXTERNAL	Allows you to use an externally defined subprogram name or overlay name as an argument.
FORMAT	Designates the structure of the records and the form of the data fields within the records of a file.
FUNCTION	Begins and defines a function subprogram.
GO TO, assigned	Transfers control to a previously ASSIGNed statement label.
GO TO, computed	Transfers control to one of several specified statements depending on the value of a specified variable.
GO TO, unconditional	Transfers control unconditionally to a specified statement.
IF, arithmetic	Transfers control conditionally to one of three statements based on the value of an arithmetic expression.
IF, logical	Conditionally executes and transfers control to a statement based on the value of a logical expression.
IMPLICIT	Changes or confirms the default data type of symbolic names.
INCLUDE	Allows you to insert a FORTRAN 5 source file in the current FORTRAN 5 program.
INTEGER	Specifies a symbolic name to have the data type INTEGER.
KILL	Terminates a task.
LOGICAL	Specifies a symbolic name to have the data type LOGICAL.
OPEN	Assigns a unit number to a file and creates the file, if necessary, according to specifications given.
OVERLAY	Specifies a subprogram as an overlay and names it.
PARAMETER	Assigns a symbolic name to a constant or to an expression.
PAUSE	Temporarily suspends program execution, waiting for operator intervention.
PRINT	Transfers data between internal storage and the line printer.
PUNCH	Transfers data between internal storage and the paper tape punch.
READ	Transfers data from a file to internal storage according to specifications in the corresponding FORMAT statement.
READ, simple	Transfers data between internal storage and the card reader.

(continued)

Statement	Function
READ BINARY	Transfers a single data record from a file to internal storage with no interpretation.
READ FREE	Transfers and converts externally recognizable data to their internal computer representation, providing a standard formatting without programmer intervention.
READ INPUT TAPE	Alternate form of READ.
READ TAPE	Alternate form of READ BINARY.
REAL	Specifies a symbolic name to have the data type REAL.
RENAME	Changes the name assigned to an existing file.
RETURN	Marks the logical end of a function or subprogram and returns control from that subprogram to the calling program unit.
REWIND	Repositions the record pointer to the beginning of a specified file.
STATIC	Places specified variables and arrays in a fixed area in memory rather than on the runtime stack.
STOP	Causes unconditional termination of program execution.
SUBROUTINE	Begins and defines a subroutine.
SUSPEND	Allows a task to suspend itself or another task.
TASK	Initiates a task.
TYPE	Allows interaction between you and your program using the console for output.
WAIT	Allows a task to suspend itself.
WAKEUP	Readies a task suspended by a WAIT or SUSPEND statement.
WRITE	Writes data from internal storage to a file or device specified by a unit number.
WRITE BINARY	Transfers a single data record from internal storage to a file with no interpretation.
WRITE FREE	Transfers and converts data according to a standard field format.
WRITE OUTPUT TAPE	Alternate form of WRITE.
WRITE TAPE	Alternate form of WRITE BINARY.

(concluded)

End of Appendix

Appendix E
Fortran 5 Runtime Databases

This appendix describes how the AOS resource manager implements the load-on-call overlay facility that FORTRAN 5 supports. It describes in detail the layout of the data areas AOS and the Fortran 5 runtime environment routines use to manage the user's program and its I/O operations. Specifically, it describes the relationships between the User Status Table (UST), the Task Control Blocks (TCBs), the TCB Extension, the Stack Partition, the Task Global Area, the Overlay Directory, the Bookkeeping Area, the File Table, and the I/O Control Block (IOCB).

This appendix describes the layout of each of these areas in more detail than Chapters 3 and 5, and depicts some of the relationships diagrammatically. You can use this information to debug your programs and examine AOS break files. The information may also be of use to you in understanding the Fortran 5 and AOS parameter files.

Please be aware that the information in this appendix is only intended to give a general description of these data areas, not depict their exact layout. This information may change between revisions of Fortran 5 or AOS.

Before you read this appendix, please read Chapter 3, "Runtime Environment Fundamentals" and Chapter 5, "The Fortran 5 Assembly Language Interface".

Runtime Environment Data Areas

The following sections describe the internal layout of important databases that AOS and the FORTRAN 5 runtime environment maintain in the program. These descriptions are specific to ECLIPSE AOS, although we mention differences between AOS and AOS/VS.

User Status Table

The UST contains information on the state of the process, including the number of tasks the process contains. The UST begins at location 400_8 in the user address space. Offsets within the UST are defined in PARU.SR in AOS and PARU.16.SR in AOS/VS. The symbols all have names beginning with "UST" (e.g., "USTEZ").

Offset USTTC (location 413_8 in AOS) contains the number of Task Control Blocks (TCBs) that Link allocated for the program. This is the largest number of tasks that may exist simultaneously. In AOS/VS, the TCB's are located in the Agent ring (ring 3), outside of the user's ring of execution (ring 7).

The following offsets apply only to AOS:

Offset	Location	Contents
USTCT	414_8	The address of the TCB for the currently active task.
USTAC	415_8	The address of the first TCB in a linked list of TCBs for active tasks (the active TCB chain).
USTFC	416_8	The address of the start of the available TCB chain. TCBs with no task associated with them are linked on this chain and are ready for use.

The first word in each TCB (offset 0) is a link address to the next TCB chain. A minus one (177777_8) in this TCB offset indicates the end of the chain.

Offset USTOD (location 420_8) contains the address of the overlay directory, which is described later in this appendix.

Task Control Blocks

AOS assigns a TCB to each task when the task is created. Single-task programs have only one TCB. Multitask programs have a number of TCBs allocated to them. Link's /TASKS= function switch specifies this number. The TCB stores information about the state of the task. AOS saves the contents of the accumulators (ACs), program counter (PC), and stack control locations (.SP, .FP, .SSE, .SOV) in the TCB for the task whenever that task is not executing.

