

**PUBLICATIONS
UPDATE**

Operating System/3 (OS/3)

Extended FORTRAN

Programmer Reference

UP-8262 Rev. 1-C

This Library Memo announces the release and availability of Updating Package C to "SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN Programmer Reference", UP-8262 Rev. 1.

This update includes the following changes to the job control procedure for release 7.1:

- Specification of catalog files
- Expanded explanation of parameters

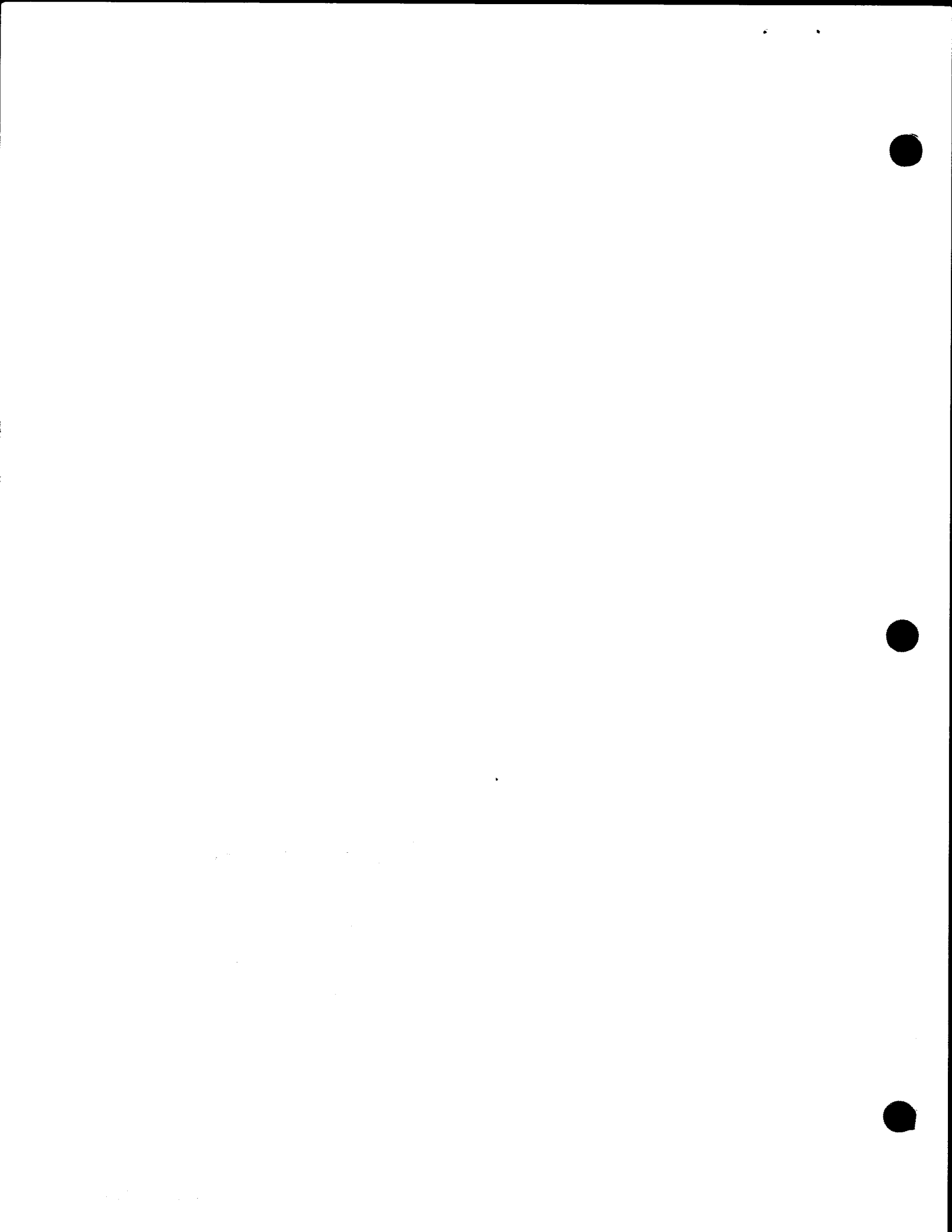
All other changes are either corrections or expanded descriptions applicable to features present in extended FORTRAN prior to the 7.1 release.

Copies of Updating Package C are now available for requisitioning. Either the updating package only, or the complete manual with the updating may be requisitioned by your local Sperry Univac representative. To receive updating package only, order UP-8262 Rev. 1-C. To receive the complete manual order UP-8262 Rev. 1.

RECEIVED
OCT 26 1981

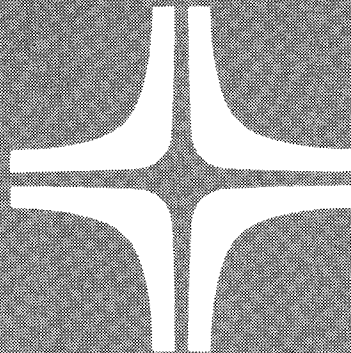
Director of Operations
Administration

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists 18, 19, 20, 21, 75 and 76 (Package C to 8262 Rev. 1, Covers and 14 pages plus Memo)	Library Memo for UP-8262 Rev. 1-C RELEASE DATE: September, 1981



Extended FORTRAN

OS/3



Programmer Reference

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry Univac representative.

Sperry Univac reserves the right to modify or revise the content of this document. No contractual obligation by Sperry Univac regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry Univac.

Sperry Univac is a division of the Sperry Corporation.

FASTRAND, SPERRY UNIVAC, UNISCOPE, UNISERVO, and UNIVAC are registered trademarks of the Sperry Corporation. ESCORT, PAGEWRITER, PIXIE, and UNIS are additional trademarks of the Sperry Corporation.

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

SPERRY UNIVAC

Operating System/3 (OS/3)

Extended FORTRAN

Supplementary Reference

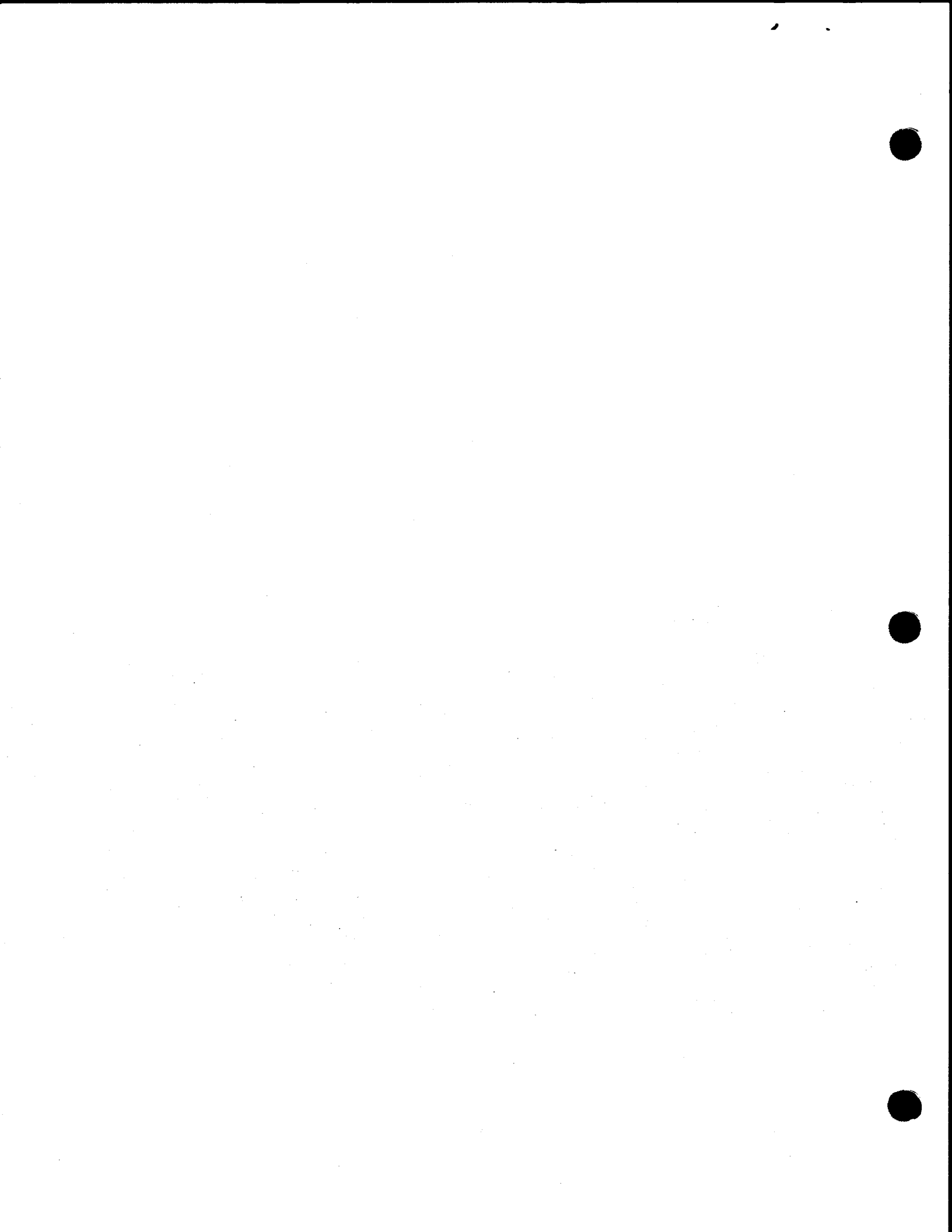
This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please contact your local Sperry Univac representative.

Sperry Univac reserves the right to modify or revise the content of this document. No contractual obligation by Sperry Univac regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry Univac.

Sperry Univac is a division of the Sperry Rand Corporation.

FASTRAND, SPERRY UNIVAC, UNISCOPE, UNISERVO, and UNIVAC are registered trademarks of the Sperry Rand Corporation. AccuScan, ESCORT, PAGEWRITER, PIXIE, and UNIS are additional trademarks of the Sperry Rand Corporation.

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.



Changes to UP-8262 Rev. 1:

Page 7-5

To the definition of a under 7.3.2, after the words
``an integer variable'', add ``(3.3.2)''.

Delete the last sentence in the paragraph preceding
7.3.2.1. Replace it with the following:

... internal representations described in 2.2 and
2.3 as specified by the format indicator, a. If
the format indicator is an array name, the array
must contain a legal FORMAT descriptor from opening
to closing parenthesis. An integer variable format
indicator must contain a FORMAT statement in an
ASSIGN statement. Our asterisk character specifies
list-directed formatting.

A COMPLEX item always requires two FORMAT editing
codes.

To the definition of a under 7.3.2.1, after the
words ``an integer variable name'', add (3.3.2).

Page 7-15

Under 7.3.4 the definition of a, after the words ``an integer variable name`` add (3.3.2).

Page 7-25

To the definition of a at the top of the page after words ``an integer variable name`` add (3.3.2) and after the words ``character asterisk`` add (7.3.5.2).

Page 7-26

Under 7.4.3, the definition of a, after the word ``assigned`` add (3.3.2) and after the word ``asterisk`` add (7.3.5.2).

```
ALTLOD= {vol-ser-no}
         {RES,$Y$RUN}
```

In the definition following the format, change the words ``file-label`` to ``file identifier`` and change \$Y\$LOD to \$Y\$RUN.

Page D-1

Replace the entire format of the FOR job control procedure with the following:


```

//[symbol] { FOR
            FORL
            FORLG } [ PRNTR= { (N
                              (1un)
                              N
                              20) } [,vol-ser-no] ) } ]

[ ,IN= { (vol-ser-no,label)
         (RES)
         (RES,label)
         (RUN,label) } ]

[ ,OUT= { (vol-ser-no,label)
          (RES,label)
          (RUN,label)
          RUN $Y$RUN } ]

[ ,SCR1= { vol-ser-no
          RES } ]

[ ,ALTLOD= { (vol-ser-no,label)
             (RES,$Y$RUN) } ] [,OPT=(D,N,X)]

[ ,MDE=I ] [,STX=option] [,CNL=k]

[ ,LIN= { filename
         LIB1 } ] [,LST=option]

```

Page D-2

Add the following immediately preceding the keyword parameter PRNTR.

Operation:

FOR

This form of the procedure call statement is used to compile an Extended FORTRAN source program.

FORL

This form of the procedure call statement is used to compile a Extended FORTRAN source program and link-edit the object modules.

FORLG

This form if the procedure call statement is used to compile a Extended FORTRAN source program, link-edit the object modules, and execute the load module.

Replace the format and description of the PRNTR keyword parameter with the following:

PRNTR= { ((N
 lun)
 N
 20) [,vol-ser-no] }

Specifies the logical unit number of the printer, and, optionally, the destination-id (vol-ser-no). If a printer device assignment set is not to be generated, the value N is coded, and the printer device assignment set must be manually inserted in the control stream.

PRNTR=(lun[,vol-ser-no])

Specifies the logical unit member (20-29) of the printer device. Optionally, the destination-id (vol-ser-no) can be specified.

PRNTR=(N[,vol-ser-no])

Indicates that a device assignment set for the printer must be manually inserted in the control stream. This permits LCB and VFB job control statements to be used in the control stream. The volume serial number can also be specified.

Keyword Parameter IN:

Change the words ``file label (label)`` to ``file identifier (label)`` wherever they occur.

Keyword Parameter OUT:

Change the words ``file label`` to ``file identifier`` wherever they occur.

Page D-3

Change the format of the keyword parameter ACTLOG to read:

ACTLOG= { (vol-ser-no,label)}
 { (RES,\$Y\$RUN) }

In the description following the format, change the words ``file label`` to ``file identifier``, and change \$Y\$LOD to \$Y\$RUN.

Changes to UP-8262 Rev. 1 (Extended FORTRAN)

Page 7-1

The last sentence in the second paragraph of 7.1 should read:

The peripheral devices are assigned unit numbers within the user's system where the unit number is a unique integer constant (k) in the range $1 \leq k \leq 99$.

MAY 11 12 30 PM '77

Page 7-5

The description of 'U' in line 3 should read:

Is a constant or an integer variable (k) designating an input or output device in the range of $1 \leq k \leq 99$.

Page 7-9

In 7.3.3.1.2. Real Descriptor - E Conversion (srEw.d), the last two lines of the first paragraph should read:

exceeds the precision of the list element, the value will be rounded to the correct size. If the value is too large or too small for the range, a zero will be substituted.

Page 7-21

In 7.3.6.2. BACKSPACE Statement the description of 'u' should be:

Is a constant or integer variable designating an I/O device.

In 7.3.6.2. BACKSPACE Statement, the fifth paragraph of the Description should read:

The BACKSPACE statement cannot be used with disc files under any conditions. It cannot be used with a file of blocked records or with a file having two I/O areas or a work area; . . .

Page 7-23

In 7.4.1. DEFINE FILE Statement, the description of 'u' should be:

Is a constant or integer variable designating an I/O device.

Page 7-25

In 7.4.2. Disc READ Statement, the description of 'u' should be:

Is a constant or integer variable designating an I/O device followed by an apostrophe.

In 7.4.3. Disc WRITE Statement, the description of ‘‘u’’ should be:

Is a constant or integer variable designating an I/O device followed by an apostrophe.

In 7.4.4. Disc FIND Statement, the description of ‘‘u’’ should be:

Is a constant or integer variable designating an I/O device followed by an apostrophe.

In 9.1. GENERAL line 2, delete the word ‘‘tape’’.

The // PARAM format should be changed as follows:

1		10
//	Δ PARAM Δ	OUT=filename, LIN=filename,LST=option,OPT-(D,N,X),NXT=LNK,CNL=K, MDE=I,STX=option,IN=module-name/filename

In line 5 under 9.2.1. Compiler Arguments, the sentence should read:

Specifies compilation of source programs residing in disc files.

Delete the word ‘‘tape’’ in line one.

Add a new paragraph 9.5. Organizing a Job Control Stream as follows:

9.5. Organizing a Job Control Stream

A special job control procedure simplifies the creation of a legal job control stream. Appendix D describes the use of this job control procedure.

If a user wishes to create his own job control cards, he should consider the following guidelines:

- The compiler requires one work file. (The JPROC, WORK1, supplies this file.)
- Use of IN and OUT options requires appropriate disc files.

- A printer with LFD PRNTR is always required.
- Because of the stacked compilation feature (9.3), the // OPTION REPEAT feature of OS/3 JCL is not required and must not be used.
- If // OPTION LINK= or // OPTION LINK,GO is specified, no linkage editor control cards or control stream data for the program are allowed because the source correction facility (9.4) or the stacked compilation feature (9.3) would mistake data sets for FORTRAN source.

Page 11-22

Under the FRECFORM=VARBLK argument, add the following sentence:

BACKSPACE is not allowed if this option is chosen.

Under the FRECFORM=FIXBLK argument, add the following sentence:

BACKSPACE is not allowed if this option is chosen.

Under the FNUMBUF=1 argument add the following sentence:

This argument is required if BACKSPACE is to be allowed.

Under the FWORKA=NO argument, add the following sentence:

This argument is required if BACKSPACE is to be allowed.

Page 11-26

Add the following bulleted item to the list of assumed defaults:

- BACKSPACE is not allowed because a work area is required.

In 11.3.4.5. Sequential Disc Files, change the first sentence of the second paragraph to read:

Sequential disc files are conceptually identical with tape files except a BACKSPACE command is not allowed.

Page B-1

Add the following sentence following the first sentence introducing the tables:

Note that the letter k when used with unit specifications represents a unique integer constant in the range of $1 \leq k \leq 99$.

Add the following bulleted item to the list in D.1. JOB CONTROL PROCEDURE after line 5:

automatic linkage or execution of a program.

Insert the following sentences in D.1 after the fourth paragraph (line 15):

The FORL procedure call generates an OPTION LINK job control statement which automatically executes the linkage editor after compilation.

The FORLG procedure call generates both OPTION LINK and OPTION GO job control statements which cause the program to be linked and executed.

NOTE:

Linkage control cards or program data is not allowed with these forms of the procedure call.

Change the format under D.1 as follows:

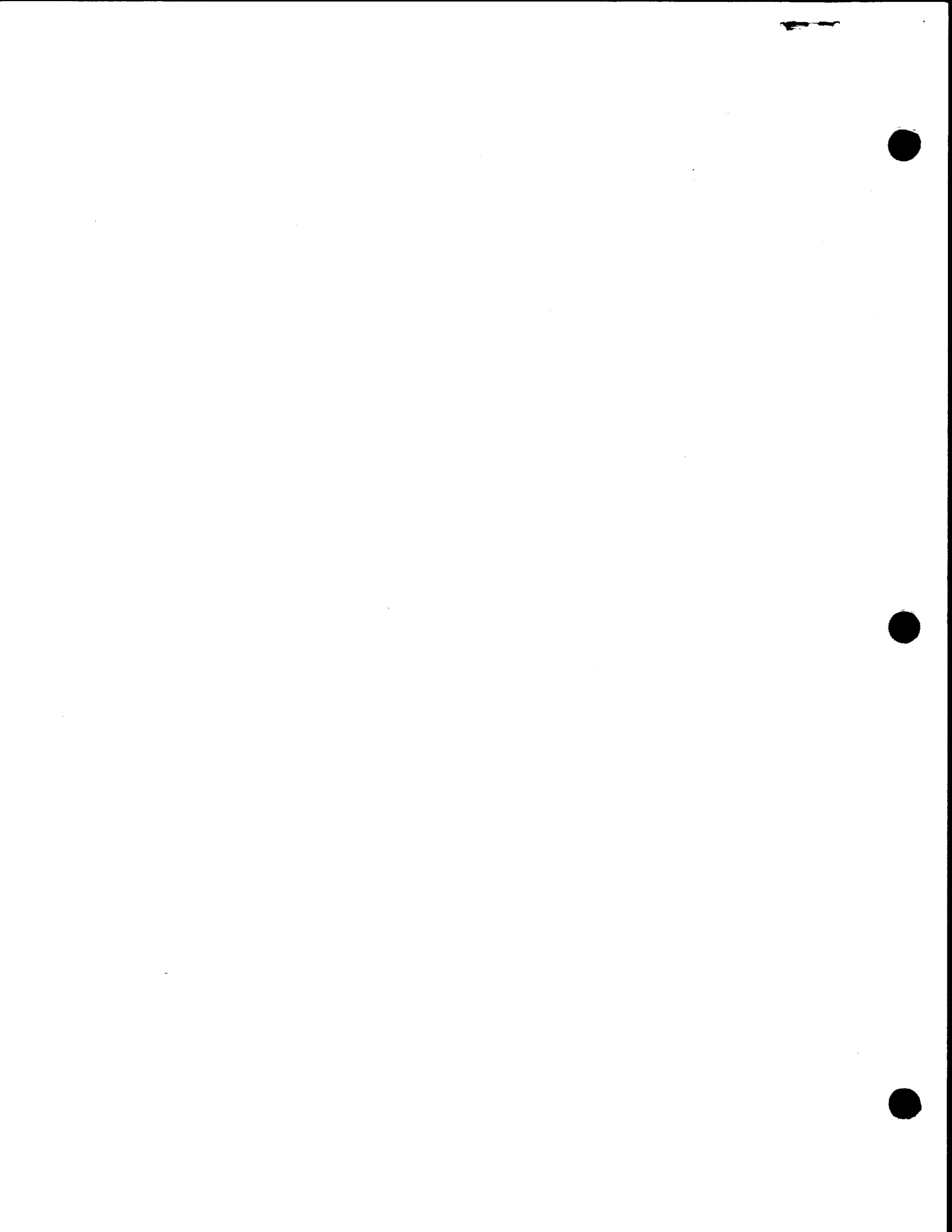
$$\begin{aligned}
 & //[\text{symbol}] \left\{ \begin{array}{l} \text{FOR} \\ \text{FORL} \\ \text{FORLG} \end{array} \right\} \left[\text{PRNTR} = \left\{ \begin{array}{l} \text{lun} \\ \underline{20} \end{array} \right\} [(\text{,vol-ser-no})] \right] \\
 & \left[\text{,IN} = \left\{ \begin{array}{l} (\text{vol-ser-no,label}) \\ (\text{RES}) \\ (\text{RES,label}) \\ (\text{RUN,label}) \end{array} \right\} \right] \left[\text{,OUT} = \left\{ \begin{array}{l} (\text{vol-ser-no,label}) \\ (\text{RES,label}) \\ (\text{RUN,label}) \end{array} \right\} \right] \\
 & \left[\text{,SCR1} = \left\{ \begin{array}{l} \text{vol-ser-no} \\ \underline{\text{RES}} \end{array} \right\} \right] \left[\text{,ALTLOD} = \left\{ \begin{array}{l} (\text{vol-ser-no,label}) \\ (\text{RES},\text{\$Y\$LOD}) \end{array} \right\} \right] \\
 & [,\text{OPT}=(\text{D,N,X})] [,\text{MDE}=1] [,\text{STX}=option] [,\text{CNL}=k] \\
 & [,\text{LIN}=filename] [,\text{LST}=option]
 \end{aligned}$$

Delete the OUT=NO parameter and description.

Add the following paragraph after the last sentence of the Appendix E.

Compile-Time Diagnostic Messages introduction to Table E-1.

For each diagnostic issued, the source statement is marked by a dollar sign (\$) below the column where the error was first recognized. When diagnosing a long statement consisting of multiple continuation card, the dollar sign occasionally appears under the wrong card, but flags the proper column. To locate the error the user should check the flagged column on each card of the statement.



PAGE STATUS SUMMARY

ISSUE: Update C – UP-8262 Rev. 1
RELEASE LEVEL: 7.1 Forward

Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level
Cover/Disclaimer		C	11 (cont)	17 thru 19 20 21, 22 23 thru 25 26 thru 28 29 thru 31 32, 33 34 thru 38 39 40 thru 42 43	A Orig. A Orig. A Orig. A Orig. B Orig. B			
PSS	1	C	12	1 thru 3	Orig.			
Preface	1	Orig.	Appendix A	1 thru 5	Orig.			
Contents	1 thru 4 5 6 thru 8	Orig. A Orig.	Appendix B	1 2, 3 4 5, 6	Orig. A Orig. A			
1	1 thru 7	Orig.	Appendix C	1 thru 6	Orig.			
2	1 thru 7	Orig.	Appendix D	1 2 2a 3, 4 5 thru 10 11, 12 13 thru 15	C A C C Orig. C Orig.			
3	1 thru 6	Orig.	Appendix E	1 2 thru 23	A Orig.			
4	1 thru 7	Orig.	Appendix F	1 thru 10	Orig.			
5	1 thru 22 23 24 thru 27 28 29	Orig. A Orig. A Orig.	Appendix G	1 thru 6	Orig.			
6	1 thru 9	Orig.	Index	1 thru 4 5 6 thru 8	Orig. A Orig.			
7	1 2, 3 4, 5 6 thru 8 9 10 thru 14 15 16 thru 20 21 22 23 24 25 thru 27	A Orig. A Orig. A Orig. A Orig. A Orig. A Orig. A	User Comment Sheet					
8	1 thru 3	Orig.						
9	1, 2 3 4 5 thru 7	A B Orig. A						
10	1 thru 5	Orig.						
11	1 2 3 thru 10 11 12, 13 14 15, 16 16a	Orig. C Orig. C A Orig. A A						

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.



Preface

This manual is one of a series designed to instruct and guide the programmer in the use of SPERRY UNIVAC Operating System/3 (OS/3). This manual specifically describes the SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN. Its intended audience is the experienced FORTRAN programmer new to SPERRY UNIVAC operating systems, and the OS/3 in particular.

The fundamentals of FORTRAN programmer reference manual, UP-7536 (current version) is also available for general information concerning FORTRAN programming. A knowledge of that manual is assumed. It is useful in reviewing the language; however, it does not present the Extended FORTRAN implementation for OS/3.

This manual is divided into three parts and seven appendixes:

- PART 1. EXTENDED FORTRAN PROGRAM STRUCTURE

Discusses Extended FORTRAN compiler, the general structure of source programs, coding form layout, character set, types of data including constants, variables, and array elements used in integer and real arithmetic.

- PART 2. FORTRAN STATEMENTS

Describes Extended FORTRAN expressions and assignment statements, control statements, statements used for functions and subroutines, specification statements, and I/O statements.

- PART 3. COMPILE, EXECUTE, AND DEBUG PROCEDURES

Discusses data initialization, compilation, configuration of the execution environment, and debugging.

- APPENDIXES

Provide additional information concerning:

- A — Character set
- B, C — UNIT options
- D — FORTRAN sample job streams
- E — Diagnostics
- F — Run-time library routines
- G — Subroutine linkage



Contents

PAGE STATUS SUMMARY

PREFACE

CONTENTS

1. INTRODUCTION

1.1.	SCOPE	1-1
1.1.1.	Compatibility	1-2
1.1.2.	Extensions	1-2
1.2.	SOURCE PROGRAMS	1-3
1.2.1.	Character Set	1-4
1.2.2.	FORTRAN Statements	1-4
1.2.3.	Comments	1-4
1.2.4.	Symbolic Names	1-5
1.2.5.	Source Statement Order	1-5
1.3.	STATEMENT CONVENTIONS	1-7

2. DATA TYPES

2.1.	GENERAL	2-1
2.2.	CONSTANTS	2-1
2.2.1.	Integer Constants	2-1
2.2.2.	Real Constants	2-2
2.2.3.	Double Precision Constants	2-3
2.2.4.	Hexadecimal Constants	2-3
2.2.5.	Complex Constants	2-4
2.2.6.	Logical Constants	2-4
2.2.7.	Literal Constants	2-5
2.3.	VARIABLES	2-5
2.4.	ARRAYS	2-6
2.4.1.	Array Element Reference	2-6
2.4.2.	Element Position Location	2-7

3. EXPRESSIONS AND ASSIGNMENT STATEMENTS

3.1.	GENERAL	3-1
3.2.	EXPRESSIONS	3-1
3.2.1.	Arithmetic Expressions	3-1
3.2.2.	Relational Expressions	3-1
3.2.3.	Logical Expressions	3-2
3.2.4.	Evaluation Order	3-2
3.2.5.	Mixed-Mode Arithmetic	3-3
3.2.6.	Arithmetic Operation User Checks	3-3
3.2.7.	Implementation of Arithmetic Operations	3-4
3.3.	ASSIGNMENT STATEMENTS	3-4
3.3.1.	Arithmetic and Logical Assignment Statements	3-5
3.3.2.	ASSIGN Statement	3-6

4. CONTROL STATEMENTS

4.1.	GENERAL	4-1
4.2.	ARITHMETIC IF	4-1
4.3.	LOGICAL IF	4-2
4.4.	UNCONDITIONAL GO TO	4-3
4.5.	COMPUTED GO TO	4-3
4.6.	ASSIGNED GO TO	4-4
4.7.	DO	4-4
4.7.1.	Transfer of Control to and from a DO Range	4-6
4.8.	CONTINUE	4-6
4.9.	STOP	4-6
4.10.	PAUSE	4-7
4.11.	END	4-7

5. FUNCTIONS AND SUBROUTINES

5.1.	GENERAL	5-1
5.2.	PROCEDURE REFERENCE	5-3
5.2.1.	Function Reference	5-3
5.2.2.	Subroutine Reference (CALL Statement)	5-3

5.3.	STATEMENT FUNCTION DEFINITION	5—4
5.4.	SUBPROGRAM DEFINITION	5—5
5.4.1.	External Functions	5—6
5.4.1.1.	FUNCTION Statement	5—6
5.4.1.2.	RETURN Statement	5—7
5.4.1.3.	ABNORMAL Statement	5—7
5.4.2.	Subroutines	5—8
5.4.2.1.	SUBROUTINE Statement	5—9
5.4.2.2.	Subroutine RETURN Statement	5—9
5.4.3.	Multiple Entry to Function and Subroutine Subprograms	5—11
5.5.	ARGUMENT SUBSTITUTION	5—12
5.5.1.	Call by Value	5—13
5.5.2.	Call by Name	5—13
5.5.3.	Symbolic Substitution	5—14
5.6.	LIBRARY PROCEDURES	5—14
5.6.1.	Intrinsic Functions	5—15
5.6.2.	Standard Library Functions	5—17
5.6.2.1.	Specification Statement Interaction	5—17
5.6.3.	Standard Library Subroutines	5—23
6.	SPECIFICATION STATEMENTS	
6.1.	GENERAL	6—1
6.2.	ARRAY DECLARATION	6—1
6.2.1.	Array Declarator	6—1
6.3.	DIMENSION STATEMENT	6—2
6.4.	TYPE STATEMENTS	6—3
6.4.1.	Explicit Type Statements	6—3
6.4.2.	IMPLICIT Statement	6—4
6.5.	EQUIVALENCE STATEMENT	6—6
6.6.	COMMON STATEMENT	6—6
6.6.1.	COMMON/EQUIVALENCE Statement Interaction	6—7
6.7.	EXTERNAL STATEMENT	6—8
6.8.	PROGRAM STATEMENT	6—9
7.	INPUT AND OUTPUT	
7.1.	GENERAL	7—1
7.2.	INPUT/OUTPUT LIST	7—1
7.2.1.	DO-Implied List	7—2

7.3.	SEQUENTIAL FILES	7-2
7.3.1.	Unformatted I/O Statements	7-3
7.3.1.1.	END and ERR Clauses	7-4
7.3.2.	Formatted READ/WRITE Statements	7-4
7.3.2.1.	I/O Compatibility Statements	7-5
7.3.3.	FORMAT Statement	7-6
7.3.3.1.	Field Descriptors	7-7
7.3.3.1.1.	Integer Descriptor (rlw)	7-8
7.3.3.1.2.	Real Descriptor — E Conversion (srEw.d)	7-9
7.3.3.1.3.	Real Descriptor — F Conversion (srFw.d)	7-9
7.3.3.1.4.	Double Precision Descriptor (srDw.d)	7-10
7.3.3.1.5.	Logical Descriptor (rLw)	7-10
7.3.3.1.6.	General Descriptor (srGw.d)	7-10
7.3.3.1.7.	Hollerith Descriptor — A Conversion (rAw)	7-10
7.3.3.1.8.	Hollerith Descriptor — H Conversion (wHc ₁ c ₂ ...cw)	7-10
7.3.3.1.9.	Hexadecimal Descriptor (rZw)	7-11
7.3.3.1.10.	Literal Descriptor ('c ₁ c ₂ ...c _n ')	7-11
7.3.3.1.11.	Blank Descriptor (wX)	7-12
7.3.3.1.12.	Record Position Descriptor (Tp)	7-12
7.3.3.1.13.	Scale Factor Effects	7-13
7.3.3.2.	Multiple Record Format Specification	7-13
7.3.3.3.	Carriage Control Conventions	7-13
7.3.3.4.	Format Interaction With the I/O List	7-14
7.3.4.	Reread	7-15
7.3.5.	List-Directed Input/Output	7-16
7.3.5.1.	NAMELIST Statement	7-17
7.3.5.2.	Simple List-Directed Input/Output	7-19
7.3.6.	Auxiliary I/O Statements	7-20
7.3.6.1.	REWIND Statement	7-20
7.3.6.2.	BACKSPACE Statement	7-21
7.3.6.3.	ENDFILE Statement	7-21
7.3.7.	Sequential File Considerations	7-22
7.4.	DIRECT ACCESS FILES	7-23
7.4.1.	DEFINE FILE Statement	7-23
7.4.2.	Disc READ Statement	7-24
7.4.3.	Disc WRITE Statement	7-26
7.4.4.	Disc FIND Statement	7-27
8.	DATA INITIALIZATION	
8.1.	GENERAL	8-1
8.2.	DATA STATEMENT	8-1
8.3.	BLOCK DATA SUBPROGRAM	8-3
8.3.1.	BLOCK DATA Statement	8-3
9.	COMPILATION	
9.1.	GENERAL	9-1

9.2.	PARAMETER STATEMENT FORMAT	9-1
9.2.1.	Compiler Arguments	9-2
9.3.	STACKED COMPILATION	9-5
9.4.	SOURCE CORRECTION FACILITY	9-6
9.5.	CREATING A JOB CONTROL STREAM	9-6 ←
10. DEBUGGING		
10.1.	GENERAL	10-1
10.2.	CONDITIONAL COMPILATION	10-1
10.3.	DEBUG STATEMENT	10-1
10.4.	DEBUGGING PACKET	10-2
10.4.1.	AT Statement	10-3
10.4.2.	TRACE ON Statement	10-3
10.4.3.	TRACE OFF Statement	10-3
10.4.4.	DISPLAY Statement	10-4
10.5.	FORMATTED MAIN STORAGE DUMP	10-5
11. CONFIGURATION OF THE EXECUTION ENVIRONMENT		
11.1.	DATA MANAGEMENT INTERFACE	11-1
11.2.	CONFIGURATIONS SUPPLIED	11-1
11.3.	PROGRAMMER-DEFINED CONFIGURATIONS	11-2
11.3.1.	File Definition Conventions	11-2
11.3.1.1.	Device Type	11-3
11.3.1.2.	Record and Block Sizes	11-3
11.3.1.3.	Record Formats	11-3
11.3.1.4.	Buffer Allocation	11-4
11.3.1.5.	File Type	11-5
11.3.2.	START Statement	11-6
11.3.3.	FORTRAN Initialization Procedure (FUNTAB)	11-6
11.3.4.	FORTRAN Unit Definition Procedure (UNIT)	11-6
11.3.4.1.	Printer File Definition	11-7
11.3.4.2.	Card Input Files	11-10
11.3.4.2.1.	Spooled Card Input File Definition	11-10
11.3.4.2.2.	Data Management Card Input File Definition	11-12
11.3.4.3.	Card Output File Definition	11-16a
11.3.4.4.	Tape File Definition	11-19
11.3.4.5.	Sequential Disc Files	11-26
11.3.4.6.	Direct Access Disc File Definition	11-32
11.3.4.7.	Reread Unit Definition	11-36
11.3.4.8.	Equivalent Unit Definition	11-37
11.3.5.	FORTRAN Unit Definition Termination Procedure (FUNEND)	11-39
11.3.6.	Error Environment Definition Procedure (ERRDEF)	11-39
11.3.7.	END Statement	11-42

12. PROGRAM COLLECTION AND EXECUTION

12.1.	GENERAL	12-1
12.2.	LINK EDITING FORTRAN PROGRAMS	12-1
12.2.1.	FORTRAN Supplied Modules	12-1
12.2.2.	Overlay and Region Structures	12-2
12.2.3.	Linkage Editor Output	12-2
12.3.	Execution FORTRAN Programs	12-3
12.3.1.	FORTRAN I/O Units	12-3
12.3.2.	Pause Messages	12-3
12.3.3.	Diagnostic Messages	12-3

APPENDIXES

A. CHARACTER SET

A.1.	SOURCE PROGRAM AND INPUT DATA CHARACTERS	A-1
A.2.	PRINTER GRAPHICS	A-3

B. SUMMARY OF UNIT OPTIONS

C. ADDITIONAL UNIT OPTIONS

C.1.	GENERAL	C-1
C.2.	PRINTER OPTIONS	C-1
C.3.	CARD READER OPTIONS	C-2
C.4.	CARD PUNCH OPTIONS	C-3
C.5.	TAPE FILE OPTIONS	C-3
C.6.	SEQUENTIAL DISC FILE OPTION	C-4
C.7.	DIRECT ACCESS DISC FILE OPTIONS	C-4
C.8.	ADDITIONAL DATA MANAGEMENT DEVICES	C-5

D. FORTRAN SAMPLE JOB STREAMS

D.1.	JOB CONTROL PROCEDURE	D-1	↓
D.2.	SAMPLE COMPILE—LINK—EXECUTE	D-9	
D.3.	SOURCE FROM DISC LIBRARY—STACKED COMPILATION	D-10	
D.4.	COMPILE—ASSEMBLE—LINK—EXECUTE	D-11	
D.5.	COMPILATIONS WITH PARAM OPTIONS	D-14	↑

E. COMPILE-TIME DIAGNOSTIC MESSAGES**F. RUN TIME MODULES****G. SUBROUTINE LINKAGE**

G.1.	CALLING FORTRAN SUBPROGRAMS	G-1	↓
G.1.1.	Save Area	G-1	
G.1.2.	Required Entry Conditions	G-2	
G.1.3.	Exit Conditions	G-2	
G.1.4.	Mathematical Library	G-3	
G.1.5.	Compiled Subprograms	G-4	
G.2.	CALLING FROM FORTRAN PROGRAMS	G-4	
G.2.1.	Parameter List Formats	G-4	
G.2.2.	Label Arguments	G-5	
G.2.3.	Conventions	G-5	
G.3.	TRACEBACK INTERFACE	G-5	↑

INDEX**USER COMMENT SHEET****FIGURES**

10-1.	DEBUG Statement and Packet	10-4
-------	----------------------------	------

TABLES

1—1.	Extended FORTRAN Character Set	1—4
1—2.	Source Statement Order	1—6
2—1.	Data Types and Optional Lengths	2—5
2—2.	Relative Location of Array Elements	2—7
3—1.	Extended FORTRAN Operators and Evaluation Order	3—3
3—2.	Result Types and Lengths for Mixed-Mode Arithmetic	3—4
3—3.	Assignment Statement Conversions	3—5
5—1.	Extended FORTRAN Procedures	5—1
5—2.	Argument Forms	5—2
5—3.	Intrinsic Functions	5—15
5—4.	Standard Library Functions	5—18
5—5.	Standard Library Subroutines	5—29
7—1.	FORMAT Statement Field Descriptors	7—7
7—2.	Carriage Control Conventions	7—14
7—3.	Permissible Associations of List Items	7—15
A—1.	EBCDIC Input Character Set	A—1
A—2.	Representative Character Set	A—3
B—1.	Summary of UNIT Arguments for Printer File	B—1
B—2.	Summary of UNIT Arguments for Spooled Card Input File	B—2
B—3.	Summary of UNIT Arguments for Card Input File	B—2
B—4.	Summary of UNIT Arguments for Card Output File	B—3
B—5.	Summary of UNIT Arguments for Tape File	B—3
B—6.	Summary of UNIT Arguments for Sequential Disc Files	B—5
B—7.	Summary of UNIT Arguments for Direct Access Disc Files	B—6
B—8.	Summary of UNIT Arguments for Reread Unit	B—6
B—9.	Summary of UNIT Arguments for Equivalent Unit	B—6
E—1.	Compile-Time Diagnostic Messages	E—2
F—1.	Extended FORTRAN Run-Time Modules	F—1
G—1.	Save Area Format	G—1
G—2.	Function Types and Corresponding Registers	G—3



1. Introduction

1.1. SCOPE

The SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN consists of the following components:

- an extended American National Standard FORTRAN language;
- a compiler, which transforms programs written in that language into a form suitable for execution;
- a library of input/output (I/O) and data formatting routines; and
- a library of commonly used mathematical functions and service routines.

The Extended FORTRAN compiler accepts programs written in the FORTRAN language and produces an object module that is suitable input to the linkage editor. Source programs may reside in the control stream or in a source program library. A job control procedure is provided to invoke the compiler, allocate scratch files, and so on. The output of the compiler must then be processed by the linkage editor; during this processing, mathematical and I/O routines are taken from the Extended FORTRAN library and included in the executable program. User-defined procedures, if they are required, are also included during the link edit. These latter procedures may be coded in FORTRAN or in some other language, such as COBOL, assembly, etc.

The output of the linkage editor is a load module that may consist of several overlay phases. During the execution of the object program, the overlay phases may be loaded by specific calls by FORTRAN statements, or they may be loaded automatically by referencing a routine in an overlay that is not currently in main storage. The load module will accept and produce ASCII files.

When the compiler is loaded, it interrogates the system to determine the amount of main storage space available to it. It then partitions the work space into an optimum allocation for table space and for I/O buffers.

During compilation, the compiler produces the following listings:

- A listing of the source program — each source statement is accompanied by any compiler-generated diagnostics; for each diagnostic, the source statement is marked at the character for which the diagnostic is produced.
- A main storage map showing the addresses allocated to the variables and arrays in the program.
- The object code in the form of a pseudo-assembly language program.

Any of the listings may be suppressed by user options.

The compiler is self-initializing, and any number of FORTRAN source programs may be processed by one call on the compiler by job control. If a FORTRAN source statement follows an END statement in the source input file, it is assumed that another program is to be processed, and the compiler reinitializes itself.

1.1.1. Compatibility

The SPERRY UNIVAC Extended FORTRAN language includes the American National Standard FORTRAN and the IBM System/360/370 DOS FORTRAN IV languages as subsets. Programs that conform to either of these specifications are accepted without change. Extended FORTRAN is also highly compatible with SPERRY UNIVAC Series 70 FORTRAN.

1.1.2. Extensions

The Extended FORTRAN language provides many extensions to *American National Standard FORTRAN, X3.9—1966*. These extensions are:

- Subscript expressions may be integer or real arithmetic expressions (2.4.1).
- Arithmetic assignment statements can be used to assign complex values to integer and real variables, or integer and real values to complex variables (3.3.1).
- A literal message is permitted with the STOP and PAUSE statements (4.9 and 4.10).
- An executable END statement is provided (4.11).
- The inclusion of statement labels (preceded by the & character) in the list of actual arguments in a subroutine call to be referenced by a RETURN statement is permitted. Thus, the subroutine can transfer control back to designated statements in the calling program (5.4.2.1).
- The ENTRY statement permits entry into a function or subroutine subprogram at points other than the beginning of the subprogram (5.4.3).
- Standard library routines are available: OVERFL, DVCHK, ERROR, ERROR1, SLITE, SLITET, SSWTCH, LOAD, FETCH, DUMP, PDUMP, and OPSYS (5.6.3).
- Arrays may have a maximum of seven dimensions (6.2.1).
- Dimension declarator subscripts are permitted in common storage.
- Optional length specifications for logical, integer, complex, and real variables and arrays can be declared (6.4.1).

- An IMPLICIT statement is provided for user-defined implicit typing of symbolic names in a program unit (6.4.2).
- End of file and error recovery are provided in READ statements (7.3.1.1).
- The applicability of the G field descriptor has been extended to cover integer and logical data (7.3.3.1.6).
- T and Z format codes are provided (7.3.3.1.9 and 7.3.3.1.12).
- Special I/O formats and statements are provided for direct access storage devices (7.4).
- The specification of hexadecimal constants in DATA statements is permitted (8.2).

The Extended FORTRAN language also includes several extensions to IBM System/360/370 FORTRAN IV; these include:

- Embedded comments (1.2.3).
- Extended exponentiation (3.2).
- An integer variable name can be used to represent the statement label of a FORMAT statement. Thus, references can be made to FORMAT statements by integer variable name or by actual statement label (3.3.2).
- Optional statement labels on arithmetic IF (4.2).
- Logical IF, PAUSE, and STOP statements can be terminal statements of DO loops (4.7).
- Array element names may be referenced on the right-hand side of statement functions (5.3).
- An ABNORMAL statement is provided for optimal code generation (5.4.1.3).
- The mathematical library may be referenced by generic names (5.6).
- The ability to initialize variables and arrays in type and DIMENSION statements (6.3 and 6.4.1).
- The ability to use the IMPLICIT statement anywhere in the specification statement group (6.4.2).
- The elimination of the restriction that all named common blocks be the same size (6.6).
- A PROGRAM statement is provided to optionally name a main program (6.8).
- Two classes of list-directed I/O statements are provided (7.3.5).
- DO-implied loops in DATA statements (8.2).
- The BLOCK DATA statement contains an optional name for the subprogram (8.3.1).
- Extended error recovery procedures are provided for the mathematical library (11.3.3).
- Blocked and buffered input/output is provided (Section 11).