Under AOS, the TCBs are allocated in the user address space, above the UST. Under AOS/VS, the TCBs are allocated in the Agent ring (ring 3). Offsets within the TCB are defined in PARU.SR in AOS and PARS.SR in AOS/VS. These offsets are symbols that begin with ?T (?LINK).

The following information applies to AOS only. The first word in each TCB, offset ?TLNK , contains the address of the next TCB in either the active TCB chain (for active TCBs) or the free TCB chain (for available TCBs). A minus one (177777_8) indicates the end of a chain. Offset ?TSTAT (offset 1) contains 16 status bits. These status bits are also defined in PARU.SR, and have symbol names beginning with ?TS ; e.g., "?TSPN". Each of the status bits in word ?TSTAT specifies a different state for the task. For example, if bit "?TSSP" is set (is one) that task has suspended itself by execution of the ?SUS (Suspend the calling task) system call. Bit ?TSIG indicates whether the task is presently executing in the Ghost context (bit 1) or executing in the primary (user) context (bit 0). Bit ?TSUF is set by the Fortran 5 runtime environment routines when a task executes the Fortran 5 WAIT or SUSPEND statements. Bit ?TSXR is set when the task is waiting as the result of a ?XMTW (Transmit a message and wait) or ?REC (Receive a message) system call. TCB offsets ?TSP, ?TFP, ?TSL, and ?TSO (offsets 1 through 5) contain the contents of locations .SP, .FP, .SSE, and .SOV when the task is not executing. Offsets ?TAC0, ?TAC1, ?TAC2, ?TAC3 and ?TPC (offsets 6_8 through 12_8) contain the contents of the task's ACs and PC when the task is not executing. Offset ?TELN (offset 14_8) contains the address of the task's TCB extension, described later in the appendix. Offset ?TFPS (offset 15_8 contains the address of an 18-word floating point save area that maintaines the state of the floating point ACs, PC, and Status when the task is not executing. Offset ?TIDPR (offset 20_8) contains the task's ID number in the left byte, and the task's priority in the right byte.

Figure E-1 depicts the relationship between the UST, TCB and TCB Extensions in ECLIPSE under AOS.

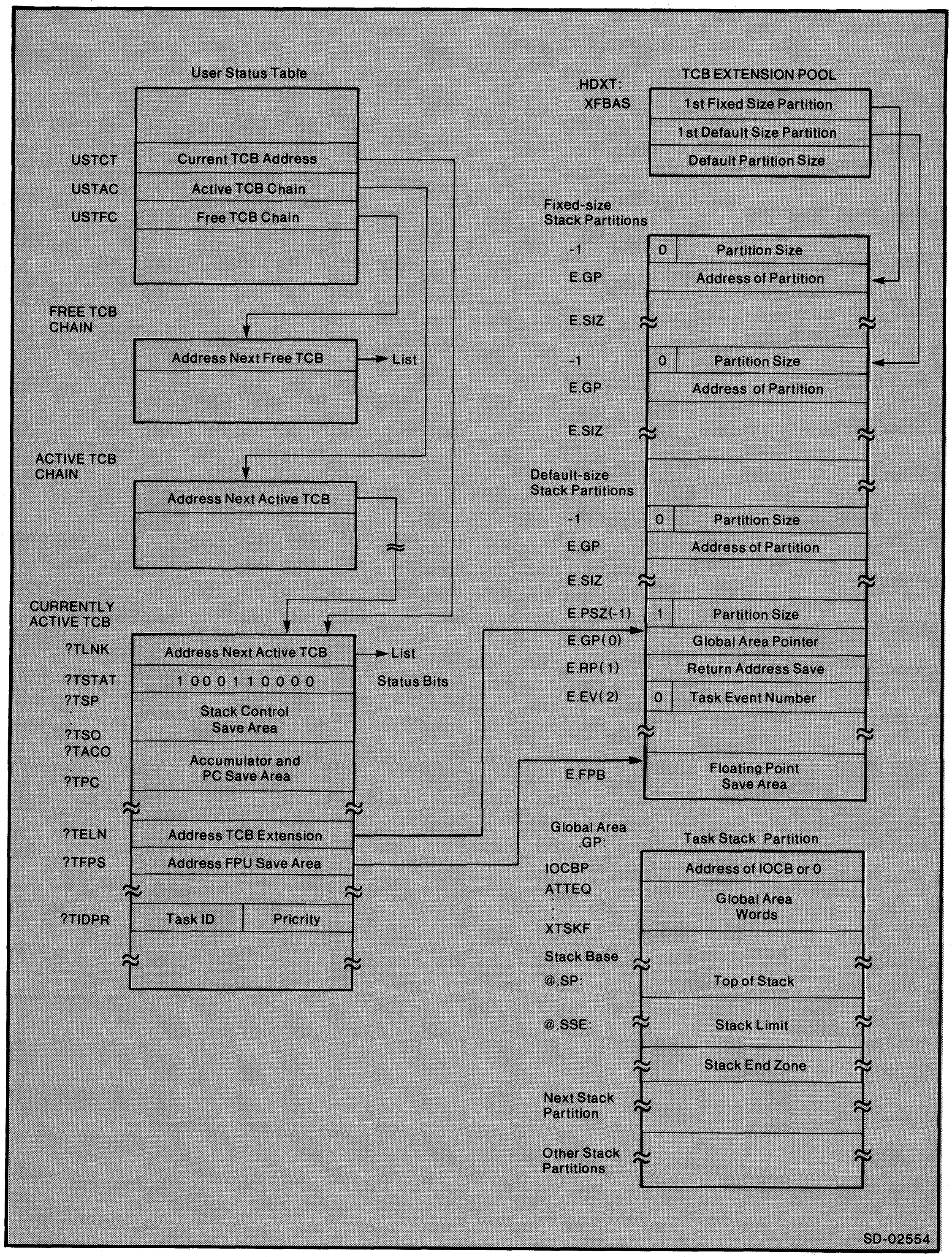

Figure E-1. The Relationship Between UST, TCB and TCB Extensions

Task Control Block Extensions

The Fortran 5 runtime environment routines maintain additional information about a task in an area known as the TCB extension. Offsets within the TCB extension are defined in F5SYM.SR. The offset symbols have names beginning with "E."; e.g., "E.PSZ". Offset E.PSZ (offset -1) contains the size of a runtime stack partition associated with that TCB extension. Offset E.GP contains the address of a Task Global Area associated with that TCB extension. When the task is executing, page zero symbol (.GP) contains the address of this global area. Since the global area is at the bottom addresses of the stack partition, E.GP represents not only the address of the task global area, but also the stack partition which contains the global area.

Each TCB extension is associated with a runtime stack partition. The runtime stack partition contains both the task global area and the runtime stack that the task owning the TCB will use. When a task is initiated by the program, the Fortran 5 runtime environment routines will allocate a stack partition for the new task. If the calling program specifies a stack size for the task, the runtime routines must find a a stack of that exact size. If the calling program does not specify a stack size, the runtime routines allocate a default size partition.

A 3-word data area called the TCB extension header contains the addresses of the fixed-size partition list and the default size partition list. The third word in the header is the size of a default size partition stack. The Fortran 5 runtime initializer creates the TCB extension header and the TCB extension pool. All of the fixed size stack partition extensions are allocated together by the runtime initializer, followed by all of the default size partition extensions. Each TCB extension contains a word that specifies the size of the stack in the stack partition associated with that TCB extension.

Offset E.PSZ (offset -1) within each TCB extension contains the size of the runtime stack in the stack partition that is associated with that TCB extension. Offset E.GP (offset 0) within the TCB extension contains the base address of the associated stack partition. E.GP contains the value of .GP when the task is not executing. Offset E.RP (offset 1) contains the value of the page-zero symbol .RP when the task is not executing. The FORTRAN 5 runtime environment routines use .RP to maintain return addresses. Offset E.EV is used to maintain Event Numbers for the Fortran 5 event number suspension mechanism (the ANTICIPATE, WAIT, and WAKEUP statements).

The leftmost bit of E.EV is used to indicate that a wakeup for an anticipated event occurred before the corresponding wakeup. The remainder of E.EV contains the event number for which the task anticipates or waits.

Offset E.FPB is the first word of the 18-word floating point save area in AOS. This area contains the contents of the floating point accumulators (FPACs), the floating point program counter (FPPC), and the floating point status register (FPSR) when the task is not executing. No floating point save area is required within the user address space under AOS/VS.

When the runtime environment allocates a TCB extension and stack partition for a new task, it obtains a pointer to either the fixed-size partition extensions or the default-size partition extensions from the TCB extension header (the address of the TCB extension header is .HDXT). The runtime environment routine that allocates stack partitions then looks at the partition size of the appropriate list of TCB extensions (either for fixed-size stacks or default-size stacks) until either a match is found or a zero-size stack partition (indicating an end of list) is found.

The leftmost (high-order) bit of the size word (offset E.PSZ) indicates that the corresponding stack partition was previously allocated. This *use-bit* is set (1) to indicate that the stack partition is in use, or cleared (0) to indicate that the partition is available for use. When the runtime environment routine finds an unused stack partition of the proper size, it sets the use-bit for that partition.

Under AOS, the runtime environment routines then place the address of the TCB extension for that partition into the TCB extension offset (?TELN) of the TCB for the new task. Under AOS/VS, the address of the TCB extension is maintained in location 16_8 (?USP), which AOS/VS maintains on a per-task basis. The runtime environment routines set the leftmost (high-order) bit to one to distinguish FORTRAN 5 tasks from non-FORTRAN 5 tasks. A non-FORTRAN 5 task must never set the high-order bit of ?USP to one. When a task is terminated, the runtime environment routines clear the use-bit of the TCB extension associated with the terminated task.

Stack Partition

The stack partition allocated to a task by Fortran 5 when the task begins execution consists of three parts:

- A Task Global Area
- A Runtime Stack
- An End Zone

The runtime stack is the stack that the task will use while it executes. The stack control locations (.SP, .FP, .SSE) all point into this runtime stack. These stack control locations are maintained in the TCB when the task is not executing.

The stack must contain a small number of words beyond the stack limit because the stack fault mechanism of the ECLIPSE[R] pushes a return block onto the stack, and the FORTRAN 5 error reporter pushes several words onto the stack. These words are called the End Zone, and insure that a stack overflow in one task will not destroy information in the stack partition immediately following the stack limit of the faulting task.

Task Global Area

Certain per-task information used by FORTRAN 5 is maintained in a data area at the bottom of the stack partition known as the Task Global Area. Page-zero symbol (.GP) points at this area. Offsets within the Global Area are defined in F5SYM.SR. Offset IOCBP (offset 0) in the Global Area contains the address of an I/O Control Block (IOCB) that the FORTRAN 5 I/O routines use.

The remainder of the words in the Global Area act as information transfer buffers between the time the options of an OPEN statement or TASK statement are processed and an IOCB or ?TASK packet is created. For example, the "ERR=" option of the OPEN statement is processed by the generated code before the I/O initializer is called to allocated an IOCB. During this time, the Task Global Area is used to maintain the ERR= branch address and the stack pointer and frame pointer at the time of the OPEN. The proper environment can then be restored after an error. Likewise, the task ID specified by the ID= option of the task statement must be maintained until a ?TASK packet is created by the Fortran 5 runtime environment routines. Again, the Global Area is used for this purpose.

Offset ATTEQ (offset 1) contains 16 bit flags that represent the possible values ATT option that the OPEN statement specifies. The symbols that specify these bit attributes are also defined in F5SYM.SR. These symbols have names beginning with FA ; e.g., FALIN . Offsets LENEQ (offset 3) through XTSKF (offset 12_8) consist of temporary storage for information passed between the I/O and tasking programmed operator routines and the FORTRAN 5 runtime environment routines. Offset LASTE contains the last runtime error detected (without the ISA offset of 3), for use by the GETERR runtime routine.