1.2. SOURCE PROGRAMS

General procedures to be followed in FORTRAN programming are presented in the following paragraphs.

1.2.1. Character Set

The character set consists of the FORTRAN character set and special characters as shown in Table 1—1. Each character is represented in the Extended Binary Coded Decimal Interchange Code (EBCDIC). EBCDIC codes not shown in the table have no graphic equivalents in the Extended FORTRAN character set, but these characters can be stored internally and transmitted to and from card, tape, and disc storage.

Table 1—1. Extended FORTRAN Character Set

FORTRAN character set	Alphabets	A through Z and \$
	Numerics	0 through 9
	Special symbols	=, () + - * / . & ' ;
	Blank	Δ or blank space
Extended character set*	Any characters capable of representation in EBCDIC, such as: ¢ > < % ! : @ # ? - "	

*The special character set can change with the options selected for the system printer, with 48 to 127 characters available, depending on printer model. See Appendix A for a detailed discussion of the character set.

1.2.2. FORTRAN Statements

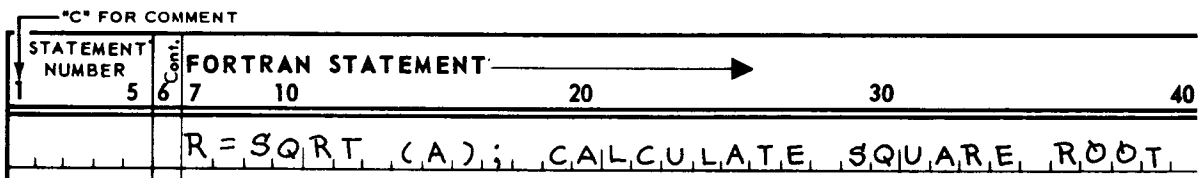
FORTRAN statements are coded on the FORTRAN coding form, where columns 1 through 72 can be used for the contents of a FORTRAN line. All characters in a line are restricted to the FORTRAN character set, except in comments and literal constants. Columns 73 through 80 are ignored and may be used in any manner; the information in these columns is printed in the source program listing, but execution of the program is not affected by this information.

Each FORTRAN statement is written in columns 7 through 72. The first line of a statement must contain either a zero or a blank character in column 6. A statement may be continued on one or more successive lines with a nonzero, nonblank character in column 6 for each line that is a continuation. A statement consists of one initial line and up to 19 continuation lines.

A statement label consists of one through five decimal digits in columns 1 through 5. The contents of these columns for continuation lines are ignored during program compilation but are shown on the program listing and may be used by the programmer. Leading zeros, and embedded and trailing blank characters, are ignored in a statement label. Each statement label must be unique within its program unit. A special use of column 1 is indicated by an X coded there for program debugging purposes (9.2).

1.2.3. Comments

The compiler provides four methods of entering comments: columns 73—80 on any line; columns 1—5 on continuation lines; the comment line; and embedded comments. A comment line is indicated by the character C in column 1. The contents of each comment line are shown on the program listing, but are ignored by the compiler. A semicolon in columns 7 through 71 in a FORTRAN statement line indicates that the information immediately following and written on the same line is to be treated as a comment; for example:



A comment following a semicolon is continued on a succeeding statement line by specifying a C in column 1:

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	Column	7	10	20	30	40
		DO 100 I=1,9; BEGIN ITERATION				
C		LOOP				

The statement, SUBROUTINE SWAP (A,B), including commentary, may be written as follows:

		SUBROUTINE; THIS SUBROUTINE				
	1	SWAP				; EXCHANGES THE VALUES
	2	(A, B)				; OF TWO REAL VARIABLES

A semicolon in a literal constant is a valid character and does not indicate a comment; a semicolon to the left of column 7 does not indicate a comment. Blank cards are ignored by the compiler.

1.2.4. Symbolic Names

Symbolic names contain up to six alphanumeric characters, the first of which must be alphabetic.

A special type of symbolic name, a label parameter, is associated with the RETURN statement. It consists of the & character immediately followed by a statement label. A label parameter can appear only in a list of actual arguments in a CALL statement (5.2.2).

1.2.5. Source Statement Order

Table 1—2 shows the order in which the source statements of each program unit must be written. Within each grouping, the statements may be written in any sequence.

Every executable program contains one main program and as many subprograms as required. A main program is a set of statements and comments that is not headed by a FUNCTION, SUBROUTINE, or BLOCK DATA statement. Subprograms are headed by one of these statements. A subprogram headed by a BLOCK DATA statement is a specification subprogram; one headed by a FUNCTION or SUBROUTINE statement is a procedure subprogram. The term "program unit" is used to refer to any main program or subprogram. All program units are terminated with an END statement. The first statement of a main program may optionally be a PROGRAM statement.

Table 1-2. Source Statement Order

LINE 1	Program Declarators:		
	BLOCK DATA FUNCTION	PROGRAM SUBROUTINE	
	Specification Statements:		
		ABNORMAL COMMON COMPLEX DIMENSION DOUBLE PRECISION EQUIVALENCE	EXTERNAL IMPLICIT INTEGER LOGICAL REAL
	DEFINE FILE	Statement Functions	
COMMENT		Executable Statements:	
	ENTRY	Arithmetic assignment Arithmetic IF ASSIGN Assigned GO TO BACKSPACE CALL Computed GO TO Logical assignment Logical IF CONTINUE DO	ENDFILE FIND PAUSE PRINT READ PUNCH RETURN REWIND STOP Unconditional GO TO WRITE
	FORMAT	DEBUG	
	NAMELIST	AT TRACE ON TRACE OFF DISPLAY	
		Any Executable Statement	
LINE n	END		

NOTES:

1. Vertical lines demarcate statements that may be freely intermixed; for example, FORMAT statements may appear anywhere between the program declarator (which may not exist) and the END statement.
2. Horizontal lines demarcate statements that must be in the order shown; for example, statement functions must follow all specification statements.
3. DATA statements must follow any specification statements that reference items to be initialized (see dotted line).

1.3. STATEMENT CONVENTIONS

Conventions used to illustrate FORTRAN statements in Sections 1 through 9 are presented throughout those sections. Conventions for illustrating statements in assembler language in Sections 10 and 11 and Appendixes C and D are as follows:

- Capital letters, parentheses (), and punctuation marks (except braces, brackets, and ellipses) must be coded exactly as shown. An ellipsis (a series of three periods) indicates the presence of a variable number of entries.
- Lowercase letters and terms represent information supplied by the user.
- Information within braces { } represents necessary entries, one of which must be chosen.
- Information within brackets [] (including commas) represents optional entries that are included or omitted depending on program requirements. Braces within brackets signify that one of the entries must be chosen if that operand is included.
- Underlined parameters are selected automatically when a parameter is omitted. These are called defaults.
- Some defaults are dependent on entries selected in other arguments. For example:

$$\left[\text{FRECSIZE} = \left\{ \begin{array}{l} k \\ \underline{80}; \text{ if FMODE=STD} \\ \underline{160}; \text{ if FMODE=BINARY} \end{array} \right\} \right]$$

The notation

FRECSIZE*4

specified as a default for an argument other than FRECSIZE, indicates that the default value for this argument consists of the value specified for the FRECSIZE argument, multiplied by 4. This default value should be used only as a default; it should not be specified as a predefinition argument.



2. Data Types

2.1. GENERAL

The data types available in SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN are integer, real, double precision, complex, logical, hexadecimal, and literal. For additional information concerning FORTRAN data types, refer to the "Writing a FORTRAN Program" section of the fundamentals of FORTRAN reference manual, UP-7536 (current version). Data types are categorized by their manipulation within the FORTRAN program; e.g., data may appear as constants, variables, or elements of an array. Each of these categories is explained in this section. Additional information on the hardware characteristics of integer and real arithmetic may be found in the discussion of the arithmetic section in the SPERRY UNIVAC 90/30 System processor reference manual, UP-8052 (current version).

2.2. CONSTANTS

A constant is an arithmetic, logical, or literal value defined by its representation in the source program. Once defined, a constant must not be redefined during program execution. An arithmetic constant is said to be signed if it is written with a plus or a minus sign, and an unsigned constant is treated as a positive value. Constants are represented internally, using 8-bit bytes organized as single units, groups of two (half words), groups of four (words), and groups of eight (double words).

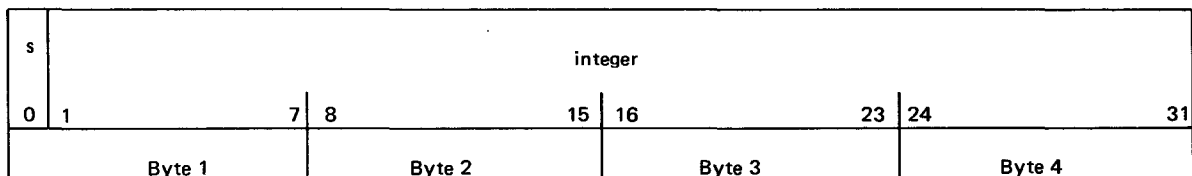
2.2.1. Integer Constants

An integer constant consists of an optional sign followed by a string of decimal digits with no decimal point. An integer constant may have a maximum of 10 digits. If the value of the constant is positive, it may be preceded by a plus sign; if the value is negative, it must be preceded by a minus sign; for example:

```

      1
     -365
100000000
    
```

An integer constant has the following 4-byte representation in storage:



where:

s

Is the sign bit (0 indicates positive; 1 indicates negative).

integer

Is a 31-bit binary integer, in twos complement representation.

The maximum absolute value for an integer is 2,147,483,647 ($2^{31}-1$).

2.2.2. Real Constants

A real constant may be written as:

- A basic real constant: an optionally signed string of up to seven significant digits with a decimal point preceding, embedded in, or following the string; for example:

—1701.001

- A basic real constant followed by a decimal exponent; the decimal exponent is expressed by the letter E followed by an optionally signed integer constant with a maximum of two significant digits; for example:

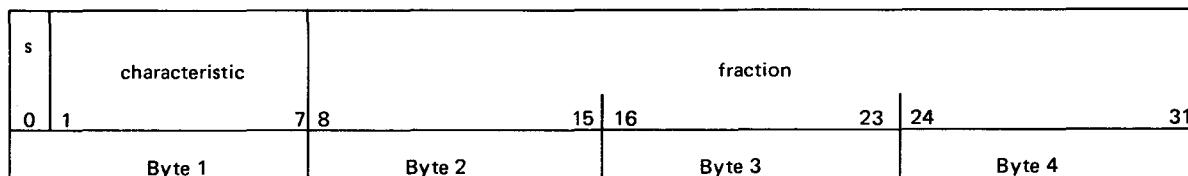
170.1E—03

- An integer constant followed by a decimal exponent; if the integer portion exceeds the permitted seven digits, truncation of the excess rightmost digits results; for example:

+1701E—4

17010E—5

Real constants occupy one word (four bytes) of storage in normalized floating-point representation. The format is:



where:

s

Is the sign bit.

characteristic

Is the exponent portion of the real number in seven bits; it is derived from the power of 16 by which the fraction must be multiplied to give the real value; the characteristic is stored as an excess 64 number.

fraction

Is six hexadecimal digits representing the fractional part of the real value. The radix point is between bits 7 and 8.

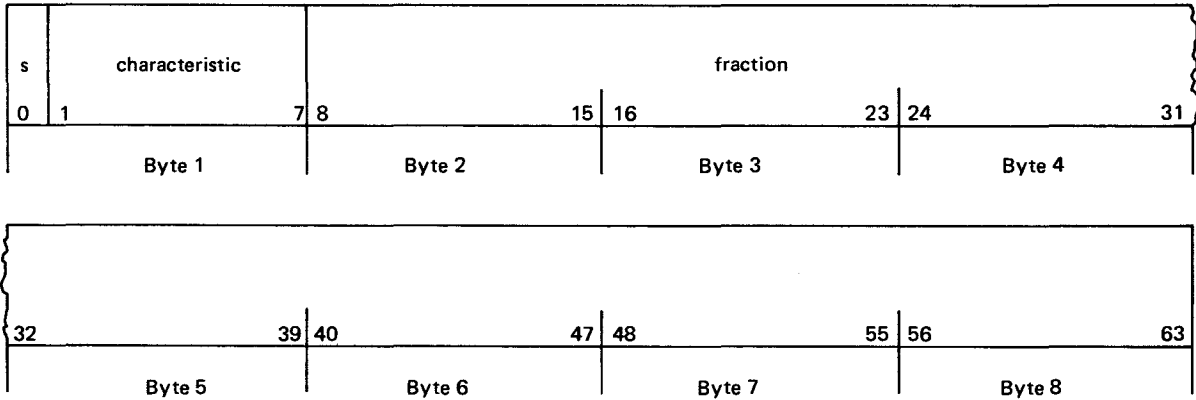
The maximum range for a real constant is from approximately 10^{-78} through 10^{75} . It may have the value 0 where the fraction is identically binary 0.

2.2.3. Double Precision Constants

A double precision constant is similar to a real constant, except that it may contain up to 16 significant digits. It is written as a basic real constant or an integer constant followed by a double precision exponent; a double precision exponent is expressed by the letter D followed by an optionally signed integer constant with a maximum of two significant digits; for example:

—.180018201840D12

A double precision constant is stored like a real constant, except that two words (eight bytes) of storage are used:



A double precision constant may range in value from approximately 10^{-78} through 10^{75} , or it may have the value 0.

2.2.4. Hexadecimal Constants

Hexadecimal constants are written as the letter Z followed by any combination of up to 32 hexadecimal digits; the hexadecimal digits and their equivalents are:

Hexadecimal Digits	Decimal Equivalents	Binary Representation
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Hexadecimal Digits	Decimal Equivalents	Binary Representation
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadecimal constants can be used only to initialize variables or arrays in specification or initialization statements. The maximum number of digits used for initialization is determined by the type of data associated with the constant. If the number of digits specified exceeds the maximum, the leftmost digits are truncated. If less than the maximum are specified, hexadecimal 0's are padded on the left. Two hexadecimal digits occupy one byte in main storage. Some examples of hexadecimal constants are:

Hexadecimal Constant	Binary Equivalent
ZF9	1111 1001
ZA8	1010 1000
ZC5	1100 0101

2.2.5. Complex Constants

A complex constant consists of an ordered pair of real constants or double precision constants, each of which may be signed, separated by a comma, and enclosed in a set of parentheses. The first portion of the complex constant is the real part, and the second is the imaginary part of the complex value. For example, (3.1415,182.) and (314D—2,—18.2D1) are valid complex constants. Complex constants are stored in either two or four words, depending on whether a double precision constant appears. The presence of a double precision constant within the parentheses causes the other constant to be treated as double precision, thus forming a double precision complex constant of 16 bytes. Integer constants in this context will be converted to real constants by the compiler. For example:

(10,50D+7)	becomes	(10.0 D+0,50D+7)
(10,10)		(10.0 E+0,10.0 E+0)
CALL A (10,10)		CALL A (10,10)
CALL A ((10,10))		CALL A ((10.0, 10.0))

2.2.6. Logical Constants

Logical constants specify the logical values .TRUE. or .FALSE. and occupy one word in storage. The value .FALSE. has a binary representation of 0; .TRUE. has a binary representation of 1.

2.2.7. Literal Constants

A literal constant consists of one or more characters from the Extended FORTRAN character set (Table 1—1). Each character in the string requires one byte of storage. Two methods of writing literal constants are:

- as a Hollerith constant in the form $wHc_1c_2\dots c_w$ where w is an unsigned integer constant and c represents a character; or
- as a character string enclosed in apostrophes: $'c_1c_2\dots c_n'$. If the apostrophe occurs in the string, it is represented by doubling that character.

The literal DO NOT would be represented by 'DO NOT' or 6HDO NOT, and the literal DON'T would be represented by 'DON'T' or 5HDON'T.

2.3. VARIABLES

A variable is represented by a symbolic name (1.2.4) that identifies a single value. A variable is associated with a data type, and in Extended FORTRAN there is both a standard and an optional length specification that determines the number of bytes assigned in main storage (Table 2—1).

The type associated with a variable is determined by either the explicit type declaration statements (6.4.1), by the IMPLICIT statement (6.4.2), or by the variable name used. Names beginning with the letters I, J, K, L, M, or N are assumed to represent integer values; names beginning with all other letters or \$ are assumed to represent real values.

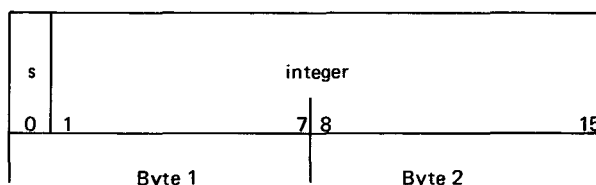
Table 2—1. Data Types and Optional Lengths

FORTRAN Name	Standard Data Type	Length in Bytes	Optional Data Type	Length in Bytes
Integer	Integer*4	4	Integer*2	2
Real	Real*4	4	Real*8	8
Double precision	Double precision	8	None	
Complex	Complex*8	8	Complex*16	16
Logical	Logical*4	4	Logical*1	1

To prevent confusion where the length can differ, the complete data type will appear in this document: a reference to 16-byte complex data will appear as complex*16. A reference to logical data without any length specification refers to logical*4 data. The optional specification for real data is real*8, the equivalent of double precision representation.

Variables of the double precision type have only a standard length. There is no variable type associated with literal or hexadecimal data. The optional length described may be specified in either the explicit type statements or the IMPLICIT statement.

The internal representation of the values is identical with that described for the proper constant type, with the exception of integer*2 and logical*1 where there are no corresponding constants. The integer*2 variable or array element occupies two bytes, with the sign stored in the most significant bit:



The maximum value for the integer*2 type is 32767 ($2^{15}-1$). The hardware does not provide overflow indications if the integer*2 is exceeded; therefore, significant numeric bits can propagate into the sign bit.

Example:

The following program prints the value -32768, with no indication of arithmetic overflow.

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
5	6	INTEGER * 2 I/32767/, J/1/, K			
		K = I + J			
		PRINT 10, K			

The logical*1 variable or array element occupies one byte in main storage. The value .FALSE. has a binary representation of 0; .TRUE. is nonzero and is usually 1.

2.4. ARRAYS

An array is an ordered set of values. Each value is called by array element, and the entire set is identified by a symbolic name called an array name. An array is described by an array declarator (Section 6). In Extended FORTRAN, the array can be declared as having a maximum of seven dimensions.

The form of the array declarator is dependent on the number of dimensions as shown in Table 2-2. For instance, an array named AGO with three dimensions, each four elements in size, has the declarator AGO(4,4,4). AGO is the array name, and the numbers in the parentheses are dimension declarators. Each dimension declarator must be an unsigned integer constant, except when a dimension is adjustable. In this case, the dimension declarator must be an integer variable with a length of four bytes.

2.4.1. Array Element Reference

Any element in an array may be referenced by using the array name, followed by parenthesized subscripts in the format:

array name (s₁,s₂,...,s_n)

where:

s

May be any integer or real arithmetic expression. The arithmetic expression must be evaluated during execution as an integer greater than 0. Each subscript is evaluated in accordance with the standard rules for evaluating mixed-mode expressions (Section 3).

n

Must correspond to the total number of subscripts in the declarator.

In an EQUIVALENCE statement, the number of subscripts may be either one (where the correspondence of elements is determined by the location of array elements as in 2.4.2) or the number of subscripts in the array declarator.

2.4.2. Element Position Location

General expressions for locating the position of an array element relative to its first element are given in Table 2—2. In the table, the first byte of the array is relative location 0; the letters a,b,...,g refer to the value of a subscript expression in an array element reference; the letters A,B,...,G refer to the values of the dimension declarators; and the m is a multiplier determined by the number of bytes of storage required for each array element.

Table 2—2. Relative Location of Array Elements

Number of Dimensions	Declarator Form	Subscript Form	Relative Location of Element in the Array
1	(A)	(a)	$(a-1)*m$
2	(A,B)	(a,b)	$((a-1)+A*(b-1))*m$
3	(A,B,C)	(a,b,c)	$((a-1)+A*(b-1)+A*B*(c-1))*m$
.	.	.	.
.	.	.	.
7	(A,B,C,D,E,F,G)	(a,b,c,d,e,f,g)	$((a-1)+A*(b-1)+A*B*(c-1)+\dots+A*B*C*D*E*F*(g-1))*m$

Examples:

If an array declarator were AGO(17), if the element referenced were AGO(4), and if the array were real, then the location of the first byte of the fourth element relative to the beginning of the array is found with the expression $(a-1)*m$. In this case: $(4-1)*4 = 12$, or the first byte of that element is the twelfth from the beginning of the array.

If AGO were declared as AGO(9,10,11) and the element to be located were AGO(3,4,5), then the calculation is $((2)+9*(3)+9*10*(4))*4$, or location 1556.



3. Expressions and Assignment Statements

3.1. GENERAL

This section discusses the use of expressions in SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN programming, and describes the assignment statements. For additional information, refer to the "FORTRAN Expressions" and "Assignment Statements" sections of the fundamentals of FORTRAN reference manual, UP-7536 (current version).

3.2. EXPRESSIONS

An expression is a group of one or more elements and operators that is evaluated as a single value during execution. Three different classes of expressions are evaluated: arithmetic, relational, and logical expressions. Each of these expressions, as well as the order of evaluation, mixed-mode arithmetic, and user checks on arithmetic operations, is discussed in the following paragraphs.

3.2.1. Arithmetic Expressions

An arithmetic expression is constructed as a numeric constant, a variable name, an array element reference, a function reference, or combinations of these by using arithmetic operators. An arithmetic expression is always evaluated as a numeric value.

3.2.2. Relational Expressions

A relational expression, actually a subset of logical expressions, consists of two arithmetic expressions separated by a relational operator. The expression is evaluated at execution as `.TRUE.` or as `.FALSE.`. No complex type of arithmetic expression may be used in a relational expression; however, the other types may be mixed in any combination.

When mixed-mode arithmetic comparisons are made, the priority of the data type is:

Data Type	Priority
Real*8 (double precision)	1
Real*4	2
Integer*4	3
Integer*2	4

The expression having lower priority is converted to the type of the higher priority, and the comparison is made. For example, if the relational expression consists of an integer expression and a real expression, the integer is converted to a real*4 type before the comparison is made.

The result of a relational expression is always logical*4 type.

3.2.3. Logical Expressions

A logical expression is:

- a relational expression, a logical constant, a logical variable reference, a logical array element reference, a logical function reference, or a logical expression in parentheses;
- a logical or relational expression preceded by the operator `.NOT.`; or
- two logical or relational expressions separated by `.AND.` or `.OR.`

If both operands of a logical expression are of the `logical*1` type, then the result is of `logical*1` type; otherwise, the result is of the `logical*4` type.

3.2.4. Evaluation Order

An expression is evaluated by the priority of the operators in Table 3—1 and then calculated as follows:

1. This process begins with the leftmost operator.
2. If no parentheses intervene, the current operator is compared with the operator on its right. If the priority of the current operator is greater than or equal to the priority of the next operator, the current operation is performed and the result becomes the operand of the prior operator. Otherwise, the next operator becomes the current operator, and this step is repeated, using it for comparison.
3. Upon encountering the right end of an expression, remaining operations are performed from right to left.
4. Sequential exponentiation is performed from right to left. For example, $X^{**}Y^{**}Z$ is evaluated as $X^{**}(Y^{**}Z)$.
5. Sequential integer division is performed from left to right. For example, $I/J/K$ is evaluated as $(I/J)/K$.
6. Expressions in parentheses are treated as single operands and evaluated first, starting with the innermost parenthesized expression, before continuing the left-to-right comparisons.

In addition to the rules already described, the order in which operations are performed may be slightly affected by optimizations:

- Sequential operations with the same priority, except exponentiation, can be performed in any order. For example, $A+B+C$ may be performed as $(A+B)+C$, or $(A+C)+B$, or $(B+C)+A$. At compilation time, an option is provided to require such expressions to be evaluated strictly from left to right. In addition, parentheses may be used to force a specific order of evaluation.
- Logical expressions are not always completely evaluated; once the value is known, evaluation ceases. For example:

```
IF (A.GT.B.OR.C.LT.FUNC(X)) GO TO 10
```

If A were greater than B, control would be transferred to statement 10 immediately, because the expression must be `.TRUE.` The function `FUNC` is not referenced.

- The removal of locally constant subexpressions from DO loops and the elimination of duplicate subexpressions may result in the calculation being performed in a place other than that specified. Therefore, if underflow, overflow, or divide check should occur for any of those locally constant calculations, they would occur upon entry to the DO loop rather than where they were originally written.

Table 3-1. Extended FORTRAN Operators and Evaluation Order

Operation	Operator	Order of Priority
Function evaluation	f(x)	1
Exponentiation	**	2
Multiplication	*	3
Division	/	
Addition or unary plus	+	4
Subtraction or unary minus	-	
Greater than	.GT.	5
Greater than or equal to	.GE.	
Less than	.LT.	
Less than or equal to	.LE.	
Equal to	.EQ.	
Not equal to	.NE.	
Logical negation	.NOT.	6
Logical product	.AND.	7
Logical sum	.OR.	8

3.2.5. Mixed-Mode Arithmetic

Mixed-mode arithmetic occurs when an operation is performed on two operands that are not the same type. The type and length of the result are shown in Table 3-2 for the arithmetic operators, including exponentiation.

3.2.6. Arithmetic Operation User Checks

The following subroutine calls enable the programmer to check the evaluation of an arithmetic expression:

- CALL DVCHK(i)

Used to check for a division by zero after the division has been executed.

- CALL OVERFL(i)

Executed after an arithmetic operation to check for an overflow or underflow condition.

- CALL ERROR1 and CALL ERROR(i)

Routines used to set and test an indicator.

See 5.6.3 for more information on these subroutines.

Table 3-2. Result Types and Lengths for Mixed-Mode Arithmetic

		First Operand: Type (Length)					
		Integer (2)	Integer (4)	Real (4)	Real (8)	Complex (8)	Complex (16)
Second Operand: Type (Length)	Integer (2)	Integer (4)	Integer (4)	Real (4)	Real (8)	Complex (8)	Complex (16)
	Integer (4)	Integer (4)	Integer (4)	Real (4)	Real (8)	Complex (8)	Complex (16)
	Real (4)	Real (4)	Real (4)	Real (4)	Real (8)	Complex (8)	Complex (16)
	Real (8)	Real (8)	Real (8)	Real (8)	Real (8)	Complex (16)	Complex (16)
	Complex (8)	Complex (8)	Complex (8)	Complex (8)	Complex (16)	Complex (8)	Complex (16)
	Complex (16)	Complex (16)	Complex (16)	Complex (16)	Complex (16)	Complex (16)	Complex (16)
	Complex (16)	Complex (16)	Complex (16)	Complex (16)	Complex (16)	Complex (16)	Complex (16)

3.2.7. Implementation of Arithmetic Operations

When the compiler generates object code for arithmetic and logical expressions, most of the FORTRAN operations are performed by using inline instructions. The size or complexity of some operations can cause the compiler to generate calls to routines provided in the Extended FORTRAN library.

Multiplication and division involving complex variables and array elements are performed by library routines. Exponentiation operations are performed by a library routine, except for cases involving integer or real bases raised to an integer constant power, where inline multiplications are generated.

These library routines are completely described in the FORTRAN mathematical library reference manual, UP-8029 (current version).

3.3. ASSIGNMENT STATEMENTS

A value is assigned to a variable or an array element by executing an assignment statement. This value is the current value until the variable or array element is redefined. There are three possible assignment statements: the arithmetic and logical, which are described in this section, and the ASSIGN statement (3.3.2).

3.3.1. Arithmetic and Logical Assignment Statements

Format:

$$v = e$$

where:

v

Is any type of arithmetic variable or array element name for an arithmetic assignment statement; or v is a logical variable or array element name for a logical assignment statement.

e

Is any type of arithmetic expression for an arithmetic assignment statement; and e is a logical expression for logical assignment statements.

Description:

The arithmetic or logical assignment statement assigns a single value to a variable or array element. The = operator is read as "is replaced by" as in "AMR is replaced by 8.19" for AMR=8.19.

For all data types except logical, Table 3—3 demonstrates the conversion of the expression e to the type of the receiving variable represented by v. Combinations of arithmetic and logical types are illegal. No conversion takes place in logical evaluations except where e is logical*4 and v is logical*1; in this case, the high order three bytes of e are ignored. The conversions are accomplished by intrinsic functions (5.6.1).

Table 3—3. Assignment Statement Conversions

		e					
Data Types	Integer*2	Integer (integer*4)	Real (real*4)	Double Precision (real*8)	Complex (complex*8)	Complex*16	
v	Integer*2	None	*	*	*	*	*
	Integer (integer*4)	**	None	IFIX(e)	IFIX(SNGL(e))	IFIX(REAL(e))	IFIX(SNGL(DREAL(e)))
	Real (real*4)	***	FLOAT(e)	None	SNGL(e)	REAL(e)	SNGL(DREAL(e))
	Double precision	***	DFLOAT(e)	DBLE(e)	None	DBLE(REAL(e))	DREAL(e)
	Complex (complex*8)	***	CMPLX(FLOAT(e),0.0)	CMPLX(e,0.0)	CMPLX(SNGL(e),0.0)	None	CMPLX(SNGL(REAL(e)),SNGL(AIMAG(e)))
	Complex*16	***	DCMPLX(DFLOAT(e),0.0)	DCMPLX(DBLE(e),0.0)	DCMPLX(e,0.0)	DCMPLX(DBLE(REAL(e)),DBLE(AIMAG(e)))	None

*Processing for integer*2 is identical with that for integer, except that the high order 16 bits of integer*4 are truncated.

**The sign is extended.

***In these cases, e is treated as an integer*4.

NOTE:

See Table 5—3 for the definitions of these intrinsic functions.

3.3.2. ASSIGN Statement

Format:

ASSIGN k TO i

where:

k

Is a statement label in the same program unit as the ASSIGN statement and is the label of another executable statement or of a FORMAT statement.

i

Is the name of an integer*4 variable.

Description:

The ASSIGN statement permits an integer variable name to represent a statement label; the variable name can then be used in the assigned GO TO statement or as a reference to a FORMAT statement. Once the integer variable name has been assigned a value by the ASSIGN statement, it can be used for no other purpose until it is redefined. For instance, it cannot be used in an arithmetic expression unless its value is redefined by an arithmetic assignment statement or a READ statement.

4. Control Statements

4.1. GENERAL

Control statements are executable statements that modify the normal sequence of program execution. The control statements used in Extended FORTRAN are identical in function with those described in the "Control Statements" section of the fundamentals of FORTRAN reference manual, UP-7536 (current version).

4.2. ARITHMETIC IF

Format:

IF (e) k₁,k₂,k₃

where:

e

Is any integer, real, or double precision arithmetic expression; complex is not permitted.

k

Is a statement label in the same program unit as the arithmetic IF control statement.

Description:

If the arithmetic expression value is negative, control is passed to the statement with the k₁ statement label; if the value is zero, k₂ receives control; and if the value is positive, k₃ receives control. If any label is omitted, control is passed to the next executable statement following the IF statement when the conditions for the missing label are met. Trailing commas may be omitted when labels are not specified.

Note that the internal representation of real and double precision values is an approximation. One of these types could be stored internally as a nonzero approximation of zero.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT	
STATEMENT NUMBER			
5	IF(I-1), 10, 20	10	20
6	IF(X-Y), 15,		
7	IF(BETA-1.5), , 20		

Statement 5 indicates that control is to be transferred to the statement labeled 10 if I is less than 1, to the statement labeled 20 if I equals 1, or to the next statement following 5 if I is greater than 1.

Statement 6 transfers control to statement 15 if Y is greater than X; otherwise, control is transferred to the next executable statement.

Statement 7 transfers control to statement 20 when BETA is greater than 1.5.

4.3. LOGICAL IF

Format:

IF (e) s

where:

e

Is any logical expression (3.2.3).

s

Is any executable statement except a DO, END, or another logical IF statement.

Description:

The logical IF statement allows the execution of a statement to be dependent on the truth of a logical expression.

Examples:

```
IF(A .AND. B) GO TO 20
IF(C .GT. D) WRITE (10) C
```

If both A and B are .TRUE., the GO TO statement is executed, and control passes to statement 20. If either A or B is .FALSE., the GO TO statement is ignored and control is transferred to the next executable statement.

The WRITE statement in the second example is executed if the value represented by C is greater than that represented by D. Otherwise, control passes to the next executable statement below the IF statement.

4.4. UNCONDITIONAL GO TO

Format:

GO TO k

where:

k

Is the statement label of an executable statement in the same program unit.

Description:

The unconditional GO TO statement provides an unconditional transfer of control to the statement with the label specified.

4.5. COMPUTED GO TO

Format:

GO TO(k₁,k₂,...,k_n),i

where:

k

Is a statement label of an executable statement in the same program unit.

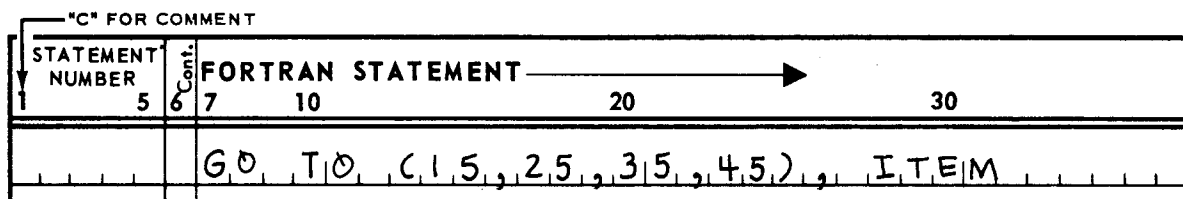
i

Is an integer variable that must be defined by using an arithmetic assignment or a READ statement before the execution of the GO TO control statement. The comma preceding i is optional.

Description:

The computed GO TO control statement permits the transfer of control to a label whose position in the GO TO control statement equals the value of an integer variable. For instance, if the value of the integer variable is 3, control is transferred to the third statement label in the computed GO TO control statement. If the integer variable is negative, is equal to 0, or is greater than the number of statement labels in the control statement, control is transferred to the next executable statement following the computed GO TO statement.

Example:



When the value of the integer variable ITEM is 4, control is transferred to statement 45; when the value of ITEM is 1, control is transferred to 15; and so on. Any value other than 1 through 4 results in control being transferred to the next executable statement following the GO TO statement.

4.6. ASSIGNED GO TO

Format:

GO TO $i, (k_1, k_2, \dots, k_n)$

where:

i

Is the name of 4-byte integer variable that must be defined by an ASSIGN statement.

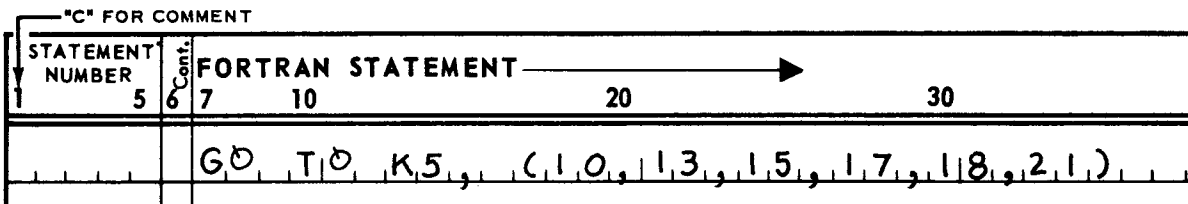
k

Is the statement label of an executable statement within the same program unit as the assigned GO TO control statement; the parenthesized list of labels and the preceding comma are optional and may be omitted. The list aids in defining the flow of control to the compiler. This list, therefore, aids the compiler in diagnosing errors and often provides significantly better code generation. When used, the label list must contain all possible destination labels.

Description:

The assigned GO TO control statement transfers program control to the statement labeled with the current value represented by an integer variable.

Example:



When the current value of the integer variable K5 is associated with one of the statement labels in parentheses, control is transferred to the system with that label. The value of the integer variable can have been defined only by the ASSIGN statement (3.3.2). When the list of statement labels (10, 13, 15, 17, 18, 21) is omitted from the assigned GO TO control statement, control is still transferred to the executable statement.

4.7. DO

Format:

DO n $i = m_1, m_2, m_3$

where:

n

Is the statement label of the terminal statement of the DO loop, which must follow the DO statement, but which cannot be another DO statement.

i

Is the control variable, which is an integer variable that may be referenced, but not redefined, within the DO range.

m₁

Is the initial parameter, the value of which is assigned to the control variable before the first execution of the DO loop; this value should be less than or equal to the value of m₂.

m₂

Is the terminal parameter; it is compared to the control variable after each execution of the DO loop; when the value of the control variable is greater than the value of m₂, the DO control statement is satisfied and control passes out of the DO range.

m₃

Is the incrementing parameter; its value is added to the control variable i after each execution of the DO loop and before the comparison of m₂ and the control variable i. When this parameter is omitted, 1 is assumed to be the increment value.

Description:

A DO control statement initiates and controls the repeated execution of the group of statements within the DO range, which extends from the first executable statement following the DO control statement to the terminal statement.

For a DO statement, the compiler generates two blocks of executable code:

- in the position of the DO statement, a block that defines the control variable to the value of the initial parameter, and
- between the terminal statement of the DO loop and the statement following it, a DO control block. Here, the control variable is incremented and tested, and program execution is resumed by either exiting or reentering the DO range.

If no control statements are present in the DO range, the loop will be executed

$$\frac{m_2 - m_1}{m_3} + 1$$

times by the actions of the DO control block. A control statement can prevent the execution of the DO control block; for example, the loop

"C" FOR COMMENT		FORTRAN STATEMENT		
STATEMENT NUMBER	Cont.	7	10	30
5	6	DO 10, I = 1, 10		
		:		
10		IF (A .GT. B) IF (C) 20, 30		

may be executed 10 times, unless the condition

$$(A \cdot GE \cdot B) \cdot AND \cdot (C \cdot GT \cdot 0 \cdot 0)$$

occurs, which would prevent the execution of the DO control block and cause premature loop exit.

Either integer constants or integer variable names may be used as parameters for the DO control statement; variables used as parameters may not be redefined within the DO range. The index variable of the DO loop should be considered to be undefined when the loop is exhausted, even though it is maintained in main storage during the loop.

4.7.1. Transfers of Control to and from a DO Range

In Extended FORTRAN programs, program control can always be transferred out of a DO loop without satisfying the DO control statement parameters. However, control can be transferred into a DO range only from the extended DO range, which consists of those statements executed between the transfer out of the innermost DO of a completely nested DO loop and the transfer back into the DO loop range. For a more complete explanation of the DO control statement, refer to the "Control Statements" section of the fundamentals of FORTRAN reference manual, UP-7536 (current version).

4.8. CONTINUE

Format:

CONTINUE

Description:

The CONTINUE control statement can serve as a terminal statement of a DO range. It produces no coding and may be used anywhere in the program, subject to the ordering in Table 1—2, without affecting the logical program execution. When used as the terminal statement of a DO range, the CONTINUE control statement must be labeled.

4.9. STOP

Formats:

STOP
STOP n
STOP 'a'

where:

n

Is a message in the form of an unsigned decimal integer constant of not more than five digits.

a

Is a message in the form of a literal of not more than 243 characters enclosed in apostrophes.

Description:

The STOP control statement terminates job step execution and returns program control to the operating system, indicating the logical end of the program. When a STOP n or a STOP 'a' is executed, a display is always produced at the job's diagnostic device (Section 11).

4.10. PAUSE

Formats:

PAUSE
PAUSE n
PAUSE 'a'

where:

n

Is a console message in the form of an unsigned decimal integer constant of not more than five digits.

a

Is a console message in the form of a literal of not more than 243 characters enclosed in apostrophes.

Description:

The PAUSE control statement halts execution of the program and produces a display at the system console. The operator has the choice of allowing the program to proceed to the next executable statement or to cancel the job.

4.11. END

Format:

END

Description:

The END control statement is an executable statement indicating the physical end of a program unit; it may not have a statement label. When the END statement is executed in a main program it is interpreted as a STOP control statement (4.9). When executed in a subroutine or function subprogram, the END statement is equivalent to a RETURN statement.



5. Functions and Subroutines

5.1. GENERAL

When a calculation or series of calculations is required repeatedly in a FORTRAN program, the statements used to perform the calculations can be coded once as a procedure. This procedure can be referenced each time the calculations are to be performed. Procedures, as explained here and described in the "Procedures and Procedure Subprograms" section of the fundamentals of FORTRAN reference manual, UP-7536 (current version), may be categorized by:

- whether the procedure coding is inserted inline by the compiler each time the procedure is referenced, or whether the procedure is compiled separately as a subprogram;
- whether the procedure is referenced by a subroutine CALL statement or by a function reference; and
- whether the procedure is written by the user or supplied with the FORTRAN library.

Table 5—1 lists the procedures and shows their relationships within these categories.

Table 5—1. Extended FORTRAN Procedures

Procedure	Coding Inline or Subprogram	Referenced By	Code Source
Statement function	Inline	Function reference	User
External function	Subprogram	Function reference	User
Intrinsic function	Inline*	Function reference	UNIVAC
Standard library function	Subprogram	Function reference	UNIVAC
Subroutine	Subprogram	CALL statement	User
Standard library subroutine	Subprogram	CALL statement	UNIVAC

*Some of the larger intrinsic functions are external subprograms. They are marked with ① in Table 5—3.

Functions are procedures referenced in expressions within FORTRAN statements. They always have at least one argument; they always return the value associated with their name when they are executed; and they return control to the expression within the referencing statement. The functions are:

- Statement functions
- External functions
- Intrinsic functions
- Standard library functions

Only statement functions and external functions are coded by the user.

Subroutines are procedures coded as subprograms; when they are referenced, control is transferred to the subroutine, it is executed, and control is then returned to the statement following the subroutine reference. Subroutines are either user-coded or supplied as standard library subroutines. Subroutines differ from functions in the method of referencing the procedure, in that multiple values or no value can be returned, and in the method by which control is returned to the referencing program unit.

Functions always transfer a value associated with the function name, but subroutines do not. When value transfers are made by subroutines, they are accomplished by redefining arguments or common storage. Arguments are included as part of the procedure definition; these are dummy arguments. Arguments are also specified in the procedure reference; these are actual arguments. Substitutions of actual for dummy arguments are made when the procedure is referenced at execution time.

The actual arguments in the procedure reference must correspond to the dummy arguments in the procedure definition. They must correspond in number, data type (except for literals), and order. The argument forms permitted for actual arguments in the user-coded procedures are shown in Table 5—2.

Table 5—2. Argument Forms

Form of Actual Arguments	Statement Functions	External Functions	Subroutines
Variable name	Yes	Yes	Yes
Expression	Yes	Yes	Yes
Function reference	Yes	Yes	Yes
Array element name	Yes	Yes	Yes
Array name	No	Yes	Yes
Literal constant	No	Yes	Yes
Label parameter (statement label preceded by &)	No	No	Yes
External procedure name	No	Yes	Yes

NOTE:

External procedure names appearing as actual arguments must be declared in an EXTERNAL statement (6.7) or referenced previously in a CALL statement or function reference.

5.2. PROCEDURE REFERENCE

Depending on whether the procedure is a function or a subroutine (Table 5—1), it is referenced by either the function reference or the subroutine CALL statement.

5.2.1. Function Reference

Statement functions, external functions, intrinsic functions, and standard library functions are all referenced in expressions with the general function reference format:

$$f(a_1, a_2, \dots, a_n)$$

where:

f

Is the symbolic name that was used to identify the user-coded function in its function definition, or that was supplied as the function name of an intrinsic or library function.

a

Represents an actual argument; at least one is required.

Actual arguments must agree in type, number, and order with the dummy arguments in the function definition, but actual argument types are not restricted by the data type of the function name. The forms permitted for actual arguments are shown in Table 5—2 for statement functions and external functions, in Table 5—3 for intrinsic functions, and in Table 5—4 for standard library functions.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
5					
	6				
		CZ=CBRT(SUZU)+CARA+YAM			
		MAICO=NORT**JAWA-INT(KS,ABL,RI)			

In the first statement, the standard library function CBRT is referenced. In the next line, a user-coded statement function, INT, is referenced, and three actual arguments are included in the function reference. An integer type value is returned to the referencing expression, although the actual arguments are both integer and real types.

5.2.2. Subroutine Reference (CALL Statement)

All subroutines, whether written by the user or supplied with the compiler, are referenced with the CALL statement.

Format:

$$\text{CALL } s(a_1, a_2, \dots, a_n)$$

where:

s

Is the symbolic name of the subroutine as defined by the user or as supplied with the standard library subroutines.

a

Is an actual argument. The use of a statement label preceded by an ampersand is allowed (5.4.2.2). The argument list is optional and must be enclosed in parentheses when used.

Description:

The CALL statement is used to transfer control to the subroutine specified by the name. The maximum number of actual arguments permitted is 255; the allowed forms are listed in Table 5-2.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT		
STATEMENT NUMBER	Cont.	7	10	20
5	6	CALL PGNUM		
		CALL ERROR(INER)		
		CALL SUB(X, Y, &10, FUNC, &20)		

Three subroutines are referenced by the calls in the example. In the first CALL statement, control is transferred to the subroutine PGNUM. When the next line is executed, the standard library subroutine ERROR is called; the actual argument INER is specified. The last line in the example references the subroutine SUB; among the arguments are two statement labels, &10 and &20, which provide an optional method of returning control from the subroutine explained in 5.4.2.2.