Input-Output Control Block

An IOCB is a large data area that the FORTRAN 5 I/O routines use during data transfer operations. An IOCB is allocated by the I/O initialization routine executed by the generated code at the beginning of a READ or WRITE statement. The IOCB consists of three primary areas:

- A runtime database of conversion information
- A data buffer
- An I/O packet

The offsets within the IOCB are defined in F5SYM.SR. Space for the IOCB is allocated on the runtime stack.

The IOCB allocation routine alters the current stack pointer to leave room for an IOCB between the current stack frame and the stack pointer. When the IOCB is removed at the end of an I/O statement, the stack pointer is restored to its previous value. The size of an IOCB depends on the size of the buffer required for the I/O statement. The buffer size depends on the length specified by the "LEN=" option for the OPEN statement. By default, the buffer is large enough for a 136-byte record.

Bookkeeping Area

If routines in your program are compiled for line number traceback at the start of every executable statement, the generated code stores the current line number in an extra word allocated in each stack frame. This extra word is called the Bookkeeping Area, and is the last word in the stack frame. If line number traceback is not used, this word is available.

Overlay Directory

In order to support the load-on-call overlay facility of the AOS resource manager, Link builds a database in the address space of every program that contains overlays. This database, called the Overlay Directory, is used by the ?RCALL processing code in URT.LB. The layout of this directory is described in an appendix of the *AOS Programmer's Manual* (093-000120). The Overlay Directory address is stored in offset USTOD (location 420_8) in the UST.

File Table

The Fortran 5 runtime environment routines maintain information about open files in a data area called the file table. This area consists of 3 major parts:

- A record length table
- A file lock table
- A channel table

The symbol .FT contains the address of the file table. Each entry in the file table consists of a single word, indexed by unit number, that contains 3 bits of flag information and 13 bits of record length. The three bits, FALIN, FAPRT, and FPBPD indicate whether or not the file is line-oriented, prepared-for-printing (i.e. recognizes ANSI carriage control) or blank-padded, respectively. The remaining 13 bits of the word contain the record length for the file (136 by default).

The file lock table consists of 64 bits of information (1 per unit number). Each bit in the lock table indicates that the corresponding unit number has been opened. The bit is set (1) if the unit has been opened, or is clear (0) if it has not been opened.

The third portion of the file table, the channel table contains the AOS channel number that is associated with the Fortran 5 unit number once the file is opened. Entries for unopened units are set to minus one (177777_8).

In addition to the file information mentioned above, the file table contains several additional data words that may conveniently be accessed relative to the file table. These additional words contain conversion constants for time units, an argument number holder for the COMARG routine and a holder for the overlay file channel number. Figure E-2 depicts the layout of the file table.

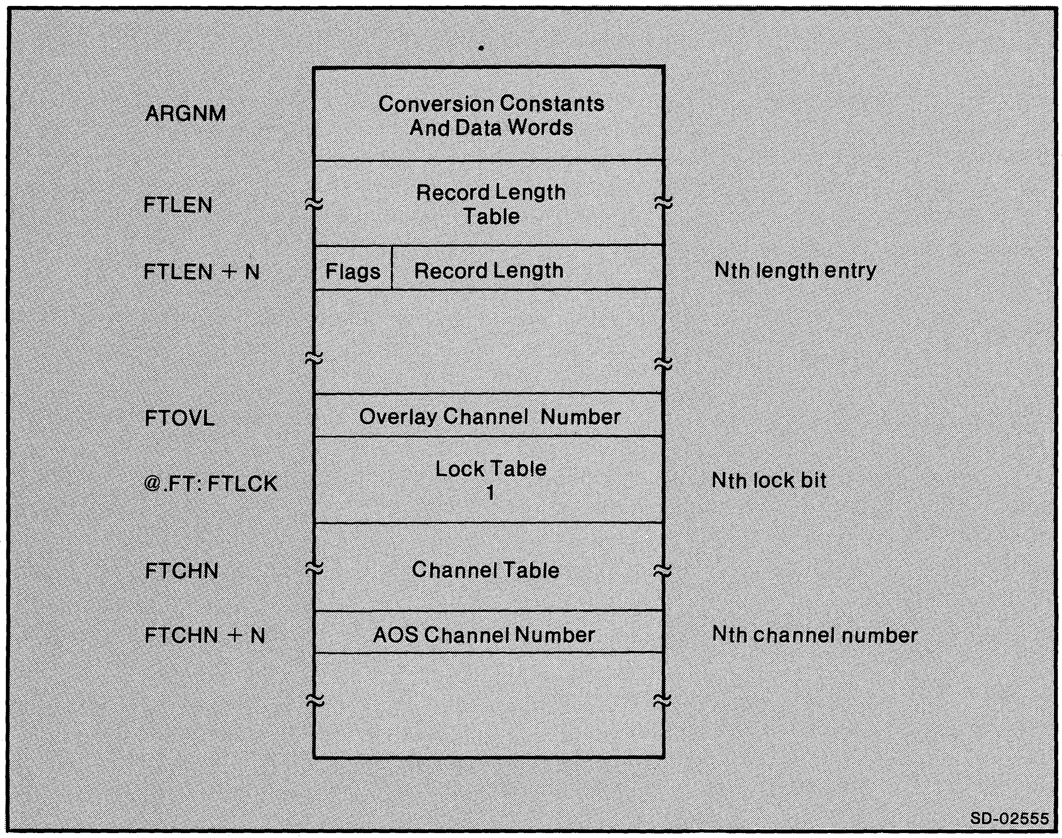

Figure E-2. File Table

Preconnection Table

Preconnected I/O in FORTRAN 5 requires a table of unit number and filename associations. The preconnection table (PCT) consists of 3 parts:

- Error file descriptors
- A statement association table
- A preconnected association table

The error file descriptors are actually ?WRITE system call packets. When a runtime error is reported, each error file descriptor is used to write all error message output. Offset EFPTR (offset -7) contains the offset of the first error file descriptor. These desriptors are contiguously allocated and have a length of EFLEN. The list terminates when the first word of a descriptor is minus one (177777_8). Offset EFBUF (offset -6) contains the address of an output buffer used for writing the error messages.

The Statement Association Table provides the mapping between various I/O statements that provide no unit number, and the unit number to which I/O should be performed. Offsets FNREA (offset -5) through FNACC (offset -1) contain the unit numbers for the various I/O statements; e.g., the PUNCH statement code refers to offset FNPUN.

The Preconnection Association Table contains the mapping between the unit numbers and the file names. This determines which unit numbers should be opened by preconnection. Each preconnection entry is four words long, and contains a unit number, a pointer to a file name, a pointer to an attribute string, and a record length.

Figure E-3 depicts the preconnection table. The first preconnection entry is used as a default. If no explicit entry for a given unit number is found in the remaining entries, this first entry is examined. If the unit number of the default entry is not minus one (177777_8), then the default preconnected file is opened. If the unit number is minus one, then no default preconnection exists, and no open is performed. The preconnection entries are terminated by a null entry containing minus one (177777_8) as the unit number.

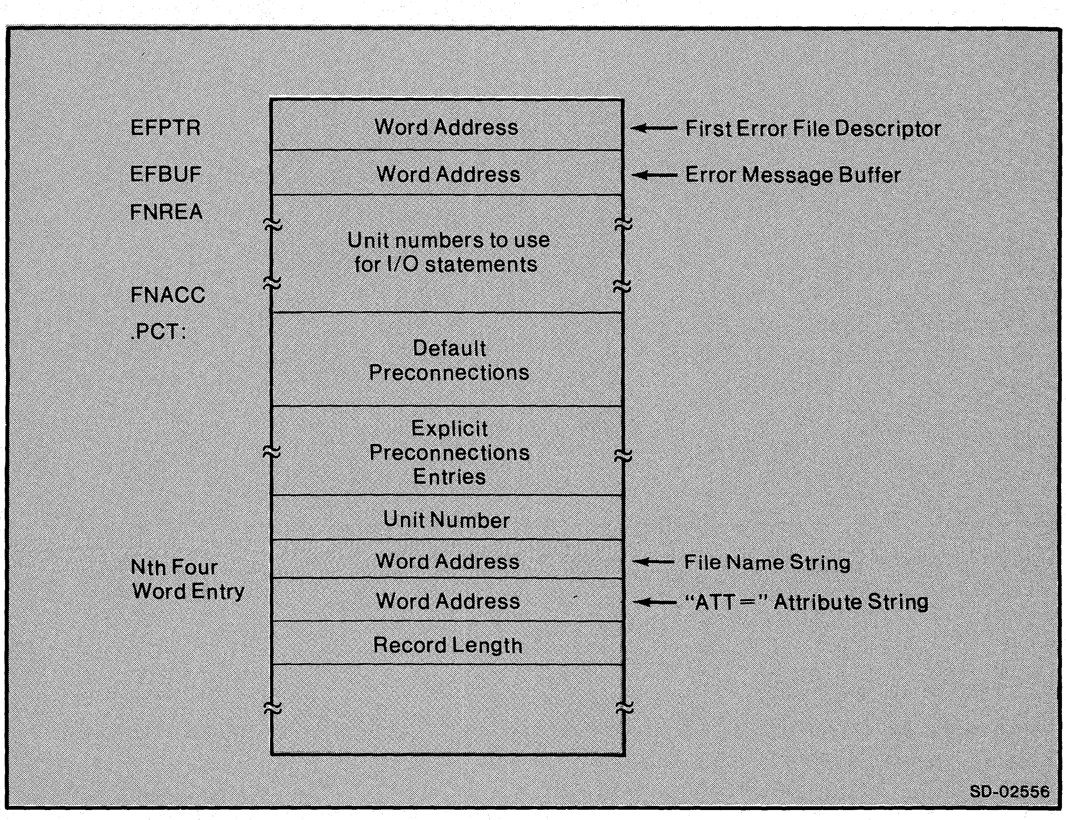

Figure E-3. The Preconnection Table

End of Appendix

Appendix F
CLRE Math Routines

The internal names of the CLRE math routines are derived from the following template:

<BASICNAME><VERSION><ENTRY>?<RESDT><ARGDT>

< BASICNAME >

A three character name assigned from the following list:

TRN	TRUNCATION
CEL	CEILING
FLR	FLOOR
FRC	FRACTIONAL PART
NIN	NEAREST WHOLE #
ABS	ABSOLUTE VALUE
REM	REMAINDER
MOD	MODULO (ANSI)
SXN	SIGN TRANSFER
PDF	POSITIVE DIFFER
MAX	MAXIMUM
MIN	MINUMUM
IMG	IMAGINARY PART
REL	REAL PART
CJG	COMPLEX CONJUGATE
SQR	SQUARE ROOT
EXP	EXPONENTIAL
LGN	NATURAL LOG
LGD	COMMON LOG 1
LGE	LOG BASE 2
SIN	SINE
COS	COSINE
TAN	TANGENT
ASN	ARCSINE
ACS	ARCTANGENT
ATT	ARCTANGENT2
HSN	HYPERBOLIC SINE
HCS	HYPERBOLIC COSINE
HTN	HYPERBOLIC TAN
AND	LOGICAL AND
IOR	LOGICAL OR
XOR	LOGICAL XOR
NOT	LOGICAL NOT
PWR	POWER
NEG	NEGATION
SGN	MULTIPLE OF SIGNS
CVT	TYPE CONVERSION

< VERSION >

A one digit specifier for the version of the routine with the following meaning:

0 This version does not use floating point hardware and does not check its arguments.

1 This version does not use floating point hardware and does check arguments.

2 This version uses floating point hardware and does not check arguments.

3 This version uses floating point hardware and does check arguments.

The FORTRAN math library does not contain all versions of all routines.