5.3. STATEMENT FUNCTION DEFINITION

The user-coded functions are the statement functions and the external functions. External functions are coded as subprograms, as described in 5.4. Statement functions, however, are user-coded procedures that are defined with only one FORTRAN statement. Statement functions require at least one argument and return one value to the referencing statement. They are referenced with the function reference described in 5.2.1. After the evaluation of the statement function, control is returned to the expression within the referencing statement.

Format:

$$f(a_1, a_2, \dots, a_n) = e$$

where:

f

Is the symbolic function name assigned to the procedure.

a

Is a dummy argument consisting of a variable name.

e

Is a limited arithmetic or logical expression.

5.4.1. External Functions

An external function is a user-coded function procedure requiring more than one FORTRAN statement for its definition. External functions require at least one argument and return at least one value to the referencing statement. They are referenced with the function reference (5.2.1). After evaluation of the external function, control is returned to the expression within the referencing statement, where computation continues, using the value associated with the function name.

An external function is defined by coding the required FORTRAN statements as a subprogram that begins with a FUNCTION statement (5.4.1.1) and ends with an END statement (4.11).

The external function returns a value of the type determined by the function or entry name, not by the data types of the arguments. The data type of the function name is decided by the first letter of the external function name (2.3), a type statement (6.4) in the same program unit as the FUNCTION declaration, or in the type specification in the FUNCTION statement.

Multiple entry into an external function is provided by the ENTRY statement (5.4.3).

5.4.1.1. FUNCTION Statement

Format:

t FUNCTION f*s (a₁,a₂,...,a_n)

where:

t

Is an optional type specification used to determine the data type of the symbolic name f, and therefore of the value returned by the external function; when this specification is omitted, the type is determined by a type statement in the same program unit or by the implicit type of the external function name. The permissible types are INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGIC.

f

Is the symbolic name identifying the procedure; routines supplied by Sperry Univac reserve the dollar sign as the third character of the function name. The name, or an ENTRY name, must be assigned a value by using a READ or assignment statement to define the function value.

***s**

Is an optional length specification for the symbolic function name (2.3). This option may be used only when the type option is used and the type specified is not DOUBLE PRECISION.

a

Is a dummy argument that may be a variable name, an array name, or a procedure name; variable names may be enclosed in slashes to use the call-by-name method of argument substitution (5.5.2).

Description:

The FUNCTION statement defines an external function and must be the first statement of the subprogram coded to define the external function.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
		INTEGER FUNCTION, XXI*2, (A)			
		:			
		RETURN			
		END			
		FUNCTION, YY1, (B, C, D, H)			
		:			
		RETURN			
		END			

In the examples, two external function subprograms are outlined. In the first, the value returned is defined as a 2-byte integer. The second subprogram returns a 4-byte real value unless the external function name YY1 is typed in the same program unit as another data type.

5.4.1.2. RETURN Statement

Format:

RETURN

Description:

The RETURN statement causes control to be transferred from the subprogram used to define the external function (or subroutine as explained in 5.4.2.2) to the program unit that referenced the subprogram.

5.4.1.3. ABNORMAL Statement

One of the functions of the SPERRY UNIVAC Operating System/3 Extended FORTRAN compiler is to increase efficiency by eliminating computational redundancies. In a statement sequence such as:

```
X = A*B+C
Y = FUNC(A)
Z = SIN(A*B)
```

the A*B is considered to be a common subexpression. The statements are usually evaluated as:

```
TEMP = A*B
X = TEMP+C
Y = FUNC(A)
Z = SIN(TEMP)
```

Other computational redundancies may be generated by the compiler and then eliminated while expanding the array element location function (Table 2—2). However, if the function FUNC redefines the value of its argument A, the reordering produces unexpected results. Functions that cause such undesirable side effects are known as abnormal functions and should be identified to the compiler.

A function is considered abnormal if it:

- redefines the value of an argument or of an entity declared in common storage (as discussed in the preceding paragraph);
- contains an input/output statement, such as a function that prints its results; or
- does not always produce the same function value when given identical arguments, such as a function that saves values between successive references.

Format:

ABNORMAL f_1, f_2, \dots, f_n

where:

f

Is the name of an abnormal function.

Description:

The ABNORMAL statement identifies abnormal functions.

When a program unit contains no ABNORMAL statement, all referenced functions are considered abnormal, except for the standard library functions (Table 5—4). In an ABNORMAL statement, all listed functions are considered abnormal; any other functions encountered are considered normal. An ABNORMAL statement without a list specifies that all functions are normal.

An abnormal function is permitted to cause side effects affecting other statements, as in the preceding example, but the function should not impact the same statement or expression. For example,

$$A(I) = I * F(I)$$

will, in general, cause unpredictable results if the function F is abnormal.

5.4.2. Subroutines

User-coded subroutines are procedures that, like external functions, are separately compiled as subprograms. Unlike function subprograms, however, subroutines:

- do not require arguments;
- do not necessarily return a value to the referencing program unit;
- have no data type associated with the subroutine name;
- are defined with the SUBROUTINE statement (5.4.2.1);
- are referenced with the CALL statement (5.2.2); and

- return control to the first executable statement after the CALL statement, or they can return control to a selected statement label in the referencing program unit (5.4.2.2).

Subroutines may have a maximum of 255 arguments; the argument forms permitted are shown in Table 5—2. Multiple entry into a subroutine is permitted (5.4.3). Subroutines are always considered to be abnormal.

5.4.2.1. SUBROUTINE Statement

Format:

SUBROUTINE s (a₁,a₂,...,a_n)

where:

s

Is a symbolic name identifying the subroutine. Avoid the use of the dollar sign as the third character of the subroutine name, since this convention is used in naming system routines. This name cannot appear elsewhere in the subprogram.

a

Is a dummy argument. The argument list is optional; when it is used, it is enclosed in parentheses. Each specification may be a variable name, an array name, a procedure name, or an asterisk. Variable names may be enclosed in slashes to specify the call-by-name method of argument substitution (5.5.2).

Description:

The SUBROUTINE statement defines the subroutine and must be the first statement of the subprogram.

An asterisk in the dummy argument list signifies that the corresponding actual argument is a label parameter preceded by an ampersand to provide an optional method of returning control to the referencing program unit.

5.4.2.2. Subroutine RETURN Statement

Format:

RETURN
or
RETURN i

where:

i

Is a positive integer constant or variable; this specification points to a label parameter in the actual argument list of the CALL statement.

Description:

The RETURN statement always returns control to the first executable statement following the CALL statement unless the optional integer specification is used. This option is not available when the RETURN statement is used to return control from an external function procedure.

5.4.3. Multiple Entry to Function and Subroutine Subprograms

Alternate entry points to external functions and subroutines are provided by the ENTRY statement.

Format:

ENTRY e (a₁,a₂,...,a_n)

where:

e

Is a symbolic name that identifies the procedure entry point.

a

Is a dummy argument corresponding to an actual argument, if any, in order, number, and type.

Description:

Arguments are optional for entry into a subroutine. At least one argument is required for entry into a function. Any dummy argument may be enclosed in slashes (5.5.2).

An ENTRY statement is nonexecutable and does not affect the normal sequence of statement execution. It defines only those formal arguments in its list; other formal arguments not defined by the ENTRY statement and used in the subprogram must have been defined by a previous reference to the subprogram. The value returned by an external function is the value type defined by the entry name. In a FUNCTION subprogram, the main storage address associated with the function name and all entry names is the same; thus, there is an implied equivalence between the names.

The following rules apply to the ENTRY statement:

1. Avoid the use of the dollar sign as the third character for the ENTRY name specification, since this is the convention for system routines.
2. In an external function subprogram, unpredictable results can occur when the entry name referenced is not assigned a value, or when the last entry name defined is of a data type different from the referenced entry name.
3. The same dummy arguments can be specified in more than one entry; the number of dummy arguments may differ at different entry points.
4. An ENTRY into an external function subprogram must specify at least one argument.
5. Only those arguments specified in the argument list of an ENTRY statement are initialized; other arguments are retained from previous function or entry references. Either the function name or at least one entry name must be assigned a value in the function subprogram.
6. The asterisk must not be used as a dummy argument in an ENTRY statement of a function.
7. A procedure subprogram, whether an external function or a subroutine, must not reference itself or any of its entry points.
8. An entry name may not be used in a statement function expression within the same subprogram.

5.5.1. Call by Value

The call-by-value method of argument substitution is the standard method of argument substitution when the dummy arguments in SUBROUTINE, FUNCTION, and ENTRY statements are simple variables. For the procedure reference

```
CALL A(B, C, D)
```

and the procedure definition

```
SUBROUTINE A (X, Y, Z)
```

the compiler generates a calling sequence for the CALL or FUNCTION reference, and a prologue for the SUBROUTINE, FUNCTION, or ENTRY statement. The calling sequence consists of a transfer of control to the start of the procedure and a list of main storage addresses where the actual arguments may be found. The prologue contains instructions that perform the argument substitution. In the preceding example, the prologue performs actions analogous to the FORTRAN statements $X = B$, $Y = C$, and $Z = D$.

This technique allows the dummy arguments to be referenced in the procedure body as though they were simple variables local to the procedure. When a RETURN statement is encountered, an epilogue is executed. An epilogue is a coding sequence that transmits the values of the dummy arguments to the calling program if they were redefined; thus, statements analogous to $B = X$, $C = Y$, and $D = Z$ are executed. The compiler generates a prologue and an epilogue for each SUBROUTINE, FUNCTION, and ENTRY statement. The RETURN statement causes the execution of the epilogue associated with the last prologue that was executed. Thus, in the following example, the subroutine on the left is treated as though it were written like the subroutine on the right.

SUBROUTINE A(B)		SUBROUTINE A	;PROLOGUE START
.		B = actual argument	
.		ASSIGN 100000 TO I	
.		GO TO 100001	;PROLOGUE END
.	100000	actual argument = B	;EPILOGUE START
.		RETURN	;EPILOGUE END
.	100001	CONTINUE	
.		.	
ENTRY C(D)		GO TO 100002	;JUMP OVER ENTRY STATEMENT
.		ENTRY C	;PROLOGUE START
.		D = actual argument	
.		ASSIGN 100003 TO I	
.		GO TO 100002	;PROLOGUE END
.	100003	actual argument = D	;EPILOGUE START
.		RETURN	;EPILOGUE END
.	100002	CONTINUE	
.		.	
RETURN		GO TO I (100000, 100003)	

5.5.2. Call by Name

The call-by-name method of argument substitution is the standard method of argument substitution when the dummy arguments in SUBROUTINE, FUNCTION, or ENTRY statements are declared to be arrays or procedure names. In these cases, the prologue copies the address of the actual argument into the procedure. Thereafter, the code generated for the array references in the procedure must retrieve the address of the array prior to accessing the array for computational purposes. See 6.2.1 for additional information on array declarator processing. As an option, the user may specify this method of argument substitution for simple variables by enclosing the dummy argument in slashes:

```
SUBROUTINE A (B,/C/,D)
```

In most cases, the choice is arbitrary, but special cases exist and can cause differing results:

```
CALL SQUARE (B,B)
.
.
SUBROUTINE SQUARE (X,Y)
.
.
X = X**2
Y = Y**2
```

Here, the introduction of slashes around X and Y will cause different results.

5.5.3. Symbolic Substitution

Symbolic substitution is the only method used for argument substitution in statement functions. This consists of the direct replacement of the statement function reference with the function expression; for example:

```
ISF(A,B,C) = 3.14159*A + B/C
.
.
90 W = ISF(X,Y,Z)/100.0
```

Statement 90 is interpreted as:

```
90 W = IFIX (3.14159*X + Y/Z)/100.0
```

This method of statement function evaluation produces faster code, and usually requires less space than a procedure containing an epilogue and a prologue. If a program contains a large number of references to a statement function, the user may choose to define an external function subprogram to save main storage space. The substituted expression is converted, if necessary, to the type of the statement function name (intrinsic function IFIX, Table 5—3).

5.6. LIBRARY PROCEDURES

The three classes of procedures provided by Sperry Univac that are available to the FORTRAN programmer are:

- Intrinsic functions, invoked with a function reference and usually associated with highly machine-dependent procedures or non-FORTRAN capabilities, such as processing a variable-length argument list (5.6.1).
- Standard library functions, invoked with a function reference and provided for evaluation of common mathematical functions in the areas of trigonometry, logarithms, roots, etc. While these procedures could be written in the FORTRAN language, Sperry Univac provides them in a library (in assembly language output form) in order to optimize accuracy, size, and performance (5.6.2).

- Standard library subroutines, invoked with the CALL statement. They are associated with the operating environment of the program and perform functions such as checking external switches, loading overlay phases, etc. (5.6.3).

Extended FORTRAN provides nearly 100 intrinsic and standard library functions, many highly similar; for example, six functions are provided to determine the absolute value of an argument, differing only in the types of their arguments and function values.

To reduce the difficulty of remembering so many names and the risk of clerical errors in programming, Extended FORTRAN provides generic function reference. These similar functions can be referenced by a single name called the generic name which, in this case, is ABS. Existing programs can reference the library using the member names of the generic class (ABS, IABS, JABS, DABS, CABS, and CDABS).

The names and properties of these functions are known to the compiler. When a function reference using a generic name is encountered, the compiler generates a reference to the proper member of the generic set by examining the types of the arguments. Generic reference is not provided for library subroutines or for user-coded procedures.

5.6.1. Intrinsic Functions

The intrinsic functions supplied with the compiler are listed in Table 5—3. Intrinsic functions are referenced with the function reference described in 5.2.1. After evaluation of the function, the function value is returned to the referencing statement at the expression containing the function reference.

Table 5—3. Intrinsic Functions (Part 1 of 2)

Generic Name	Use	Number Arguments	Member Function Name	Member Argument Type	Member Function Type
ABS	Determine the absolute value of the argument	1	ABS IABS JABS DABS	Real*4 Integer*4 Integer*2 Double precision	Real*4 Integer*4 Integer*2 Double precision
{ ABS } { CABS }	Determine the absolute value of the argument	1	CABS* CDABS*	Complex*8 Complex*16	Real*4 Double precision
AINT	Truncation; eliminate the fractional portion of argument	1	AINT DINT	Real*4 Double precision	Real*4 Double precision
INT	Truncation; eliminate the fractional portion of argument	1	INT IDINT	Real*4 Double precision	Integer*4 Integer*4
MOD	Remaindering; defined as $a_1 - [x] a_2$, where $[x]$ is the greatest integer whose magnitude does not exceed the magnitude of a_1/a_2 and whose sign is the same as a_1/a_2	2 (Argument 2 must be nonzero.)	JMOD AMOD MOD DMOD	Integer*2 Real*4 Integer*4 Double precision	Integer*2 Real*4 Integer*4 Double precision

Table 5-3. Intrinsic Functions (Part 2 of 2)

Generic Name	Use	Number Arguments	Member Function Name	Member Argument Type	Member Function Type
{ MAX } { MAX0 }	Select the largest value	≥ 2	JMAX0 ① AMAX0 ① AMAX1 ② { MAX ① } { MAX0 ① } MAX1 ② DMAX1 ①	Integer*2 Integer*4 Real*4 Integer*4 Real*4 Double precision	Integer*2 Real*4 Real*4 Integer*4 Integer*4 Double precision
{ MIN } { MIN0 }	Select the smallest value	≥ 2	JMIN0 ① AMIN0 ② AMIN1 ① { MIN* ① } { MIN0** ② } MIN1 ① DMIN1 ①	Integer*2 Integer*4 Real*4 Integer*4 Real*4 Double precision	Integer*2 Real*4 Real*4 Integer*4 Integer*4 Double precision
	Convert argument from integer to real or double precision	1	FLOAT ② DFLOAT ② HFLOAT ② DHFLOT ②	Integer*4 Integer*4 Integer*2 Integer*2	Real*4 Double precision Real*4 Double precision
	Convert argument from real to integer	1	IFIX ② HFIX ②	Real*4 Real*4	Integer*4 Integer*2
SIGN	Replace the algebraic sign of the first argument with the sign of the second argument	2	JSIGN SIGN ISIGN DSIGN	Integer*2 Real*4 Integer*4 Double precision	Integer*2 Real*4 Integer*4 Double precision
DIM	Positive difference; subtract the smaller of the two arguments from the first argument	2	JDIM DIM IDIM DDIM	Integer*2 Real*4 Integer*4 Double precision	Integer*2 Real*4 Integer*4 Double precision
SNGL	Convert double precision to real	1	SNGL CSNGL	Double precision Complex*16	Real*4 Complex*8
REAL	Get real part of a complex number	1	REAL DREAL	Complex*8 Complex*16	Real*4 Double precision
AIMAG IMAG	Get imaginary part of a complex number	1	IMAG, AIMAG DIMAG	Complex*8 Complex*16	Real*4 Double precision
DBLE	Convert from real to double precision	1	DBLE CDBLE	Real*4 Complex*8	Double precision Complex*16
CMPLX	Convert two real arguments to a complex number	2	CMPLX DCMPLX	Real*4 Double precision	Complex*8 Complex*16
CONJG	Get conjugate of a complex number	1	CONJG DCONJG	Complex*8 Complex*16	Complex*8 Complex*16

① This function is an external procedure supplied in the Extended FORTRAN library.

② This function is accessible only through its member name.

5.6.2. Standard Library Functions

The standard library functions (Table 5—4) are function subprograms supplied with the compiler. They are accessed with a function reference (5.2.1) and return control to the referencing program unit within the expression of the referencing statement. Detailed information on performance, size, and mathematical methods is available in the Series 90 FORTRAN mathematical library programmer reference manual, UP-8029 (current version).

5.6.2.1. Specification Statement Interaction

This section details the effects of listing the name of an intrinsic or standard library function in an **ABNORMAL**, **EXTERNAL**, or type statement.

- If the generic name of an intrinsic function is specified in an **EXTERNAL** statement, generic reference to that function is no longer possible, and the function is treated as a user-provided function. Members of the generic class whose names differ from the generic name are available for specific reference. If the procedure name appears in an actual argument list, that name is transmitted to the referenced procedure.

For example, if the name **ABS** appears in an **EXTERNAL** statement, any function reference to **ABS**, regardless of the type or number of arguments, is presumed to be a reference to a user **ABS** function. An explicit reference to **DABS** is still recognized as an intrinsic function reference.

When the procedure name **ABS** appears in an actual argument list, the name **ABS** is transmitted.

- If a member name of an intrinsic function is specified in an **EXTERNAL** statement, generic reference is still possible to the class to which it belongs, except for the cited member, which is no longer available. In all procedure references and argument lists, the name is assumed to be a user-supplied procedure.

For example, if **DABS** is mentioned in an **EXTERNAL** statement, references to **ABS** with integer or **real*4** arguments are still permitted. An **ABS** reference with a double precision argument will cause a diagnostic message to be printed during compilation, and the function will not be evaluated. **DABS** is presumed to be a user-supplied procedure.

- If the generic name or a member name of a standard library function appears in an **EXTERNAL** statement, complete generic or specific reference is permitted. If the name appears in an argument list, that particular name is transmitted. For example, if **LOG** is cited as **EXTERNAL**, **LOG** function references cause either **ALOG**, **DLOG**, **CLOG**, or **CDLOG** to be invoked. When it appears as a procedure name in an actual argument list, the name **LOG** is transmitted.
- If the generic name or a member name of an intrinsic or standard library function appears in a type statement with an initial data declaration, the name is considered to be a simple variable. The name may never be followed by a left parenthesis, as in a function reference. If it is a member name that is different from the generic name, generic function reference is permitted to invoke all other members of the generic class.
- If the generic name or a member name of an intrinsic or standard library function appears in a type statement without an initial data declaration and the type does not conflict with the function type of a member name in Table 5—2 or 5—3, then the type declaration has no effect, since the types of these functions are already known to the compiler.
- If the generic name of an intrinsic or standard library function appears in an **ABNORMAL** statement, or if it appears in a type statement that conflicts with the function type of a member name in Table 5—2 or 5—3, it is considered to be a user-supplied function. Specific reference to other members of the same generic class is still permitted.
- If a member name of an intrinsic or standard library function appears in an **ABNORMAL** statement, or if it appears in a type statement that conflicts with its function type as specified in Table 5—2 or 5—3, it is considered to be a user-supplied function. Generic reference to that generic class is permissible as long as it does not cause the cited member to be invoked; a diagnostic is issued when this occurs, and no function reference is generated.

Table 5-4. Standard Library Functions (Part 1 of 5)

General Operation	Generic Name	Member Name	Mathematical Definition	Argument			Function Value Type and Range
				Number	Type	Range	
Trigonometric	SIN	SIN	$y = \sin(x)$	1	real+4 (in radians)	$ x < (2^{18}.\pi)$	real+4 $-1 \leq y \leq 1$
		DSIN		1	real+8 (in radians)	$ x < (2^{50}.\pi)$	real+8 $-1 \leq y \leq 1$
		CSIN	$y = \sin(z)$	1	complex+8 (in radians)	$ x_1 < (2^{18}.\pi)$ $ x_2 \leq 174.673$	complex+8 $-M \leq y_1, y_2 \leq M$
		CDSIN		1	complex+16 (in radians)	$ x_1 < (2^{50}.\pi)$ $ x_2 \leq 174.673$	complex+16 $-M \leq y_1, y_2 \leq M$
	COS	COS	$y = \cos(x)$	1	real+4 (in radians)	$ x < (2^{18}.\pi)$	real+4 $-1 \leq y \leq 1$
		DCOS		1	real+8 (in radians)	$ x < (2^{50}.\pi)$	real+8 $-1 \leq y \leq 1$
		CCOS	$y = \cos(z)$	1	complex+8 (in radians)	$ x_1 < (2^{18}.\pi)$ $ x_2 \leq 174.673$	complex+8 $-M \leq y_1, y_2 \leq M$
		CDCOS		1	complex+16 (in radians)	$ x_1 < (2^{50}.\pi)$ $ x_2 \leq 174.673$	complex+16 $-M \leq y_1, y_2 \leq M$
	TAN	TAN	$y = \tan(x)$	1	real+4 (in radians)	$ x < (2^{18}.\pi)$	real+4 $-M \leq y \leq M$
		DTAN		1	real+8 (in radians)	$ x < (2^{50}.\pi)$	real+8 $-M \leq y \leq M$
	{ COTAN COT }	{ COTAN COT }	$y = \cotan(x)$	1	real+4 (in radians)	$ x < (2^{18}.\pi)$	real+4 $-M \leq y \leq M$
		{ DCOTAN DCOT }		1	real+8 (in radians)	$ x < (2^{50}.\pi)$	real+8 $-M \leq y \leq M$

NOTES:

1. $M = 16^{63} \cdot (1 - 16^{-6})$ for real+4 and $16^{63} \cdot (1 - 16^{-14})$ for real+8
2. z is a complex number of the form $x_1 + x_2 i$

Table 5-4. Standard Library Functions (Part 2 of 5)

General Operation	Generic Name	Member Name	Mathematical Definition	Argument			Function Value Type and Range	
				Number	Type	Range		
Trigonometric (cont.)	{ ASIN ARSIN }	{ ASIN ARSIN }	$y = \arcsin(x)$	1	real*4	$ x \leq 1$	real*4 (in radians) $-\pi/2 \leq y \leq \pi/2$	
		{ DASIN DARSIN }		1	real*8	$ x \leq 1$	real*8 (in radians) $-\pi/2 \leq y \leq \pi/2$	
	{ ACOS ARCOS }	{ ACOS ARCOS }	$y = \arccos(x)$	1	real*4	$ x \leq 1$	real*4 (in radians) $0 \leq y \leq \pi$	
		{ DACOS DARCOS }		1	real*8	$ x \leq 1$	real*8 (in radians) $0 \leq y \leq \pi$	
	ATAN	ATAN	$y = \arctan(x)$	1	real*4	any real argument	real*4 (in radians) $-\pi/2 \leq y \leq \pi/2$	
		DATAN		1	real*8	any real argument	real*8 (in radians) $-\pi/2 \leq y \leq \pi/2$	
	ATAN2	ATAN2	$y = \arctan\left(\frac{x_1}{x_2}\right)$	2	real*4	any real arguments except (0,0)	real*4 (in radians) $-\pi \leq y \leq \pi$	
		DATAN2		2	real*8	any real arguments except (0,0)	real*8 (in radians) $-\pi \leq y \leq \pi$	
	Hyperbolic	SINH	SINH	$y = \frac{e^x - e^{-x}}{2}$	1	real*4	$ x < 175.366$	real*4 $-M \leq y \leq M$
			DSINH		1	real*8	$ x < 175.366$	real*8 $-M \leq y \leq M$
CSINH			$y = \frac{e^z - e^{-z}}{2}$	1	complex*8	$ x_1 \leq 174.673$ $ x_2 < 2^{18} \cdot \pi$	complex*8 $-M \leq y_1, y_2 \leq M$	
CDSINH				1	complex*16	$ x_1 \leq 174.673$ $ x_2 < 2^{50} \cdot \pi$	complex*16 $-M \leq y_1, y_2 \leq M$	
COSH		COSH	$y = \frac{e^x + e^{-x}}{2}$	1	real*4	$ x < 175.366$	real*4 $-M \leq y \leq M$	
		DCOSH		1	real*8	$ x < 175.366$	real*8 $-M \leq y \leq M$	



Table 5-4. Standard Library Functions (Part 3 of 5)

General Operation	Generic Name	Member Name	Mathematical Definition	Argument			Function Value Type and Range
				Number	Type	Range	
Hyperbolic (cont.)	COSH	CCOSH	$y = \frac{e^z + e^{-z}}{2}$	1	complex*8	$ x_1 \leq 174.673$ $ x_2 < 2^{18} \cdot \pi$	complex*8 $-M \leq y_1, y_2 \leq M$
		CDCOSH		1	complex*16	$ x_1 \leq 174.673$ $ x_2 < 2^{50} \cdot \pi$	complex*16 $-M \leq y_1, y_2 \leq M$
	TANH	TANH	$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	1	real*4	any real argument	real*4 $-1 \leq y \leq 1$
		DTANH		1	real*8	any real argument	real*8 $-1 \leq y \leq 1$
Exponential	EXP	EXP	$y = e^x$	1	real*4	$x \geq -180.218$ $x \leq 174.673$	real*4 $0 \leq y \leq M$
		DEXP		1	real*8	$x \geq -180.218$ $x \leq 174.673$	real*8 $0 \leq y \leq M$
		CEXP	$y = e^z$	1	complex*8	$x_1 \leq 174.673$ $ x_2 < (2^{18} \cdot \pi)$	complex*8 $-M \leq y_1, y_2 \leq M$
		CDEXP		1	complex*16	$x_1 \leq 174.673$ $ x_2 < (2^{50} \cdot \pi)$	complex*16 $-M \leq y_1, y_2 \leq M$
Base 10 Exponential	EXP10	EXP10	$y = 10^x$	1	real*4	$x \geq -180.216/\ln(10)$ $x \leq 174.673/\ln(10)$	real*4 $0 \leq y \leq M$
		DEXP10		1	real*8	$x \geq -180.216/\ln(10)$ $x \leq 174.673/\ln(10)$	real*8 $0 \leq y \leq M$
Natural logarithm	{ ALOG LOG }	{ ALOG LOG }	$y = \log_b x$ or $y = \ln(x)$	1	real*4	$x > 0$	real*4 $y \geq -180.218$ $y \leq 174.673$
		DLOG		1	real*8	$x > 0$	real*8 $y \geq -180.218$ $y \leq 174.673$



Table 5-4. Standard Library Functions (Part 4 of 5)

General Operation	Generic Name	Member Name	Mathematical Definition	Argument			Function Value Type and Range
				Number	Type	Range	
Natural logarithm	{ ALOG } { LOG }	CLOG	$y = PV \log_e(z)$ PV is principle value, which means y_2 between $-\pi$ and π .	1	complex*8	$z \neq 0 + 0i$	complex*8 $y = y_1 + y_2 i$ $y_1 \geq -180.218$ $y_1 \leq 175.021$ $-\pi \leq y_2 \leq \pi$
		CDLOG		1	complex*16	$z \neq 0 + 0i$	complex*16 $y = y_1 + y_2 i$ $y_1 \geq -180.218$ $y_1 \leq 175.021$ $-\pi \leq y_2 \leq \pi$
Common logarithm	{ ALOG10 } { LOG10 }	{ ALOG10 } { LOG10 }	$y = \log_{10} x$	1	real*4	$x > 0$	real*4 $y \geq -78.268$ $y \leq 75.859$
		DLOG10		1	real*8	$x > 0$	real*8 $y \geq -78.268$ $y \leq 75.859$
Square root	SQRT	SQRT	$y = \sqrt{x}$ or $y = x^{1/2}$	1	real*4	$x \geq 0$	real*4 $0 \leq y \leq M^{1/2}$
		DSQRT		1	real*8	$x \geq 0$	real*8 $0 \leq y \leq M^{1/2}$
		CSQRT	$y = \sqrt{z}$ or $y = z^{1/2}$	1	complex*8	any complex argument	complex*8 $0 \leq y_1 \leq 1.0987 (M^{1/2})$ $ y_2 \leq 1.0987 (M^{1/2})$
		CDSQRT		1	complex*16	any complex argument	complex*16 $0 \leq y_1 \leq 1.0987 (M^{1/2})$ $ y_2 \leq 1.0987 (M^{1/2})$
Cube root	CBRT	CBRT	$y = x^{1/3}$	1	real*4	any real argument	real*4 $-M^{1/3} \leq y \leq M^{1/3}$
		DCBRT		1	real*8	any real argument	real*8 $-M^{1/3} \leq y \leq M^{1/3}$

Table 5-4. Standard Library Functions (Part 5 of 5)

General Operation	Generic Name	Member Name	Mathematical Definition	Argument			Function Value Type and Range
				Number	Type	Range	
Distribution	ERF	ERF	$y = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$	1	real*4	any real argument	real*4 $-1 \leq y \leq 1$
		DERF		1	real*8	any real argument	real*8 $-1 \leq y \leq 1$
	ERFC	ERFC	$y = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-u^2} du$	1	real*4	any real argument	real*4 $0 \leq y \leq 2$
		DERFC		1	real*8	any real argument	real*8 $0 \leq y \leq 2$
	GAMMA	GAMMA	$y = \int_0^{\infty} u^{x-1} e^{-u} du$	1	real*4	$x > 2^{-252}$ and $x < 57.5744$	real*4 $0.88560 \leq y \leq M$
		DGAMMA		1	real*8	$x > 2^{-252}$ and $x < 57.5744$	real*8 $0.88560 \leq y \leq M$
	ALGAMA LGAMMA	ALGAMA	$y = \log_e \Gamma(x) \text{ or}$ $y = \log_e \int_0^{\infty} u^{x-1} e^{-u} du$	1	real*4	$x > 0$ and $x < 4.2913 \cdot 10^{73}$	real*4 $-0.12149 \leq y \leq M$
		DLGAMA		1	real*8	$x > 0$ and $x < 4.2913 \cdot 10^{73}$	real*8 $-0.12149 \leq y \leq M$



5.6.3. Standard Library Subroutines

The standard library subroutines are procedures available in subprograms supplied with the compiler. These subroutines are invoked by the CALL statement, and control is returned to the main program at the first executable statement immediately following the CALL statement. All of the standard library subroutines may be overridden; a user may supply his own routine with any of the FORTRAN names, such as OVERFL, ERROR, etc. Such routines must be included with an INCLUDE control card at the time the program is linked. Note that each library name has a \$ generated as the second character (e.g., O\$ERFL and E\$ROR).

The subroutines provided by Extended FORTRAN are presented here and summarized in Table 5—5.

■ Arithmetic Overflow and Underflow Test (OVERFL)

The overflow check subroutine, OVERFL, informs the program when computational results are not within the maximum or minimum magnitude permitted for a value. A real computation always yields a correct fraction, but the exponent is incorrect by 128 for an overflow and by -128 for an underflow. An overflow during an integer computation yields unpredictable results. An overflow or underflow causes a program check interrupt; when this occurs, various switches are set and program execution resumes at the next instruction, which may be in the same FORTRAN statement. These switches are interrogated by the OVERFL subroutine:

CALL OVERFL (i)

where:

i

Is an integer*4 variable.

The variable is assigned a value of 1, 2, or 3 to indicate the status of the interrupt switches.

The OVERFL subroutine operates in three separate modes for compatibility with other FORTRAN systems:

— Extended FORTRAN Mode

Integer and real overflow and real underflow are monitored. Only the last event, either overflow or underflow, is reflected in the interrupt switches. The i values assigned are:

- 1 — An overflow interrupt has occurred. A previous underflow interrupt will not be reported, and the overflow/underflow interrupt switch is reset.
- 2 — Neither overflow nor underflow has occurred.
- 3 — An underflow interrupt has occurred. A previous overflow interrupt will not be reported, and the overflow/underflow interrupt switch is reset.

Integer overflows are reported only if the // OPTION BOF is in the job control stream of the executable program.

For example, the statements

```
X = (10E75*10E75) + (10E-75*10E-75)
CALL OVERFL (I)
CALL OVERFL (J)
```

set the value of I to 3 and J to 2, indicating, respectively, that an underflow was the last interrupt and that there are no conditions to report. If the arithmetic statement is written as

```
X = (10E-75*10E-75)+(10E75*10E75)
```

I has the value 1, indicating that an overflow was the last event.

— SPERRY UNIVAC Series 70 Mode

Integer and real overflow and real underflow are monitored independently. The *i* values assigned are:

- 1 — An overflow has occurred. The overflow switch is reset. OVERFL should be entered again to determine if an underflow has also occurred.
- 2 — Neither overflow nor underflow has occurred.
- 3 — An underflow has occurred. The underflow interrupt switch is reset.

→ The module FL\$OVW70 must be included with an INCLUDE control card during linkage editing and the // OPTION BOF must be specified in the job control stream.

For example, the statements

```
X = (10E75*10E75)+(10E-75*10E-75)
CALL OVERFL (I)
CALL OVERFL (J)
CALL OVERFL (K)
```

set the value of *I* to 1, *J* to 3, and *K* to 2, indicating, respectively, an overflow, an underflow, and that there are no conditions to report.

— IBM System 360/370 Mode

Real overflow and underflow are monitored, but integer overflow is ignored. The *i* values assigned are identical with those for the Extended FORTRAN mode.

The desired mode of operation is selected when the executable program is linked and executed. Selection of IBM mode causes the DVCHK subroutine to ignore integer division by 0.

■ Divide Check Subroutine (DVCHK)

The divide check subroutine, DVCHK, informs the program when an integer or real division by 0 occurs or an integer result of a division exceeds $\pm 2,147,483,647$. In both cases, an indicator is set, and the computation yields the original dividend. This indicator is interrogated with the statement:

```
CALL DVCHK (i)
```

where:

i
Is an integer*4 variable.

The values assigned to *i* by DVCHK are:

- 1 — when a divide check has occurred. The indicator is reset.
- 2 — when a divide check has not occurred.

Integer divide checks are reported only if the job control statement // OPTION BOF is present in the control stream of the executable program.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Column	7	10	20	30
		CALL DVCHK(I)			
		GO TO (10,20), I			
10		STOP 'TERMINATION ON DVCHK'			
20		CONTINUE			

If a division by 0 was attempted ($I = 1$), program control is transferred to statement 10; otherwise, control goes to statement 20.

- Error Indicator Test (ERROR)

This standard library subroutine tests an indicator to determine if a function error condition or an I/O ERR exit has occurred:

CALL ERROR (i)

where:

i

Represents an integer*4 variable.

The integer variable is assigned the following values:

- 1 — if a function error condition exists after a reference to a standard library function (Table 5-4) or to the ERROR1 subroutine;
- 2 — if no function or I/O error exists;
- 3 — if an ERR exit was taken from an I/O statement because of a data transmission error;
- 4 — if an ERR exit was taken from an I/O statement because of improper data; and
- 5 — if an ERR exit was taken because of an unrecoverable I/O error. No further references to the file are permitted.

A subsequent call of the ERROR subroutine, prior to additional I/O or function references, always returns a value of 2.

- Error Indicator Setting Subroutine (ERROR1)

This subroutine is used in conjunction with the ERROR subroutine; CALL ERROR1 sets the function error indicator tested by the ERROR subroutine. This is also performed by the standard library functions. The reference to the ERROR1 subroutine is:

CALL ERROR1

Example:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER					
5	6	7	10	20	30
		Z = XRAY (Q)			
		CALL ERROR (I)			
		GO TO (30, 40), I			
40		CONTINUE			
30		error condition routine			
		FUNCTION XRAY (B)			
		IF (B) 10, 20, 10			
20		CALL ERROR 1			
		RETURN			
10		CONTINUE			

Indicator Setting Subroutine (SLITE)

The SLITE standard library subroutine sets or resets one or more of four indicators internal to the subprogram. This subroutine is used with the SLITET subroutine, which tests these indicators. The format of the CALL statement is:

CALL SLITE (e)

where:

e

Is an integer expression. The value of the expression determines the indicator settings:

0 — if all four indicators are to be reset.

1, 2, 3, or 4 — to set the corresponding sense indicator.

-1, -2, -3, or -4 — to reset the corresponding indicator.

- Indicator Testing Subroutine (SLITET)

The SLITET subroutine tests the indicators controlled by the SLITE subroutine. The format of the CALL statement is:

CALL SLITET (e,i)

where:

e

Is an integer expression with a value corresponding to the sense indicator to be tested.

i

Is an integer variable name returning the results of the test.

If the indicator specified by e is set, the integer variable i is set to 1. If the indicator is not set, or if e is outside the range $1 \leq e \leq 4$, then i is set to 2. Execution of the SLITET subroutine does not affect the indicator settings.

- Control Information Check (SSWTCH)

The SSWTCH standard library subroutine allows the FORTRAN programmer to check control information during program execution. This control information is provided prior to execution of the program on a // SET UPSI job control card used in the operating system.

The format of the CALL statement is:

CALL SSWTCH (e,i)

where:

e

Is an integer expression with a value of 1 through 4, representing a binary switch position.

i

Is the integer variable name used to return the result of the switch position test.

If the specified binary switch is set, the variable has the value 1; otherwise, its value is 2. Execution of the SSWTCH subroutine does not alter the switch settings.

- Main Storage Dump Routines (DUMP and PDUMP)

These subroutines cause a dump or listing of the main storage assigned to the program; the subroutines are described in Section 10.

- EXIT Subroutine

The EXIT standard library subroutine terminates the program. The CALL EXIT statement is equivalent to the FORTRAN STOP statement (4.9).

↓

- FETCH Subroutine

The FETCH subroutine loads an executable program and transfers control to its transfer address. Processing in the calling program is not resumed. An I/O error during the load causes immediate job termination. The CALL statement has the format:

CALL FETCH (s)

where:

s

Is a load module name which must be either a 6- or 8-character name enclosed in apostrophes, or a double precision or complex variable containing a load module name.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT					
STATEMENT NUMBER	5	6	7	10	20	30	40
			:				
			DOUBLE PRECISION	DNAME	/'LOADMX' /		
			CALL	FETCH	(DNAME)		
			:				
			CALL	FETCH	('LOADMX')		

↑

The two calls of the FETCH standard library subroutine are equivalent.

- LOAD Subroutine

The LOAD standard library subroutine loads subprogram overlays. Control is not transferred to the subprogram but returns to the statement immediately following the CALL statement requesting the overlay. An I/O error during the load causes immediate job termination. The loaded subprogram cannot share the same main storage addresses as the procedure containing this CALL statement.

The format of the CALL statement is:

CALL LOAD (s)

where:

s

Is a phase name that must be an 8-character name enclosed in apostrophes, or a double precision or complex variable containing a phase name.

- OPSYS Subroutine

The OPSYS subroutine loads subprogram overlays and transfers control to the statement following the CALL statement.

The format of the CALL statement is:

CALL OPSYS ('LOAD',s)

where:

s

Is a phase name that must be an 8-character name enclosed in apostrophes, or a double precision or complex variable containing a phase name.

This statement is equivalent to the CALL LOAD (s) statement.

Table 5-5. Standard Library Subroutines

Subroutine	Format	Use
OVERFL	CALL OVERFL (i)	Tests for overflow or underflow.
DVCHK	CALL DVCHK (i)	Tests for invalid division.
ERROR	CALL ERROR (i)	Tests for function or I/O error conditions.
ERROR1	CALL ERROR1	Sets the function error indicator.
SLITE	CALL SLITE (e)	Sets the sense indicators specified.
SLITET	CALL SLITET (e,i)	Tests for the setting of specified sense indicators.
SSWTCH	CALL SSWTCH (e,i)	Tests the binary switch specified by the integer expression and returns a value in the integer variable name.
DUMP	CALL DUMP (list)	Dumps main storage assigned to the program; program execution terminates.
PDUMP	CALL PDUMP (list)	Dumps main storage assigned to the program; program execution continues.
EXIT	CALL EXIT	Terminates the program.
FETCH	CALL FETCH (s)	Loads and transfers control to the overlay specified by the phase name.
LOAD	CALL LOAD (s)	Loads subprogram overlays and transfers control to the program statement after the CALL statement.
OPSYS	CALL OPSYS ('LOAD', s)	Loads subprogram overlays and transfers control to the program statement after the CALL statement; equivalent to CALL LOAD statement.





6. Specification Statements

6.1. GENERAL

Specification statements are nonexecutable statements that inform the compiler about program data and main storage allocation. See the "Specification Statements" section of the fundamentals of FORTRAN reference manual, UP-7536 (current version). All statements in this section are order dependent. Refer to Table 1—2.

6.2. ARRAY DECLARATION

An array is an ordered set of elements identified by a symbolic name (2.4). An array may be declared in a DIMENSION statement, a COMMON statement, or in an explicit type statement (INTEGER, REAL, DOUBLE PRECISION, COMPLEX, or LOGICAL).

6.2.1. Array Declarator

Format:

$v (i_1, i_2, \dots, i_7)$

where:

v

Is a symbolic name identifying the array.

i

Is a unsigned integer constant or integer variable (for adjustable dimensions); an integer variable used to declare an adjustable dimension must be a COMMON variable or a dummy argument of the integer*4 type; from one to seven dimensions may be declared.

Description:

The array declarator specifies the name and the dimensions of an array. If the array name is a dummy argument, the array is a dummy array, and the dimensions may be specified as integer variables. In the interest of efficiency, dummy arrays are processed at execution time in a special fashion. The procedure prologue (5.5.1, 5.5.2) saves the subscripts in dimension declarators from the argument list or common storage, and derives a partial solution to the equation used to locate array elements (Table 2—2). Thereafter, subscript calculations in the body of the procedure can be performed more quickly. A side effect of this technique, however, is that it is impossible to redeclare array dimensions within procedures; for example, in the code sequence:

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	Cont.	7	10	20	30	40
		DIMENSION B (5,10)				
		CALL A (B, 5, 10)				
		:				
		SUBROUTINE A (X, I, J)				
		DIMENSION X (I, J); DECLARES (5, 10)				
5		J = 5				
10		I = 10				
		X (M, N) =				

statements 5 and 10 do not change the array from X(5,10) to X(10,5).

6.3. DIMENSION STATEMENT

Format:

DIMENSION v₁ (i₁)/c₁ / v₂(i₂)/c₂ / v_n(i_n)/c_n /

where:

v(i)

Is an array declarator (6.2.1).

c

Is an optional list of constants, each element of which is either a constant or a constant preceded by an unsigned integer constant multiplier in the format j*c. The constants are used to initialize the array.

Description:

The DIMENSION statement declares and optionally initializes arrays. Elements in the array are initialized starting with the first element. An array may be partially initialized, but an array that is a dummy argument may not be initialized. COMMON arrays should be initialized only in BLOCK DATA subprograms.

The constants in the list should agree in type with the array to be initialized. The compiler will convert numeric constants to the type of the corresponding array, but truncation may occur. In addition, the list may contain hexadecimal and literal constants.

The length implied by type (t), with or without the optional length specification (*s), applies to every name in the list unless it is specifically overridden by a specification for the individual name. See 5.6.2.1 for a discussion of specifying intrinsic and standard library functions in type statements.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
5		REAL LOAF, IOTA/5.2/, JOKE/7.5/,			
		*MATRIX(3,4,5)/60*0.0/			
		REAL*8 A, B, C			
		REAL A, B*8, C			

The first statement specifies LOAF, IOTA, JOKE, and MATRIX as real types. In addition, the statement indicates that IOTA is assigned a value of 5.2; JOKE, 7.5; and the array MATRIX consists of 60 elements and is initialized with 0.0 in every element.

In the second explicit type statement, the variables A, B, and C all are typed as double precision due to the length specification.

The third type statement specifies A and C as real variables; B is a double precision variable because of its length specification.

6.4.2. IMPLICIT Statement

Format:

IMPLICIT t*s(a₁,a₂,...,a_n),t*s(a_{n+1}—a_m,...),...

where:

t

Is the type, specified as INTEGER, REAL, DOUBLE PRECISION, COMPLEX, or LOGICAL.

a

Is a letter (A through Z and \$) associated with the specified data type. The format of this specification may be A,B,C, etc., with commas separating each letter, or it may be A—D, to specify a range of letters.

***s**

Is the optional length specification.