To check an argument, the system tests whether the argument is within range of a given routine. Those routines that do not check will produce undefined results for arguments out of range.

< Entry >

A one digit entry descriptor with the following meaning:

1 This is the entry point for the actual code for this routine.

2 This is the entry point for the page zero entry to the routine.

3 This is the entry point for the ?RCALL version of this routine.

4 This is the entry point used when the specified routine is passed as an argument in a subroutine call and then is called by that subroutine.

The NREL and ZREL parts of an intrinsic function are now separated into two separate binaries. Because of this, you can invoke an intrinsic function without using any ZREL locations (by means of a EJSR). The ZREL entry point routine consists of a single ZREL location that contains the address of the actual address of the code for the routine.

< ARGDT >

A one digit specifier of the result and the argument, respectively, from the following list:

1 16-bit integer
3 real
4 double precision
5 complex
6 double precision complex

If the data types of the argument and the result are identical, use only one digit.

For power routines, the two digits represent base and exponent respectively.

Examples:

SIN21?3 is the name of the entry point for the single precision SIN routine.

SIN22?3 is the name of the ZREL entry point for the above routine. A call to the SIN routine could take these forms:

```
JSR    @SIN22?3

?RCALL SIN23?3

EJSR  SIN21?3
```

< RESDT >

See description for <ARGDT>

Calling Sequence

The way in which the calling routine passes the arguments to and from the routines depends upon the data types of the following entities.

Integer

Single Argument	The calling routine passes the address of the argument in AC1 and returns the result in AC1.
Two Arguments	The calling routine passes the address of the first argument in AC1, the second argument in AC2, and returns the result in AC1.
> Two Arguments	The calling routine pushes the addresses of all arguments and the result onto the stack.

Real and Double Precision

Single Argument	The calling routine passes the address of the argument in the FPAC (FPAC0 on the ECLIPSE), and returns the result in the FPAC.
Two Arguments	The calling routine passes the address of the first argument in the FPAC, the second argument in AC2, and returns the result in the FPAC.
> Two Arguments	The calling routine passes the addresses of all arguments and the result onto the stack.

The MAX and MIN functions are treated as > 2 arguments.

The conversion functions, that have different argument and result data types, pass parameters according to the above specifications for each parameter. Therefore, the conversion from REAL to INTEGER passes the argument in the FPAC and the result is returned in AC1.

Complex

The calling routine passes the addresses of all arguments onto the stack.

End of Appendix

Appendix G
ASCII Table

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.

LEGEND:

Character code in decimal
EBCDIC equivalent hexadecimal code
Character

	10_	
0	64 / 7C	@

OCTAL	00_			01_			02_			03_			04_			05_			06_			07_		
0	0	00	NUL	8	16	BS (BACK-SPACE)	16	10	DLE ↑P	24	18	CAN ↑X	32	40	SPACE	40	4D	(	48	F0	0	56	F8	8
1	1	01	SOH ↑A	9	05	HT (TAB)	17	11	DC1 ↑Q	25	19	EM ↑Y	33	5A	!	41	5D	)	49	F1	1	57	F9	9
2	2	02	STX ↑B	10	15	NL (NEW LINE)	18	12	DC2 ↑R	26	3F	SUB ↑Z	34	7F	" (QUOTE)	42	5C	*	50	F2	2	58	7A	:
3	3	03	ETX ↑C	11	0B	VT (VERT. TAB)	19	13	DC3 ↑S	27	27	ESC (ESCAPE)	35	7B	#	43	4E	+	51	F3	3	59	5E	;
4	4	37	EOT ↑D	12	0C	FF (FORM FEED)	20	3C	DC4 ↑T	28	1C	FS ↑\	36	5B	$	44	6B	, (COMMA)	52	F4	4	60	4C	<
5	5	2D	ENQ ↑E	13	0D	RT (RETURN)	21	3D	NAK ↑U	29	1D	GS ↑]	37	6C	%	45	60	-	53	F5	5	61	7E	=
6	6	2E	ACK ↑F	14	0E	SO ↑N	22	32	SYN ↑V	30	1E	RS ↑↑	38	50	&	46	4B	. (PERIOD)	54	F6	6	62	6E	>
7	7	2F	BEL ↑G	15	0F	SI ↑O	23	26	ETB ↑W	31	1F	US ↑←	39	7D	' (APOS)	47	61	/	55	F7	7	63	6F	?

OCTAL	10_			11_			12_			13_			14_			15_			16_			17_		
0	64	7C	@	72	C8	H	80	D7	P	88	E7	X	96	79	` (GRAVE)	104	88	h	112	97	p	120	A7	x
1	65	C1	A	73	C9	I	81	D8	Q	89	E8	Y	97	81	a	105	89	i	113	98	q	121	A8	y
2	66	C2	B	74	D1	J	82	D9	R	90	E9	Z	98	82	b	106	91	j	114	99	r	122	A9	z
3	67	C3	C	75	D2	K	83	E2	S	91	8D	[	99	83	c	107	92	k	115	A2	s	123	C0	{
4	68	C4	D	76	D3	L	84	E3	T	92	E0	\	100	84	d	108	93	l	116	A3	t	124	4F	|
5	69	C5	E	77	D4	M	85	E4	U	93	9D	]	101	85	e	109	94	m	117	A4	u	125	D0	}
6	70	C6	F	78	D5	N	86	E5	V	94	5F	↑ or ^	102	86	f	110	95	n	118	A5	v	126	A1	~ (TILDE)
7	71	C7	G	79	D6	O	87	E6	W	95	6D	← or _	103	87	g	111	96	o	119	A6	w	127	07	DEL (RUBOUT)

SD-00217 Character code in octal at top and left of charts.