Description:

The IMPLICIT statement permits the user to specify his own implicit type conventions for each program unit. The IMPLICIT statement types symbolic names by the first letter of the name, including the dollar sign.

If \$ is to be included in a range specification (two letters separated by a minus sign), it must be last. The dollar sign indicates real data by the standard typing conventions.

Symbolic names that start with a letter not covered by the IMPLICIT statement are typed according to the standard convention in 2.3. Any implicit typing, whether standard or specified by the IMPLICIT statement, is superseded by explicit typing.

Symbolic names that appear in the program before the IMPLICIT statement are typed by standard conventions, except for dummy arguments in a SUBROUTINE or FUNCTION statement and the function name in a FUNCTION statement, which are redefined by the first IMPLICIT statement, but not subsequent IMPLICIT statements. IMPLICIT statements must appear in the specification group (Table 1-2).

Example:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER					
5		7	10	20	30 40
		IMPLICIT REAL*8(A-D,F), LOGICAL(L),			
		*INTEGER*2(N,Q,U,V), INTEGER(X-\$)			

After processing the IMPLICIT statement in the example, names beginning with the letters of the character set are typed as follows:

- A through D are double precision, as specified by the IMPLICIT statement, because real*8 is the equivalent of double precision;
- E is real, because of the standard convention;
- F is double precision, as specified by the IMPLICIT statement;
- G and H are real because of the standard convention;
- I, J, and K are integer because of the standard convention;
- L is logical, as specified by the IMPLICIT statement;
- M is integer because of the standard convention;
- N is integer*2, as specified by the IMPLICIT statement;
- O and P are real because of the standard convention;
- Q is integer*2, as specified by the IMPLICIT statement;
- R through T are real because of the standard convention;
- U and V are integer*2, as specified by the IMPLICIT statement;
- W is real because of the standard convention; and
- X through \$ are implicitly typed as integer by the IMPLICIT statement.

6.5. EQUIVALENCE STATEMENT

Format:

EQUIVALENCE (k₁),(k₂),....,(k_n)

where:

k

Is a list of the form a₁,a₂,....,a_m and each a is a variable name, an array element name, or an array name. Each name specified in the list shares assigned storage. Dummy arguments may not appear in the list.

Description:

The EQUIVALENCE statement permits sharing of a main storage unit by two or more entities specified within parentheses. The equivalence provided by the statement is in relation to the first, or leftmost, byte of the entities specified. (See 6.6.1 for a discussion of the effects of the interaction of EQUIVALENCE and COMMON statements.)

Program execution time is increased whenever a variable that does not have a proper boundary alignment is referenced. To achieve proper alignment, a variable must have an assigned main storage address that is an integral multiple of its length. Complex*16 variables require an 8-byte (double word) and complex*8 requires a 4-byte alignment. There are no boundary requirements for the logical*1 variables.

The first variable in each EQUIVALENCE group is assigned to a main storage address that is a multiple of 8 if possible. If erroneous boundaries are present in the EQUIVALENCE group, the addresses in the group are increased successively by 2, 4, and 6 in an attempt to correct the error. Thereafter, it is the programmer's responsibility to ensure that the variables in the EQUIVALENCE group have the proper alignment.

A variable with incorrect boundary alignment is recognized during compilation, and a warning diagnostic is provided. When the program is linked, a library routine is provided which receives control when the hardware interrupt caused by a reference to a variable with an improper alignment occurs. The subroutine repeats the instruction that caused the interrupt, after having moved the operand to the proper boundary.

6.6. COMMON STATEMENT

Format:

COMMON /x₁/a₁/.../x_n/a_n

where:

x

Is an optional symbolic name identifying the COMMON block. If no symbolic name appears between the slashes or if x₁ with its associated slashes is omitted, blank COMMON is assumed.

a

Is a nonempty list of variable names, array names, or array declarators. No dummy arguments are permitted.

Description:

The COMMON statement allows sharing of a common main storage area by different program units. When block names are specified, the compiler treats each block as a separate control section (CSECT) whose allocation will appear separately on the linker map. When no block name is specified (blank COMMON), the compiler uses a CSECT name that is not assigned by the programmer. It is the programmer's responsibility to ensure that every variable and array in COMMON has the proper boundary alignment. Automatic boundary error recovery is provided (6.5), but this increases execution time.

Every named or blank COMMON block is assigned a main storage address that is a multiple of 8. Each COMMON variable or array is assured of proper alignment if it is placed in the block in descending length: complex*16 variables and double precision first, then real and complex*8, and so on until logical*1 variables. In differing program units, when multiple definitions of a COMMON block specify different sizes for the block, the largest definition is accepted.

6.6.1. COMMON/EQUIVALENCE Statement Interaction

The compiler does not process COMMON and EQUIVALENCE statements individually in the sequence in which they are encountered. Instead, these statements are processed in three consecutive phases:

1. COMMON storage is allocated by processing all COMMON statements without regard to boundary requirements.
2. EQUIVALENCE groups that do not contain COMMON variables or arrays are processed, and storage is allocated. In any group containing improper boundaries, address adjustments are attempted.
3. EQUIVALENCE groups that contain COMMON variables or arrays are allocated storage without regard to boundary requirements. This may have the effect of lengthening COMMON at the right end of the list; COMMON cannot be extended at the left end of the list.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	5	6	7	10	20	30
			DIMENSION A(3)			
			COMMON B, C, D			
			EQUIVALENCE (A, D)			
			COMMON E			

produces a blank COMMON configuration of

B	C	D shares storage with A (1)	E shares storage with A (2)	A (3)
---	---	-----------------------------------	-----------------------------------	-------

The first three statements can be ordered in any arbitrary sequence with the same result. Replacement of the third line with

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
5					
6					
		EQUIVALENCE (A(3), B)			

is an illegal extension of COMMON.

6.7. EXTERNAL STATEMENT

Format:

EXTERNAL v_1, v_2, \dots, v_n

where:

v

Is the name of an external function or an external subroutine.

Description:

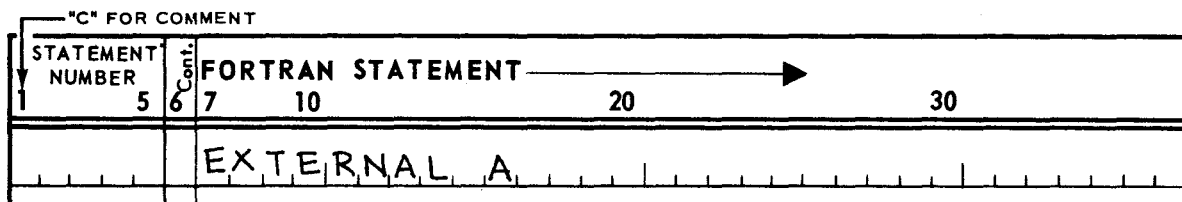
The EXTERNAL statement specifies function or subroutine names used as actual arguments to an external procedure. If an intrinsic function name appears in an EXTERNAL statement, that procedure is assumed to have been written by the user, and no assumptions about its properties are made. This is also true with the standard library function names. (See 5.6.2.1 for a discussion of specifying intrinsic and standard library functions in an EXTERNAL statement.)

A procedure name can appear both as an actual argument and as a dummy argument. This can occur when the procedure name is passed through multiple levels of procedure reference. In such a case, an EXTERNAL statement must appear at every level of procedure call.

When the context of the program uniquely identifies a symbolic name to be a procedure name, the EXTERNAL statement is unnecessary:

10	CALL A
20	CALL B(A)

No EXTERNAL statement is needed, but if statements 10 and 20 were reversed in sequence, the statement



would be needed.

6.8. PROGRAM STATEMENT

Format:

PROGRAM s

where:

s

Is the symbolic name used to identify a main program.

Description:

The PROGRAM statement may be optionally used to identify a main program for later reference by the linkage editor and librarian. When present, the PROGRAM statement is the first statement of the program unit. In the absence of this statement, the compiler assumes the name \$MAIN for main programs. Two main programs cannot be compiled in the same job if this statement is not specified, since the second main program will otherwise supersede the first.

The symbolic name s is a special name that bears no relationship to any variable or array name in the program unit. It must be unique with respect to the SUBROUTINE, FUNCTION, BLOCK DATA, and COMMON block names in the executable program.



7. Input and Output

7.1. GENERAL

This section describes the characteristics of the input/output system and the SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN statements required for input and output control. For further information, refer to the "Input/Output and FORMAT Statements" section of the fundamentals of FORTRAN reference manual, UP-7536 (current version). Also see Section 11 in this manual, which describes the usage of the OS/3 data management system.

The FORTRAN input and output statements are READ and WRITE. These statements designate an I/O device and reference an I/O list; they may reference a FORMAT statement. The input and output devices, that may be used in FORTRAN for sequential files include: card reader, printer, card punch, magnetic tape, and disc subsystems used sequentially. Direct access processing is also possible with disc subsystems. The peripheral devices are assigned unit numbers within the user's system where the unit number is a unique integer constant (k) in the range between 1 and 99.

7.2. INPUT/OUTPUT LIST

The purpose of an I/O list is to identify variables, arrays, and array elements so that they may be transferred to and from external devices. The I/O list is an ordered set of items with the format:

$$a_1, a_2, \dots, a_n$$

where:

a

Is a simple I/O list which may be a variable, array element, or array name;

Is two simple lists separated by a comma;

Is a simple I/O list in parentheses; or

Is a DO-implied list (7.2.1).

Example:

```
V2,ARRAY,MATRIX(5)
```

This I/O list consists of a variable, an array name, and an array element.

In an unformatted input/output statement, the I/O list directly determines record length; in a formatted statement, record length is determined by the interaction between the list and the FORMAT specifications. Section 11 discusses record length limitations with regard to various devices and file access methods.

7.2.1. DO-Implied List

Format:

```
(k,d)
```

where:

k

Is an I/O list (7.2).

d

Is a DO specification with the form: $i=m_1, m_2, m_3$ where parameter interpretation is identical with the DO statement (4.7).

Description:

The DO-implied list allows the transfer of list elements in the sequence specified by the DO parameters. DO-implied lists may be nested to a maximum of seven levels.

Example:

```
((AX(I,J,K),I=1,5),J=1,5),K=1,5)
```

If the 3-level DO-implied list in the example is used in a WRITE statement, the group of 125 elements of array AX is transferred to the specified external medium. The transfer would be to storage if the list were used in a READ statement. See 2.4.1 for the general expression to determine the location of array elements.

7.3. SEQUENTIAL FILES

The use of the American National Standard FORTRAN I/O statements READ, WRITE, BACKSPACE, REWIND, and ENDFILE is defined in the following paragraphs. The FORMAT statement, used for editing values represented by character strings on the external media, is also described.

Files referenced with the standard statements are always treated as sequential, even when they reside on disc storage.

7.3.1. Unformatted I/O Statements

An entire list of variables, arrays, and array elements transferred to an external device by an unformatted WRITE statement exists as a single logical record for a subsequent unformatted READ or BACKSPACE order. The formats are:

WRITE (u) k

READ (u,EOF=label₁,ERR=label₂) k

READ (u,END=label₁,ERR=label₂) k

where:

u

Is a constant or integer variable designating an I/O device.

EOF=label

Is an optional specification denoting the statement label of the statement to receive control if an end of file condition occurs;

END=label

May be substituted for the EOF=label specification.

ERR=label

Is an optional specification denoting the statement label of the statement to receive control if an error condition occurs.

k

Is an I/O list, which may be empty for a READ statement to indicate the record is to be skipped.

NOTE:

If both the EOF and ERR specifications, or both the END and ERR specifications, are present, their order may be interchanged.

Description:

The unformatted I/O statements initiate and control the transfer of unformatted data between a designated peripheral device and main storage.

Unformatted I/O is designed for high efficiency data transfer and, consequently, no data conversion operations take place; the variables are in the representation specified in 2.2 and 2.3. Only minor input validity checking is performed in keeping with this emphasis on throughput.

If the list for a WRITE statement consists of two integers followed by three double precision values, the only valid READ statements for that record are:

READ (u) ; bypass the record
READ (u) I
READ (u) I,I
READ (u) I,I,D
READ (u) I,I,D,D
READ (u) I,I,D,D,D

Even more efficiency can be achieved by reducing a list to a single element. Compare the following program segments:

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	6	7	10	20	30	40
		DIMENSION A(10), B(20), C(30)				
		DOUBLE PRECISION B				
		:				
		WRITE (9) A, B, C				

		DIMENSION A(10), B(20), C(30), DUMMY(80)				
		DOUBLE PRECISION B				
		EQUIVALENCE (DUMMY, A), (DUMMY(11), B),				
		(C, DUMMY(51))				
		:				
		WRITE (9) DUMMY				

The contiguous ascending storage addresses implied by DUMMY in the second segment allow greater efficiency in the data transfer.

7.3.1.1. END and ERR Clauses

The END and ERR specifications may appear in any order after the integer unit designation. EOF is an alternate form for END and is identical in function in Extended FORTRAN. If the END parameter is not present in a READ statement, the program is terminated with an informational message if the end of data is encountered. If either the END or EOF specification is present, control is transferred to the specified statement label when the end of data is encountered.

The ERR parameter specifies a statement label to which control is passed when it is impossible to completely process the current list. Other records in the file might still be available for processing. To describe the situation, the indicators tested by the ERROR subroutine (5.6.3) are set. If the ERR parameter is not specified, the program is terminated with an informational message when a record cannot be processed. Refer to 11.3.1.4.

7.3.2. Formatted READ/WRITE Statements

Formats:

READ (u,a) k

→ READ (u,a,EOF=label₁) k

READ (u,a,END=label₁) k

READ (u,a,END=label₁,ERR=label₂) k

WRITE (u,a) k

where:

u

Is a constant or an integer variable designating an input or output device, and has a value of from 1 to 99. ←

a

Is an array name, an integer variable (3.3.2), a NAMELIST name (7.3.5.1), the label of a FORMAT statement (7.3.3), or the asterisk character (7.3.5.2).

EOF=label

Is an optional specification indicating that if an end of file condition is encountered on input, the program is to branch to the label specified.

END=label

Accomplishes the same as EOF=label.

ERR=label

Is the optional specification of a label to which control is passed on encountering an error condition.

k

Is an optional I/O list.

Description:

The formatted READ/WRITE statements initiate and control the transfer of formatted data between designated peripheral device and main storage. Data is always converted from and to character strings on external media and the internal representations specified in 2.2 and 2.3 as specified by the format indicator, a. If the format indicator is an array name, the array must contain a legal FORMAT descriptor from opening to closing parenthesis. An integer variable format indicator must contain a FORMAT statement label from an ASSIGN statement. An asterisk character specifies list-directed formatting. A COMPLEX item always requires two FORMAT editing codes. ↓

7.3.2.1. I/O Compatibility Statements

The following FORTRAN II statements are accepted by the Extended FORTRAN processor:

READ a,k

PUNCH a,k

PRINT a,k

where:

a

Is the statement label of a FORMAT statement, an array name, an integer variable name (3.3.2), or the asterisk character (7.3.5.2).

k

Is an I/O list. ↑

NOTE:

No unit specification is made because it is unnecessary; the compiler addresses the appropriate device in the user's system configuration.

7.3.3. FORMAT Statement

Format:

I **FORMAT** (*q*₁*t*₁*z*₁*t*₂*z*₂...*t*_{*n*-1}*z*_{*n*-1}*t*_{*n*}*z*_{*n*})

where:

I

Is the label of the FORMAT statement.

q

Is an optional group of one or more slashes; each time a slash appears in the FORMAT statement, it signals the end of a logical record.

t

Is a field descriptor (7.3.3.1) or a group of field descriptors specifying the data conversion or the action to be executed.

z

Is a field separator (either a slash or a comma) required when more than one field descriptor is used; commas are not required when they follow fields described by blank (wX), Hollerith (wHc₁c₂...c_w) and literal ('c₁c₂...c_n') descriptors; slashes end a logical record.

Description:

The FORMAT statement specifies editing information for transforming formatted data (character strings), from and to internal representations. The FORMAT statement descriptors are described in the following paragraphs.

Examples:

"C" FOR COMMENT		STATEMENT NUMBER		FORTRAN STATEMENT	
5	6	7	10	20	30
100		FORMAT (' FIRST PAGE' /)			
110		FORMAT (/ / / I 1 2 , 2 X I 1 2 /)			

If referenced by a WRITE statement, the first FORMAT statement causes the transfer of the literal FIRST PAGE and provides an additional blank logical record. The second format statement skips three logical records and then describes a record with a 12-byte integer field, two blanks and another 12-byte integer field, plus another blank record.

7.3.3.1. Field Descriptors

The field descriptors specify the kind of I/O data conversion or action to be executed. Extended FORTRAN allows the descriptors listed in Table 7-1.

Table 7-1. *FORMAT Statement Field Descriptors*

Classification	Field Descriptor
Integer	rlw
Real (E conversion)	srEw.d
Real (F conversion)	srFw.d
Double precision	srDw.d
Logical	rLw
General	srGw.d
Hollerith (A conversion)	rAw
Hollerith (H conversion)	wHc ₁ c ₂ ...c _w
Hexadecimal	rZw
Literal	'c ₁ c ₂ ...c _n '
Blank	wX
Record position	Tp

LEGEND:

r = a repeat count ($0 < r \leq 255$)

w = the field width ($0 < w \leq 255$)

s = the scale factor nP ($-128 < n < +127$)

d = decimal positions ($0 \leq d \leq w$)

c = character

p = character position in the external record ($0 < p \leq 32767$)

The specifications within the field descriptors are explained below, and the input and output actions accomplished by the descriptors are described in 7.3.3.1.1 through 7.3.3.1.12.

- Repeat Count

The repeat count allows a field descriptor to be repeated a maximum of 255 times. The repeat count specification must be an unsigned integer constant. The field descriptor 5L3 is the same as L3,L3,L3,L3,L3.

- Field Width

The field width specification is an unsigned integer constant indicating the number of character positions the data occupies, or will occupy, in the external medium. The specification must not exceed 255.

- Scale Factor

Input and output using the E, F, D, and G conversion codes can be scaled up or down (multiplied or divided) by the specified power of 10, when the scaling specification in the format nP is included in the field descriptor. A complete description is available in the fundamentals of FORTRAN reference manual, UP-7536 (current version). Refer also to 7.3.3.1.13 in this manual.

- Decimal Positions

The specification describes the number of digits to the right of the decimal point; if none exist, a zero must be specified.

- Character

Any character of the Extended FORTRAN character set is permissible.

- Character Position

See 7.3.3.1.12.

Field descriptors may be grouped by using parentheses. The left parenthesis may be preceded by a group repeat count indicating the number of times the enclosed descriptors are to be repeated. The maximum is 255. Nesting to three levels is permitted. The result of the basic group and repeat count 2(2X,2I5,F10.0) is 2X,15,15,F10.0,2X,I5,I5,F10.0.

7.3.3.1.1. Integer Descriptor (rlw)

On input operations, if the value exceeds the range, only the least significant digits are stored with the sign, if any. An integer, which consists of a signed integer constant where the positive sign is optional, may contain, or be preceded by, embedded zeros or blanks. Blanks are interpreted as zeros.

If the value exceeds the permissible range of $\pm 32,768$ for integer*2 or $\pm 2,147,483,647$ for integer*4, the list element is defined to be the least significant 16 or 32 bits.

On output, the external field is preceded by a minus sign if the value is negative, and may be preceded by blanks, space permitting, if the value is positive. If the internal value cannot be converted into the w characters specified, the output field is set to w asterisks.

7.3.3.1.2. Real Descriptor — E Conversion (srEw.d)

On input, the external field consists of a string of digits optionally preceded by blanks or zeros preceded by an optional sign. Blanks are interpreted as zeros. The digit string may specify a decimal point, which overrides the d specification in the descriptor. The digit string may be followed by exponent notation, E or D followed by an optionally signed integer constant. If the integer constant is signed, the E or D may be omitted. If the number of significant digits exceeds the precision of the list element, the value will be rounded to the correct size. If the value is too small or too large for the range, a zero will be substituted. ←

On output, the external field has the following format:

$$s_1 0.n_1 n_2 \dots n_d E s_2 ee$$

where:

- s_1 Is the sign of the value, either blank or —.
- n Is a decimal digit.
- s_2 Is the sign of the exponent, either blank or —.
- ee Is the 2-digit exponent.

Note the decimal point preceding the digits.

For a complete representation of all values, the w specification should provide at least seven more additional field positions than the d specification.

The rules governing the output form when w is not at least 7 greater than d are:

- If (w—d) is 6, the zero character preceding the decimal point is deleted from the output form.
- If (w—d) is 5 and the value is nonnegative, both the s_1 and the zero characters preceding the decimal point are deleted from the output form.
- If neither of the above conditions holds, the entire output field is set to asterisks.

7.3.3.1.3. Real Descriptor — F Conversion (srFw.d)

For input action, refer to the E conversion description (7.3.3.1.2). On output, the external field has the following form:

$$s i_1 i_2 \dots i_{w-d-1} \cdot f_1 f_2 \dots f_d$$

where:

- s Is the sign of the value, either blank or —.
- i Is a digit within the integer portion of the output value.

f

Is a digit within the fractional portion of the output value.

Sufficient space must be provided for a minus sign if the value is negative. If the integer part of the value is nonnegative and requires more than (w-d-1) character positions for its representation, or is negative and requires more than (w-d-2) character positions, then the E conversion is used instead of the F conversion specified by the descriptor. If neither F nor E conversions suffice to represent the value, the entire field is set to asterisks.

7.3.3.1.4. Double Precision Descriptor (srDw.d)

For input action, refer to the E conversion description in 7.3.3.1.2. On output, also discussed in 7.3.3.1.2, the external field has the following form:

$$s_1 0 . n_1 n_2 \dots n_d D s_2 e e$$

7.3.3.1.5. Logical Descriptor (rLw)

The logical field descriptor allows the input or output of logical values. On input, the field is scanned until a T or an F is encountered; if no T or F is found, the list element is set to .FALSE.. On output, a T or an F is inserted in the record. The character is right-justified and is preceded by w-1 blanks.

7.3.3.1.6. General Descriptor (srGw.d)

This descriptor provides the capabilities of the E, F, I and the L conversion codes. During an input operation, this descriptor accepts any real data form with or without an exponent. During an output operation, the F conversion code is automatically selected if sufficient field width is specified in the descriptor; if not, the standard E or D exponential form is selected for output. The G descriptor may also be used to transfer integer, double precision, and logical data fields. For double precision data, the G descriptor is, in effect, the same as a D descriptor. For integer and logical data, the G descriptor is interpreted as an I or an L descriptor, respectively. The d and s editing information in the format may be omitted when transferring integer or logical data; it is ignored when present.

7.3.3.1.7. Hollerith Descriptor — A Conversion (rAw)

This descriptor requires a corresponding variable or array element name in the I/O list. The maximum number of characters that can be transmitted to a variable or array element is equal to the length, in bytes, of the variable or array element.

On input, if the descriptor specifies fewer than the maximum number of characters, the data field is transferred to main storage and left-justified; blanks are inserted in the remaining storage positions. If the descriptor specifies more than the maximum number of characters, only the rightmost characters of the data field are transferred to main storage. The remaining characters are skipped.

On output, if the descriptor specifies fewer characters than can be represented in the variable type, the leftmost characters of the data field are transferred from main storage. If the descriptor specifies more characters than can be represented in the variable type, the data field, right-justified and preceded by blanks, is transferred from main storage to the external field.

7.3.3.1.8. Hollerith Descriptor — H Conversion (wHc₁c₂...c_w)

On input, the next w characters transferred from the external device replace the current Hollerith data specified in the format statement. On output, the Hollerith data currently contained in the FORMAT statement is transferred to an external device.

7.3.3.1.9. Hexadecimal Descriptor (rZw)

This descriptor is used to transfer hexadecimal digits, any two of which may be stored in one byte in the list item. The number of digits associated with the data types are:

Type	Number of Hexadecimal Digits (k)
Logical*1	2
Logical*4	8
Integer*2	4
Integer*4	8
Real*4	8
Double precision	16
Complex*8	16
Complex*16	32

On input, the hexadecimal digits are stored two to a byte, right-justified and zero filled; blanks are interpreted as zeros. If a minus sign precedes the value, the leftmost bit of the variable is set to 1.

On output, a sign position is never produced, and when w is less than k in the above table, hexadecimal digits are truncated on the left. When w exceeds k , $(w-k)$ blanks precede the value.

7.3.3.1.10. Literal Descriptor ('c₁c₂...c_n')

This format code, similar in function to the H conversion, causes alphanumeric information to be read into or written from the literal data in the FORMAT statement. It is not necessary to specify an external field width. No I/O list item in a READ or WRITE statement is associated with this form of alphanumeric transmission. If an apostrophe is required in a Hollerith string, two successive apostrophes must be specified. For example, the characters DON'T are represented as 'DON''T'. The effect of the literal format code depends on whether it is used with an input or an output statement.

■ Input

The characters in the external field replace the literal data in the FORMAT specification in main storage. Contiguous inner apostrophes in the FORMAT specification are consolidated into a single apostrophe. Field width is determined by the literal length after contiguous apostrophes are eliminated. For example, the FORMAT descriptor 'A'' 'B' causes the next four characters to be input. Each apostrophe in the external field is treated as a separate character.

For example, if the input data in positions 1 through 10 is COUNTER△△△ and the following statements are used, the READ statement causes the 10 characters specified COUNTER△△△ to be transferred, replacing the characters HEADING△△△ in the FORMAT statement.

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Col.				
5	6	7	10	20	30 40
		READ (1, 20)			
20		FORMAT ('HEADING')			

■ Output

All characters, including blanks, within the apostrophes and the characters representing the literal constant are written as part of the output data. The descriptor 'DON'T' causes the five characters DON'T to be written.

Example:

		WRITE (30, 10)			
10		FORMAT ('THESE ARE SAMPLE PROBLEMS')			

Execution of the WRITE statement causes the following record to be written:

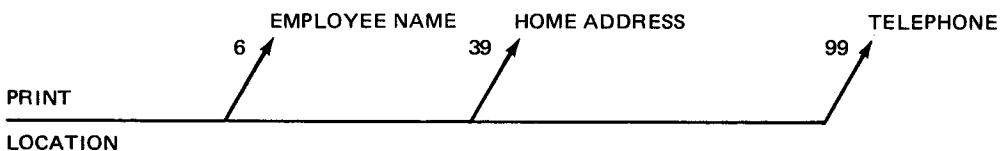
THESE ARE SAMPLE PROBLEMS

7.3.3.1.11. Blank Descriptor (wX)

This descriptor omits the next w consecutive characters on input. On output, the blank descriptor skips w positions in the output record. At the time each output formatted record is started, it is filled with blanks.

7.3.3.1.12. Record Position Descriptor (Tp)

This descriptor specifies the position in a FORTRAN record where data transfer is to begin. Input and output may begin at any position by using the Tp descriptor. The value of p represents the start position. As noted for the X descriptor, each output formatted record is blank filled at the time it is started. For example, the format specification (T7, 13EMPLOYEE△NAME,T100,9HTELEPHONE,T40,12HHOME△ADDRESS) causes record positions not specified in the field specification to be filled with blanks. However, for print records, the position specified becomes print column t-1, because the first character of a print record is interpreted as the carriage control character (Table 7-2), which is not printed. Thus, a print record for the format shown in the example would be:



The following statements cause the 10 characters starting from position 20 of the record to be converted according to the F10.3 code and stored in Y, and the 5 characters starting from position 1 to be converted according to the F5.1 specification and stored in B.

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
1					
		READ (3, 2) Y, B			
2		FORMAT (T20, F10.3, T1, F5.1)			

7.3.3.1.13. Scale Factor Effects

Scale factors have the form nP , where n is an optionally signed integer constant, and affect only D, E, F, and G format codes. Scale factors associated with other format codes are not meaningful.

READ and WRITE statements set an effective nP at their outset. By using an nP directly preceding either a format code or its associated repeat specification (if any), all the following D, E, F, and G format codes will be treated as though each were preceded by nP until a new scale factor is encountered. This rule applies even when a rescan of the entire FORMAT statement is required. For variables of type real or complex, a scale factor will either shift the decimal point n positions or have no effect, according to the following rules:

- Scaling has no effect when an input field contains an exponent or, for G output, when the internal value is within the range of effective F conversion.
- When an exponent is produced by a D, E, F, or G output conversion, scaling multiplies the basic real number by 10^n and reduces the produced exponent by n . Thus, external value = internal value.
- In all other cases, the scale factor implies a change of value according to the rule: external value = internal value $\cdot 10^n$.

7.3.3.2. Multiple Record Format Specification

The slash (/) is a record delimiter and a field separator. If a list of field specifications is followed by a slash, the remainder of the record being edited is ignored on input or filled with spaces on output. Any editing codes following the slash are used to edit the next record. The outer right parenthesis of the FORMAT statement is also a record delimiter if I/O list elements of the corresponding I/O statement remain at the time it is scanned.

7.3.3.3. Carriage Control Conventions

The first position of a printer output record does not print, but determines the action of the printer carriage. The action executed for a given carriage control symbol is described in Table 7-2.

Table 7—2. Carriage Control Conventions

Symbol	Meaning
△	1-line advance
0	2-line advance
+	No advance
1	Skip to top of next page
—	3-line advance

NOTE:

All actions take place before printing.

7.3.3.4. Format Interaction With the I/O List

During the execution of an I/O statement, the FORMAT specification is scanned from left to right. Editing codes of the form wH, 'h₁...h_n', wX and Tp, as well as slashes, are interpreted and acted upon without reference to the I/O list. When any other editing code is encountered, one of two possible actions is taken:

1. if a list element remains to be transmitted, it is converted and transmitted, and the FORMAT scan continues; or
2. if no list elements remain, both the current external record and the READ or WRITE statement are terminated.

A maximum of three levels of parentheses is permitted in a FORMAT statement:

```

LABEL FORMAT    ( .. ( .. ( .. ) .. ) .. ( .. ( .. ) .. ) .. )
                  1  2  3  3  2  2  3  3  2  1

```

When the right parenthesis at level 1 is encountered and a list element remains to be transmitted, a new record is started and one of two possible actions is taken:

1. if level 2 parenthetical groups exist, the FORMAT scan is resumed at the repeat count preceding the rightmost level 2 grouping; or
2. the scan is resumed at the beginning of the FORMAT.

An occurrence of a complex variable in an I/O list requires two real editing codes, and complex*16 variable requires two double precision editing codes.

List items must be associated as shown in Table 7—3.

Table 7-3. Permissible Associations of List Items

Descriptor	Data Types of List Items
Integer	Integer*2, integer*4
Real (E conversion, F conversion), double precision	Real*4, real*8, the real or imaginary part of complex*8 or complex*16 types
Logical	Logical*1, logical*4
General, Hollerith (A conversion), hexadecimal	Integer*2, integer*4, real*4, real*8 logical*1, logical*4, the real or imaginary part of complex*8 or complex*16 types

7.3.4. Reread

Format:

READ (u,a)k

where:

u

Is a constant or integer variable designating the reread unit.

a

Is the statement label of a FORMAT statement, an integer*4 variable (3.3.2), or an array name.

k

Is an I/O list.

Description:

The reread form of the READ statement allows the previous record transferred to main storage to be reread using a different FORMAT statement. This order neither selects nor initiates action on a peripheral device.

The Extended FORTRAN library contains a unit table that associates unit numbers with files. In this discussion, it is assumed that unit 29 has been associated with the reread feature; actually, any one or more units can be designated (Section 11).

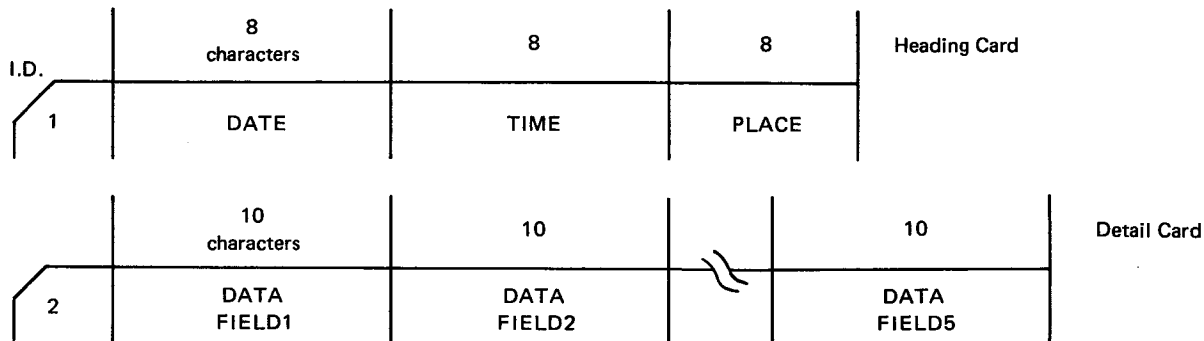
The reread feature is used when the program must determine the kind of information in a record. For instance, both header and detail records may be intermixed, and each kind of record may require different editing information in a FORMAT statement. After a READ order transfers a record to main storage, the record is identified by the program. If the correct format was applied, the program performs the necessary action on the data; if not, the program may execute a

READ (29,a) k

in conjunction with the desired FORMAT statement.

No ERR return is allowed with a reread. If an END or EOF label is specified and the previous read encountered an end of file, control is returned to the specified label. An unformatted record may not be reread.

Example:



"C" FOR COMMENT		STATEMENT NUMBER	FORTTRAN STATEMENT
5	6	7	10
		20	30
			40
			DOUBLE PRECISION DATA, TIME, PLACE, D(5),
C	READ		RECORD
			READ (20, 15) I, DATE, TIME, PLACE
15			FORMAT (I1, 3A8)
C	IDENTIFY		RECORD
			IF (I-1), 99 ; GO PROCESS HEADING CARD
C	CARD		IS DETAIL, SO FORMAT EDIT AGAIN
			READ (29, 30) D
30			FORMAT (1X, 5D10.4)

7.3.5. List-Directed Input/Output

Two classes of list-directed input/output statements are provided in Extended FORTRAN. Both classes process only formatted records, with the Extended FORTRAN system automatically supplying the necessary FORMAT specifications.

- Namelist input/output records contain variable or array element names with their associated values. The entire list is named, and on input the file is automatically searched to locate the name (7.3.5.1).
- Simple list-directed input/output records contain values without variable or array names. The statements are syntactically simple and require less main storage during program execution (7.3.5.2).

The general formats of I/O statements used in conjunction with the NAMELIST statement are:

READ (unit, namelist-name, END=label₁, ERR=label₂)

WRITE (unit, namelist-name)

Note that the END and ERR clauses are optional and that no list is present.

The general form of data for input is:

$\Delta n \Delta a_1 = c_1, a_2 = c_2,$

.

.

$a_n = c_n, \&END$

where:

n

Is a namelist name, a name identical with the name specified in the NAMELIST statement.

a

Is a variable, array element or array name of any data type.

c

Is a single, optionally signed, constant of the same type as the associated name; or, if the name is an array name, c is a list of one or more elements, each element separated by a comma, where an element is either an optionally signed constant or a list of identical, optionally signed, constants preceded by an unsigned integer repeat count of the form k*

The following rules pertain to input data:

1. The first character in a logical record must be blank. The second must be an ampersand immediately followed by the namelist name without any embedded blanks. The namelist is separated from the succeeding symbolic name by a blank or blanks. A comma after the last data unit is optional. The end of the NAMELIST record is signaled by &END.
2. When an array element or an array occurs in a NAMELIST record, the data is an optionally signed constant of the same type as its associated name. The constants can be preceded by an unsigned integer and an asterisk to indicate repetition. An array need not be filled by its data list.
3. No blanks may be embedded in constants.
4. If logical constants are used, the acceptable values are T, or .TRUE., and F, or .FALSE..
5. Literal constants can be transferred on input by using either apostrophes or the wH field descriptor. Literal constants may appear on an input record as TITLE='DON'T' or TITLE=5HDON'T.

A READ statement referencing a namelist name causes the next record to be read and tested for the proper namelist name. If the name is found, the first variable or array name is read and compared with the list of names defined in the NAMELIST statement. If the variable or array name is found in the list, the data value or values are assigned, and the next name is accessed. If the record does not contain the namelist name, subsequent records are read from the external medium until the record containing the name is found. If, after the proper record is found, a variable or array name that is not in the list of names appears in the input record, an error message is produced and the program is terminated.

Output data contains the namelist name followed by variables, array elements, and/or array names and their corresponding values. An array is written out by columns. Data fields are large enough to contain all the significant digits. Output data can be read by an input statement referencing the namelist name. Literal data is never produced as output.

7.3.5.2. Simple List-Directed Input/Output

List-directed I/O statements are identical in concept with formatted READ and WRITE statements except for the lack of a specific FORMAT statement reference. They are distinguished by the presence of the character asterisk (*) in place of the usual FORMAT reference, as in:

```
READ (10,*,END=30) A,B,C
```

These statements initiate and control the transfer of formatted data between a designated unit and main storage. Format control is provided by the Extended FORTRAN system based on the types of the list items and the record length associated with the unit. When preparing input data, the programmer must ensure that it conforms to the requirements of this list-directed format, specifically in regard to the use of the comma, slash, and blank characters. List-directed output records are, of course, acceptable as list-directed input.

■ Input Data Format

An input record consists of a list of constants, each demarcated by a separator. Separators are the characters:

blank (or a series of blanks)

comma (preceded and followed by zero or more blanks)

end-of-record

slash (preceded by zero or more blanks)

Since the blank is considered a separator, no embedded blanks may appear in arithmetic constants; blank, comma, or slash may appear within a literal constant enclosed within apostrophes, and end-of-record forces a read of the next sequential record. For card input, end-of-record is determined by the fixed length of 80 positions. For other input, such as tape or disc, the length specification given at the time the record was written is the determining factor. The slash separator causes termination of the READ statement. Real constants must be associated with real list items; integer and literal constants may have any association. The exponent identifiers E and D are considered equivalent. The real and imaginary parts of a COMPLEX constant must be separated by a comma and enclosed in parentheses. A repeat count may precede a constant using the form:

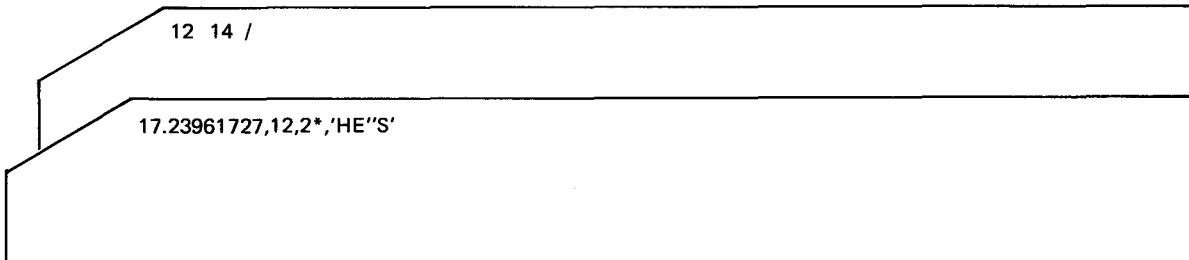
```
r*constant
```

Two or more consecutive comma separators (with any number of blanks or end of records intervening) indicates that the corresponding list items are not to be redefined. Multiple numbers of these "null items" may be indicated by:

```
(separator)r*(separator)
```

Example:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	LINE	7	10	20	30
5	6				
		INTEGER E,F,G			
		READ(U,*) A,B,C,D,E,F,G,H,I			



After the READ statement is executed, the values of the list items will be:

A	17.2396 (or 17.23961727 if real*8)
B	12.0
C,D	unchanged
E	HE'S
F	12
G	14
H,I	unchanged

■ Output Data Format

The output records consist of a list of constants, each separated by a comma. Output records never contain repeat items ($r*\text{constant}$) or literals. The maximum precision commensurate with the list item will be represented.

7.3.6. Auxiliary I/O Statements

Auxiliary I/O statements control the demarcation of files and the positioning of files to desired points of reference.

7.3.6.1. REWIND Statement

Format:

REWIND u

where:

u

Represents an integer constant or variable designating a sequential file on tape or disc.

Description:

The REWIND statement positions the file to a point immediately preceding all records of the file. The file is closed before a rewind operation. A REWIND statement issued to an unopened file is a null operation. A REWIND statement issued after an ENDFILE statement is issued allows the file to be reopened at the time of the next READ or WRITE statement.

7.3.6.2. BACKSPACE Statement

Format:

BACKSPACE u

where:

u

Is an integer constant or variable designating an I/O device. ←

Description:

The BACKSPACE statement activates the designated unit and causes a backspace of one logical record.

A record for a formatted file is defined by the termination of a WRITE statement, a slash encountered during format control, or the last parenthesis encountered in the format when other list items exist in the corresponding READ or WRITE statement. It is illegal for a format to demand a record longer than is present at the current file position.

In an unformatted environment, a record is defined by a single WRITE statement. The BACKSPACE statement has no effect if the file associated with a unit is currently positioned immediately preceding the first record. This statement should not be used when the file was used for list-directed input/output.

A BACKSPACE statement issued to an unopened file is a null operation. Logically, a BACKSPACE statement can follow only a READ statement or a WRITE statement to that file. A BACKSPACE statement after a WRITE statement closes and positions the file; the file is open after a legal BACKSPACE statement.

The BACKSPACE statement cannot be used with disc files under any conditions, with a file of blocked records, nor with a file having two I/O areas or a work area. However, these restrictions do not apply when backspacing over a file's end-of-file record. ←

7.3.6.3. ENDFILE Statement

Format:

ENDFILE u

where:

u

Is an integer constant or variable designating a card, tape, or sequential disc output file.

Description:

The ENDFILE statement closes the file specified by the unit number. Only a REWIND statement is allowed after an ENDFILE statement is issued; all other commands produce error messages. An ENDFILE statement issued to an unopened file is a null operation physically.

7.3.7. Sequential File Considerations

The I/O statements may not be executed in arbitrary sequences; the following table shows instances where specific commands are prohibited or ignored.

Current Operation Previous Operation	READ	WRITE	ENDFILE	BACKSPACE	REWIND
Successful READ			File truncated		
EOF encountered during READ			P		
WRITE	P				
ENDFILE	P	BACKSPACE missing (warning)	P	N	
BACKSPACE					
REWIND				I	I
No previous operation				I	I

LEGEND:

I indicates an ignored operation.

P indicates a prohibited operation.

N indicates that the operation is noted, but the file is still positioned following the last logical record. A second BACKSPACE must be issued to position the file in front of the last logical record in the file.

Further, not all operations are permitted on all devices; the following table shows prohibited combinations.

Operation File type	READ	WRITE	ENDFILE	BACKSPACE	REWIND
TAPE	*	*			
DISC	*	*		P	
CARD READ		P	P	P	P
CARD PUNCH	P			P	P
PRINTER	P			P	P
REREAD		P	P	P	P

*This operation may be prohibited when the files are defined as input only or output only.
See 11.3.1.4 for further details.

Formatted and unformatted records may be freely intermixed on output tape and disc files, but it is a user responsibility to read these records in the same mode as they were written.

7.4. DIRECT ACCESS FILES

Extended FORTRAN direct access statements are used to control disc subsystems. The term "direct access" refers to the ability of the disc to access a specified record of a file without accessing all preceding records. Disc subsystems need not be accessed directly; these devices may be used with sequential files in the same manner as for tape units. In this case, the only I/O statements required are those described in 7.3.

The direct access I/O statements are DEFINE FILE, FIND, READ, and WRITE. The direct access I/O statements can transmit either formatted or unformatted records.

7.4.1. DEFINE FILE Statement

Format:

```
DEFINE FILE u1(r1,m1,x1,v1),u2(r2,m2,x2,v2),...,un(rn,mn,xn,vn)
```

where:

u

Is a file identifier or an integer constant designating an I/O device.

r

Is an integer constant specifying the number of records in the file.

m

Is an integer constant specifying the maximum size of a record in the file in terms of characters (bytes), main storage locations (bytes), or main storage units (words), depending on the specification for x.

7.4.3. Disc WRITE Statement

Format:

WRITE (u'p,a) k

where:

u

Is a constant or integer variable designating an I/O device, followed by an apostrophe.

p

Is an integer expression designating the position of the record in the file.

a

Is an optional FORMAT statement label, an array name, an integer variable to which the statement label of a FORMAT statement has been assigned (3.3.2), or the character asterisk (7.3.5.2).

k

Is an I/O list.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT
STATEMENT NUMBER	Cont.	
1	5	6 7 10 20 30
		DEFINE FILE 4(1,50,3,6,L,FILE 4)
		:
		LOGICAL L
		:
		DOUBLE PRECISION D
		:
		FILE 4 = 2
		:
		WRITE (4' FILE 4+1,2) I,R,D,L
2		FORMAT (I8,F12.2,D15.5,L1)

Thirty-six bytes (8 + 12 + 15 + 1) are transferred from storage to the third record in the file. The format specification indicates the number of bytes for the integer, real, double precision, and logical values transferred. If the WRITE statement does not specify a format label, an unformatted WRITE statement is executed. In this case, 20 bytes are transferred.

Variable Name	Type	Number of Bytes
I	Integer	4
R	Real	4
D	Double precision	8
L	Logical	4
		20 Total

7.4.4. Disc FIND Statement

Format:

FIND (u'p)

where:

u

Is a constant or integer variable designating an I/O device, followed by an apostrophe.