↑ means CONTROL

End of Appendix

Appendix H
Entry points for FORTRAN 5 Runtime Environment Routines

Link loads runtime routines from the FORTRAN 5 runtime libraries to perform actions specified by the source code in your program. Link also loads runtime routines for each program in order to support the program's runtime environment.

In Table H-1, we list the entry points of the majority of these runtime environment routines. This list will assist you in debugging your FORTRAN 5 programs and in computing the size of your program's runtime code. You can also use it when you write assembly language routines.

In the right-hand column is a code number or letter which refers to a note at the end of the Appendix. These notes indicate the type of the calling convention you use for the routine. The calling conventions may change between revisions of FORTRAN 5. Entry points are defined as a page zero (.ZREL) entry except those listed as .NREL. (See the "Notes" section of this Appendix.)

Table H-1. Runtime Routine Entry Points

Entry Point Name	Description	Calling Sequence
.ANTI	Perform the ANTICIPATE statement	1
.BITR	1-bit FLD reference (RHS of assignment)	2
.BITW	1-Bit FLD assignment (LHS of assignment)	2
.BKSP	Perform the BACKSPACE statement	2
.BRDC	Perform a binary read of a complex entity	3
.BRDD	Perform a binary read of a double precision entity	3
.BRDI	Perform a binary read of an integer entity	3
.BRDL	Perform a binary read of a logical entity	3
.BRDR	Perform a binary read of a real entity	3
.BRDX	Perform a binary read of a double precision complex entity	3
.BWRC	Perform a binary write of a complex entity	3
.BWRD	Perform a binary write of a double precision entity	3
.BWRI	Perform a binary write of an integer entity	3

(continues)

Table H-1. Runtime Routine Entry Points

Entry Point Name	Description	Calling Sequence
.BWRL	Perform a binary write of a logical entity	3
.BWRR	Perform a binary write of a real entity	3
.BWRS	Perform a binary write of a character string	3
.BWRX	Perform a binary write of a double precision complex entity	3
.BYTR	BYTE function reference (RHS of assignment)	2
.BYTW	BYTE function assignment (LHS of assignment)	2
.CACC	Accept an input line up to a NEWLINE	0
.CGO	Perform the computed GO TO statement	4
.CNMD	Write a decimal number to the error files	4
.CNMO	Write an octal number to the error files	4
.CRLF	Write a NEWLINE to the error files	0
.CVB	Internal conversion of ASCII characters to binary	5
.CVD	Internal conversion of binary to ASCII characters	5
.CWCH	Write a single character to the error files	6
.CWRL	Write a null-delimited string to the error files	7
.ERET	Internal invocation of the runtime error reporter	6
.F5INIT	Entry point for the FORTRAN 5 runtime initializer	A
.F5PC	Entry point for task initialization code	A
.F5PX	Entry point for task initialization code	A
.FCLO	Perform the CLOSE statement	1
.FDEL	Perform the DELETE statement	1
.FIOPREP	I/O unit number to channel number translation	1
.FKILL	Perform the KILL statement	1
.FLDR	FLD function reference (RHS of assignment)	9
.FLDW	FLD function assignment (LHS of assignment)	9
.FOP	Internal file open	10
.FOPE	Perform the OPEN statement	10
.FPTRAP	Entry point for the floating point fault handler	A
.FRDC	Perform a formatted read of a complex entity	3

(continued)

Table H-1. Runtime Routine Entry Points

Entry Point Name	Description	Calling Sequence
.FRDD	Perform a formatted read of a double precision entity	3
.FRDI	Perform a formatted read of an integer entity	3
.FRDL	Perform a formatted read of a logical entity	3
.FRDR	Perform a formatted read of a real entity	3
.FRDX	Perform a formatted read of a double precision complex entity	3
.FREN	Perform the RENAME statement	10
.FSUS	Perform the SUSPEND statement	10
.FTAS	Perform the TASK statement for parametric entry points	11
.FTSK	Perform the TASK statement	12
.FWAI	Perform the WAIT statement	1
.FWRC	Perform a formatted write of a complex entity	3
.FWRD	Perform a formatted write of a double precision entity	3
.FWRI	Perform a formatted write of an integer entity	3
.FWRL	Perform a formatted write of a logical entity	3
.FWRR	Perform a formatted write of a real entity	3
.FWRS	Perform a formatted write of a character string	3
.FWRX	Perform a formatted write of a double precision complex entity	3
.GCH	Internal routine to get a character from a buffer	6
.GMEM	Internal routine to increase the unshared area by one page	0
.GREC	Internal routine to read a record from a file	5
.IACC	Initialization for the ACCEPT statement	0
.IATT	Process the ATT= option of the OPEN statement	4
.IBRD	Initialization for a binary read	4
.IBWR	Initialization for a binary write	4
.IDEC	Initialization for the DECODE statement	10
.IENC	Initialization for the ENCODE statement	10
.IEND	Process for the END= option for I/O statements	1
.IERR	Process for the ERR= option for I/O statements	1
.IFILE	Perform an implicit open of a preconnected file	6

(continued)

Table H-1. Runtime Routine Entry Points

Entry Point Name	Description	Calling Sequence
.IFRD	Initialization for the formatted read statement	10
.IFWR	Initialization for the formatted write statement	10
.IIBADDR	INFOS® interface routine	8
.IID	Process the ID= option for the TASK statement	1
.IINDSTR	INFOS interface routine	8
.ILEN	Process the LEN= option for the OPEN statement	1
.IOCB	Internal routine to allocate an IOCB	A
.IOPREP	Internal routine to convert a unit number to a channel	1
.IPRI	Process the PRI= option for the TASK statement	1
.IPRT	Initialization for the PRINT statement	13
.IPUN	Initialization for the PUNCH statement	13
.IREA	Initialization for the READ statement without format	13
.IREC	Process the REC= option for the OPEN statement	1
.ISAERR	Set the ISA error return variable to an error code	6
.ISANORM	Set the ISA error return variable for a normal return	0
.ISTK	Process the STK= option for the TASK statement	1
.ITYP	Initialization for the TYPE statement	0
.IURD	Initialization for the unformatted READ statement	1
.IUWR	Initialization for the unformatted WRITE statement	1
.LIERR	Language Independent Error Reporting Routine	A
.LINO	Perform initialization for line-number traceback	0
.LINE	Perform line number indication for traceback	12
.NCAL	Perform NREL internal call	0
.NFMT	Process a formatted I/O item	5
.PAUS	Perform the PAUSE statement	11
.PCH	Internal routine to output a character	6
.PCR	Internal routine to output a line terminator	6
.PNM	Internal routine to output a digit	6
.PREC	Internal routine write a record to a file	0