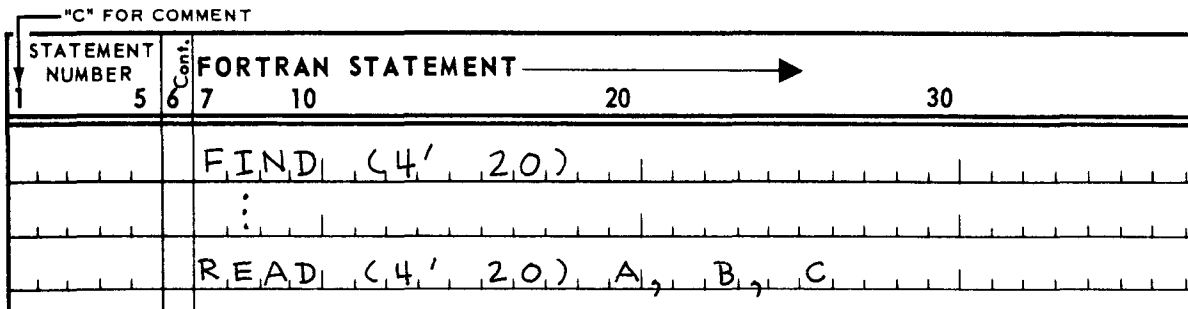
p

Is an integer expression designating the position of a record in the file.

Description:

The FIND statement can decrease the time required to execute an object program requiring records from disc. This statement positions the access arms to a disc address specified by a file identifier and a record position. During the time the arms are being positioned, execution of the object program can continue. After positioning, a READ statement accessing the record addressed in the FIND statement may be executed, and the record is transferred to main storage; thus, data transfer is completed more quickly when the arms are pre-positioned to a required track address prior to the execution of a READ statement. The FIND statement is never logically required in a program.

Example:



This example shows the relationship between a READ statement and a FIND statement. While the access arms are being positioned, the statements between the FIND statement and the READ statement are executed.



8. Data Initialization

8.1. GENERAL

Data initialization for SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN programs is described in this section. For more information, refer to the fundamentals of FORTRAN reference manual, UP-7536 (current version). See the DIMENSION and type statements (6.3 and 6.4.1), which have an initialization capability. In the absence of initialization, variables and array elements must be defined prior to reference.

8.2. DATA Statement

Format:

DATA $k_1/d_1/,k_2/d_2/, \dots, k_n/d_n/$

where:

k

Is a list of variable names, array names, array element names, or implied DO lists separated by commas.

d

Is a list of constants, any of which may be preceded by r^* to specify a repeat count, where r is an unsigned integer constant; items in the list are separated by commas.

Description:

The DATA statement initializes values represented by a variable, an array, and specified array elements. None of these items should be in blank COMMON; they should be in labeled COMMON only if the DATA statement appears in a block data subprogram.

Array element names may appear in DATA statements if their usage conforms to the following conventions:

- Subscript expressions are restricted to the standard American National Standard forms: c , v , $c*v$, $c*v+k$, $c*v-k$, and $v-k$, where c and k are positive integer constants and v is an integer variable.
- When v appears in a subscript, the array element name must be within the range of an implied DO list in which v is the control variable.
- The initial, terminal, and incremental values of the control variable of any implied DO list must be specified as positive integer constants.

Constants may be of the integer, real, double precision, complex, hexadecimal, logical, or literal type. When the corresponding variable is of a differing type (except for logical or literal), the constant will be converted, possibly causing truncation.

The DATA statement may be used to initialize arrays and variables with literal data. When initializing an array element or variable, a long literal string is truncated on the right to the correct size and a shorter string is filled with blanks on the right to the correct size.

Several consecutive elements of an array may be initialized with a single literal constant by using the array name without a subscript or by using an array element as the last item in the list. The long literal constant is placed in as many consecutive array elements as needed to contain it. If the last used position is only partially filled, that element is padded on the right with blanks. Truncation occurs if the literal string exceeds the limit of the array.

Example:

```
DIMENSION ARR(6)
DATA ARR/'ABCDEFGHIJKLM'/'
```

produces

```
ARR(1) contains 'ABCD'
ARR(2) contains 'EFGH'
ARR(3) contains 'IJKL'
ARR(4) contains 'M△△△'
ARR(5) not initialized
ARR(6) not initialized
```

A long literal may be overlaid if the constant list contains more than one constant.

Example:

```
DIMENSION ARR(6)
DATA ARR,VAR/'ABCDEFGHIJKLM',99/'
```

produces

```
ARR(1) contains 'ABCD'
ARR(2) contains 99.0
ARR(3) contains 'IJKL'
ARR(4) contains 'M△△△'
ARR(5) not initialized
ARR(6) not initialized
VAR not initialized
```


Initialization may commence at any point in the array.

Example:

```
DIMENSION ARR(6)
DATA VAR,ARR(3)/17,10HABCDEFGHIJ/
```

produces

```
VAR contains 17.0
ARR(1) not initialized
ARR(2) not initialized
ARR(3) contains 'ABCD'
ARR(4) contains 'EFGH'
ARR(5) contains 'IJΔΔ'
ARR(6) not initialized
```

8.3. BLOCK DATA SUBPROGRAM

A block data subprogram is an independently compiled specification subprogram. It is used to initialize values in labeled common blocks. The subprogram can contain only DATA, EQUIVALENCE, COMMON, DIMENSION, type, and IMPLICIT statements. The block data subprogram is headed by the BLOCK DATA statement. The order of statements is governed by the rules shown in Table 1—2.

8.3.1. BLOCK DATA Statement

Format:

```
BLOCK DATA s
```

where:

s

Is an optional symbolic name used to identify the BLOCK DATA subprogram.

Description:

The BLOCK DATA statement is the first statement in a block data subprogram, the statement indicating the beginning of a block data subprogram to the compiler. For a discussion of the effects of s, see the PROGRAM statement (6.8). In the absence of s, the compiler supplies the name \$BLOCK.



9. Compilation

9.1. GENERAL

The SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN compiler accepts source programs from card or disc files. Programs may be placed in disc files for storage and maintenance by using the SPERRY UNIVAC OS/3 system service programs. Refer to the current versions of OS/3 system service programs (SSP) programmer reference (UP-8209), introduction to the SSP (UP-8043), and SSP user guide (UP-8062). ←

The Extended FORTRAN compiler is named FOR and requires one work file, allocated in the job control stream. The compiler requires $CC00_{16}$ (X'CC00') bytes of main storage plus space for the prologue. Additional storage is utilized to increase compiler capacity. See Appendix D for examples of compilations using FOR.

9.2. PARAMETER STATEMENT FORMAT

Parameter statements for the compiler appear as punched cards in the job control stream.

Format:

1		10
//	Δ PARAM Δ	$n_1=d_1, n_2=d_2, \dots$

The // sequence must be in columns 1 and 2; columns 73 through 80 are not used. One or more blanks are required before and after PARAM, and one or more blanks are permitted after a comma. Each argument consists of a name (n), an equal sign, and a compiler directive (d). An argument may not contain embedded blanks. Multiple PARAM statements are permitted, but continuation is not. An argument may not continue on another card. For an explanation of statement conventions that apply to this section, refer to 1.3.

9.2.1. Compiler Arguments

A list of arguments provided by the compiler follows. Descriptions of the arguments follow the list. Refer to Appendix D for additional compilation options for users experienced in assembly language.

Format:

1	10
<code>//ΔPARAMΔ</code>	<code>OUT=filename, LIN=filename,LST=option,OPT=(D,N,X),NXT=LNK,CNL=K, MDE=I,STX=option, IN=module-name/filename</code>

The occurrence of an IN argument signals the end of the scanning for PARAM statements in each set of program units to be compiled. Arguments following an IN argument on a given // PARAM card are ignored. Subsequent // PARAM statements may contain IN arguments to allow for stacked compilations. (See 9.3.)

Output Argument:

OUT=filename

Specifies the file in which the compiler is to place object modules.

A one to eight alphanumeric character identifier is specified by filename. If OUT is not specified, the compiler places all object modules in the temporary scratch file \$Y\$RUN.

Library Input Argument:

LIN=filename

Specifies the name of the default file in which the source modules reside.

A one to eight alphanumeric character identifier is specified by filename. If LIN is not specified, the compiler assumes the default filename of LIB1. This argument is used in conjunction with the IN argument.

Listing Argument:

LST=option

Specifies the quantity of listings produced by the compiler.

One option may be chosen. The options include:

N

Specifies an abbreviated listing consisting of only the compiler identification, parameters, error counts, and termination conditions.

- S Specifies, in addition to the N listing, the source code listing with any serious diagnostics.
- M Specifies, in addition to the S listing, a storage map showing the addresses assigned to variables and arrays.
- W Specifies, in addition to the M listing, academic and warning diagnostics.
- O Specifies, in addition to the W listing, an object code listing showing the SPERRY UNIVAC 90 instructions generated for the executable statements.

The LST argument remains in effect for succeeding compilations until another LST argument is encountered. If no LST PARAM is specified, the M option is assumed.

Options Argument:

OPT=(D, N, X)

Specifies compilation options.

One or all options may be chosen. The options include:

- D Specifies double spacing of the compiler listing.
- N Specifies that no object program is to be generated. The program units are merely compiled and cannot be executed.
- X Specifies compilation of all cards with the character X in column 1. If this option is not specified, these cards will be treated as comments (10.2).

The default for the OPT argument is single spacing with the absence of the N and X specifications. All OPT options remain in effect until another OPT specification is encountered. If only one OPT argument is specified, the parentheses are optional.

Automatic Linker Call Argument:

NXT=LNK

Specifies the execution of the OS/3 linkage editor will automatically be invoked after the compilation of all program units in the next data set. ←

For card compilations, the linkage editor begins reading the PARAM statements immediately following the /* which terminated the FORTRAN source card input.

For a disc compilation, the linkage editor begins reading the PARAM statements immediately following the // PARAM card containing the IN argument which specified the disc file to be compiled and its associated correction file. ←

There is no default for the NXT argument. NXT can be used to simplify the job control stream and improve performance.

Compiler Termination Argument:**CNL=k**

Specifies compiler termination if a diagnostic with a severity level k is generated.

The values for k are:

- 2, which indicates academic messages, e.g., a truncated constant;
- 4, which indicates warning diagnostics, e.g., an extraneous comma in a list;
- 6, which indicates serious diagnostics, e.g., an array reference without a preceding array declarator;
or
- 8, which indicates fatal errors, e.g., insufficient storage to complete the compilation.

If the CNL argument is not specified, the compiler processes all program units in the control stream, regardless of errors encountered. When specified, the CNL argument remains in force until redefined.

Mode Argument:**MDE=I**

Specifies that the compiler is to evaluate expressions in a strict left-to-right order when there is a choice, and that storage is to be allocated for variables and arrays in the sequence in which they were encountered.

This argument is recommended for use when compiling programs originally developed under the IBM System/360 Disc Operating System. When specified, the MDE argument remains in force for all remaining compilations.

STXIT Macro Instruction Argument:

When the Extended FORTRAN Compiler generates code for a main program, a call to a FORTRAN IV library subprogram is produced. This causes the execution of two STXIT macro instructions, locates the diagnostic device, and sets up the program mask in the program status word (PSW).

The two STXIT macro instructions, for program checks and abnormal termination enable the library to:

- maintain switches for the OVERFL and DVCHK subroutines;
- recover boundary alignment errors caused by COMMON and EQUIVALENCE statements and argument substitution; and
- provide for orderly shutdown of the program when fatal errors occur.

The STX argument for the compiler parameter statement provides user control of the execution of STXIT macro instructions:

STX=Y

Causes the execution of two STXIT macro instructions at the beginning of a SUBROUTINE or a FUNCTION subprogram.

STX=N

Suppresses the execution of two STXIT macro instructions at program initiation. STX=N is used only for main programs.

If the STX argument is not specified, a specification of Y is assumed for main programs, and a specification of N is assumed for SUBROUTINE or FUNCTION subprograms.

The STX argument is operative only for the current subprogram to be compiled. This argument is useful when integrating COBOL and assembler object modules with FORTRAN object modules.

Input Argument:

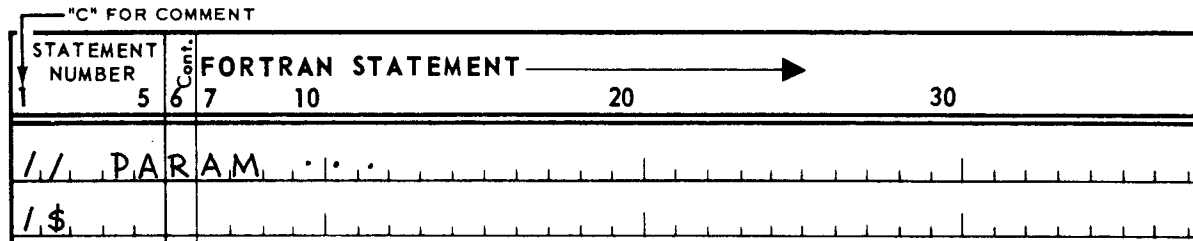
IN=module-name/filename

Specifies compilation of source programs residing in disc files.

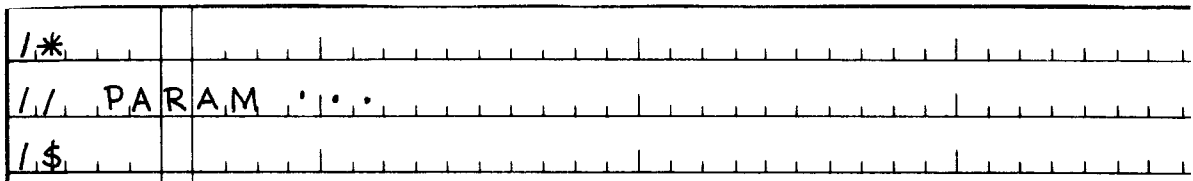
Module-name is a one to eight alphanumeric character identifier indicating the name of a source module to be compiled. Filename is a one to eight alphanumeric character identifier indicating the name of a file in which the module resides. If /filename is not specified, a default name is assumed and is described via the LIN argument.

9.3. STACKED COMPILATION

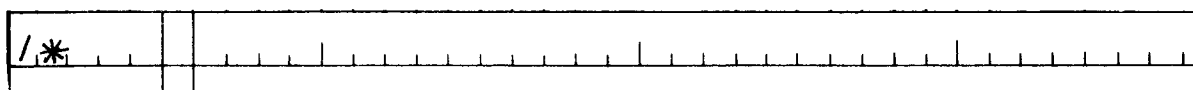
The compiler is capable of processing arbitrary numbers of source program units during a single execution. When the source programs are on punched cards, one or more units may be placed between the /\$ and /* data set delimiters. The data set is preceded by compilation // PARAM statements. Some compiler parameters are global and apply to all programs compiled. When a global parameter is to be changed, the job control stream should be organized into two or more data sets, each preceded by the required parameters. For example:



⋮ (one or more program units)



⋮ (one or more program units)



→ When the source programs are on disc files, the programs are identified by using a librarian module name. A source module consists of one or more FORTRAN program units. The IN compiler parameter is used to identify source files to the compiler. Global parameters can be redefined in the same way as for cards. For example:

"C" FOR COMMENT		FORTRAN STATEMENT	
STATEMENT NUMBER	Column	10	30
5	6	7	20
1			
1			
1			
1			

9.4. SOURCE CORRECTION FACILITY

When source programs reside on disc, it is possible to change the source as it is read into the compiler. If a /\$ and /* data set immediately follow the // PARAM statement with the IN argument, the compiler assumes that the data set contains correction cards to the source file. The method of correction is the same for the OS/3 system librarian's COR function. Refer to the current versions of OS/3 system service programs (SSP) programmer reference (UP-8209), introduction to the SSP (UP-8043), and SSP user guide (UP-8062). The corrections apply only to this compilation and the original source is not changed. When the compilation is complete, the next card available in the control stream immediately follows the /* card. For example:

```
// PARAM IN=MODA/FILEA

/$

    update of correction cards

/*

// PARAM . . .
```

NOTE:

A data set to be compiled from cards may not immediately follow an IN card because it will be mistaken for a correction deck.

9.5. CREATING A JOB CONTROL STREAM

The problem of creating a legal job control stream is greatly simplified by using the proper jproc (job control procedure). How to use the jprocs is described in Appendix D. However, if you wish to create your own job control stream, the following rules must be observed.

- The FORTRAN compiler requires one work file. The jproc WORK1 will supply this file.
- If the IN or OUT options are specified, the appropriate disc files must be defined.

- A printer device is required and must be defined by the // DVC 20 and // LFD PRNTR job control statements.
- Because of the stacked compilation feature (9.3), the // OPTION REPEAT feature of job control is not required and, in fact, must not be used.
- If the // OPTION LINK or the // OPTION LINK,GO job control statement is specified, no linkage editor control cards or control stream data for the program is allowed; the source correction facility (9.4) or the stacked compilation feature (9.3) would mistake the data sets for FORTRAN source code.





10. Debugging

10.1. GENERAL

Debugging aids are provided with the SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN compiler. These debugging aids consist of the standard I/O statements (especially list-directed statements described in 7.3.5), conditional compilation, the DEBUG statement and its associated packets, and the DUMP and PDUMP standard library subroutines for formatted main storage dumps.

10.2. CONDITIONAL COMPILATION

The compiler accepts a parameter which enables conditional compilation of any line which contains the character X in position 1 of the line. When this parameter is enabled, the line will be compiled; otherwise, the line is treated as a comment.

Example:

"C" FOR COMMENT	
STATEMENT NUMBER	FORTRAN STATEMENT
X 5	PRINT 10, A, B, C
X 10	FORMAT (3F15.6)

This coding would be used to print intermediate results during the debugging of a program. When debugging is complete, these statements can remain dormant in the source to be used at a later date if necessary. See Section 9 for the format of the PARAM statement.

10.3. DEBUG STATEMENT

Format:

DEBUG o,....o

where:

o

Is an option and may be any of the following:

INIT (n_1, n_2, \dots, n_n)

- n specifies a variable or array name whose new value is to be displayed when the value is changed. If name refers to an array, the changed element is output; if the list of names, with its associated parenthesis, is omitted, a display is performed every time any array element or variable has a value change. When the INIT option is omitted, changes of value are not reflected in the output of the debugging operations.

SUBCHK (n_1, n_2, \dots, n_n)

- n specifies an array name, identifying the array where subscript validity is to be checked. If the list of names, with its associated parenthesis, is omitted, all arrays are checked by comparing the total subscript value and array size, and a message is produced for each failure. When the SUBCHK option is omitted, no subscript checking occurs.

SUBTRACE

Specifies that the name of the subroutine, function, or entry being debugged is to be displayed each time the subprogram is entered; notification is also given when execution of the subprogram is complete.

TRACE

Specifies the display of the program flow by statement label within the program being debugged. A TRACE ON statement must appear in the debugging packet.

UNIT(i)

- i specifies an integer constant specifying the output unit for the debugging information. The debugging information consists of the displays produced by the DEBUG and DISPLAY statements and the DUMP and PDUMP subroutines. If a debug unit is not specified in the main program, this information will be displayed on the standard diagnostic device until a subprogram that defines the debug unit is executed. Subsequent definitions of the debug unit are ignored.

Description:

The DEBUG statement is a specification statement informing the compiler that the debugging aids are to be used and specifies the debugging operations to be performed. One DEBUG statement is permitted for each program unit, as shown in Table 1—2.

An example of the DEBUG statement is given in Figure 10—1.

10.4. DEBUGGING PACKET

One or more debugging packets can be included for each program unit to be examined. These packets begin with the AT statement and end with another AT statement specifying a new packet or with the program unit's END statement. The debugging packet consists of an AT statement, debugging statements as required (TRACE ON, TRACE OFF, and DISPLAY), and other FORTRAN source statements, as required.

General rules for using debugging packets are:

1. DO loops must not extend beyond the originating packet.
2. Only executable statements may appear in a debugging packet.

A simple debugging packet is illustrated in Figure 10—1.

10.4.1. AT Statement

Format:

AT a

where:

a

Is the statement label of an executable statement in the program unit to be debugged.

Description:

The AT statement, the first statement of a debugging packet, specifies the point in the program unit at which debugging starts. The debugging operations specified in the packet beginning with the AT statement are performed before the statement in the program unit referenced by the AT statement is executed.

10.4.2. TRACE ON Statement

Format:

TRACE ON

Description:

The TRACE ON statement allows the tracing of the program flow by statement number. When a statement with a label is executed in the main program unit, the statement number is displayed in the debugging information.

The TRACE ON statement takes effect as specified by the AT statement, and the trace operates through any level of subprogram call and return.

The TRACE option in the DEBUG statement must have been specified to use the TRACE ON statement. If control in the program unit is transferred to a program unit for which the TRACE option in a DEBUG statement was not specified, no trace output within that unit is made.

10.4.3. TRACE OFF Statement

Format:

TRACE OFF

Description:

The TRACE OFF statement stops the tracing operations initiated by the TRACE ON statement.

10.4.4. DISPLAY Statement

Format:

DISPLAY k

where:

k

Is a list of variable and/or array names separated by commas.

Description:

The DISPLAY statement is used in a debugging packet to display data in a NAMELIST format (7.3.5.1). The specification

DISPLAY k

has the same effect as the statements:

NAMELIST /n/k

WRITE (u,n)

Array elements may not appear in the list.

"C" FOR COMMENT		STATEMENT NUMBER	FORTRAN STATEMENT
5	6	7	10 20 30 40 50
			SUBROUTINE X(A, B)
			DIMENSION Y(10, 10)
			⋮
	1		DO 10, I=1, 10
			DO 10, J=1, 10
			⋮
			Y(I, J) =
			IF (Y(I, J)) .10, 20, 20
	20		⋮
			⋮
	10		CONTINUE
			⋮
X			DEBUG INIT(Y), SUBCHK(Y), SUBTRACE, TRACE,
X			UNIT(3)
X			AT 1
X			TRACE ON
X			AT 20
X			TRACE OFF
X			DISPLAY I, J
			END

Figure 10-1. DEBUG Statement and Packet

10.5. FORMATTED MAIN STORAGE DUMP

Two Extended FORTRAN standard library subroutines, DUMP and PDUMP, are provided to display variables or arrays. These two subroutines are identical, except that DUMP terminates the calling program and PDUMP does not.

Format:

CALL p (u₁,l₁,f₁u₂l₂,f₂,...,u_nl_nf_n)

where:

p

Is either DUMP or PDUMP.

u

Is a variable or array element name indicating the upper address boundary for the display.

l

Is a variable or array element name indicating the lower address boundary for the display.

f

Is an integer indicating the desired interpretation of the storage area.

The u and l specifications may be interchanged; their positions in the CALL statement do not influence the dump. The argument list enclosed in parentheses is optional.

The codes used for the format specification f are:

f	Display Interpretation
0	Hexadecimal
1	Logical*1
2	Logical*4
3	Integer*2
4	Integer*4
5	Real*4
6	Real*8
7	Complex*8
8	Complex*16
9	Literal

The output of these subroutines is directed to the debug unit or to the standard diagnostic unit. If no argument list is present, the dump is for the entire program and is in hexadecimal format.



11. Configuration of the Execution Environment

11.1. DATA MANAGEMENT INTERFACE

This section describes the interface between SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN and the OS/3 data management system, including:

- the relationships between unit numbers and external files;
- the kinds of devices supported;
- performance considerations, such as record blocking and buffering; and
- system defaults, that is, assumptions made by the system when specific directions are not provided.

Default actions taken when various errors are detected during program execution and how these defaults are changed to suit application requirements are also described. An example of a complete execution environment is given in 11.3.6.

An executable program requires a group of subroutines to support the FORTRAN I/O statements and to provide an interface to the data management system. These subroutines, individually called by the compiler, are automatically placed in the executable program by the linkage editor. One module, the control module, is central to the entire I/O scheme, because it contains the following tables:

- a unit table containing a unit number and FORTRAN control information and having an entry for each unit number implicit in the FORTRAN source program;
- a unit control table (a DTF in data management terminology) required by the data management system; and
- buffers and work areas for record processing.

A few control modules suitable for many application programs are contained in the Extended FORTRAN library (11.2). For more complex programs, the control module must be configured, using the Extended FORTRAN unit definition procedures (UNITS). Only one control module can exist in an executable program.

11.2. CONFIGURATIONS SUPPLIED

The following configurations are supplied by Sperry Univac for general use in simple applications. The unit numbers selected are industry standard.

■ Control Module FL\$IO

<u>Unit</u>	<u>Device</u>	<u>Notes</u>
1	Card read	Cards in control stream
3	Printer	Also used for diagnostics
5	Card read	Equivalent to unit 1
6	Printer	Equivalent to unit 3
29	Reread	

■ Control Module FL\$IO1

<u>Unit</u>	<u>Device</u>	<u>Notes</u>
1	Card read	Cards in control stream
2	Card punch	
3	Printer	Also used for diagnostics
5	Card read	Equivalent to unit 1
6	Printer	Equivalent to unit 3
11,12	Tapes	508-byte variable unblocked records, no labels, workfile
29	Reread	To reread cards, but not tapes

When additional configurations are being generated for general use at a site, it is suggested that module names FL\$102, FL\$103, etc., be used.

11.3. PROGRAMMER-DEFINED CONFIGURATIONS

The execution environment is configured using an assembly language source module with the form:

Continuation | ↘
72

1	10	
name		START file initialization file definition₁ file definition₂ . . . file definition_n file termination error definition END

Each element of the preceding assembly module is discussed in detail in this section. For an explanation of the statement conventions applicable to this section, refer to 1.3.

11.3.1. File Definition Conventions

Basic information about various arguments specified in defining a file is presented in the following paragraphs. This information applies to all files for which these features are specified.

11.3.1.1. Device Type

The device type is specified by the FDEVICE argument. This required argument is the basic criterion against which all other arguments are validated. For example, if the device is specified as a printer, the specification of a 5000-character record is rejected.

The specification for FDEVICE is one of the primary considerations in selecting default values for other arguments. For example, if the device is specified as card input, the Extended FORTRAN system assumes the card length to be 80, unless the user specifies otherwise.

File support provided by Extended FORTRAN is largely device independent. The user need not be concerned with whether the device is a UNISERVO VI-C, VIII-C, 12, 16, or 20 Magnetic Tape Unit, for example, because the system dynamically adapts itself to the varying requirements of these devices. The few features that cannot be supported in a device-independent fashion are noted in this section.

11.3.1.2. Record and Block Sizes

Record and block sizes are specified by the two optional arguments FRECSIZE, which specifies the record size, and FBKSZ, which specifies the block size.

The default value for FRECSIZE is selected by the FORTRAN system, based on the device type specified. For FBKSZ, the default value is computed from the record size.

FBKSZ is associated with the tape and disc devices and must always be greater than or equal to the record size.

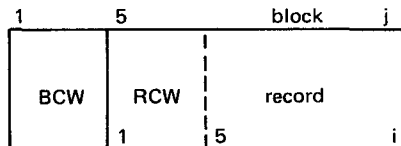
11.3.1.3. Record Formats

Four different record forms are available, including variable-length unblocked, variable-length blocked, fixed-length unblocked, and fixed-length blocked records, and are specified by the FRECFORM argument.

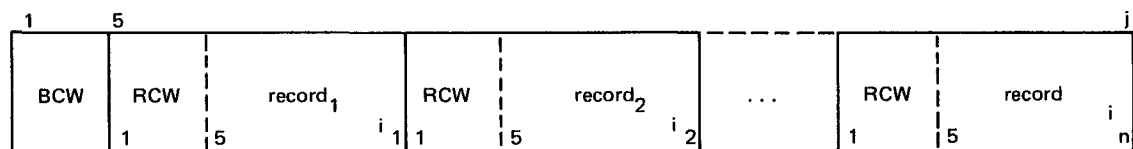
- Variable-Length Records

Formats for variable-length unblocked and blocked records follow.

- Variable-Length Unblocked Records (VARUNB)



- Variable-Length Blocked Records (VARBLK)



For both unblocked and blocked records, *i* specifies record size, *j* specifies block size, BCW specifies a data management block control word, and RCW specifies a data management record control word.

The FORMAT statement (7.3.3.) may not specify a record larger than *i*-4 for variable-length records. For unformatted input/output, no size limitation exists, since large FORTRAN records are automatically segmented into multiple data management records, using the record control words to identify beginning, middle, and end segments of the I/O list.

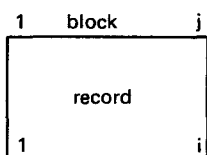
The BCW and RCW are controlled by Extended FORTRAN and the data management system and are not accessible through the FORTRAN language. The FBKSZ and FRECSIZE arguments are interpreted as maximums; shorter records will be accepted, and generated if possible, to save space on the external file and to reduce channel contention for main storage access.

- Fixed-Length Records

Formats for fixed-length unblocked and blocked records follow.

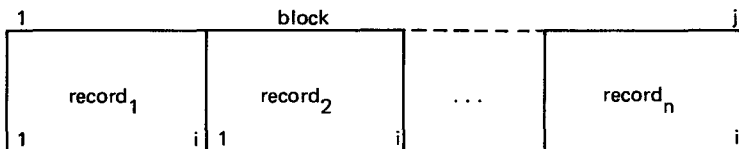
- Fixed-Length Unblocked Records (FIXUNB)

Format:



- Fixed-Length Blocked Records (FIXBLK)

Format:



For both unblocked and blocked records, *i* is the size specified for the FRECSIZE argument, and *j* is the size specified for the FBKSZ argument. For unblocked records, *i* and *j* must be equal. For blocked records, *j* is an integral multiple of *i*. The last block of the file may be less than *j* bytes, but it is always a multiple of *i*.

The FORMAT statement may not require more than *i* character positions for fixed-length records. In an unformatted I/O list, no more than *i* bytes may be required for a record.

11.3.1.4. Buffer Allocation

The amount of main storage used to support a unit is controlled by three interacting optional arguments: FBUFPOOL, which specifies buffer pooling; FNUMBUF, which specifies the number of buffers to be allocated to a unit; and FWORKA, which specifies whether a work area is to be allocated.

Buffer pooling must be used with discretion, or unpredictable results will occur. When multiple units with pooled buffers are active, only unblocked records may be processed, and only one buffer can be used. The term active covers the time period from the first reference to the unit until termination, which, on input, means an END clause return, and on output, means the execution of an ENDFILE statement. If only one unit using buffer pooling is active at a given time, record blocking and double buffering can be used.

During the processing of each unit definition procedure, a data management control block (DTF) is generated. Then, assuming buffer pooling is not requested, one or two buffers are allocated, using the FBKSZ value or its default to determine the block size. After the last unit definition procedure is processed, space for a work area is allocated.

Work area size is determined by the largest record for which work area processing was requested.

Similarly, one or two buffers are allocated for units using pooled buffers. The largest blocksize specified for such use is selected.

A work area is automatically assumed for output variable length blocked files.

For any application, a tradeoff can be made between main storage economy and program performance by use of these arguments and blocksize adjustments. This is especially useful when the program processes large tape or sequential disc files and is to be executed relatively often. In other cases, the system defaults are generally best.

Of the eight possible combinations, the following four are generally of greatest utility.

- One buffer, no work area, buffer pooling

This configuration gives greatest main storage economy. There is no overlap between computational and I/O activity, and blocked files cannot be processed if more than one file is using pooled buffers.

- One buffer, no work area, no buffer pooling

This configuration requires more main storage, but allows unrestricted use of blocked files

There is no overlap between computational and I/O activity.

- One buffer, work area, no buffer pooling

This is the usual Extended FORTRAN default. This requires slightly more main storage, but allows overlap between computational and I/O activity. The central processor loading is slightly increased because of record movement, but overall performance is usually improved.

- Two buffers, no work area, no buffer pooling

This configuration requires a still greater amount of main storage, provides overlap, and reduces computational loading due to the absence of record movement.

There is no requirement to allocate buffers for all units in the same fashion. The most attention should be given to the highest activity files.

11.3.1.5. File Type

The type of file — input, output, or work — is specified by the FTYPEFLE argument. This argument is not necessary for most devices. A printer, for example, is incapable of performing input functions and is always classified as an output device.

For tape and disc devices, the specification of an input or an output file permits the system to eliminate support coding and reduce the size of the executable program. The specification of a work file causes support coding for both input and output functions to be included.

11.3.2. START Statement

The START statement, a subprogram declarator statement required by the assembler, is the first statement of the configuration definition.

Format:

1	10
name	START

A 1- to 8-character symbolic name used to reference the control module on a linkage editor INCLUDE statement is specified by name. START is coded as shown.

This statement is always followed by the FUNTABL procedure call.

11.3.3. FORTRAN Initialization Procedure (FUNTAB)

The FUNTAB statement follows the START statement and precedes all other statements. It initializes either Basic FORTRAN or Extended FORTRAN parameters needed by statements which follow. To initialize the Extended FORTRAN parameters, the FUNTAB statement is coded as follows:

Format:

1	10	17
	FUNTAB	SYS=FOR

NOTE:

Omitting the SYS=FOR operand from the FUNTAB call initializes Basic FORTRAN parameters.

11.3.4. FORTRAN Unit Definition Procedure (UNIT)

Each file definition consists of a call on the FORTRAN unit definition procedure (UNIT), with arguments specifying characteristics of the file.

There are major syntactical differences between FORTRAN and assembly language:

- In the assembler, the statement continuation character is required for lines 1 through (n-1) in column 72, whereas in FORTRAN it is required in lines 2 through n in column 6.
- No embedded blanks are permitted, and all continuation lines must start in column 16, as is illustrated in subsequent examples.

Format:

1	10	16	Continuation ↓ 72
	UNIT	$n_1=c_1,$ $n_2=c_2,$. . . $n_n=c_n$	x x . . x

Each argument consists of an identifying name (n), an equal sign, and a particular characteristic (c) of the file being defined. All arguments must start in column 16. If an argument is not required, it is omitted, and the comma is deleted.

11.3.4.1. Printer File Definition

A single printer file is defined by using the UNIT procedure call presented in this paragraph. Following is a listing, in order of relative importance and utility, of the arguments that may appear on this UNIT procedure call. Following the listing, descriptions of UNIT arguments and a UNIT example are presented.

Work areas and buffer pooling are not supported for printers. The default number of buffers is 2.

Format:

1	10	16
	UNIT	FDEVICE=PRINTER $FUNIT = \left\{ \begin{array}{l} k \\ \text{PRINT} \\ \text{PUNCH} \end{array} \right\}$ $[FFILEID = \left\{ \begin{array}{l} \text{filename} \\ \text{FORT}k; \text{ if } FUNIT = k \\ \text{PRNTR}; \text{ if } FUNIT = \text{PRINT} \\ \text{PUNCH}; \text{ if } FUNIT = \text{PUNCH} \end{array} \right\}]$ $[FRECSIZE = \left\{ \begin{array}{l} k \\ \underline{121} \end{array} \right\}]$ $[FNUMBUF = \left\{ \begin{array}{l} 1 \\ \underline{2} \end{array} \right\}]$ [FDIAGNOS=YES] $[FPRINTOV = \left\{ \begin{array}{l} \text{SKIP} \\ \text{NOSKIP} \end{array} \right\}]$ $[FCHAR = \left\{ \begin{array}{l} \text{OFF} \\ \text{ON} \end{array} \right\}]$ [FOPTION=YES]

Device Identification Argument:

FDEVICE=PRINTER

Specifies that this is a printer file.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant (k) in the range $1 \leq k \leq 99$.

FUNIT=PRINT

Specifies PRINT as the unit identifier.

FUNIT=PUNCH

Specifies PUNCH as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module.

The identifiers PRINT and PUNCH are provided for reference by the FORTRAN II statements PRINT and PUNCH, respectively, since these statements contain no specific unit identification. When a FORTRAN II statement is executed and one of these special identifiers has not been provided in the control module, the first printer device specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

File Name Argument:

FFILEID=filename

Specifies a 1- to 7-character FORTRAN style symbolic name (filename).

FFILEID=FORTk

Specifies the file name as FORTk, where $1 \leq k \leq 99$. If the FFILEID argument is not specified, and FUNIT=k has been specified, FORTk is the default file name.

FFILEID=PRNTR

Specifies the file name as PRNTR. If the FFILEID argument is not specified and FUNIT=PRINT has been specified, PRNTR is the default file name.

FFILEID=PUNCH

Specifies the file name as PUNCH. If the FFILEID argument is not specified and FUNIT=PUNCH has been specified, PUNCH is the default file name.

Record Size Argument:

FRECSIZE= $\left. \begin{array}{c} k \\ 121 \end{array} \right\}$

Specifies a positive integer constant (k), in the range $1 \leq k \leq 161$. If this argument is omitted, 121 is the default record size. This accommodates a 120-character SPERRY UNIVAC 0773 Printer, with one additional character for carriage control. Other printers may specify up to 160 print positions.



Buffer Allocation Argument:

FNUMBUF=1

Specifies one buffer to be allocated to a unit.

FNUMBUF=2

Specifies two buffers to be allocated to a unit.

Diagnostic Messages Argument:

The Extended FORTRAN runtime environment always requires a device for diagnostic purposes.

FDIAGNOS=YES

Specifies the current unit as the diagnostic device. If FRECSIZE is specified, its value must be 101 or more. Debugging information may also be written to this device (9.3). This argument is not available for input files.

Printer Forms Control Argument:

This argument specifies whether the forms control loop (or an electronic equivalent) contained in the printer device for locating the top and bottom of the page is to cause automatic skipping across the seam of the paper.

FPRINTOV=SKIP

Specifies that the printer is to skip to the top of the next page (home paper) when the bottom of the current page (forms overflow) is detected.

FPRINTOV=NOSKIP

Specifies that no automatic forms control is desired. Spacing is then under sole control of the carriage control characters (7.3.3.3.).

Invalid Character Processing Argument:

This argument specifies the action to be taken when a character with no corresponding printer graphic is encountered.

FCHAR=OFF

Specifies that a blank is to be substituted for the character and that the line is to be written to the printer with no error notification.

FCHAR=ON

Specifies that a device error is to be generated and the program is to be terminated.

Optional Units Argument:

FOPTION=YES

Specifies an optional unit, a unit not always required during program execution.

When this argument is specified, and the file has not been allocated by job control statements, WRITE statements are effectively ignored. A unit need not be declared as optional if the logic of the program does not cause a reference to the unit.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT					
STATEMENT NUMBER	5	6	7	10	20	30	72
							X
							X
							X
							X

A printer is defined, unit 10, with 100 printable characters per line, which is to be used also for diagnostic purposes. No automatic forms overflow is to take place, and device error recovery is requested. The FORTRAN system assumes defaults of:

- file name is FORT10;
- two buffers;
- substitution of blanks for nonprinting characters; and
- file is required if a reference occurs.

11.3.4.2. Card Input Files

Two methods, the operating system spooling facility and the data management card read procedures, are provided to read data cards. The operating system spooling facility reads cards and transcribes them to a disc file before the executable program is activated. When a card image is requested by the program, the operating system reads the card image from disc and delivers it to the program. The data management card read procedures require the allocation of a card reader device to the executable program and activate the device in synchronization with program requests for card images. This method requires more main storage and is most suited to high volume applications. The two methods are described in the following paragraphs.

11.3.4.2.1. Spooled Card Input File Definition

A spooled card input file is defined by using the UNIT procedure call presented in this section. Following is a listing, in order of their relative importance and utility, of the arguments that may appear on the UNIT procedure call. Following the listing, descriptions of UNIT arguments, programming considerations, and a UNIT example are presented.

Only one spooled card input file is permitted for a given application.

Format:

1	10	16
	UNIT	FDEVICE=SPOOLIN $FUNIT = \left\{ \begin{array}{l} k \\ \text{READ} \end{array} \right\}$ [FREREAD=YES] $[FBKSZ = \left\{ \begin{array}{l} k \\ 400 \end{array} \right\}]$ [FBUFPOOL=YES] $[FRECSIZE = \left\{ \begin{array}{l} k \\ 80 \end{array} \right\}]$

Device Identification Argument:

FDEVICE=SPOOLIN

Specifies that this is a spooled card input file (embedded data set).

Unit Identifier Argument:

FUNIT=kSpecifies a unique integer constant in the range $1 \leq k \leq 99$.**FUNIT=READ**

Specifies READ as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module.

The identifier READ is provided for reference by the FORTRAN II READ statement, since this statement contains no specific unit identification. When a FORTRAN II statement is executed and one of these special identifiers has not been provided, the first spoolin device specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

Reread Argument:

FREREAD=YES

Specifies that the unit is to participate in the reread feature (7.3.4).

The reread unit consists of a single buffer to which each formatted input record is transferred. To conserve central processor time, this data movement is inhibited unless specifically requested.

Block Size Argument:

FBKSZ = $\left\{ \begin{array}{l} k \\ 400 \end{array} \right\}$

Specifies a positive integer constant (k) that is an integral multiple of FRECSIZE. A large multiple of FRECSIZE reduces operating system overhead. If a number is specified that is not an integral multiple of FRECSIZE, the block size is rounded downward to the nearest multiple. The default block size is the largest integral value of FRECSIZE that is less than or equal to 400.

Buffer Pooling Argument:

FBUFPOOL=YES

Specifies that buffer pooling is to be used.

The buffers are to be logically equivalent with all other units for which buffer pooling is specified.

When multiple units with pooled buffers are active, only unblocked records may be processed and only one buffer can be used. A unit is active from the first reference to the unit until termination, which means an END clause return. If only one unit using buffer pooling is active at a given time, record blocking can be used.

Record Size Argument:

**FRECSIZE= { k }
 { 80 }**

Specifies the record size for a spooled card input file, where k may have a value from 1 to 128. The default record size of the spooled card input file is 80.

Programming Considerations:

Spooled input consists of one or more sets of cards, each headed with a card containing

/ \$

in columns 1 and 2 and terminated with a card containing

/ *

in columns 1 and 2.

The / \$ card is always bypassed by the Extended FORTRAN library and is not accessible as a data card; the / * card causes control to be transferred to the label specified in the END clause or, in the absence of an END clause, causes program termination.

Example:

"C" FOR COMMENT		STATEMENT NUMBER	FORTRAN STATEMENT			
5	6	7	10	20	30	72
			UNIT	FDEVICE=	SPOOLIN	X
			FUNIT=	2,		X
			FREREAD=	YES,		X
			FBKSZ=	240		

This UNIT procedure call defines a spooled card input file, unit 2, that participates in the reread feature. Three cards (240 characters) at a time are read into the buffer to reduce operating system overhead. As a default, the FORTRAN system assumes a unique, nonpooled buffer.

11.3.4.2.2. Data Management Card Input File Definition

A single data management card input file is defined by using the UNIT procedure call presented in this paragraph. Following is a listing, in the order of this relative importance and utility, of the arguments that may appear on the UNIT procedure call. Following the listing, descriptions of the UNIT arguments and a UNIT example are presented.

The only limitation on the number of data management card input files is the system configuration and the number of devices that can be allocated to the application. Cards may be read from a card punch if the device is equipped with the optional read feature.

Format:

1	10	16
	UNIT	FDEVICE=CARDIN FUNIT={ k READ } [FFILEID={ filename FORTk; if FUNIT=k READER; if FUNIT=READ }] [FREREAD=YES] [FBUFPOOL=YES] [FNUMBUF={ 1 2 }] [FWORKA={ YES; if FNUMBUF=1 NO; if FNUMBUF=2 }] [FRECSIZE={ k 80 }] [FBKSZ={ k FRECSIZE }] [FSTUB={ 51 66 }] [FOPTION=YES] [FAUE=YES]



Device Identification Argument:

FDEVICE=CARDIN

Specifies that this is a card input file.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant in the range 1 ≤ k ≤ 99.

FUNIT=READ

Specifies READ as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module.

The identifier READ is provided for reference by the FORTRAN II READ statement, since this statement contains no specific unit identification. When a FORTRAN II statement is executed and this special identifier has not been provided, the first card in device specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

File Name Argument:

FFILEID=filename

Specifies a 1- to 7-character FORTRAN style symbolic name (filename)

FFILEID=FORTk

Specifies the file name as FORTk, where $1 \leq k \leq 99$. If the FFILEID argument is not specified and FUNIT=k has been specified, FORTk is the default file name.

FFILEID=READER

Specifies the file name as READER. If the FFILEID argument is not specified and FUNIT=READ has been specified, READER is the default file name.

Reread Argument:

FREREAD=YES

The reread unit consists of a single buffer to which each formatted input record is transferred. To conserve central processor time, this data movement is inhibited unless specifically requested.

Buffer Pooling Argument:

FBUFPOOL=YES

The buffers are to be logically equivalent with all other units for which buffer pooling is specified.

When multiple units with pooled buffers are active, only unblocked records may be processed, and only one buffer can be used. A unit is active from the first reference to the unit until termination, which means an END clause return. If only one unit using buffer pooling is active at a given time, double buffering can be used.

Buffer Allocation Argument:

FNUMBUF=1

Specifies one buffer to be allocated to the unit.

FNUMBUF=2

Specifies two buffers to be allocated to the unit.

Work Area Allocation Argument:

This argument specifies whether records are to be processed directly in the buffer or moved from a work area for processing.

FWORKA=YES

Specifies that space for a work area is to be allocated. If this argument is omitted and FNUMBUF=1 is specified, the default is that space is allocated for a work area.

FWORKA=NO

Specifies that no space for a work area is to be allocated. If this argument is omitted and FNUMBUF=2 is specified, the default is that no space is allocated for a work area.

Record Size Argument:

This argument specifies record size.

$$\text{FRECSIZE} = \left\{ \begin{array}{l} k \\ \underline{80} \end{array} \right\}$$

Specifies a positive integer constant (k) in the range $1 \leq k \leq 128$.

If 96-column cards are to be read, 96 must be specified. If this argument is omitted, 80 is the default record size.

If an 8413 diskette is to be read, the record size specified must be that actually recorded on the diskette.

If the rightmost columns of an 80-column card are not meaningful to the program, this argument may be used to save main storage space by specifying a shorter record size.

Block Size Argument:

This argument specifies the 8413 diskette block size.

$$\text{FBKSZ} = \left\{ \begin{array}{l} k \\ \text{FRECSIZE} \end{array} \right\}$$

Specifies a positive integer constant ($k \leq 1024$) that should be an integral multiple of FRECSIZE. A large multiple of FRECSIZE will reduce overhead. The default value is the FRECSIZE specification.

If a number is specified that is not an integral multiple of FRECSIZE, the block size is rounded downward to the nearest multiple.

Stub Card Argument:

This argument specifies cards physically shorter than 80 columns.

FSTUB=51

Specifies a 51-column card.

FSTUB=66

Specifies a 66-column card.

The card reader must be equipped with the proper optional feature if this argument is specified. If stub cards are to be read, FSTUB must be specified. FSTUB is completely independent of the record size.

Optional Units Argument:

FOPTION=YES

Specifies an optional unit, a unit not always required during program execution.

When this argument is specified and the file has not been allocated by job control statements, the first READ reference causes an end-of-file condition to occur.

A unit need not be declared as optional if the logic of the program does not cause a reference to the unit.

Rejection of Mispunched Cards Argument:

FAUE=YES

Specifies that cards with an illegal hole combination in a column are to be bypassed and will not be delivered to the program.

When the device being used is a SPERRY UNIVAC 0716 Card Reader, the erroneous card is also sorted into a unique error stacker.

If this argument is not specified, the card reader is stopped, and operator intervention is sought when an illegal hole combination is detected.

Example:

"C" FOR COMMENT		STATEMENT NUMBER	FORTRAN STATEMENT				
1	5	6	7	10	20	30	72
			UNIT FDEVICE=CARDIN,				X
			FUNIT=READ,				X
			FNUMBUF=2				X
			FREC SIZE=56,				X
			FAUE=YES				

This UNIT procedure call defines a card reader device, or a card punch device with the optional read feature, to be referenced by using the FORTRAN II READ statement. Two buffers are allocated for efficiency, and only the first 56 characters on each card are to be transferred to main storage. Cards with erroneous punches are ignored. The defaults assumed are:

- file name is READER;
- records will not be reread;
- nonshared buffers with no work area;
- no stub cards; and
- file required if a reference occurs.

11.3.4.3. Card Output File Definition

A single card or 8413 diskette output file is defined by using the UNIT procedure calls presented in this paragraph. Following is a listing, in the order of their relative importance and utility, of the arguments that may appear on the UNIT procedure call. Following the listing, descriptions of UNIT arguments and a UNIT example are presented. ←

Format:

1	10	16
	UNIT	FDEVICE=CARDOUT FUNIT={ ^k PUNCH } [FFILEID={ filename FORTk; if FUNIT=k PUNCH; if FUNIT=PUNCH }] [FBUFPOOL=YES] [FNUMBUF={ ¹ 2 }] [FWORKA={ YES; if FNUMBUF=1 } NO; if FNUMBUF=2 }] [FRECSIZE={ ^k 80 }] [FBKSZ={ ^k FRECSIZE }] ← [FCRDERR=RETRY] [FOPTION=YES]



Device Identification Argument:

FDEVICE=CARDOUT

Specifies that this is a card output file.

Unit Identification Argument:

FUNIT=k

Specifies a unique integer constant (k) in the range $1 \leq k \leq 99$.

FUNIT=PUNCH

Specifies PUNCH as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module.

The identifier PUNCH is provided for reference by the FORTRAN II PUNCH statement, since this statement contains no specific unit identification. When a FORTRAN II statement is executed and this special identifier has not been provided, the first cardout device specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

File Name Argument:

FFILEID=filename

Specifies a 1- to 7-character FORTRAN style symbolic name (filename).

FFILEID=FORTk

Specifies the file name as FORTk where $1 \leq k \leq 99$. If the FFILEID argument is not specified and FUNIT=k has been specified, FORTk is the default file name.

FFILEID=PUNCH

Specifies the file name as PUNCH. If the FFILEID argument is not specified and FUNIT=PUNCH has been specified, PUNCH is the default file name.

Buffer Pooling Argument:

FBUFPOOL=YES

Specifies that buffer pooling is to be used. The buffers are to be logically equivalent with all other units for which buffer pooling is specified.

When multiple units with pooled buffers are active, only one buffer can be used. A unit is active from the first reference to the unit until termination, which means the execution of an ENDFILE statement. If only one unit using buffer pooling active at a given time, double buffering can be used.

Buffer Allocation Argument:

FNUMBUF=1

Specifies one buffer to be allocated to the unit.

FNUMBUF=2

Specifies two buffers to be allocated to the unit.

Work Area Allocation Argument:

This argument specifies whether records are to be processed directly in the buffer or moved from a work area for processing.

FWORKA=YES

Specifies that space for a work area is to be allocated. If this argument is omitted and FNUMBUF=1 is specified, the default is that space is allocated for a work area.

FWORKA=NO

Specifies that no space for a work area is to be allocated. If this argument is omitted and FNUMBUF=2 is specified, the default is that no space is allocated for a work area.

Record Size Argument:

FRECSIZE= $\left\{ \begin{array}{l} k \\ 80 \end{array} \right\}$

→ Specifies a positive integer constant (k) in the range $1 \leq k \leq 128$.

↓ If this argument is omitted, 80 is the default record size.

Block Size Argument:

This argument specifies the 8413 diskette block size.

FBKSZ= $\left\{ \begin{array}{l} k \\ \underline{\text{FRECSIZE}} \end{array} \right\}$

↑ Specifies a positive integer constant ($k \leq 1024$) that should be an integral multiple of FRECSIZE. A large multiple of FRECSIZE will reduce overhead. The default value is the FRECSIZE specification.

If a number is specified that is not an integral multiple of FRECSIZE, the block size is rounded downward to the nearest multiple.

Device Error Recovery Argument:

FCRDERR=RETRY

Specifies that error recovery coding is included in the executable program.

If this argument is not specified or if the recovery attempt is unsuccessful, program termination is initiated when device errors occur. Mispunched cards are automatically segregated into an error card stacker. This argument is not meaningful if card output is spooled (transmitted to disc for later transcription to a card punch).

Optional Units Argument:

FOPTION=YES

Specifies an optional unit, a unit not always required during program execution.

When this argument is specified and the file has not been allocated by job control statement, WRITE statements are effectively ignored.

A unit need not be declared as optional if the logic of the program does not cause a reference to the unit.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	Cont.	7	10	20	30	72
5	6					
						X
						X
						X

This FUNDEF procedure call defines a card punch device, unit 32, with a pooled buffer. In the event of a device error, automatic retry is to be attempted. The defaults assumed are:

- file name is FORT32;
- one buffer and one work area;
- record size of 80; and
- file is required if a reference occurs.

11.3.4.4. Tape File Definition

A single tape file is defined by using the UNIT procedure call presented in this paragraph. Following is a listing, in the order of their relative importance and utility, of the arguments that may appear on the UNIT procedure call. Following the listing, descriptions of UNIT arguments and a UNIT example are presented.

Format:

1	10	16
	UNIT	FDEVICE=TAPE
		FUNIT={ k READ PUNCH }
		[FFILEID={ filename FORTk; if FUNIT=k READER; if FUNIT=READ PUNCH; if FUNIT=PUNCH }]
		[FTYPEFLE={ INOUT WORK; if FUNIT=k INPUT; if FUNIT=READ OUTPUT; if FUNIT=PUNCH }]
		[FRECFORM={ VARUNB VARBLK FIXUNB FIXBLK }]

1

10

16

UNIT
(cont)
$$\left[\text{FNUMBUF} = \left\{ \begin{array}{l} 1 \\ 2 \end{array} \right\} \right]$$
$$\left[\text{FWORKA} = \left\{ \begin{array}{l} \text{YES}; \text{ if FNUMBUF}=1 \\ \text{NO}; \text{ if FNUMBUF}=2 \end{array} \right\} \right]$$

[FBUFPOOL=YES]

$$\left[\text{FRECSIZE} = \left\{ \begin{array}{l} k \\ 508 \end{array} \right\} \right]$$
$$\left[\text{FBKSZ} = \left\{ \begin{array}{l} k \\ |\text{FRECSIZE}|; \text{ if FRECFORM}=\text{FIXUNB} \\ |\text{FRECSIZE}+4|; \text{ if FRECFORM}=\text{VARUNB} \\ |\text{FRECSIZE}*4|; \text{ otherwise} \end{array} \right\} \right]$$

[FREREAD=YES]

[FDIAGNOS=YES]

[FBKNO=YES]

$$\left[\text{FERROPT} = \left\{ \begin{array}{l} \text{IGNORE} \\ \text{SKIP} \end{array} \right\} \right]$$

[FCKPT=YES]

$$\left[\text{FFILABL} = \left\{ \begin{array}{l} \text{STD} \\ \text{NO} \end{array} \right\} \right]$$

[FCKPTREC=YES]

$$\left[\text{FCLRW} = \left\{ \begin{array}{l} \text{RWD} \\ \text{NORWD} \\ \text{UNLOAD} \end{array} \right\} \right]$$

[FOPRW=NORWD]

[FOPTION=YES]

Device Identification Argument:

FDEVICE=TAPE

Specifies that this is a tape file.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant in the range $1 \leq k \leq 99$.

FUNIT=READ

Specifies READ as the unit identifier.

FUNIT=PUNCH

Specifies PUNCH as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module.

The identifiers READ and PUNCH are provided for reference by the FORTRAN II statements READ and PUNCH, respectively, since these statements contain no specific unit identification. When a FORTRAN II statement is executed and one of these special identifiers has not been provided, the applicable device specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

File Name Argument:

FFILEID=filename

Specifies a 1- to 7-character FORTRAN style symbolic name (filename).

FFILEID=FORTk

Specifies the file name as FORTk, where $1 \leq k \leq 99$. If the FFILEID argument is not specified and FUNIT=k has been specified, FORTk is the default file name.

FFILEID=READER

Specifies the file name as READER. If the FFILEID argument is not specified and FUNIT=READ has been specified, READER is the default file name.

FFILEID=PUNCH

Specifies the file name as PUNCH. If the FFILEID argument is not specified and FUNIT=PUNCH has been specified, PUNCH is the default file name.

Type-of-File Argument:

FTYPEFLE=WORK or **FTYPEFLE=INOUT**

Specifies a work file. If this argument is not specified and FUNIT=k has been specified, WORK is the FTYPEFLE default. FTYPEFLE=WORK should be specified if the tape is to be read and written. WORK files are limited to a single volume (reel). ←

FTYPEFLE=INPUT

Specifies an input file. If this argument is not specified and FUNIT=READ has been specified, INPUT is the FTYPEFLE default. FTYPEFLE=INPUT should be specified if the tape is to be read but never written.

FTYPEFLE=OUTPUT

Specifies an output file. If this argument is not specified and FUNIT=PUNCH has been specified, OUTPUT is the FTYPEFLE default. FTYPEFLE=OUTPUT should be specified if the tape is to be written but never read.

Record Format Argument:

FRECFORM=VARUNB

Specifies variable-length unblocked records.

FRECFORM=VARBLK

Specifies variable-length blocked records. If this option is chosen, BACKSPACE is not allowed.

FRECFORM=FIXUNB

Specifies fixed-length unblocked records.

FRECFORM=FIXBLK

Specifies fixed-length blocked records. If this option is chosen, BACKSPACE is not allowed.

Buffer Allocation Argument:

FNUMBUF=1

Specifies one buffer to be allocated to a unit. This argument is required if BACKSPACE is to be allowed.

FNUMBUF=2

Specifies two buffers to be allocated to a unit.

Work Area Allocation Argument:

This argument specifies whether records are to be processed directly in the buffer or moved to and from a work area for processing.

FWORKA=YES

Specifies that space for a work area is to be allocated. If this argument is omitted and FNUMBUF=1 is specified, the default is that space is allocated for a work area.

FWORKA=NO

Specifies that no space for a work area is to be allocated. If this argument is omitted and FNUMBUF=2 is specified, the default is that no space is allocated for a work area. This argument is required if BACKSPACE is to be allowed.

Buffer Pooling Argument:

FBUFPOOL= YES

Specifies that buffer pooling is to be used.

The buffers are to be logically equivalent with all other units for which buffer pooling is specified.

When multiple units with pooled buffers are active, only unblocked records may be processed, and only one buffer can be used. A unit is active from the first reference to the unit until termination, which on input means an END clause return and on output means the execution of an ENDFILE statement. If only one unit using buffer pooling is active at a given time, record blocking and double buffering can be used.

Record Size Argument:

**FRECSIZE= { K }
 { 508 }**Specifies a positive integer constant (k) in the range $18 \leq k \leq 32767$ if fixed records are specified and $14 \leq k \leq 32767$ if variable records are specified. If this argument is omitted, 508 is the default record size.

Extended FORTRAN pads out all variable-length records to 18 bytes if necessary. This implies that it is impossible to detect all instances when the program requests records longer than the length written. Fixed-length records must be at least 18 bytes.

Block Size Argument:

This argument specifies the block size, which must always be greater than or equal to the record size. The default values for FBKSZ depend on the absolute value of the FRECSIZE specification and on the record form used.

FBKSZ=k

Specifies the block size (k) as a positive integer constant in the range $18 \leq k \leq 32767$.

FBKSZ=|FRECSIZE|

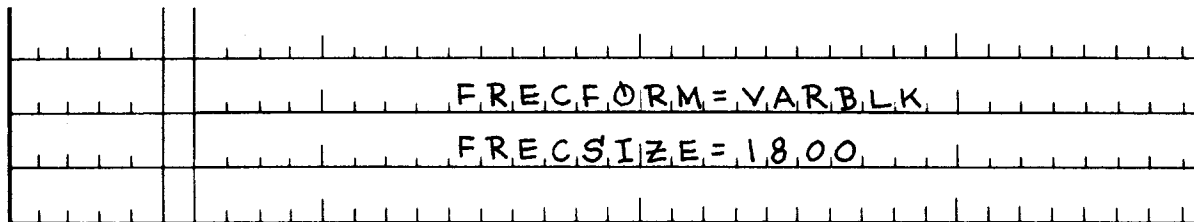
Indicates the blocksize is equal to the record size. If this argument is not specified, and fixed unblocked records have been specified, this is the default block size.

FBKSZ=|FRECSIZE+4|

Indicates the block size is four more than the record size. If this argument is not specified, and variable unblocked records have been specified, this is the default block size.

FBKSZ=|FRECSIZE*4|

Indicates the block size as four times the record size. If this argument is not specified and blocked records have been specified, this is the default. Files containing blocked records cannot be backspaced.

Example:

In this example FBKSZ is not specified. Since FRECFORM=VARBLK is specified, the default value for FBKSZ is equal to four times the value of FRECSIZE.

Reread Argument:**FREREAD=YES**

Specifies that a unit is to participate in the reread feature (7.3.4).

The reread unit consists of a single buffer to which each formatted input record is transferred. To conserve central processor time, this data movement is inhibited unless specifically requested.

Diagnostic Messages Argument:**FDIAGNOS=YES**

Specifies the current unit as the diagnostic device. If FRECSIZE is specified, its value must be 101 or more. Debugging information may also be written to this device (9.3). This argument is not available for input files.

The Extended FORTRAN run-time environment always requires a device for diagnostic purposes.

Block Numbering Argument:**FBKNO=YES**

Specifies that sequence numbers are to be encoded in each block before it is written and checked after each block is read. These block numbers are not visible to the FORTRAN programmer.

Device Error Processing Arguments:

Two arguments are used to specify device error processing.

■ FERROPT

Specifies action to be taken when an erroneous data block is encountered.

If omitted, specifies that control is to be transferred to the ERR clause of the READ statement; abnormal termination procedures are to be initiated if the ERR clause is not present.

FERROPT=IGNORE

Specifies that the erroneous block is to be accepted.

FERROPT=SKIP

Specifies that the erroneous block is to be bypassed by reading the next block.

SKIP and IGNORE should be used with discretion, since device position may be lost for unformatted files and NAMELISTs.

When the problem program receives control at the ERR label, the ERROR subroutine (5.6.3) should be referenced to determine the error type. If the error is unrecoverable, the unit cannot be referenced again. Unrecoverable errors can be caused by severe device failure, parity errors that cause inconsistent control information, or any error on a list-directed statement, which always implies loss of position.

If the error is recoverable, the device is considered operable. Further references to the unit deliver subsequent logical records; the erroneous record is bypassed. A parity or wrong length error on a blocked file causes an ERR return for every logical record in the erroneous block. The term "logical record" is interpreted identically with the BACKSPACE statement (7.3.6.2).

■ FRECERR

FRECERR=YES

Specifies that formatted records in blocks with parity or wrong length errors are to be moved to the reread buffer. Access to these records is required by some application programs.

After an ERR return, the reread unit may be referenced to recover the data, which may contain one or more erroneous bits. The next reference to the unit in error delivers the next record or causes another ERR return. A reread unit must be defined to access the reread buffer (11.3.4.7). Refer also to the ERRDEF procedure (11.3.6).

Tape Label Checking Argument:

FFILABL=STD

Specifies that system standard labels are assumed.

FFILABL=NO

Specifies that tapes are to be read and written without labels.

Checkpoint Processing Argument:

FCKPT=YES

Specifies that an input tape file contains operating system checkpoint dumps used to restart programs after a catastrophic failure.

The block size must be 20 bytes or larger when this argument is used. FCKPT must be specified when checkpoint dumps are present.

Tape Rewind Arguments:

Two arguments may be used to specify tape rewinding. They have no effect on the FORTRAN REWIND command.

■ FCLRW

FCLRW=RWD

Specifies that the tape is to be rewound to loadpoint when the STOP statement is executed.

FCLRW=NORWD

Specifies that there is to be no rewind when the STOP statement is executed.

FCLRW=UNLOAD

Specifies that there is to be rewind with interlock when the STOP statement is executed and that the tape is inaccessible to subsequent steps in the job without operator intervention.

■ FOPRW

FOPRW=NORWD

Specifies that the tape is not to be rewound to load point when it is first referenced.

Optional Units Argument:

FOPTION= YES

Specifies an optional unit, a unit not always required during program execution.

When this argument is specified and the file has not been allocated by job control statements, WRITE statements are effectively ignored, and the first READ reference will cause an end-of-file condition to occur.

A unit need not be declared as optional if the logic of the program does not cause a reference to the unit.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT					
STATEMENT NUMBER	LINE	5	10	20	30	40	72
				UNIT	FDEVICE=TAPE,		X
				FUNIT=7,			X
				FTYPEFILE=INPUT,			X
				FREC.FDRM=VARBLK,			X
				FWORKA=YES,			X
				FRECSIZE=400,			X
				FBKSZ=1000,			X
				FCKPT=YES			

This example defines a tape file, unit 7, used for input only. The records are variable in length, with a maximum size of 400 bytes, and blocked into a maximum blocksize of 1000 bytes. The file is processed by using a work area, and checkpoint records are present and are to be bypassed when encountered.

The assumed defaults are:

- file name is FORT7;
 - no buffer pooling, one unique buffer;
 - no reread;
 - not the diagnostic device;
 - no block numbering;
 - device errors to be returned to ERR labels with the record bypassed and not made available to reread;
 - no labels;
 - rewinds at start and end of processing; and
 - file is required if a reference occurs.
- ■ BACKSPACE is not allowed because a work area is requested.

11.3.4.5. Sequential Disc Files

A single sequential disc file is defined by using the UNIT procedure call presented in this paragraph. Following is a listing, in the order of their relative importance and utility, of the arguments that may appear on the UNIT procedure call. Following the listing, descriptions of UNIT arguments and a UNIT example are presented.

→ Sequential disc files are conceptually identical with tape files, except that BACKSPACE is not allowed. Most arguments are treated identically with tape file arguments.

Format:

1	10	16
	UNIT	<p>FDEVICE=SDISC</p> <p>FUNIT=$\left\{ \begin{array}{l} k \\ \text{READ} \\ \text{PUNCH} \end{array} \right\}$</p> <p>[FSECTOR=$\left\{ \begin{array}{l} \text{NO} \\ \underline{\text{YES}} \end{array} \right\}$] ←</p> <p>[FFILEID=$\left\{ \begin{array}{l} \text{filename} \\ \underline{\text{FORTk}}; \text{ if FUNIT=k} \\ \underline{\text{READER}}; \text{ if FUNIT=READ} \\ \underline{\text{PUNCH}}; \text{ if FUNIT=PUNCH} \end{array} \right\}$] ←</p> <p>[FTYPEFLE=$\left\{ \begin{array}{l} \text{INOUT} \\ \underline{\text{WORK}}; \text{ if FUNIT=k} \\ \underline{\text{INPUT}}; \text{ if FUNIT=READ} \\ \underline{\text{OUTPUT}}; \text{ if FUNIT=PUNCH} \end{array} \right\}$] ←</p> <p>[FRECFORM=$\left\{ \begin{array}{l} \underline{\text{VARUNB}} \\ \underline{\text{VARBLK}} \\ \underline{\text{FIXUNB}} \\ \underline{\text{FIXBLK}} \end{array} \right\}$] ←</p> <p>[FNUMBUF=$\left\{ \begin{array}{l} 1 \\ \underline{2} \end{array} \right\}$] ←</p> <p>[FBUFPOOL=YES]</p> <p>[FWORKA=$\left\{ \begin{array}{l} \underline{\text{YES}}; \text{ if FNUMBUF=1} \\ \underline{\text{NO}}; \text{ if FNUMBUF=2} \end{array} \right\}$]</p> <p>[FRECSIZE=$\left\{ \begin{array}{l} k \\ \underline{508} \end{array} \right\}$]</p> <p>[FBKSZ=$\left\{ \begin{array}{l} k \\ \underline{\text{FRECSIZE}}; \text{ if FRECFORM=FIXUNB} \\ \underline{\text{FRECSIZE} + 4}; \text{ if FRECFORM=VARUNB} \\ \underline{\text{FRECSIZE} * 4}; \text{ otherwise} \end{array} \right\}$]</p> <p>[FREREAD=YES]</p> <p>[FDIAGNOS=YES]</p> <p>[FERROPT=$\left\{ \begin{array}{l} \text{IGNORE} \\ \underline{\text{SKIP}} \end{array} \right\}$]</p> <p>[FRECCERR=YES]</p> <p>[FOPTION=YES]</p> <p>[FVERIFY=YES]</p>

Device Identification Argument:

FDEVICE=SDISC

Specifies that this is a sequential disc file.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant (k) in the range $1 \leq k \leq 99$.

FUNIT=READ

Specifies READ as the unit identifier.

FUNIT=PUNCH

Specifies PUNCH as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module. The identifiers READ and PUNCH are provided for reference by the FORTRAN II statements READ and PUNCH, respectively, since these statements contain no specific unit identification. When a FORTRAN II statement is executed and one of these special identifiers has not been provided, the applicable device specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

Sector Processing Argument:

FSECTOR=YES

Specifies that processing on a sectorized disc is expected (e.g., an 8416). FSECTOR parameter is valid for all file types: input, output, or work. The Extended FORTRAN I/O system ensures that all I/O areas, including pooled I/O areas, are integral multiples of 256 bytes in length. This is necessary to prevent program termination or destruction of data.

FSECTOR=NO

Specifies that processing on a nonsectorized disc is expected. This argument is used to conserve space in main storage.

File Name Argument:

FFILEID=filename

Specifies a 1- to 7-character FORTRAN style symbolic name (filename).

FFILEID=FORTk

Specifies the file name as FORTk, where $1 \leq k \leq 99$. If the FFILEID argument is not specified and FUNIT=k has been specified, FORTk is the default file name.

FFILEID=READER

Specifies the file name as READER. If the FFILEID argument is not specified and FUNIT=READ has been specified, READER is the default file name.

FFILEID=PUNCH

Specifies the file name as PUNCH. If the FFILEID argument is not specified and FUNIT=PUNCH has been specified, PUNCH is the default file name.

Type of File Argument:

FTYPEFLE=WORK or FTYPEFLE=INOUT

Specifies a work file. If this argument is not specified and FUNIT=k has been specified, WORK is the FTYPEFLE default. FTYPEFLE=WORK should be specified if the disc is to be read and written.

FTYPEFLE=INPUT

Specifies an input file. If this argument is not specified and FUNIT=READ has been specified, INPUT is the FTYPEFLE default. FTYPEFLE=INPUT should be specified if the disc is to be read but never written.

FTYPEFLE=OUTPUT

Specifies an output file. If this argument is not specified and FUNIT=PUNCH has been specified, PUNCH is the default file name. FTYPEFLE=OUTPUT should be specified if the disc is to be written but never read.

Record Format Argument:

FRECFORM=VARUNB

Specifies variable-length unblocked records.

FRECFORM=VARBLK

Specifies variable-length blocked records.

FRECFORM=FIXUNB

Specifies fixed-length unblocked records. If specifying a FIXBLK file (read to end-of-file and then write), FSECTOR=YES should be specified since I/O on system sectorized discs must be done to prepare for extension. ←

FRECFORM=FIXBLK

Specifies fixed-length blocked records.

Buffer Allocation Argument:

FNUMBUF=1

Specifies one buffer to be allocated to a unit.

FNUMBUF=2

Specifies two buffers to be allocated to a unit.

Buffer Pooling Argument:

FBUFPOOL=YES

Specifies that buffer pooling is to be used.

The buffers are to be logically equivalent with all other units for which buffer pooling is specified.

When multiple units with pooled buffers are active, only unblocked records may be processed and only one buffer can be used. A unit is active from the first reference to the unit until termination, which on input means an END clause return and on output means the execution of an ENDFILE statement. If only one unit using buffer pooling is active at a given time, record blocking and double buffering can be used.

Work Area Allocation Argument:

This argument specifies whether records are to be processed directly in the buffer or moved to and from a work area for processing.

FWORKA=YES

Specifies that space for a work area is to be allocated. If this argument is omitted and FNUMBUF=1 is specified, the default is that space is allocated for a work area.

FWORKA=NO

Specifies that no space for a work area is to be allocated. If this argument is omitted and FNUMBUF=2 is specified, the default is that no space is allocated for a work area.

Record Size Argument:

$$\text{FRECSIZE} = \left\{ \begin{array}{l} k \\ \underline{508} \end{array} \right\}$$

Specifies the record size as a positive integer constant (k). If this argument is omitted, 508 is the default record size. See the specific disc subsystem reference manuals for maximum and minimum record size specifications.

Block Size Argument:

This argument specifies the block size, which must always be greater than or equal to the record size.

The default value for FBKSZ depend on the absolute value of the FRECSIZE specification and on the record form used.

FBKSZ=k

Specifies the block size as a positive integer constant in the range $3 \leq k \leq 3625$. The upper limit can be increased to 7294 bytes for SPERRY UNIVAC 8414/8424/8425 Disc Drive Units and to 13030 bytes for SPERRY UNIVAC 8430 Disc Drive Units.

FBKSZ=FRECSIZE

Indicates the block size is equal to the record size. If this argument is not specified and fixed unblocked records have been specified, this is the default block size.

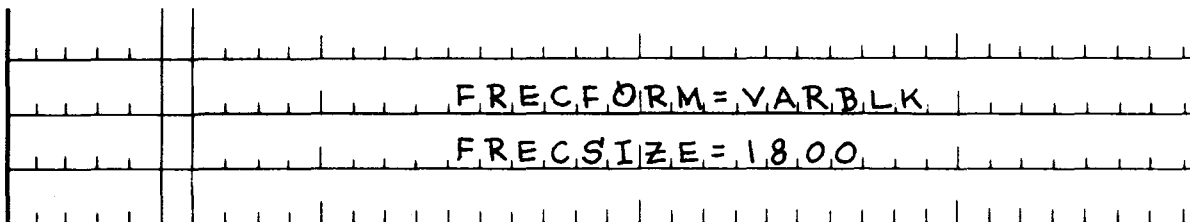
FBKSZ=FRECSIZE+4

Indicates the block size is four more than the record size. If this argument is not specified and variable unblocked records have been specified, this is the default block size.

FBKSZ=FRECSIZE*4

Indicates the block size is four times the record size. If this argument is not specified and blocked records have been specified, this is the default block size.

Example:



In this example, FBKSZ is not specified. Since FRECFORM=VARBLK is specified, the default value for FBKSZ is equal to four times the value of FRECSIZE.

Reread Argument:

FREREAD=YES

Specifies that a unit is to participate in the reread feature (7.3.4).

The reread unit consists of a single buffer to which each formatted input record is transferred. To conserve central processor time, this data movement is inhibited unless specifically requested.

Diagnostic Messages Argument:

FDIAGNOS=YES

Specifies the current unit as the diagnostic device. If FRECSIZE is specified, its value must be 101 or more. Debugging information may also be written to this device (9.3). This argument is not available for input files.

The Extended FORTRAN runtime environment always requires a device for diagnostic purposes.

Device Error Processing Arguments:

Two arguments are used to specify device error processing.

■ **FERROPT**

Specifies action to be taken when an erroneous data block is encountered.

If omitted, specifies that control is to be transferred to the ERR clause of the READ statement; abnormal termination procedures are to be initiated if the ERR clause is not present.

FERROPT=IGNORE

Specifies that the erroneous block is to be accepted.

FERROPT=SKIP

Specifies that the erroneous block is to be bypassed by reading the next block.

SKIP and IGNORE should be used with discretion, since device position may be lost for unformatted files and NAMELISTs.

When the problem program receives control at the ERR label, the ERROR subroutine (5.6.3) should be referenced to determine the error type. If the error is unrecoverable, the unit cannot be referenced again. Unrecoverable errors can be caused by severe device failure, parity errors that cause inconsistent control information, or any error on a list-directed statement, which always implies loss of position.

If the error is recoverable, the device is considered operable. Further references to the unit deliver subsequent logical records; the erroneous record is bypassed. A parity or wrong length error on a blocked file causes an ERR return for every logical record in the erroneous block. The term "logical record" is interpreted identically to the BACKSPACE statement (7.3.6.2).

■ **FRECERR****FRECERR=YES**

Specifies that formatted records in blocks with parity or wrong length errors are to be moved to the reread buffer. Access to these records is required by some application programs.

After an ERR return, the reread unit may be referenced to recover the data, which may contain one or more erroneous bits. The next reference to the unit in error delivers the next record or causes another ERR return. A reread unit must be defined to access the reread buffer (11.3.4.7). Refer also to the ERRDEF procedure (11.3.6).

Optional Units Argument:

FOPTION=YES

Specifies an optional unit, a unit not always required during program execution.

When this argument is specified and the file has not been allocated by job control statements, WRITE statements are effectively ignored, and the first READ reference will cause an end-of-file condition to occur.

A unit need not be declared as optional if the logic of the program does not cause a reference to the unit.

Sector Processing Argument:

→ **FSECTOR=YES**

Specifies that processing on a sectorized disc is expected (e.g., an 8416 or 8418). FSECTOR parameter is valid for all file types: input, output, or work. The Extended FORTRAN I/O system ensures that all I/O areas, including pooled I/O areas, are integral multiples of 256 bytes in length. This is necessary to prevent program termination or destruction of data.

↓
↑ **FSECTOR=NO**

Specifies that processing on a nonsectorized disc is expected. This argument is used to conserve space in main storage.

Write Verification Argument:

FVERIFY=YES

Specifies that all WRITE statements cause the data to be automatically read back to ensure proper recording on the disc surface.

This increased reliability necessarily causes some performance degradation.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT						
STATEMENT NUMBER	5	6	7	10	20	30	40	72
								X
								X
								X
								X

This FUNDEF procedure call specifies a sequential disc file, unit 9, intended for output of variable unblocked records with a maximum size of 1000 bytes. Each record is read checked after it is written. The defaults assumed are:

- file is named FORT9;
- variable unblocked records and one unique 1004-byte buffer with a work area;
- not a diagnostic device and not optional to the program;
- records never reread and not available in the reread buffer after ERR returns.

11.3.4.6. Direct Access Disc File Definition

A direct access disc file is defined by using the UNIT procedure call presented in this paragraph. Following is a listing, in the order of their relative importance and utility, of the arguments that may appear on the FUNDEF procedure call. Following the listing, descriptions of FUNDEF arguments, programming considerations, and a FUNDEF example are presented.

Direct access disc file options are treated identically with sequential disc file options.

Format:

1	10	16
	UNIT	FDEVICE=DISC FUNIT=k [FSECTOR={ NO }] [FSECTOR={ YES }] [FFILID={ filename }] [FFILID={ FORTk; where k = FUNIT }] [FTYPEFLE={ INPUT }] [FTYPEFLE={ OUTPUT }] [FBUFPOOL=YES] [FRECSIZE={ k }] [FRECSIZE={ 512 }] [FREREAD=YES] [FRECERR=YES] [FVERIFY=YES]

Device Identification Argument:

FDEVICE=DISC

Specifies that this is a direct access disc file.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant(k) in the range $1 \leq k \leq 99$.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT and PUNCH) may be specified by a control module.

Sector Processing Argument:

FSECTOR=YES

Specifies that processing on a sectorized disc is expected (e.g., an 8416). FSECTOR parameter is valid for all file types: input, output, or work. The Extended FORTRAN I/O system ensures that all I/O areas, including pooled I/O areas, are integral multiples of 256 bytes in length. This is necessary to prevent program termination or destruction of data.

FSECTOR=NO

Specifies that processing on a nonsectorized disc is expected. This argument is used to conserve space in main storage.

File Name Argument:

FFILEID=filename

Specifies a 1- to 7-character FORTRAN style symbolic name.

FFILEID=FORTk

Specifies the file name as FORTk, where $1 \leq k \leq 99$. If the FFILEID argument is not specified and FUNIT=k has been specified, FORTk is the default file name.

Type of File Argument:

FTYPEFLE=INPUT

Specifies an input file.

FTYPEFLE=OUTPUT

Specifies an output file.

See Programming Considerations.

Buffer Pooling Argument:

FBUFPOOL=YES

Specifies that buffer pooling is to be used.

The buffers are to be logically equivalent with all other units for which buffer pooling is specified.

When multiple units with pooled buffers are active, only one buffer can be used. A unit is active from the first reference to the unit until termination, which on input means an END clause return and an output means the execution of an ENDFILE statement.

Record Size Argument:

**FRECSIZE= { k }
 { 512 }**

Specifies the record size as a positive integer constant. If this argument is omitted, 512 is the default record size. See the specific disc subsystem reference manuals for maximum and minimum record size specifications.

Reread Argument:

FREREAD=YES

Specifies that a unit is to participate in the reread feature (7.3.4).

The reread unit consists of a single buffer to which each formatted input record is transferred. To conserve central processor time, this data movement is inhibited unless specifically requested.

Device Error Processing Argument:

FRECERR=YES

Specifies that formatted records in blocks with parity errors or wrong length errors are to be moved to the reread buffer. Access to these records is required by some application programs.

After an ERR return, the reread unit may be referenced to recover the data, which may contain one or more erroneous bits. The next reference to the unit in error delivers the next record or causes another ERR return. A reread unit must be defined to access the reread buffer (11.3.4.7). Refer also to the ERRDEF procedure (11.3.6).

Write Verification Argument:

FVERIFY= YES

Specifies that all WRITE statements cause the data to be automatically read back to ensure proper recording on the disc surface.

This increased reliability causes some performance degradation.

Programming Considerations:

The TYPEFLE specification does not restrict the use of READ and WRITE statements. The only implication of TYPEFLE is that, for INPUT, label checking is performed and, for OUTPUT, labels are written. When the associated DEFINE FILE statement is executed, it must specify a record size less than or equal to FRECSIZE. The Extended FORTRAN system thereafter enforces the record size specified on the DEFINE FILE statement, but always transfers records of FRECSIZE bytes to and from the disc.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT					
STATEMENT NUMBER		5	10	20	30	40	72
							X
							X
							X

This UNIT procedure call specifies a direct access disc file, unit 13, which is to be read only. The record size is 348. The defaults are:

- file name is FORT13;
- no buffer pooling;
- buffer size is 348;
- no reread and records not available in the reread buffer after ERR returns; and
- no verification.

11.3.4.7. Reread Unit Definition

The reread unit is defined by using the UNIT procedure call presented in this paragraph. Following is a listing, in the order of their relative importance and utility, of the arguments that may appear on the UNIT procedure call. Following the listing, description of UNIT arguments, programming considerations, and a UNIT example are presented.

A single VARUNB buffer is automatically constructed with a size equivalent to the largest record size of all the units in the reread feature.

Format:

1	10	16
	UNIT	FDEVICE=REREAD
		FUNIT={ ^k READ }

Device Identification Argument:

FDEVICE=REREAD

Specifies that this is the reread unit.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant (k) in the range $1 \leq k \leq 99$.

FUNIT=READ

Specifies READ as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT and PUNCH) may be specified by a control module. The identifier READ is provided for reference by the FORTRAN II READ statement, since this statement contains no symbolic unit identification. When a FORTRAN II statement is executed and this special identifier has not been provided, the applicable unit specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

Programming Considerations:

The record in the reread buffer is redefined only after a formatted READ statement is issued to a unit specified with FREREAD=YES or FRECERR=YES.

Example:

STATEMENT NUMBER	FORTRAN STATEMENT	
5	UNIT, FDEVICE=REREAD,	72
	FUNIT=29	X

The reread unit is defined to be unit 29.

11.3.4.8. Equivalent Unit Definition

An equivalent unit is defined by using the UNIT procedure call presented in this paragraph. Following is a listing, in the order of their relative importance and utility, of the arguments that may appear on the UNIT procedure call. Following the listing, descriptions of UNIT arguments and a UNIT example are presented.

The function of an equivalent unit is to provide another reference number for a file. For example, an input file might be referenced with both an extended FORTRAN statement with a unit number and FORTRAN II statement that implies the special name READ. An equivalent unit can be used to resolve conflicts of this type.

Format:

1	10	16
	UNIT	FDEVICE=EQUIV
		$\text{FUNIT} = \left\{ \begin{array}{l} k \\ \text{READ} \\ \text{PRINT} \\ \text{PUNCH} \end{array} \right\}$
		$\text{FEQUIV} = \left\{ \begin{array}{l} k \\ \text{READ} \\ \text{PRINT} \\ \text{PUNCH} \end{array} \right\}$

Device Identification Argument:

FDEVICE=EQUIV

Specifies that this is an equivalent unit.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant (k) in the range $1 \leq k \leq 99$.

FUNIT=READ

Specifies READ as the unit identifier.

FUNIT=PRINT

Specifies PRINT as the unit identifier.

FUNIT=PUNCH

Specifies PUNCH as the unit identifier.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module. The identifiers READ, PRINT, and PUNCH are provided for reference by the FORTRAN II statements READ, PRINT, and PUNCH, respectively, since these statements contain no specific unit identification. When a FORTRAN II statement is executed and one of these special identifiers has not been provided, the applicable device specified is used. The units are searched in the order in which they are defined. In an executable program, only one such unit may be defined.

Establishing Equivalence Argument:

This argument is used to specify the unit that is to be treated as equivalent to the unit specified for FUNIT. When a file reference to the unit specified for FUNIT occurs, device action takes place on the unit specified for FEQUIV.

FEQUIV=k

Specifies a unique integer constant (k) in the range $1 \leq k \leq 99$.

FEQUIV=READ

Specifies READ as the equivalent unit.

FEQUIV=PRINT

Specifies PRINT as the equivalent unit.

FEQUIV=PUNCH

Specifies PUNCH as the equivalent unit.

Examples:

"C" FOR COMMENT		STATEMENT NUMBER		FORTRAN STATEMENT			
1	5	6	7	10	20	30	72
				UNIT	FDEVICE=	EQUIV,	X
					FUNIT=	PRINT,	X
					FEQUIV=	5,	

This UNIT procedure call specifies an equivalent unit that has no number; it can be referenced only by using a FORTRAN II PRINT statement. When referenced, unit 5 is activated; unit 5 must be defined by using another UNIT procedure call.

Circular equivalences, such as the following, are not permitted.

				UNIT	FDEVICE=	EQUIV,	X
					FUNIT=	1,	X
					FEQUIV=	2,	
				UNIT	FDEVICE=	EQUIV,	X
					FUNIT=	2,	X
					FEQUIV=	1,	

11.3.5. FORTRAN Unit Definition Termination Procedure (FUNEND)

The list of units specified with UNIT procedure calls is terminated by the FUNEND procedure call. The FUNEND procedure call:

- terminates the unit list;
- generates a work area large enough to accommodate any unit for which FWORKA=YES is specified;
- generates one or two buffers large enough to accommodate any unit for which FBUFPOOL=YES is specified;
- generates a reread buffer large enough to accommodate any unit for which FREREAD=YES or FRECERR=YES is specified; and
- guarantees the presence of a diagnostic unit.

If FDIAGNOS=YES was specified for a unit, no action takes place. If a unit was specified as, or defaulted to, FFILEID=PRNTR, that unit is specified as the diagnostic device. If neither of the preceding conditions holds, a UNIT procedure call with the following form is generated, and a warning diagnostic is issued.

"C" FOR COMMENT		STATEMENT NUMBER		FORTRAN STATEMENT			
1	5	6	7	10	20	30	72
					UNIT	FDEVICE=PRINTER,	X
					FUNIT=PRINT,		X
					FDIAGNOS=YES,		X
					FRECSIZE=101		

Format:

1	10	
		FUNEND

11.3.6. Error Environment Definition Procedure (ERRDEF)

During the execution of the object program, the Extended FORTRAN system monitors program operations for consistency and legality, insofar as it is practical. The errors detected are grouped into seven classes, each having a limit on the number of times the error is to be accepted before program termination and on the number of diagnostic messages to be produced. The seven error classes include program, arithmetic, argument, alignment, read, and data errors, explained in the following paragraphs, and fatal errors, which are catastrophic errors forcing immediate program termination. A table in the library contains the limit information for each class. This table is automatically included in the executable program if the table is not explicitly redefined by the programmer. For this purpose, the error definition procedure (ERRDEF) is provided.

Treatment of nonfatal errors can be controlled by using the ERRDEF procedure call. Following is a listing, in order of relative importance and utility, of the arguments that may appear on the ERRDEF procedure call. Following the listing, descriptions of ERRDEF arguments and an ERRDEF example are presented. The ERRDEF procedure call should follow the FUNEND procedure call in the configuration module.

Format:

1	10	16
	ERRDEF	$\left[\text{FPROG} = \left(\left\{ \begin{matrix} i \\ \text{ALL} \\ \underline{1} \end{matrix} \right\}, \left\{ \begin{matrix} j \\ \text{ALL} \\ \underline{1} \end{matrix} \right\} \right) \right]$ $\left[\text{FARITH} = \left(\left\{ \begin{matrix} i \\ \text{ALL} \end{matrix} \right\}, \left\{ \begin{matrix} j \\ \text{ALL} \\ \underline{10} \end{matrix} \right\} \right) \right]$ $\left[\text{FARG} = \left(\left\{ \begin{matrix} i \\ \text{ALL} \end{matrix} \right\}, \left\{ \begin{matrix} j \\ \text{ALL} \\ \underline{10} \end{matrix} \right\} \right) \right]$ $[\text{FUNDFLO} = \text{YES}]$ $\left[\text{FALIGN} = \left(\left\{ \begin{matrix} i \\ \text{ALL} \end{matrix} \right\}, \left\{ \begin{matrix} j \\ \text{ALL} \\ \underline{10} \end{matrix} \right\} \right) \right]$ $\left[\text{FREAD} = \left(\left\{ \begin{matrix} i \\ \text{ALL} \end{matrix} \right\}, \left\{ \begin{matrix} j \\ \text{ALL} \\ \underline{10} \end{matrix} \right\} \right) \right]$ $\left[\text{FDATA} = \left(\left\{ \begin{matrix} i \\ \text{ALL} \end{matrix} \right\}, \left\{ \begin{matrix} j \\ \text{ALL} \\ \underline{10} \end{matrix} \right\} \right) \right]$ $\left[\text{FERROPT} = \left(\begin{matrix} \text{NONE} \\ \text{READ} \\ \text{READ,DATA} \\ \text{READ,UNREC} \\ \text{READ,DATA,UNREC} \\ \text{DATA} \\ \text{DATA,UNREC} \\ \text{UNREC} \end{matrix} \right) \right]$

where:

i

Is a positive integer constant less than 32,768, with $i \geq j$, specifying the number of times the error is to be accepted before program termination. For a fatal error, *i* is assumed to be 1.

j

Is a positive integer constant less than 32,768, with $j \leq i$, specifying the number of diagnostic messages to be produced. For a fatal error, *j* is assumed to be 1.

ALL

Specifies that there is no limit on the number of times the error is to be accepted before program termination or that there is no limit on the number of diagnostic messages to be produced.

During execution, the first *j* errors cause diagnostic messages to be produced; when the *ith* error occurs, a diagnostic is issued, and program termination is initiated.

Program Error Argument (FPROG):

This argument is used to control system action when the flow of execution encounters a statement for which code cannot be generated because of a syntax or other error or when an error occurs in FORMAT-I/O list interaction.