(continued)

Table H-1. Runtime Routine Entry Points

Entry Point Name	Description	Calling Sequence
.RLEF	Internal routine to turn off LFE mode	A
.ROUND	Internal routine to perform number rounding	5
.RTER	The runtime error reporter	6
.RTRN	Perform an alternate RETURN statement	6
.RWND	Perform the REWIND statement	1
.SBCH	Perform a subscript check operation	14
.SEEK	Process the REC= option for READ and WRITE statements	10
.SLEF	Internal routine to turn on LEF mode	A
.SOVL	Entry point for the stack fault handler	A
.STOP	Perform the STOP statement	11
.TACC	Perform termination for the ACCEPT statement	0
.TBRD	Perform termination for the binary READ statement	0
.TBWR	Perform termination for the binary WRITE statement	0
.TDEC	Perform termination for the DECODE statement	0
.TENC	Perform termination for the ENCODE statement	0
.TFRD	Perform termination for the formatted READ statement	0
.TFWR	Perform termination for the formatted WRITE statement	0
.TPRT	Perform termination for the PRINT statement	0
.TPUN	Perform termination for the PUNCH statement	0
.TRACE	VAL invocation of the runtime error traceback routine	B
.TREA	Perform termination for the READ statement without format	0
.TRTN	Internal routine for return address resolution	A
.TTYP	Perform termination for the TYPE statement	0
.TURD	Perform termination for the unformatted READ statement	0
.TUWR	Perform termination for the unformatted WRITE statement	0
.UFMT	Process an unformatted I/O item	5
.VIOPREP	Open routine for virtual data file	0
.WAKE	Perform the WAKEUP statement	1
.XTPP	.NREL routine for passing parameters to a new task	A

(continued)

Table H-1. Runtime Routine Entry Points

Entry Point Name	Description	Calling Sequence
.IACC	.NREL entry for ACCEPT initialization (.IACC)	A
?DIOCB	.NREL entry to free an I/O control block (entered via a JMP)	A
?IDEC	.NREL entry for DECODE initialization (.IDEC)	A
?IENC	.NREL entry for ENCODE initialization (.IENC)	A
?IFRD	.NREL entry for formatted READ initialization (.IFRD)	A
?IFWR	.NREL entry for formatted WRITE initialization (.IFWR)	A
?IPRT	.NREL entry for PRINT initialization (.IPRT)	A
?IPUN	.NREL entry for PUNCH initialization (.IPUN)	A
?IREA	.NREL entry for READ initialization (.IREA)	
?ITYP	.NREL entry for TYPE initialization (.ITYP)	A
?IURD	.NREL entry for unformatted READ initialization (.IURD)	A
?IUWR	.NREL entry for unformatted WRITE initialization (.IUWR)	A
?TACC	.NREL entry for ACCEPT termination (.TACC)	A
?TFRD	.NREL entry for formatted READ termination (.TFRD)	A
?TFWR	.NREL entry for formatted WRITE termination (.TFWR)	A
?TTYP	.NREL entry for TYPE termination (.TTYP)	A
?TURD	.NREL entry for unformatted READ termination (TURD)	A
?TUWR	.NREL entry for unformatted WRITE termination (.TUWR)	A
?UKIL	Task kill post-processing routine	A
?UTSK	Task initialization pre-processing routine	A

(concluded)

End of Appendix

Index

Within this index, "f" or "ff" after a page number means "and the following page" or "pages", respectively. In addition, primary page references of the runtime routines appear in italics. Commands, calls, and acronyms are in uppercase letters (e.g. OPEN); all others are lowercase.

G

I

Data General

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

educational services

012-1780

CUT ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

Data General

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

134-784-01

moisten & seal

CUSTOMER DOCUMENTATION COMMENT 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/MIS Manager
- ☐ Senior Systems Analyst
- ☐ Engineer
- ☐ Analyst/Programmer
- ☐ Operator
- ☐ End User
- ☐ Other ______________

How do you use this manual? *(List in order: 1 = Primary Use)*

- ___ Introduction to the product
- ___ Reference
- ___ Tutorial Text
- ___ Operating Guide
- ___ Other ______________

fold

About the manual:

	Yes	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 wish to order manuals, use the enclosed TIPS Order Form (USA only).

Comments:

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

Data General

Customer Documentation
62 T.W. Alexander Drive
Research Triangle Park, NC 27709-9990

134-784-01

134-784-01

moisten & seal

CUSTOMER DOCUMENTATION COMMENT 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/MIS Manager
- ☐ Senior Systems Analyst
- ☐ Engineer
- ☐ Analyst/Programmer
- ☐ Operator
- ☐ End User
- ☐ Other ______________

How do you use this manual? *(List in order: 1 = Primary Use)*

- ___ Introduction to the product
- ___ Reference
- ___ Tutorial Text
- ___ Operating Guide
- ___ Other ______________

fold

About the manual:		Yes	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 wish to order manuals, use the enclosed TIPS Order Form (USA only).

Comments:

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

Data General

Customer Documentation
62 T.W. Alexander Drive
Research Triangle Park, NC 27709-9990

134-784-01

Data General Corporation, Westboro, MA 01580

093-000154-02