Arithmetic Error Argument (FARITH):

This argument is used to control system action when a program check interrupt occurs for overflow, underflow, or divide check. The standard library functions (Table 5-4) cannot cause this error.

Argument Error Argument (FARG):

This argument is used to control system action when an out-of-range argument is transmitted to a standard library function (Table 5-4).

Improper argument values can cause error reporting by the standard library functions (Table 5-3) because:

- the function is mathematically undefined for the argument, as SQRT (-10);
- the function value is insignificant, as SIN (10E60); or
- the function value is too large to be represented, as 10E50** 10E50. This is analogous to an overflow condition.

As a default, a function value too small to be represented (an underflow) is approximated by 0 and is not reported or considered on the FARG counts.

Underflow Error Argument (FUNDFLO):

FUNDFLO=YES

Used to control system action when underflows occur. This argument indicates that underflow will be reported and counted.

Alignment Error Argument (FALIGN):

This argument is used to control system action when a program check interrupt occurs for an instruction referencing an illegal main storage boundary. This can occur because of improper COMMON and EQUIVALENCE statements and during argument substitution in prologues and epilogues.

Read Error Argument (FREAD):

This argument is used to control system action when an input device error occurs. The error counts associated with FREAD are meaningful only when an ERR clause is present in the referencing statement. If no ERR clause is present, the program is immediately terminated, regardless of the specifications for the number of times the error may be accepted or the number of diagnostic messages that may be produced.

Data Error Argument (FDATA):

This argument is used to control system action when the input data contains illegal characters. The error counts associated with FDATA are meaningful only when an ERR clause is present in the referencing statement. If no ERR clause is present, the program is immediately terminated, regardless of the specifications for the number of times the error may be accepted or the number of diagnostic messages that may be produced.

Error Options Argument (FERROPT):

This argument specifies the meaning of the ERR clause. The Extended FORTRAN default is to pass control to the ERR label only for parity or wrong length errors. The ERROR subroutine (5.6.3) is used to determine the error type. Eight possible combinations of the following FERROPT specifications are available.

NONE

Used to control system action when the ERR clause feature is to be disabled.

READ

Used to control system action when control is to be passed to the ERR label for parity or wrong length errors only.

DATA

Used to control system action when control is to be passed to the ERR label for data errors; this class is composed of invalid input characters.

UNREC

Used to control system action when control is to be provided at the ERR clause for unrecoverable device errors only. No further references to the unit are permitted.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT					
STATEMENT NUMBER	6	7	10	20	30	72	
5		ERRDEF FPRDG=(100,100),					X
		FALIGN=(ALL,50),					X
		FERROPT=(DATA)					

This ERRDEF procedure call indicates that if a program error occurs, the error may be accepted 100 times, and 100 diagnostic messages may be produced. If an alignment error occurs, there is no limit on the number of times the error may be accepted, and 50 diagnostic messages may be produced. DATA, specified for error options, indicates that control is to be passed to the ERR label if data errors occur. The standard defaults are taken for all arguments not specified.

11.3.7. END Statement

The END statement, a source program terminator statement required by the assembler, indicates the end of the definition of the execution environment.

Format:

1	10
END	

The END statement, coded as shown, follows the ERRDEF procedure call, or if the ERRDEF procedure call is not present, the FUNEND procedure call.

Example:

An example of a complete execution environment follows.

"C" FOR COMMENT		STATEMENT NUMBER	FORTRAN STATEMENT	
		5		
	Cont.	6		
		7	10	20
			30	72
			MYIO, START	
			FUNTAB SYS=FDR	
			UNIT FDEVICE=PRINTER,	X
			FUNIT=12,	X
			FDIAGNOS=YES	
			UNIT FDEVICE=TAPE,	X
			FUNIT=11,	X
			FTYPEFLE=INPUT,	X
			FRECSIZE=200	
			FUNEND	
			ERRDEF FERROPT=(DATA)	←
			END	



Format:

1	10	
	END	

The END statement, coded as shown, follows the ERRDEF procedure call, or if the ERRDEF procedure call is not present, the FUNEND procedure call.

Example:

An example of a complete execution environment follows.

"C" FOR COMMENT		FORTRAN STATEMENT				72
STATEMENT NUMBER	Cont.	7	10	20	30	
MYID		START				
		FUNTAB SYS=FOR				
		UNIT FDEVICE=PRINTER,				X
		FUNIT=12,				X
		FDIAGNOS=YES				
		UNIT FDEVICE=TAPE,				X
		FUNIT=11,				X
		FTYPEFILE=INPUT,				X
		FRECSIZE=200				
		FUNEND				
		FERRDEF FERROPT=(DATA)				
		END				



12. Program Collection and Execution

12.1. GENERAL

Before a set of program units compiled by the SPERRY UNIVAC Operating System/3 (OS/3) Extended FORTRAN can be executed, they must be collected and the necessary FORTRAN supporting routines made available to them. The OS/3 linkage editor performs this task.

After the program units are link edited, all physical devices required for execution are assigned via OS/3 job control statements.

12.2. LINK EDITING FORTRAN PROGRAMS

Several special interfaces of the OS/3 linkage editor are used by Extended FORTRAN and described in this section. The user should be aware of these interfaces to use the linkage editor successfully with FORTRAN compiled programs. The linkage editor options listed in the system service programs programmer reference manual, UP-8209 (current version) can be used only if they do not conflict with requirements of Extended FORTRAN.

12.2.1. FORTRAN Supplied Modules

After programs are compiled by the FORTRAN compiler, various mathematical functions, service routines, and system routines may have to be connected to the programs. This entire group of modules must then be converted into executable format. The functions SIN, ALOG, and CBRT, the subroutines DUMP and DVCHK, and the service routines read-write, integer editing, and error detection are examples of the services which may need to be supplied before a FORTRAN program is executable.

A complete list of functions and services supplied with Extended FORTRAN may be found in Appendix F.

12.2.2. Overlay and Region Structures

Sometimes the executable program created as linkage editor output is too large to fit into the required main storage limits. The OS/3 linkage editor provides overlay and region segmentation methods to assist in creating smaller executable load modules.

Programs compiled by the Extended FORTRAN compiler reference each subprogram with the automatic overlay feature of the OS/3 system. Thus, an overlay structure may be used with no changes to the FORTRAN programs. A few restrictions should be observed, however, so that the FORTRAN service routines operate correctly:

- The root phase of the overlay structure should contain the following:
 - All common areas
 - The execution environment module
 - The modules FL\$IOCOM, FL\$ABTRM, FL\$ERCTL
 - The main program of the execution
- ↓ ■ Any direct access associated variables should be in a common area.
- If the explicit overlay control statements CALL LOAD, CALL FETCH, or CALL OPSYS are used, the automatic overlay feature will not operate correctly. The linkage editor option, NOV, must be specified to suppress normal V-CON processing.
- ↑ ■ Local variables become undefined upon exit from a subprogram if the subprogram is in an overlay.

The user should take care when building overlay structures since program execution speed can be seriously affected.

12.2.3. Linkage Editor Output

The executable module created by the linkage editor is placed in the system file, \$Y\$RUN. The program may be executed directly from this file, or it may be saved with the OS/3 system service routines.

In addition to load modules, the linkage editor produces a listing and a storage map for each load module. All linkage editor errors should be resolved before attempting to execute the program. The storage maps should be saved to aid in debugging the program.

The following example causes a FORTRAN program (\$MAIN) and an execution environment module (MYIO) to be linked to an executable module (TEST). All supporting FORTRAN run-time modules needed by \$MAIN and MYIO programs are automatically included into TEST from the system object module library, \$Y\$OBJ.

```
// WORK 1

// EXEC LNKEDT

/$

LOADM TEST

INCLUDE $MAIN

INCLUDE MYIO

/*
```

12.3. Execution FORTRAN Programs

The Extended FORTRAN compiler uses the OS/3 Operating System and Data Management System to execute its compiled programs. The following information describes the various OS/3 interfaces that FORTRAN requires.

12.3.1. FORTRAN I/O Units

The FORTRAN I/O unit module that is linked to the executable program specifies which units and devices may be used during this execution. The user is responsible for supplying the actual devices which connect to the units in the I/O unit module.

To connect an actual device to an executable program, the user supplies appropriate JCL statements which allocate the device for this job or job step. He must assign a file on the device via the LFD job control statement where the filename on the LFD statement is the same as the FFILEID parameter described in Section 11.

The FORTRAN diagnostic device must always be allocated to the executing program. In all Extended FORTRAN default I/O configurations, this device is a printer with the FFILEID=PRNTR. When a device is not used during an execution of a program, the device need not be assigned.

12.3.2. Pause Messages

If a PAUSE statement is executed, the text of the PAUSE message is displayed on the system console. The program then waits for a response from the operator.

There are three allowable responses to a PAUSE message:

- CONT — continue the program execution.
- STOP — terminate the program normally.
- DUMP — terminate the program with a dump.

If any other response is made, the PAUSE message is reissued.

12.3.3. Diagnostic Messages

The FORTRAN run-time system has many diagnostic messages which may be displayed during execution. These messages are output to the FORTRAN unit assigned for diagnostic information (11.3.4.1, FDIAGNOS=YES).

The amount of information output by the FORTRAN run-time system may be controlled by the error definition procedure (11.3.6); however, the STOP message and execution summary information are always output. Therefore, when using preprinted forms or when printing final draft output, the user should assign the diagnostic device separate from his good copy printer. For a complete list of run-time diagnostics, refer to the system messages programmer/operator reference, UP-8076 (current version).

Diagnostic messages that can be generated during compilation are listed and described in Appendix E.



Appendix A. Character Set

A.1. SOURCE PROGRAM AND INPUT DATA CHARACTERS

Table A-1 shows the EBCDIC input character set for Extended FORTRAN. The table indicates, in the upper portion of each square, the card punches used to represent the hexadecimal value and, in the lower portion of the square, the corresponding character on the keyboard of a SPERRY UNIVAC 1700 Series keypunch. The SPERRY UNIVAC 1004 Card Processor changes to the character blank (hexadecimal Z40) all card punch combinations marked with a dagger. For example, the hexadecimal value C1 is encoded with the keyboard character A, 12-1 hole combination on the punched card.

Table A-1. EBCDIC Input Character Set (Part 1 of 4)

DIGIT 2 DIGIT 1	0	1	2	3	4	5	6	7
0	0, 1, 8, 9 †	12, 1, 9 †	12, 2, 9 †	12, 3, 9 †	12, 4, 9 †	12, 5, 9 †	12, 6, 9 †	12, 7, 9 †
1	12, 11, 1, 8, 9 †	11, 1, 9 †	11, 2, 9 †	11, 3, 9 †	11, 4, 9 †	11, 5, 9 †	11, 6, 9 †	11, 7, 9 †
2	11, 0, 1, 8, 9 †	0, 1, 9 †	0, 2, 9 †	0, 3, 9 †	0, 4, 9 †	0, 5, 9 †	0, 6, 9 †	0, 7, 9 †
3	12, 11, 0, 1, 8, 9 †	1, 9 †	2, 9 †	3, 9 †	4, 9 †	5, 9 †	6, 9 †	7, 9 †
4	(space)	12, 0, 1, 9 †	12, 0, 2, 9 †	12, 0, 3, 9 †	12, 0, 4, 9 †	12, 0, 5, 9 †	12, 0, 6, 9 †	12, 0, 7, 9 †
5	12 &	12, 11, 1, 9 †	12, 11, 2, 9 †	12, 11, 3, 9 †	12, 11, 4, 9 †	12, 11, 5, 9 †	12, 11, 6, 9 †	12, 11, 7, 9 †
6	11 - (minus)	0, 1 /	11, 0, 2, 9 †	11, 0, 3, 9 †	11, 0, 4, 9 †	11, 0, 5, 9 †	11, 0, 6, 9 †	11, 0, 7, 9 †
7	12, 11, 0 †	12, 11, 0, 1, 9 †	12, 11, 0, 2, 9 †	12, 11, 0, 3, 9 †	12, 11, 0, 4, 9 †	12, 11, 0, 5, 9 †	12, 11, 0, 6, 9 †	12, 11, 0, 7, 9 †

Table A-1. EBCDIC Input Character Set (Part 2 of 4)

DIGIT 2 \ DIGIT 1	8	9	A	B	C	D	E	F
0	12, 8, 9 †	12, 1, 8, 9 †	12, 2, 8, 9 †	12, 3, 8, 9 †	12, 4, 8, 9 †	12, 5, 8, 9 †	12, 6, 8, 9 †	12, 7, 8, 9 †
1	11, 8, 9 †	11, 1, 8, 9 †	11, 2, 8, 9 †	11, 3, 8, 9 †	11, 4, 8, 9 †	11, 5, 8, 9 †	11, 6, 8, 9 †	11, 7, 8, 9 †
2	0, 8, 9 †	0, 1, 8, 9 †	0, 2, 8, 9 †	0, 3, 8, 9 †	0, 4, 8, 9 †	0, 5, 8, 9 †	0, 6, 8, 9 †	0, 7, 8, 9 †
3	8, 9 †	1, 8, 9 †	2, 8, 9 †	3, 8, 9 †	4, 8, 9 †	5, 8, 9 †	6, 8, 9 †	7, 8, 9 †
4	12, 0, 8, 9 †	12, 1, 8 †	12, 2, 8 †	12, 3, 8 †	12, 4, 8 †	12, 5, 8 †	12, 6, 8 †	12, 7, 8 †
5	12, 11, 8, 9 †	11, 1, 8 †	11, 2, 8 †	11, 3, 8 †	11, 4, 8 †	11, 5, 8 †	11, 6, 8 †	11, 7, 8 †
6	11, 0, 8, 9 †	0, 1, 8 †	12, 11 †	0, 3, 8 †	0, 4, 8 †	0, 5, 8 †	0, 6, 8 †	0, 7, 8 †
7	12, 11, 0, 8, 9 †	1, 8 †	2, 8 †	3, 8 †	4, 8 †	5, 8 †	6, 8 †	7, 8 †

Table A-1. EBCDIC Input Character Set (Part 3 of 4)

DIGIT 2 \ DIGIT 1	0	1	2	3	4	5	6	7
8	12, 0, 1, 8 †	12, 0, 1 †	12, 0, 2 †	12, 0, 3 †	12, 0, 4 †	12, 0, 5 †	12, 0, 6 †	12, 0, 7 †
9	12, 11, 1, 8 †	12, 11, 1 †	12, 11, 2 †	12, 11, 3 †	12, 11, 4 †	12, 11, 5 †	12, 11, 6 †	12, 11, 7 †
A	11, 0, 1, 8 †	11, 0, 1 †	11, 0, 2 †	11, 0, 3 †	11, 0, 4 †	11, 0, 5 †	11, 0, 6 †	11, 0, 7 †
B	12, 11, 0, 1, 8 †	12, 11, 0, 1 †	12, 11, 0, 2 †	12, 11, 0, 3 †	12, 11, 0, 4 †	12, 11, 0, 5 †	12, 11, 0, 6 †	12, 11, 0, 7 †
C	12, 0 12-0	12, 1 A	12, 2 B	12, 3 C	12, 4 D	12, 5 E	12, 6 F	12, 7 G
D	11, 0 11-0	11, 1 J	11, 2 K	11, 3 L	11, 4 M	11, 5 N	11, 6 O	11, 7 P
E	0, 2, 8 0-2-8	11, 0, 1, 9	0, 2 S	0, 3 T	0, 4 U	0, 5 V	0, 6 W	0, 7 X
F	0 0	1 1	2 2	3 3	4 4	5 5	6 6	7 7

Table A-2. Representative Character Set (Part 2 of 4)

DIGIT 2 \ DIGIT 1	8	9	A	B	C	D	E	F
0								
1								
2								
3								
4			¢	. (period)	<	(+	
5			!	\$	*)	:	⌋
6			\	, (comma)	%	_ (underline)	>	?
7			:	#	@	' (apostrophe)	=	" (quote mark)

Table A-2. Representative Character Set (Part 3 of 4)

DIGIT 2 \ DIGIT 1	0	1	2	3	4	5	6	7
8		a	b	c	d	e	f	g
9		j	k	l	m	n	o	p
A		~ (tilde)	s	t	u	v	w	x
B								
C	{	A	B	C	D	E	F	G
D	}	J	K	L	M	N	O	P
E			S	T	U	V	W	X
F	0	1	2	3	4	5	6	7

Table A-2. Representative Character Set (Part 4 of 4)

DIGT 2 DIGT 1	B	9	A	B	C	D	E	F
8	h	i						
9	q	r						
A	y	z						
B								
C	H	I						
D	Q	R						
E	Y	Z						
F	B	9						



Appendix B. Summary of UNIT Options

Summaries of UNIT arguments and the types of files they are used to define are presented in Tables B—1 through B—8.

Table B—1. Summary of UNIT Arguments for Printer File

Argument	Use
FDEVICE=PRINTER	Specifies the device to be used for the file.
FUNIT= $\left. \begin{array}{l} k \\ \text{PRINT} \\ \text{PUNCH} \end{array} \right\}$	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.
[FFILEID= $\left. \begin{array}{l} \text{filename} \\ \text{FORTK}; \text{ if FUNIT} = k \\ \text{PRNTR}; \text{ if FUNIT} = \text{PRINT} \\ \text{PUNCH}; \text{ if FUNIT} = \text{PUNCH} \end{array} \right\}$]	Specifies job control file reference name (LFD). Defaults PRNTR and PUNCH taken only for FORTRAN II FUNIT.
[FRECSIZE= $\left. \begin{array}{l} k \\ \underline{121} \end{array} \right\}$]	Specifies logical record size.
[FNUMBUF= $\left. \begin{array}{l} 1 \\ 2 \end{array} \right\}$]	Specifies number of input/output buffers.
[FDIAGNOS=YES]	Specifies the unit as the diagnostic device.
[FPRINTOV= $\left. \begin{array}{l} \text{SKIP} \\ \text{NOSKIP} \end{array} \right\}$]	Specifies printer action when bottom of page is encountered.
[FCHAR= $\left. \begin{array}{l} \text{OFF} \\ \text{ON} \end{array} \right\}$]	Specifies printer action for illegal characters. OFF causes blank substitution; ON causes program termination.
[FOPTION=YES]	Specifies a file not logically required. If the file is not allocated, output is ignored.

Table B-2. Summary of UNIT Arguments for Spooled Card Input File

Argument	Use
FDEVICE=SPOOLIN	Specifies the device to be used for the file.
FUNIT= $\left\{ \begin{array}{l} k \\ \text{READ} \end{array} \right\}$	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.
[FREREAD=YES]	Specifies that a copy of each formatted input record is to be transferred to the reread buffer.
[FBKSZ= $\left\{ \begin{array}{l} k \\ \underline{400} \end{array} \right\}$]	Specifies size of unit buffer.
[FBUFPOOL=YES]	Specifies that the buffer for the unit is to be pooled.
[FRECSIZE = $\left\{ \begin{array}{l} k \\ \underline{80} \end{array} \right\}$]	Specifies card size to be read.

Table B-3. Summary of UNIT Arguments for Card Input File

Argument	Use
FDEVICE=CARDIN	Specifies device to be used for the file.
FUNIT= $\left\{ \begin{array}{l} k \\ \text{READ} \end{array} \right\}$	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.
[FFILEID= $\left\{ \begin{array}{l} \text{filename} \\ \text{FORT}k; \text{ if FUNIT}=k \\ \text{READER}; \text{ if FUNIT}=\text{READ} \end{array} \right\}$]	Specifies job control file reference name (LFD). Default READER taken only for FORTRAN II FUNIT.
[FREREAD=YES]	Specifies that a copy of each formatted input record is to be transferred to the reread buffer.
[FBUFPOOL=YES]	Specifies that buffers for the unit are to be pooled, or shared, with all other units so specified.
[FNUMBUF= $\left\{ \begin{array}{l} 1 \\ 2 \end{array} \right\}$]	Specifies number of input buffers.
[FWORKA= $\left\{ \begin{array}{l} \text{YES}; \text{ if FNUMBUF}=1 \\ \text{NO}; \text{ if FNUMBUF}=2 \end{array} \right\}$]	Specifies that logical records are to be processed in a work area rather than in the buffer.
[FRECSIZE= $\left\{ \begin{array}{l} k \\ \underline{80} \end{array} \right\}$]	Specifies logical record size.
[FSTUB= $\left\{ \begin{array}{l} 51 \\ 66 \end{array} \right\}$]	Specifies that cards shorter than 80 columns are to be processed.
[FOPTION=YES]	Specifies a file not logically required. If the file is not allocated, and input causes an end return.
[FAUE=YES]	Specifies that misspunched cards are to be ignored.
[FBKSZ= $\left\{ \begin{array}{l} k \\ \text{FRECSIZE} \end{array} \right\}$]	Specifies the block size for the 8413 diskette.

Table B-4. Summary of UNIT Arguments for Card Output File

Argument	Use
FDEVICE=CARDOUT	Specifies device to be used for the file.
FUNIT= { ^k PUNCH }	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.
[FFILEID= { filename FORTk; if FUNIT=k PUNCH; if FUNIT=PUNCH }]	Specifies job control file reference name (LFD). Default PUNCH taken only for FORTRAN II FUNIT.
[FBUFPOOL=YES]	Specifies that buffers for the unit are to be pooled, or shared, with all other units so specified.
[FNUMBUF= { ¹ 2 }]	Specifies number of input buffers.
[FWORKA= { YES; if FNUMBUF=1 } NO; if FNUMBUF=2 }]	Specifies that logical records are to be processed in a work area rather than in the buffer.
[FRECSIZE= { ^k 80 }]	Specifies logical record size.
[FCRDERR=RETRY]	Specifies that automatic error recovery is to be attempted for mispunched cards.
[FOPTION=YES]	Specifies a file not logically required. If the file is not allocated, output is ignored.
[FBKSZ= { ^k FRECSIZE }]	Specifies the block size for the 8413 diskette.

Table B-5. Summary of UNIT Arguments for Tape File (Part 1 of 2)

Argument	Use
FDEVICE=TAPE	Specifies device to be used for the file.
FUNIT= { ^k READ PUNCH }	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.
[FFILEID= { filename FORTk; if FUNIT=k READER; if FUNIT=READ PUNCH; if FUNIT=PUNCH }]	Specifies job control file reference name (LFD). Defaults READER and PUNCH taken only for FORTRAN II FUNIT.
[FTYPEFLE= { INOUT WORK; if FUNIT=k INPUT; if FUNIT=READ OUTPUT; if FUNIT=PUNCH }]	Specifies level of data management support.
[FRECFORM= { VARUNB VARBLK FIXUNB FIXBLK }]	Specifies records as variable or fixed and blocked or unblocked.
[FNUMBUF= { ¹ 2 }]	Specifies number of input/output buffers.
[FWORKA= { YES; if FNUMBUF=1 } NO; if FNUMBUF= 2 }]	Specifies that logical records are to be processed in a work area rather than in the buffer.

Table B-5. Summary of UNIT Arguments for Tape File (Part 2 of 2)

Argument	Use
[FBUFPOOL=YES]	Specifies that buffers for the unit are to be pooled, or shared, with all other units so specified.
[FRECSIZE= { ^k 508 }]	Specifies logical record size. Taken as maximum for variable records.
[FBKSZ= { ^k FRECSIZE ; if FRECFORM=FIXUNB FRECSIZE+4 ; if FRECFORM=VARUNB FRECSIZE*4 ; otherwise }]	Specifies the size of the unit buffer.
[FREREAD=YES]	Specifies that a copy of each formatted input record is to be transferred to the reread buffer.
[FDIAGNOS=YES]	Specifies the unit as the diagnostic device
[FBKNO=YES]	Specifies that output tape blocks are to be sequentially numbered and input tape blocks are to be checked.
[FERROPT= { IGNORE } { SKIP }]	Specifies action for device errors. IGNORE and SKIP disable the ERR clause for parity/length.
[FRECERR=YES]	Specifies that records with bad parity or wrong length are to be moved to the reread buffer.
[FFILABL= { STD } { NO }]	Specifies standard or missing labels on magnetic tape.
[FCKPT=YES]	Specifies checkpoint dumps used to restart programs after a catastrophic failure are present on input tapes.
[FCLRW= { RWD } { NORWD } { UNLOAD }]	Specifies positioning at end of program execution for input and output tapes.
[FOPRW=NORWD]	Specifies that rewind is disabled at first reference to tape file.
[FOPTION=YES]	Specifies a file not logically required. If the file is not allocated, output is ignored, and input causes an end return.

Table B-6. Summary of UNIT Arguments for Sequential Disc Files

Argument	Use
FDEVICE=SDISC	Specifies device to be used for the file.
FUNIT= $\left. \begin{matrix} k \\ \text{READ} \\ \text{PUNCH} \end{matrix} \right\}$	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.
[FSECTOR= $\left. \begin{matrix} \text{NO} \\ \text{YES} \end{matrix} \right\}$]	Specifies processing on a sectorized disc expected.
[FFILEID= $\left. \begin{matrix} \text{filename} \\ \text{FORTk; if FUNIT=k} \\ \text{READER; if FUNIT=READ} \\ \text{PUNCH; if FUNIT=PUNCH} \end{matrix} \right\}$]	Specifies job control file reference name (LFD). Defaults READER and PUNCH taken only for FORTRAN II FUNIT.
[FTYPEFLE= $\left. \begin{matrix} \text{INOUT} \\ \text{WORK; if FUNIT=k} \\ \text{INPUT; if FUNIT=READ} \\ \text{OUTPUT; if FUNIT=PUNCH} \end{matrix} \right\}$]	Specifies level of data management support.
[FRECFORM= $\left. \begin{matrix} \text{VARUNB} \\ \text{VARBLK} \\ \text{FIXUNB} \\ \text{FIXBLK} \end{matrix} \right\}$]	Specifies records as variable or fixed and blocked or unblocked.
[FBUFPOOL=YES]	Specifies that buffers for the unit are to be pooled, or shared, with all other units so specified.
[FNUMBUF= $\left. \begin{matrix} 1 \\ 2 \end{matrix} \right\}$]	Specifies number of input/output buffers.
[FWORKA= $\left. \begin{matrix} \text{YES; if FNUMBUF=1} \\ \text{NO; if FNUMBUF=2} \end{matrix} \right\}$]	Specifies that logical records are to be processed in a work area rather than in the buffer.
[FRECSIZE= $\left. \begin{matrix} k \\ 508 \end{matrix} \right\}$]	Specifies logical record size. Taken as maximum for variable records.
[FBKSZ= $\left. \begin{matrix} k \\ \text{FRECSIZE}; \text{ if FRECFORM=FIXUNB} \\ \text{FRECSIZE} + 4; \text{ if FRECFORM=VARUNB} \\ \text{FRECSIZE} * 4; \text{ otherwise} \end{matrix} \right\}$]	Specifies the size of the unit buffer.
[FREREAD=YES]	Specifies that a copy of each formatted input record is to be transferred to the reread buffer.
[FDIAGNOS=YES]	Specifies the unit as the diagnostic device.
[FERROPT= $\left. \begin{matrix} \text{IGNORE} \\ \text{SKIP} \end{matrix} \right\}$]	Specifies action for device errors. IGNORE and SKIP disable the ERR clause for parity/length.
[FRECCERR=YES]	Specifies that records with bad parity or wrong length are to be moved to the reread buffer.
[FOPTION=YES]	Specifies a file not logically required. If the file is not allocated, output is ignored, and input causes an end return.
[FVERIFY=YES]	Specifies a reread of each written block to ensure proper parity.



Table B-7. Summary of UNIT Arguments for Direct Access Disc Files

Argument	Use
FDEVICE=DISC	Specifies device to be used for the file.
FUNIT=k	Specifies Extended FORTRAN unit reference number.
[FSECTOR= { NO } { YES }]	Specifies processing on a sectorized disc expected.
[FFILEID= { filename FORTk; where k=FUNIT }]	Specifies job control file reference name (LFD).
[FTYPEFLE= { INPUT OUTPUT }]	Specifies level of data management support. (11.3.3.6).
[FBUFPOOL=YES]	Specifies that buffers for the unit are to be pooled, or shared, with all other units so specified.
[FRECSIZE= { k 512 }]	Specifies logical record size.
[FRECERR=YES]	Specifies that records with bad parity or wrong length are to be moved to the reread buffer.
[FREREAD=YES]	Specifies that a copy of each formatted input record is to be transferred to the reread buffer.
[FVERIFY=YES]	Specifies a reread of each written block to ensure proper parity.

Table B-8. Summary of UNIT Arguments for Reread Unit

Argument	Use
FDEVICE=REREAD	Specifies device to be used for the file.
FUNIT= { k READ }	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.

Table B-9. Summary of UNIT Arguments for Equivalent Unit

Argument	Use
FDEVICE=EQUIV	Specifies device to be used for the file.
FUNIT= { k READ PRINT PUNCH }	Specifies Extended FORTRAN unit reference number or FORTRAN II statement reference.
FEQUIV= { k READ PRINT PUNCH }	Specifies the unit to be activated.

Appendix C. Additional UNIT Options

C.1. GENERAL

Additional options for execution environment configuration not presented in Section II are described in this appendix. Users familiar only with FORTRAN, however, should ignore this entire appendix, since it requires detailed knowledge of both assembly language and the data management system.

In the following descriptions of options for the various devices, a symbol required by an argument must be provided in assembler language following the FUNEND procedure call or be defined in another module. If it is defined in another module, the symbol must be named on an EXTRN statement in the UNIT module and on an ENTRY statement in the module defining the symbol. For further information, refer to the data management reference manual, UP-8068 (current version). For an explanation of the statement conventions applicable to this appendix, refer to 1.4.

C.2. PRINTER OPTIONS

Extended FORTRAN Printer Device Control Subroutines:

FORTRAN language rules require an advance and print sequence, but many printers accept only print and advance sequences. To prevent significant loss of performance, a print line is not delivered to the data management system until the spacing requirements of the next line are known. This requires special coding if the executable program contains assembly language procedures that deliver images to any printer defined by a UNIT procedure call.

Two subroutines are available for Extended FORTRAN printer device control.

- **FL\$PRNT**

CALL FL\$PRNT i,j,k

Used by the assembly language procedure to write a print line.

The address of the unit number, an INTEGER*4 variable, is specified by i. When i is negative, the Extended FORTRAN unit PRINT is assumed. The address of the line length, an INTEGER*4 variable, is specified by j. The address of the output line, the first character of which is a device independent control character, is specified by k.

A save area must previously have been specified in register 13.

■ **FL\$CLS**

CALL FL\$CLS i

Used to close the printer file if it is desirable for the assembly language procedure to perform standard PUT macro instructions.

The address of the unit number is specified by i.

The DTF may now be opened and standard data management processing continued, using register 3 as the IOREG. The automatic skip to home paper position may be lost if the file is closed immediately after forms overflow is detected.

C.3. CARD READER OPTIONS

Additional options for the card reader are described in the following paragraphs.

Binary Card Input Argument:

FMODE=STD

Specifies standard translation mode.

FMODE=BINARY

Specifies binary translation mode. Binary card input defines two bytes for each card column. Holes 12 through 3 are mapped onto bits 2⁵ through 2⁰ of byte 1; holes 4 through 9 are mapped onto bits 2⁵ through 2⁰ of byte 2, etc. Bits 2⁷ and 2⁶ of each byte are set to 0. When the BINARY option is specified, the default value for FRECSIZE is changed from 80 to 160.

→ Binary cards should be read by using an unformatted statement or an A FORMAT code; 96-column cards can not be read with BINARY mode.

ASCII Character Set Argument:

FASCII=YES

Specifies the ASCII character set.

If this argument is not specified, the EBCDIC character set is used. If FMODE=BINARY is specified, FASCII cannot be specified. ASCII does not imply a larger character set than EBCDIC, but is the accepted standard for information interchange.

C.4. CARD PUNCH OPTIONS

Additional options for the card punch are specified in the following paragraphs.

Binary Card Output Argument:

FMODE=STD

Specifies standard translation mode.

FMODE=BINARY

Specifies binary translation mode. Binary card output defines two bytes for each card column. Holes 12 through 3 are mapped onto bits 2⁵ through 2⁰ of byte 1; holes 4 through 9 are mapped onto bits 2⁵ through 2⁰ of byte 2, etc. Bytes 2⁷ and 2⁶ of each byte are not transmitted to the unit. When the BINARY option is specified, the default value for FRECSIZE is changed from 80 to 160.

ASCII Character Set Argument:

FASCII=YES

Specifies the ASCII character set.

If this argument is not specified, the EBCDIC character set is used. If FMODE=BINARY is specified, FASCII cannot be specified. ASCII does not imply a larger character set than EBCDIC, but is merely the accepted standard for information interchange.

C.5. TAPE FILE OPTIONS

Additional options for magnetic tape are specified in the following paragraphs.

User Header and Trailer Labels Arguments:

The FFILABL and FLABADDR arguments are used to specify user header and trailer labels.

- **FFILABL**

This argument specifies the type of labels to be used. In addition to the STD and NO options presented in Section 11, a third option is available.

FFILABL=NSTD

Specifies nonstandard labels.

A nonstandard labeled tape with user trailer labels cannot be extended or backspaced after ENDFILE has been encountered.

- **FLABADDR**

FLABADDR=symbol

Specifies that user header and trailer labels are to be processed.

The address of the user label routine is specified by symbol. This argument should be specified if FFILABL=NSTD is specified.

ASCII Tape Files Arguments:

- FASCII

FASCII=YES

Specifies ASCII files.

- FBUFFOFF

FBUFFOFF=k

Specifies that a block length field of 0 to 99 bytes is to be prefixed on each block. FBUFFOFF may be specified only if FASCII=YES has been specified. A value of 0-99 is specified by k.

If a value other than 4 is specified for k, the block length field is assumed to be destined for, or received from, an alien operating system and is ignored. If the block size is determined by default, FBUFFOFF is added afterward.

- FLENCHK

FLENCHK=YES

Specifies that, for variable length records, the block length field is automatically set on output and checked on input. FLENCHK may be specified only if FASCII=YES and FBUFFOFF=4 have been specified.

C.6. SEQUENTIAL DISC FILE OPTION

An additional option for sequential disc follows.

User Header and Trailer Argument:

FLABADDR=symbol

Specifies that the user header and trailer labels are to be processed. The address of the user label routine is specified by symbol.

C.7. DIRECT ACCESS DISC FILE OPTIONS

Additional options for direct access disc are specified in the following paragraphs.

- FLABADDR

FLABADDR=symbol

Specifies that user header labels are to be processed. The address of the user label routine is specified by symbol.

- FTRLBL

FTRLBL=YES

Specifies that user trailer labels are to be processed. This argument may be specified only if the FLABADDR argument has been specified.

C.8. ADDITIONAL DATA MANAGEMENT DEVICES

Files for sequential devices supported by the SPERRY UNIVAC Operating System/3 (OS/3) Data Management System and not presented in Section 11, such as optical document readers, paper tape, etc., are defined by using the following UNIT procedure call. A listing, in the order of their relative importance and utility, of the arguments that may appear on this UNIT procedure call is followed by descriptions of the arguments.

Format:

1	10	16
	UNIT	FDEVICE=DMS FUNIT=k FWORKA=YES [FFILEID= { filename } { <u>FORT</u> k }] [FRECFORM= { <u>VARUNB</u> VARBLK } FIXUNB } FIXBLK }] [FRECSIZE= { k <u>508</u> }] [FREREAD=YES]

Device Identification Argument:

Specifies that this file is for a sequential device supported by the OS/3 data management system.

Unit Identifier Argument:

FUNIT=k

Specifies a unique integer constant in the range $1 \leq k \leq 99$.

A maximum of 102 unique unit identifiers (values 1—99 and READ, PRINT, and PUNCH) may be specified by a control module.

Work Area Allocation Argument:

FWORKA=YES

Specifies that records are to be moved to and from a work area for processing. Space for a work area is to be allocated.

File Name Argument:

FFILEID=filename

Specifies a 1- to 7-character FORTRAN style symbolic name (filename).

FFILEID=FORTk

Specifies the file name as FORTk, where $1 \leq k \leq 99$. If the FFILEID argument is not specified, FORTk is the default file name.

The UNIT procedure call generates an address constant that references the specification for FFILEID. A define the file (DTF) macro instruction labeled with the file name must be provided. An EXTRN statement is automatically generated for the label specified in FFILEID.

Record Formats Argument:

FRECFORM=VARUNB

Specifies variable-length unblocked records.

FRECFORM=VARBLK

Specifies variable-length blocked records.

FRECFORM=FIXUNB

Specifies fixed-length unblocked records.

FRECFORM=FIXBLK

Specifies fixed-length blocked records.

Record Size Argument:

This argument specifies the record size and is used only to ensure that the common work area is large enough for all units using it. No I/O areas are allocated; these must be defined by the user.

FRECSIZE= $\left\{ \begin{array}{c} k \\ \underline{508} \end{array} \right\}$

Specifies a positive integer constant.

If this argument is omitted, 508 is the default record size.

Reread Argument:

FREREAD=YES

Specifies that a unit is to participate in the reread feature (7.3.4).

The reread unit consists of a single buffer to which each formatted input record is transferred. To conserve central processor time, this data movement is inhibited unless specifically requested.

Appendix D. FORTRAN Sample Job Streams

D.1. JOB CONTROL PROCEDURE

The FOR procedure call statement generates the necessary job control statements to compile an Extended FORTRAN program. Optionally, it can generate job control statements that specify the following:

- input – source library;
- output – object library;
- PARAM control statements defining the compiler processing logic; and
- automatically link and/or execute the program.

The input may be embedded data cards, (/ \$, source deck, /*) immediately after the FOR procedure call, or a module in any library as defined by the IN parameter. This results in the appropriate DVC—LFD control statement sequence with an LFD name, LIB1, and the PARAM control statement, PARAM IN=module-name/LIB1, unless the PARAM LIN statement is specified.

The object code is written in \$Y\$RUN by default, but a specific output library can be specified by the OUT parameter. This results in the appropriate DVC—LFD control statement sequence with an LFD name, OUTFPUT, and the PARAM control statement, PARAM OUT=OUTFPUT.

The ALTLOD parameter generates the necessary DVC—LFD control statement with an LFD name, ALTLOD, and the appropriate EXEC control statement to load and execute the FORTRAN compiler from a private library other than \$Y\$LOD.

Format:

$$\begin{array}{l}
 //[\text{symbol}] \left\{ \begin{array}{l} \text{FOR} \\ \text{FORL} \\ \text{FORLG} \end{array} \right\} \left[\text{PRNTR} = \left\{ \begin{array}{l} N \\ \left(\begin{array}{l} \text{lun} \\ N \end{array} \right) \\ 20 \end{array} \right\} [\text{vol-ser-no}] \right] \left[\text{IN} = \left\{ \begin{array}{l} (\text{vol-ser-no}, \text{label}) \\ (\text{RES}) \\ (\text{RES}, \text{label}) \\ (\text{RUN}, \text{label}) \\ (*, \text{label}) \end{array} \right\} \right] \\
 \left[\text{OUT} = \left\{ \begin{array}{l} (\text{vol-ser-no}, \text{label}) \\ (\text{RES}, \text{label}) \\ (\text{RUN}, \text{label}) \\ (*, \text{label}) \\ (\text{RUN}, \$Y\$RUN) \end{array} \right\} \right] \left[\text{SCR1} = \left\{ \begin{array}{l} \text{vol-ser-no} \\ \text{RES} \end{array} \right\} \right] \\
 \left[\text{ALTLOD} = \left\{ \begin{array}{l} (\text{vol-ser-no}, \text{label}) \\ (\text{RES}, \text{label}) \\ (\text{RUN}, \text{label}) \\ (*, \text{label}) \\ (\text{RES}, \$Y\$LOD) \end{array} \right\} \right] [\text{OPT} = (\text{D}, \text{N}, \text{X})] \\
 [\text{MDE} = 1] [\text{STX} = \text{option}] [\text{CNL} = \text{option}] \\
 \left[\text{LIN} = \left\{ \begin{array}{l} \text{filename} \\ \text{LIB1} \end{array} \right\} \right] [\text{LST} = \text{option}]
 \end{array}$$

Label:

symbol

Specifies the 1- to 6-character source module name; only needed when the IN parameter is used.

Operation:

FOR

This form of the procedure call statement is used to compile an Extended FORTRAN source program.

FORL

This form of the procedure call statement is used to compile an Extended FORTRAN source program and link-edit the object modules.

FORLG

This form of the procedure call statement is used to compile an Extended FORTRAN source program, link-edit the object modules, and execute the load module.

Keyword Parameter PRNTR:

$$\text{PRNTR} = \left\{ \left(\left(\begin{array}{c} N \\ \text{lun} \\ N \\ \underline{20} \end{array} \right) \left[\text{vol-ser-no} \right] \right) \right\}$$

Specifies the logical unit number of the printer, and, optionally, the destination-id (vol-ser-no). If a printer device assignment set is not to be generated, the value N is coded, and the printer device assignment set must be manually inserted in the control stream.

PRNTR=(lun[,vol-ser-no])

Specifies the logical unit number (20—29) of the printer device. Optionally, the destination-id (vol-ser-no) can be specified.

PRNTR=(N[,vol-ser-no])

Indicates that a device assignment set for the printer must be manually inserted in the control stream. This permits LCB and VFB job control statements to be used in the control stream. The volume serial number can also be specified.

Keyword Parameter IN:

This parameter specifies the input file referenced by the PARAM IN control statement. If omitted, the source input is assumed to be embedded data cards (/ \$, source deck, /*).

IN=(vol-ser-no,label)

Specifies the volume serial number (vol-ser-no) and the file identifier (label) where the source input is located.

IN=(RES)

Specifies that the source input is located on the SYSRES device in \$Y\$SRC.

IN=(RES,label)

Specifies that the source input is located on the SYSRES device, in the file identified by the file identifier (label).

IN=(RUN,label)

Specifies that the source input is located on the job's \$Y\$RUN file with the file identifier (label) specified by the user.

IN>(* ,label)

Specifies that the source input is located on a catalog file identified by the file identifier (label). ←

Keyword Parameter OUT:

This parameter specifies the output file definition. If omitted, the object code is located on the job's \$Y\$RUN file.

OUT=(vol-ser-no,label)

Specifies the volume serial number (vol-ser-no) and the file identifier (label) of the file where the object code is to be located.

OUT=(RES,label)

Specifies that the object code is to be located on the SYSRES device, within the file specified by the label parameter.

OUT=(RUN,label)

Specifies that the object code is to be located on the job's \$Y\$RUN file identified by a user specified file identifier (label).

OUT>(* ,label)

Specifies that the object code is to be located on a catalog file identified by the file identifier (label). ←

Keyword Parameter SCR1:

SCR1= { **vol-ser-no** }
 { **RES** }

Specifies the volume serial number of the work file labeled \$SCR1. If omitted, the work file is assumed to be on the SYSRES device.

Keyword Parameter ALTLOD:

This parameter specifies the location of the alternate load library. If omitted, the compiler is loaded from \$Y\$RUN. ↓

ALTLOD=(vol-ser-no,label)

Specifies the volume serial number (vol-ser-no) and file identifier (label) of an alternate load library that contains the Extended FORTRN compiler.

ALTLOD=(RES,label)

Specifies that the alternate load library is located on the job's SYSRES device, in the file identified by the file identifier (label).

ALTLOD=(RUN,label)

Specifies that the alternate load library is located on the job's \$Y\$RUN file with the file identifier (label) specified by the user.

ALTLOD>(* ,label)

Specifies that the alternate load library is located on a catalog file identified by the file identifier (label). ↑



Keyword Parameter OPT:

OPT=(D, N, X)

Specifies one or all of the following compilation options.

- D Specifies double spacing of the compiler listing.
- N Specifies that no object program is to be generated. The program units are merely compiled and cannot be executed.
- X Specifies compilation of all cards with the character X in column 1. If this option is not specified, these cards will be treated as comments.

The default for the OPT argument is single spacing with the absence of the N and X specifications. All OPT options remain in effect until another OPT specification is encountered. If only one OPT argument is specified, the parentheses are optional.

Keyword Parameter MDE:

MDE=I

Specifies that the compiler is to evaluate expressions in a strict left-to-right order when there is a choice, and that storage is to be allocated for variables and arrays in the sequence in which they were encountered.

This parameter is recommended for use when compiling programs originally developed under the IBM System/360 Disc Operating System. When specified, the MDE parameter remains in force for all remaining compilations.

Keyword Parameter STX:

When the Extended FORTRAN compiler generates code for a main program, a call to a FORTRAN IV library subprogram is produced. This causes the execution of two STXIT macro instructions, locates the diagnostic device, and sets up the program mask in the program status word (PSW).

The two STXIT macro instructions, for program checks and abnormal termination enable the library to:

- maintain switches for the OVERFL and DVCHK subroutines;
- recover boundary alignment errors caused by COMMON and EQUIVALENCE statements and argument substitution; and
- provide for orderly shutdown of the program when fatal errors occur.

The STX parameter provides user control of the execution of STXIT macro instructions.

STX=Y

Causes the execution of two STXIT macro instructions at the beginning of a SUBROUTINE or a FUNCTION subprogram.

STX=N

Suppresses the execution of two STXIT macro instructions at program initiation. STX=N is used only for main programs.

If the STX parameter is not specified, a specification of Y is assumed for main programs, and a specification of N is assumed for SUBROUTINE or FUNCTION subprograms.

The STX parameter is operative only for the current subprogram to be compiled. This argument is useful when integrating COBOL and assembler object modules with FORTRAN object modules.

Keyword Parameter CNL:

→ **CNL=option**

Specifies compiler termination if a diagnostic with a severity level is generated.

The values are:

- 2, indicates academic messages, e.g., a truncated constant;
- 4, indicates warning diagnostics, e.g., an extraneous comma in a list;
- 6, indicates serious diagnostics, e.g., an array reference without a preceding array declarator; or
- 8, indicates fatal errors, e.g., insufficient storage to complete the compilation.

If the CNL parameter is not specified, the compiler processes all program units in the control stream, regardless of errors encountered. When specified, the CNL parameter remains in force until redefined.

Keyword Parameter LIN:

LIN=filename

Specifies the name of the default filename in which the source modules reside.

A 1- to 8-alphanumeric-character identifier is specified by filename. If the LIN parameter is not specified, the compiler assumes the default filename of LIB1. This parameter is used in conjunction with the IN parameter.

Keyword Parameter LST:

LST=option

Specifies the quantity of listings produced by the compiler. One of the following options may be chosen.

- N Specifies an abbreviated listing consisting of only the compiler identification, parameters, error counts, and termination conditions.

- S Specifies, in addition to the N listing, the source code listing with any serious diagnostics.
- M Specifies, in addition to the S listing, a storage map showing the addresses assigned to variables and arrays.
- W Specifies, in addition to the M listing, academic and warning diagnostics.
- O Specifies, in addition to the W listing, an object code listing showing the SPERRY UNIVAC 90/30 instructions generated for the executable statements.

The LST parameter remains in effect for succeeding compilations until another LST parameter is encountered. If no LST PARAM is specified, the M option is assumed.

Example 1a:

The following example illustrates the use of the FOR procedure call statement in its basic form:

1	LABEL	ΔOPERATIONΔ	OPERAND	Δ
		10	16	
1.	// JOB	FRTRN1A		
2.	// FOR			
3.	/\$			
4.		.		
5.	Source	deck		
6.		.		
7.	/*			

Line	Explanation
1	Indicates that the name of the job is FRTRN1A.
2	Indicates the name of the procedure being called (FOR). No keyword parameters specifying special options for this compile are used.
3	Indicates start of data.
4—6	Represents the source deck to be compiled.
7	Indicates end of data.



Example 1b:

The basic form generates the following control stream:

1	LABEL	Δ OPERATION Δ	10	16	OPERAND	Δ	72
1	///	JOB	FR	TRN	LIB		
2	///	DVC	20	///	LFD	PRNTR	
3	///	DVC	RES				
4	///	EXT	ST	3	CYL	1	
5	///	LBL	\$SCR1	///	LFD	\$SCR1	
6	///	EXEC	FOR				
7	/	\$					
8							
9		source	deck				
10							
11	/	*					

Line Explanation

- 1 Indicates that the name of the job is FRTRN1B.
- 2 Indicates the default logical unit number and LFD name of the printer.
- 3—5 Indicates that the work file needed for compiling is, by default, on the SYSRES device, has both a file-label and LFD name of \$SCR1, and uses the sequential access technique; that allocation is contiguous; that three cylinders are allocated for the secondary increment; and that one cylinder is allocated for the first extent.
- 6 Loads the Extended FORTRAN compiler from \$Y\$LOD.
- 7 Indicates start of data.
- 8—10 Represents the source deck to be compiled.
- 11 Indicates end of data.

Example 2a:

The following example illustrates the use of a FOR procedure call statement that defines all the keyword parameters:

1	///	JOB	FR	TRN	2A		
2	///	PROG	NM	FOR	PRNTR=Z1, IN=(DSC1, U\$SCR1),		X
3	///				OUT=(DSC2, U\$OBJ), LST=S,		X
4	///	2			SCR1=DISC2, ALTLOD=(DSC3, ALTLODLIB)		
5	/	\$					



<u>Line</u>	<u>Explanation</u>
1	Indicates that the name of the job is FRTRN2A.
2	Indicates the name of the procedure being called (FOR). The source module name is PROGNM. The logical unit number of the printer is 21, and the input file has a volume serial number of DSC1, with a file-label of U\$SRC.
3	Indicates that the output file volume serial number is DSC2, with a file-label of U\$OBJ. The format of the compiler listing is supplied by the LST parameter.
4	Indicates that the work file needed for compiling has a volume serial number of DSC2. The Extended FORTRAN compiler is located on the device with a volume serial number of DSC3 in the file labeled ALTLODLIB.
5	End of job.

Example 2b:

By using the keyword parameters in example 2a, the following control stream is generated.

1	LABEL	△OPERATION△	OPERAND	△
		10	16	
1	//	JOB	FRTRN2B	
2	//	DVC 21	// LFD PRNTR	
3	//	DVC 50	// VOL DSC1	
4	//	LBL U\$SRC	// LFD LIB1	
5	//	DVC 51	// VOL DSC2	
6	//	LBL U\$OBJ	// LFD OUTPUT	
7	//	DVC 51	// VOL DSC2	
8	//	EXT ST,C,3	CYL,1	
9	//	LBL \$SCRI	// LFD \$SCRI	
10	//	DVC 52	// VOL DSC3	
11	//	LBL ALTLODLIB	// LFD ALTLOD	
12	//	EXEC FOR	ALTLOD	
13	//	PARAM	OUT=OUTPUT	
14	//	PARAM	LST=S	
15	//	PARAM	IN=PROGNM/LIB1	
16	//	&		

<u>Line</u>	<u>Explanation</u>
1	Indicates that the name of the job is FRTRN2B.
2	Indicates that the printer is to be assigned to the logical unit number 21, with an LFD name of PRNTR. This was obtained from line 2 in example 2a.

↓

<u>Line</u>	<u>Explanation</u>
3	Indicates that the input file volume serial number is DSC1. This was obtained from the IN parameter of line 2 in example 2a. It is assigned to the device with a logical unit number of 50, which was the first available number in the range of 50—54.
4	Indicates that the input file is labeled U\$SRC with an LFD name of LIB1. This was obtained from the IN parameter of line 2 in example 2a.
5	Indicates that the output file volume serial number is DSC2. This was obtained from the OUT parameter of line 3 in example 2a. It is assigned to the device with a logical unit number of 51, which was the next available number in the range of 50—54. Logical unit number 50 was already assigned to the device with a volume serial number of DSC1 (line 3).
6	Indicates that the output file is labeled U\$OBJ with an LFD name of OUTFPUT. This was obtained from the OUT parameter of line 3 in example 2a.
7—9	Indicates the work file for the compiler has a volume serial number of DSC2. Because this volume serial number was already used, this work file uses the same device logical unit number of 51. This work file has both a file-label and LFD name of \$SCR1 and uses the sequential access technique; allocation is contiguous; three cylinders are allocated for the secondary increment; and one cylinder is allocated for the first extent. This was obtained from line 4 in example 2a.
10	Indicates that the alternate load library for the compiler has a volume serial number of DSC3. It is assigned to the device with a logical unit number of 52, which was the next available number in the range of 50—54. This was obtained from the ALTLOD parameter of line 4 in example 2a.
11	Indicates that the alternate load library has a label of ALTLODLIB with an LFD name of ALTLOD. This was obtained from the ALTLOD parameter of line 4 in example 2a.
12	Loads the Extended FORTRAN compiler from the file labeled ALTLOD.
13—15	PARAM control statements, which identify the processing options for the FORTRAN compiler. These are generated in the following manner: Line 13 — The filename OUTFPUT is generated automatically when the OUT parameter is used. Line 14 — Indicates that compiler identification, parameters, error counts, termination conditions, source code, and diagnostics will be listed. This was obtained from the LST parameter in line 3 of example 2a. Line 15 — The filename LIB1 is generated automatically when the IN parameter is specified. The module name PROGNM is generated from the label field in line 2 of example 2a.
16	End of job.

↑

D.2. SAMPLE COMPILE-LINK-EXECUTE

The following job control stream example illustrates a simple compilation from cards, linking the program EX1, and execution of the bound program TEST1 from \$Y\$RUN.

Example:

1	LABEL	Δ OPERATION Δ	OPERAND	Δ	COMMENTS
		10	16		
1.	// JOB	EXAMPLE1			
2.	// DVC	20 // LFD	PRNTR		PRINTER FOR ALL PROCESSORS
3.	// WORK				ONE WORK FILE
4.	// EXEC	FOR			BEGIN COMPILATION
5.	// PARAM				PARAMETERS (AS NEEDED)
6.	/\$				
		PROGRAM	EX1		
		{	program body	}	
		END			
7.					
8.	/*				END COMPILATION
9.	// WORK				ONE WORK FILE
10.	// EXEC	LNKEDT			BEGIN LINK EDIT
11.	/\$				START OF INPUT TO LINKAGE EDITOR
12.		LOADM	TEST1		
13.		INCLUDE	EX1		
14.	/*				END LINK EDIT
15.	// EXEC	TEST1, \$Y\$RUN			BEGIN EXECUTION
16.	/&				END OF JOB

Line	Explanation
1	Indicates the job name, EXAMPLE1
2	Indicates the printer device number for all processors
3	Specifies one work file for Extended FORTRAN compiler execution
4	Begins compilation
5	Adds parameters here as per job requirements
6	Start of data to compiler (source program)
7	End of source data
8	End of compilation
9	Specifies one work file for the linkage editor

<u>Line</u>	<u>Explanation</u>
10	Begins the link edit
11	Start of data to linkage editor
12	Names new load module TEST1
13	Links source module named EX1
14	Ends link edit
15	Begins program execution
16	End of job

D.3. SOURCE FROM DISC LIBRARY — STACKED COMPILATION

This job control stream represents the source module from the disc library for a stacked compilation. Source programs on disc files are identified using a librarian module name. Each source module consists of one or more FORTRAN program units.

Example:

	LABEL	△OPERATION△	OPERAND	△	COMMENTS
	1	10	16		
1.	// JOB	EXAMPLE2			
2.	// DVC	20	// LFD PRNTR		
3.	// WORK	1			
4.	// DVC	50	// VOL DISCOO // LBL FORSOURCE, // LFD INPUT		
5.	// EXEC	FOR			
6.	// PARAM	IN=MODULE 1/	INPUT		
7.	// PARAM	IN=MODULE 2/	INPUT		
8.	:				
9.	:				
10.	// PARAM	IN=MODULE n/	INPUT		
11.	//&				

<u>Line</u>	<u>Explanation</u>
1	Indicates job name, EXAMPLE2
2	Indicates the printer device number
3	Specifies one work file for Extended FORTRAN compiler
4	Specifies that the file, FORSOURCE, on disc DISCOO, device 50, is the INPUT file.
5	Begins compilation

Line	Explanation
6-7	Identifies the first and second source programs' module names/filenames to the compiler
8-10	Identifies all succeeding and last source program module names/filenames to the compiler
11	End of job

D.4. COMPILE-ASSEMBLE-LINK-EXECUTE

This example shows the user-specified execution environment, spoolin input, and print and tape output.

Example:

1	LABEL	△OPERATION△ 10	16	OPERAND	△	72
1.	//	JOB	EXAMPLE	3		
2.	//	DVC	20	//	LFD	PRNTR
3.	//	WORK1				
4.	//	EXEC	FOR			
5.	/\$					
		{program body}				
6.	/*					
7.	//	WORK1				
8.	//	WORK2				
9.	//	EXEC	ASM			
10.	//	PARAM	LST=NC			
11.	/\$					
12.	MYIO	START				
13.		FUNTAB	SYS=FOR			
14.		UNIT	FDEVICE=PRINTER,			X
15.			FUNIT=1			
16.		UNIT	FDEVICE=SPOOLIN,			X
17.			FUNIT=READ,			X
18.			FBKSZ=80			
19.		UNIT	FDEVICE=TAPE,			X
20.			FUNIT=10,			X
21.			FFILEID=XYZ,			X
22.			FRECFORM=FIXBLK,			X
23.			FRECSIZE=256			

Example: (cont)

1	LABEL	ΔOPERATIONΔ	16	OPERAND	Δ	72
24		FUNEND				
25		EJECT				
26		ERRDEF				
27		END				
28	/*					
29	// WORK1					
30	// EXEC	LNKEDT				
31	/\$					
32		LOADM	TEST3			
33		INCLUDE	\$MAIN			
34		INCLUDE	MYIO			
35	/*					
36	// DVC 21	// LFD	FORT1			
37	// DVC 90	// VOL	TAPE00 // LFD	XYZ		
38	// EXEC	TEST3,	\$Y\$RUN			
39	/\$					
		{SPOOLIN	data}			
40	/*					
41	/&					



- | Line | Explanation |
|------|---|
| 1 | Indicates job name, EXAMPLE3 |
| 2 | Indicates the printer device number |
| 3 | Specifies one work file for Extended FORTRAN compiler |
| 4 | Begins FORTRAN compilation |
| 5 | Start of data to compiler (source program) |
| 6 | End of data |
| 7-8 | Specifies two work files for the assembler |
| 9 | Begins Assembler execution |
| 10 | Specifies no cross-reference listing |

<u>Line</u>	<u>Explanation</u>
11	Start of data to the Assembler
12	Start of execution environment module (MYIO)
13	Initializes file for Extended FORTRAN
14—15	Defines first file (UNIT definition procedure) specifying a printer file
15	Specifies printer unit number 1
16—18	Defines second file (UNIT definition procedure) specifying a spooled input file
17	Identifies the reader as input device
18	Indicates input blocksize
19—23	Defines third file (UNIT definition procedure) specifying a tape file
20	Specifies tape unit number 10
21	Indicates the tape filename, XYZ
22	Specifies fixed-length blocked records
23	Specifies a tape record length of 256 bytes
24	Terminates UNIT procedure calls
25	File termination
26	Includes library table of error information in executable program
27	Terminates source program
28	End of data to Assembler
29	Specifies one work file for the linkage editor
30	Begins link edit
31	Start of data to linkage editor
32	Names the new load modules TEST3
33—34	Links the modules \$MAIN and MYIO to TEST3
35	End of data to linkage editor
36—37	Connects devices assigned before execution of TEST3 to the FORTRAN unit table via their LFD names
38	Executes load module TEST3 from \$Y\$RUN

Line	Explanation
39	Start of spoolin data
40	End of spoolin data
41	End of job

NOTE:

The default FFILEID for the printer is FORT1.

D.5. COMPILATIONS WITH PARAM OPTIONS

The following example illustrates the use of special options specified via the // PARAM statement:

Example:

1	LABEL	Δ OPERATION Δ	OPERAND	Δ	COMMENTS
		10	16		
1.	// JOB	EXAMPLE4			
2.	// DVC	20 // LFD	PRNTR		
3.	// WORK				
4.	// EXEC	FOR			
5.	// PARAM	LST=D,OPT=(X,D),NXT=LNK			
6.	/\$				
		{program data}			
7.	/*				
8.	/\$				
9.		LOADM	TEST4		} Params for Link Editor
10.		INCLUDE	\$MAIN		
11.	/*				
12.	// EXEC	TEST4, \$Y\$RUN			
13.	//&				

Line	Explanation
1	Names job EXAMPLE4
2	Assigns device 20 to printer
3	Specifies one work file for the Extended FORTRAN compiler and the linkage editor
4	Begins compilation
5	Specifies a double-spaced object code listing. Also indicates that cards with X in column 1 will be accepted for compilation as FORTRAN statements.
6	Start of program data

<u>Line</u>	<u>Explanation</u>
7	End of program data
8	Linkage editor is called directly from the compiler and this is start of linkage editor data
9	Names new load module TEST4
10	Links \$MAIN module to TEST4
11	End of data to linkage editor
12	Executes load module TEST4 from \$Y\$RUN
13	End of job



Appendix E. Compile-Time Diagnostic Messages

The compile-time diagnostics are listed and described in Table E—1. All messages are prefixed with *FC* to identify them as being generated by the FORTRAN compiler. The 3-digit number following *FC* in the *Message Number* column explicitly identifies the diagnostic. The degree of severity is shown in the *Severity Code* column, where:

- 2 = academic
- 4 = warning
- 6 = serious
- 8 = fatal

The *Diagnostic Message* column shows the message as it appears when printed. The cause of the message and the action to be taken are described in the remaining columns.

For each diagnostic message issued, the source statement is marked by a dollar sign (\$) below the column where the error was first recognized. When a long statement consisting of multiple continuation cards is diagnosed, the dollar sign occasionally appears under the wrong card, but flags the proper column. To locate the error, check the flagged column on each card of the statement.



Table E-1. Compile-Time Diagnostic Messages (Part 1 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC000	6	ERROR	Error in source code. Also the compiler and message file are incompatible. Submit a Software User Report (SUR).	Attempt to locate error by using diagnostic marker \$ which points to the error.
FC001	6	- ERROR	An obsolete diagnostic that is incompatible with the message file has been issued by the compiler. Submit a SUR.	Attempt to locate error by using the diagnostic marker \$.
FC002	6	- DIMENSIONED VARIABLE HAS INVALID SUBSCRIPT.	A valid subscript is an integer or real arithmetic expression. Complex and logical expressions are invalid.	Correct source code.
FC003	2	variable NEVER REFERENCED	Cited variable defined but not referenced in the program.	Check for misspelling.
FC004	2	variable NEVER DEFINED	Cited variable is not defined by an assignment statement READ. It does not appear in an argument list to a subroutine or abnormal function, or is not in a COMMON statement.	Check for misspelling.
FC005	4	variable WILL CAUSE BOUNDARY ERROR	Variable specified is on improper main storage boundary. Execution can proceed but efficiency is compromised.	If the program is to be executed often, COMMON and EQUIVALENCE statements should be reorganized.
FC006	4	EQUIVALENCE STATEMENT HAS INVALID VARIABLE NAME	EQUIVALENCE statement contains dummy argument name or procedure name.	Correct source program.
FC007	6	VARIABLE USED AS ARRAY BUT NEVER DIMENSIONED	Program context indicates that the array name indicated by the diagnostic marker \$ should have been preceded by a dimension declarator in a DIMENSION, COMMON or TYPE statement.	Provide a dimension declarator or correct misspelling of the symbolic name.
FC008	4	ARRAY WITH INCORRECT NUMBER OF SUBSCRIPTS	Any array declared with K dimension declarator subscripts may be referenced in an EQUIVALENCE statement with either one or K reference subscripts.	Correct source program.

Table E-1. Compile-Time Diagnostic Messages (Part 2 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC009	6	EQUIVALENCED VARIABLE HAS INVALID SUBSCRIPT	In an EQUIVALENCE statement, subscripts must be constants and within the boundaries of the array.	Correct source program.
FC010	4	SAME VARIABLE REPEATED IN GROUP	Same entity need not be mentioned more than once in an equivalence group or an equivalence set (one or more interacting groups).	Check for possible misspelling and correct source code.
FC011	4	GROUP HAS ONLY ONE VARIABLE	Equivalence group is not meaningful since a given entity is to share storage with an unknown number of associates.	Correct source code.
FC012	6	DIFFERENT COMMON BLOCK EQUIVALENCED TOGETHER	Common blocks must be unique entities which cannot be associated via EQUIVALENCE statements.	Correct source program, correcting possible misspelling in either COMMON or EQUIVALENCE statements.
FC013	6	VARIABLES WITHIN SAME COMMON BLOCK EQUIVALENCED	Common block entities must be allocated in ascending address sequence. EQUIVALENCE statements are not permitted to violate this basic rule.	Delete common entities from the EQUIVALENCE statement and correct possible misspelling in the COMMON statements.
FC014	4	COMMON BLOCK EXTENDED BACKWARD	Violation of a basic ANS/ECMA rule. Unless this diagnostic appears in every program unit referencing the block program execution, results will be incorrect in the source code.	Change EQUIVALENCE statements in the source code.
FC015	6	INCONSISTENT EQUIVALENCE STATEMENT	Equivalence set is attempting to distort the structure of an array which demands the contiguity of successive array elements.	Correct EQUIVALENCE statement.
FC016	6	EQUIVALENCE CLASS TOO LARGE	Excessive number of equivalence sets in the program unit. The compiler can address only 65K in the equivalence table.	Reduce number of equivalence groups.
FC018	2	STATEMENT LABEL NEVER REFERENCED	Label on this statement is meaningless; control can never be transferred directly to this label.	Check program logic to determine if a GO TO reference is needed.

Table E-1. Compile-Time Diagnostic Messages (Part 3 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC019	6	LABEL NOT DEFINED ON EXECUTABLE STATEMENT	Transfer of control to nonexecutable statement not permitted.	Change label.
FC020	4	CONTROL CANNOT REACH THIS STATEMENT	This statement is preceded by an unconditional transfer of control and should be labeled.	Check other diagnostics produced by this compilation and correct the program; the block of code is either unnecessary or missing a reference label.
FC021	6	END OF DO BLOCK CANNOT BE REACHED	An unconditional transfer of control has been used as the terminal statement of a DO loop. The DO terminal block is inaccessible and only one iteration of the DO is possible.	Correct source program.
FC022	4	MULTIPLY DEFINED LABEL	Statement labels must be unique within a program unit.	Change label and all references to it.
FC023	4	ASSIGNED GO TO LABEL NEVER ASSIGNED	Possible destination label did not appear in an ASSIGN statement.	Check for keypunch or logic errors.
FC024	4	RECURSIVE BRANCH	A block jumps only to itself. No other exit exists so the loop never terminates.	Correct source program.
FC025	2	ILLEGAL BRANCH INTO DO LOOP	A DO loop has been entered without executing the DO statement, which sets the control variable to its initial value.	Correct source program.
FC026	6	ILLEGAL BRANCH INTO DEBUG REGION	It is illegal to branch into a debug packet from outside the packet and illegal to branch out of the packet to another packet or the main program.	Correct branch logic.
FC027	2	ARITHMETIC CONSTANT INITIALIZES LOGICAL VARIABLE	DATA statement processor cannot perform a meaningful conversion.	Correct DATA statement.

Table E-1. Compile-Time Diagnostic Messages (Part 4 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC028	2	LOGICAL CONSTANT INITIALIZES NON LOGICAL VARIABLE	DATA statement processor cannot perform a meaningful conversion.	Correct DATA statement.
FC029	2	LITERAL TRUNCATED ON RIGHT	Character string length exceeded length of variable, example: A/'ABCDE'	Correct DATA statement.
FC030	6	INSUFFICIENT NUMBER OF SUBSCRIPTS	Except in an EQUIVALENCE statement, an array element reference must contain the same number of subscripts as the array declarator.	Add missing subscripts.
FC031	6	ILLEGAL VARIABLE IN SUBSCRIPT	Variable is not the induction variable of a controlling implied DO loop.	Correct subscript.
FC032	4	INSUFFICIENT NUMBER OF CONSTANTS	DATA statement has more variables than constants.	Correct DATA statement.
FC033	6	CONTROL VARIABLE OF IMPLIED DO IS DUPLICATED	Control variable appears twice in a nest or is itself initialized within the loop.	Correct statement.
FC034	2	TOO MANY CONSTANTS	More constants than variables appear in the DATA statement.	Correct DATA statement.
FC035	6	FORMAT STATEMENT LABEL label-name IS UNDEFINED	The FORMAT statement label was referenced in a READ/WRITE statement but not defined in the source program.	Provide FORMAT statement or correct the READ/WRITE.
FC036	6	LABEL label-name IS UNDEFINED. USED IN ASSIGN STATEMENT	The label specified was not defined.	Either define the label or correct the ASSIGN statement.
FC037	4	FORMAT STATEMENT LABEL label-name NEVER REFERENCED	As stated.	Correct the format label or remove the unnecessary FORMAT statement.
FC051	8	NO STATEMENTS IN PROGRAM UNIT	Null data set or only comments in source module.	Correct job control, or correct source module.



Table E-1. Compile-Time Diagnostic Messages (Part 5 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC052	8	PROGRAM CONTAINS NO USABLE STATEMENTS	A main program, subroutine, or function contains no executable statements or a block subprogram contains no initial block declarations.	Correct subprogram.
FC053	6	END STATEMENT IS MISSING.	End of file encountered on data set but no END statement detected.	Provide END statement.
FC054	6	STATEMENT CANNOT BE CLASSIFIED.	Compiler cannot identify the statement. For large statements containing keywords such as FORMAT, COMMON, etc, the keyword must be within the first 3 lines.	Correct or compress the statement.
FC055	6	STATEMENT IS OUT OF ORDER	See ordering requirements in Section 1.	Place statement in proper sequence.
FC056	6	MISSING AT STATEMENT.	Debug packet not preceded by an AT statement has been encountered.	Provide AT statement or change source sequence.
FC057	6	STATEMENT INVALID FOR BLOCK DATA SUBPROGRAM	A block data subprogram may only contain specification and data initialization statements.	Remove statement.
FC058	6	MISSING EQUAL SIGN OR UNCLASSIFIABLE STATEMENT	No recognizable keyword, and syntax invalid for arithmetic assignment, logical assignment, or statement function.	Correct statement.
FC059	4	REDUNDANT RIGHT PARENTHESIS	No corresponding left parenthesis for marked delimiter.	Check statement.
FC060	6	MISSING OPERATOR. REMAINDER OF STATEMENT IGNORED	An arithmetic or logical operator was expected where indicated.	Correct statement.
FC061	6	STATEMENT NOT CLASSIFIABLE	Statement cannot be classified because of misspelled keyword or unrecognizable syntax.	Correct statement.
FC062	6	RETURN STATEMENT IN MAIN PROGRAM.	A RETURN statement should only appear in a subroutine or function subprogram.	Change statement to STOP or CALL EXIT.

Table E-1. Compile-Time Diagnostic Messages (Part 6 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC063	6	MISSING (AFTER KEYWORD 'IF'.	An IF statement is of the form IF ().	Correct source program.
FC064	6	MISSING) AFTER KEYWORD 'IF'.	An IF statement must be of the form IF ().	Correct source program.
FC065	6	INVALID EXPRESSION IN ARITHMETIC IF	The expression cannot be evaluated as negative, zero, or positive.	Correct statement.
FC066	6	COMPLEX EXPRESSION NOT PERMITTED IN ARITHMETIC IF	As stated.	Correct program.
FC067	6	STATEMENT NOT SCANNED AFTER KEYWORD	Keyword is not acceptable; statement scanning is discontinued.	Correct statement.
FC068	6	ILLEGAL CONVERSION IN ASSIGNMENT statement	In an ASSIGNMENT statement of the form V=E, when V is logical, E must also be logical. When V is arithmetic, E must also be arithmetic.	Correct statement or specify additional data typing.
FC069	4	SCAN TERMINATED. REMAINDER OF EXPRESSION IGNORED	Compiler could not recover from syntax error.	Correct statement.
FC070	6	THE OBJECT STATEMENT OF THIS LOGICAL IF IS ILLEGAL	A DO or logical IF cannot be the object statement of a logical IF.	Correct statement.
FC071	6	ILLEGAL DESTINATION LABEL(S) FOR ARITHMETIC IF	The labels are for nonexecutable statements or constitute an illegal entry into a DO nest.	Correct flow of control.
FC072	6	INVALID SYNTAX IN NAMELIST STATEMENT	Array element names and implied DO loops are not permitted.	Correct syntax of statement.

Table E-1. Compile-Time Diagnostic Messages (Part 7 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC073	6	NAMelist NAME HAS BEEN PREVIOUSLY DEFINED	Namelist names must be unique within a program unit.	Change name.
FC074	6	INVALID DELIMITER	The character indicated by the marker is invalid, example is '5(' in an expression.	Correct statement.
FC075	6	EXTERNAL STATEMENT REDEFINES USAGE OF NAME	The name is being declared to be a procedure name, but is used in a different context in another statement.	Correct program to use name consistently.
FC076	6	COMMON BLOCK NAME ALSO USED AS SUBPROGRAM NAME	The linker requires that all procedure and common block names must be unique.	Change one of the names.
FC077	4	EQUIVALENCE GROUP NOT FOUND	Empty parenthetical grouping.	Correct source statement.
FC078	6	SUBSCRIPT ILLEGAL OR NOT ALLOWED HERE	A subscript in an EQUIVALENCE statement must be an integer constant.	Correct statement.
FC079	4	SUBSCRIPT ON NON ARRAY IGNORED	In a data statement, a variable name followed by a left parenthesis has been found. Since the variable was not declared previously in a dimension statement the name is flagged as a nonarray. Recovery is made to the next name in the variable list.	Either remove the extraneous parenthesis or dimension the array.
FC080	4	LABEL ON SPECIFICATION STATEMENT IGNORED	Label is not accepted as the destination point of a control transfer.	Warning only.
FC081	6	MORE THAN SEVEN SUBSCRIPTS	An array may have from one to seven dimensions.	Correct statement.

Table E-1. Compile-Time Diagnostic Messages (Part 8 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC082	4	MORE THAN ONE DIMENSION DECLARATOR FOR THIS ARRAY	The dimensions of this array have been declared previously.	Delete declarator. Do not delete the array name if a type or COMMON statement.
FC083	4	NO CONSTANTS IN DATA CONSTANT LIST	As stated. Problem may be due to misspelled constant or extraneous delimiter in list.	Correct the statement by inserting the constant list or correcting punctuation.
FC084	6	ILLEGAL FORM FOR DIMENSION DECLARATOR	A dimension declarator must be a constant or integer variable.	Correct subscript.
FC085	4	ONLY ONE EXPLICIT TYPE DECLARATOR ALLOWED	A symbolic name may appear only once in an explicit type statement.	Remove second declaration.
FC086	6	NAME ILLEGAL FOR DATA	NAME is not a variable or array name, or is a dummy argument, or a blank common.	Correct statement.
FC087	6	ADJUSTABLE DIMENSIONS PROHIBITED IN MAIN PROGRAMS	As stated.	Correct program logic.
FC088	6	NON-LOGICAL USED IN LOGICAL EXPRESSION	Improper formation of logical expression.	Correct expression.
FC089	6	LOGICAL PRIMARIES USED WITH RELATIONAL OPERATORS	Operators GT, GE, EQ, NE, LT, LE cannot be used with logical entries.	Correct expression.
FC090	6	LOGICAL PRIMARY DETECTED IN ARITHMETIC EXPRESSION	As stated.	Correct expression.

Table E-1. Compile-Time Diagnostic Messages (Part 9 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC091	6	ILLEGAL EXPRESSION	Expression is malformed.	Correct expression.
FC092	6	MISSING RIGHT PARENTHESIS	Right parenthesis missing or redundant; left parenthesis present.	Add or delete a parenthesis.
FC093	4	SCAN RESUMED AT THIS POINT	As stated.	Correct statement.
FC094	6	LEFT PARENTHESIS MISSING AFTER UNIT NUMBER	DEFINE FILE should have a parenthesis after the unit number.	Correct statement.
FC095	6	MISSING COMMA BETWEEN LIST ITEMS	I/O list does not have a comma between the two symbolic names.	Correct statement.
FC096	4	MISSING RIGHT PARENTHESIS IN READ/WRITE	A right parenthesis must be present following the format reference or END/ERR clauses.	Correct statement.
FC097	4	1 THROUGH 99 ARE ONLY VALID UNIT NUMBERS	Unit number out of range.	Change unit reference.
FC098	6	UNIT MUST BE INTEGER CONSTANT OR VARIABLE	Noninteger data types not permitted as unit number reference.	Change unit reference.
FC099	4	FILE SIZE TOO LARGE	File size must be less than 25 bits.	Reduce size.
FC100	6	FILE SIZE MUST BE INTEGER 4 CONSTANT	The file size specified in a DEFINE FILE statement must be an integer 4 type.	Correct size specification.
FC101	4	RECORD SIZE IN DEFINE FILE STATEMENT IS TOO LARGE	Record size exceeds 13030 bytes.	Reduce record size.

Table E-1. Compile-Time Diagnostic Messages (Part 10 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC102	6	MAX RECORD MUST BE 14 CONSTANT	Max record size specified in a DEFINE FILE statement must be an integer 4 constant.	Correct size specification.
FC103	6	TRANSFER LETTER MUST BE 'L', 'U', OR 'E'	As stated.	Correct DEFINE FILE statement.
FC104	6	MISSING UNIT SPECIFICATION	The unit specification on a DEFINE FILE statement is missing.	Supply missing integer constant unit number.
FC105	6	ILLEGAL SYNTAX IN DATA STATEMENT	As stated.	Correct statement.
FC106	6	BAD SUBSCRIPT FORMAT FOR DATA ARRAY	Subscripted arrays in the variable list of a DATA statement must be of the form 'C * V + K' where: C and K are positive integer constants, V is an integer variable.	Correct subscript.
FC107	6	RECORD POINTER MUST BE INTEGER EXPRESSION	Record pointer in a DAM statement must be of type integer greater than zero and less than or equal to the number of records in the file.	Use IFIX function or ASSIGNMENT statement to correct expression type.
FC108	6	BAD NESTING OF IMPLIED DO IN DATA STATEMENT	As stated. An implied DO list must be completely enclosed in any surrounding implied DO list.	Change statement.
FC109	6	NAMELIST INCOMPATIBLE WITH FORTRAN II STATEMENTS	A namelist may be referenced only with the FORTRAN IV statements READ/ WRITE (unit, namelist).	Correct statement.
FC110	6	ILLEGAL IMPLIED DO SPECIFICATION IN DATA STATEMENT	As stated.	Correct statement.

Table E-1. Compile-Time Diagnostic Messages (Part 11 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC111	4	DUPLICATE END CLAUSE	Two END clauses in one statement.	Probably intended to be an ERR clause.
FC112	4	END CLAUSE NOT VALID FOR WRITE STATEMENT	Not a supported feature of FORTRAN IV.	Change program logic.
FC113	6	MULTIPLY DEFINED FORMAT STATEMENT LABEL	Format labels must be unique.	Change label and all references.
FC114	6	MISSING OR INVALID LABEL	END/ERR clause has no transfer label, contains the label of a nonexecutable statement, or has an illegal transfer of control.	Change destination label.
FC115	4	DUPLICATE ERR CLAUSE	Two ERR clauses.	Correct statement. Probably should be END clause.
FC116	6	INVALID EDITING CODE	Edit code cannot be recognized.	Correct statements.
FC117	4	NO I/O LIST ALLOWED WITH NAMELIST	The variable list must appear in the NAMELIST statement and not in the READ/WRITE statement.	Correct both statements.
FC118	6	ILLEGAL I/O LIST ITEM	The symbolic name indicated by the marker cannot appear in an I/O list.	Correct statement.
FC119	6	REPEAT COUNT ON T	No repeat count is permitted on a format 'T' field descriptor. The legal form of this descriptor is 'Tp' where p is an integer such that $0 < p \leq 32767$.	Remove repeat count.
FC120	4	SCALE FACTOR OUT OF RANGE	The absolute value of the scale factor cannot exceed 127.	Correct scale factor.

Table E-1. Compile-Time Diagnostic Messages (Part 12 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC121	2	INVALID USAGE OF SCALE FACTOR	Scale factor on a nonreal editing code.	Correct statement.
FC122	4	FIELD WIDTH OR GROUP COUNT MUST BE LESS THAN 256	As stated.	Reorganize format.
FC123	6	MISSING GROUP OR FIELD	As stated.	Correct statement where indicated by marker.
FC124	6	ILLEGAL SYNTAX IN FIND STATEMENT	As stated.	Correct statement.
FC125	6	ITEM NOT PERMITTED ON LEFT SIDE OF EQUALS OPERATOR	Item is not a variable or array element name or function name.	Correct statement.
FC126	6	ITEM ILLEGALLY REFERENCED	Name indicated is not a legal primary.	Check other diagnostics in program.
FC127	6	ITEM ILLEGAL FOR I/O LIST	Name in an I/O list must be either an array element, array name, or a variable name.	Correct statement.
FC128	6	NAME IS NOT A SUBROUTINE NAME	Conflict in usage of this name.	The name was used in different context earlier in the program. Correct one of the usages.
FFC129	6	ITEM NAMED NOT ARITHMETIC ARGUMENT	An invalid name has been scanned in a function argument list. Valid arguments are constants, variables, arrays, or procedure names.	Correct argument.
FC130	6	FUNCTION OR UNDECLARED ARRAY	Statement function out of sequence or undeclared array.	Correct sequence or supply array declarator.

Table E-1. Compile-Time Diagnostic Messages (Part 13 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC131	6	UNDECLARED ARRAY OR REPEATED STATEMENT FUNCTION	Appears in a context such as A(I) =2.1, X=10, A(I) ... where the compiler assumed the first statement was a statement function.	Correct program.
FC132	6	UNDECLARED ARRAY NAME	This array name did not appear in a dimension declarator.	Correct spelling or insert the declarator.
FC133	6	SYMBOLIC NAMES CANNOT EXCEED SIX CHARACTERS	As stated.	Shorten name.
FC134	6	DO PARAMETER MUST BE A CONSTANT OR SIMPLE VARIABLE	Array element names, procedure names, etc, not acceptable as DO parameters.	Correct statement and rerun job.
FC135	6	VARIABLE MUST BE OF TYPE INTEGER	As stated.	Correct statement and rerun job.
FC136	6	ITEM NOT A SIMPLE VARIABLE	Array element encountered in improper context.	Move array element to a simple variable (scalar) in a previous statement.
FC137	6	CONTROL VARIABLE SHOULD FOLLOW LOOP LABEL	Control variable missing from DO statement.	Correct statement.
FC138	6	VALUE WITH T ILLEGAL OR TOO LARGE	The record position cannot exceed 32767.	Correct format.
FC139	6	MISSING EQUAL SIGN FOLLOWING DO CONTROL VARIABLE	As stated.	Correct DO statement.
FC140	6	ILLEGAL FORMAT DESCRIPTOR	The format code is not recognizable.	Correct statement.

Table E-1. Compile-Time Diagnostic Messages (Part 14 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC141	6	MISSING OR ILLEGAL SUB-ROUTINE NAME	As stated.	Name is illegal if used in a differing context in prior statements.
FC142	6	MISSING OPERATOR OR UNDECLARED ARRAY	Left parenthesis encountered after a name known to be a scalar.	Correct source statement.
FC143	6	A SUBSCRIPT EXPRESSION MAY ONLY BE INTEGER OR REAL	Complex and logical expressions may not be used as subscripts.	Correct subscript.
FC144	6	TOO FEW SUBSCRIPTS FOR THIS ARRAY	Array element reference contains fewer subscripts than the array declarator.	Correct either the declarator or the reference.
FC145	6	MISSING FORMAT LABEL	The label on the format statement is missing. Labels on format statements are mandatory.	Supply a label.
FC146	2	WARNING-REPEATED SCALE FACTOR	Previous scale factor cannot take effect.	Correct source statement.
FC147	6	ACTUAL AND DUMMY ARGUMENTS ARE OF DIFFERING TYPES	In a statement function reference, the actual and dummy arguments must be of the same type.	Correct either the reference or statement function, or insert a type statement.
FC148	6	TOO MANY ARGUMENTS FOR THIS FUNCTION	As stated. This diagnostic is for implicit and basic external functions only.	Correct function reference.
FC149	6	TOO FEW ARGUMENTS FOR THIS FUNCTION	As stated.	Add missing argument.
FC150	6	MISSING LEFT PARENTHESIS	As stated.	Add a parenthesis.

Table E-1. Compile-Time Diagnostic Messages (Part 15 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC151	6	DO LIST TOO SHORT	In scanning the DO statement list, the terminal parameter has not been found. Consequently, the DO range is undefined.	Supply terminal parameter.
FC152	4	SUBCHK LIST MAY ONLY CONTAIN ARRAY NAMES	Simple variable or procedure name encountered. Subscript checking is only meaningful for arrays.	Delete or correct symbolic name.
FC153	6	CONTROL VARIABLE OF DO MUST BE INTEGER TYPE	As stated.	Change variable to integer.
FC154	6	ARGUMENT TYPE CONFLICT	The argument specified is illegal for the (generic) function specified.	Correct function reference.
FC155	6	IMPROPERLY NESTED DO LOOPS	A DO loop must be completely enclosed by any surrounding DO loop. See fundamentals of FORTRAN reference manual, UP-7536 (current version).	Correct structure of loops.
FC156	6	LABEL FOR DO MISSING OR BAD	Error in processing label designating DO loop terminator.	Fix label in DO statement.
FC157	4	DUPLICATE AT STATEMENT IGNORED	Two AT statements specify the same statement label.	Select proper debug packet.
FC158	6	RECURSIVE STATEMENT FUNCTION OR UNDECLARED ARRAY	Occurs for statement such as $A(I)=A(I)+1.0$ when A does not appear in a dimension declarator.	Correct program.
FC159	6	UNDECLARED ARRAY OR BAD STATEMENT FUNCTION	Statement cannot be classified either as a statement function or an assignment statement.	Correct statement or add dimension declarator.
FC160	6	REPEATED DUMMY ARGUMENTS IN STATEMENT FUNCTION	Each dummy argument name must be unique.	Correct statement.

Table E-1. Compile-Time Diagnostic Messages (Part 16 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC161	2	EXTRANEIOUS DATA AFTER CLOSING PARENTHESIS	As stated.	Check that right parenthesis is not in an NH string with an improper N.
FC162	6	ILLEGAL STATEMENT FUNCTION DEFINITION	Statement function argument not a simple variable. This diagnostic can also occur for an undeclared array.	Correct statement function.
FC163	6	A DEBUG STATEMENT OUT OF ORDER	AT, TRACE ON, TRACE OFF, and DISPLAY statements may only occur after the DEBUG statement.	Correct order of statements.
FC164	6	ENTRY STATEMENT NOT PERMITTED IN A MAIN PROGRAM	These statements are permitted only in subroutine and function subprograms.	Remove statement and associated program logic.
FC165	6	ILLEGAL LIST SYNTAX FOR DISPLAY STATEMENT	The list may contain only variable and array names (without subscripts). Dummy arguments called by name are not permitted.	Correct statement.
FC166	6	ENTRY NAMED USED IN ANOTHER CONTEXT	As stated.	Correct misspelling or program logic.
FC167	6	DUPLICATE DUMMY ARGUMENTS	Dummy argument names must be unique.	Correct statement.
FC168	6	CALL BY NAME/ VALUE CONFLICT FOR THIS ARGUMENT	The method of argument processing for this argument is different in a previous entry, subroutine, or function statement.	Select either CALL by NAME or CALL by VALUE and use consistently throughout program.
FC169	6	NO LABEL SPECIFIED IN AT STATEMENT	The debug packet has no reference point (entry label) in the source code and cannot be executed.	Provide label after AT statement.
FC170	6	DUPLICATE ENTRY NAME	As stated. Entry names must be unique.	Change entry name.

Table E-1. Compile-Time Diagnostic Messages (Part 17 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC171	6	FUNCTIONS MAY NOT USE LABELS AS ARGUMENTS	As stated.	Add a status variable to argument list, or use the ERROR/ERROR1 subroutines for control purposes.
FC172	6	FUNCTIONS MUST HAVE AT LEAST ONE ARGUMENT	As stated.	For functions such as random number generators, an argument must be present even if it is never referenced.
FC173	6	ILLEGAL LIST SYNTAX FOR DUMMY ARGUMENT LIST	A dummy argument list may consist only of simple variable names which are unique in the list.	Correct list.
FC174	4	MISSING NAME ON PROGRAM STATEMENT	As stated.	Provide a name; system default is \$MAIN.
FC175	4	EXTRANEIOUS DATA AFTER PROGRAM NAME	As stated.	Remove extraneous characters.
FC176	4	EXTRANEIOUS DATA AFTER VALID DEBUG OPTION	As stated.	Remove extraneous characters.
FC177	4	ONLY ONE DEBUG STATEMENT ALLOWED	As stated.	Convert to a continuation of the previous debug statement.
FC178	6	ILLEGAL LIST SYNTAX	Improper debug statement list.	Correct statement at the point indicated by the marker.
FC179	4	LABELS NOT ALLOWED ON AT STATEMENTS	AT is a nonexecutable statement. A label is not allowed.	Remove label from statement.
FC180	4	NON ARRAY OR VARIABLE INIT	An attempt has been made to specify the Debug Init option on an item which is neither an array or simple variable name.	Either correct or remove the name.

Table E-1. Compile-Time Diagnostic Messages (Part 18 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC181	6	COMMON VARIABLE MULTIPLY DEFINED	A variable may only occur in one common block.	As stated.
FC200	4	RECOVERED MISSING FIELD WIDTH TO 10	Field width in the format specification is missing. The compiler substitutes a value of 10 and continues scanning.	Define width of field.
FC201	2	WARNING SCALE OUT OF RANGE	The absolute value of the scale factor cannot exceed 127.	Correct the scale factor.
FC202	2	IGNORED DIGITS AFTER FIFTH IN STOP OR PAUSE	A maximum of 5 digits may be specified on a STOP or PAUSE statement. All excess digits were ignored.	Truncate the number or change it to a literal.
FC203	2	HEXADECIMAL CONSTANT CONTAINS INVALID HEX DIGIT	A hexadecimal constant may only contain the digits 0 through 9 and A through F. The constant may be a maximum of 32 characters in length.	Correct constant.
FC204	4	WARNING INSERTED LEFT PAREN TO START GROUP	The opening parenthesis denoting an implicit, equivalence or format list is missing. The compiler inserts the parenthesis and continues scanning the statement.	Correct statement.
FC205	4	WARNING INSERTED RIGHT PAREN TO CLOSE GROUP	The closing parenthesis denoting the end of an implicit, equivalence, or format group is missing. The compiler inserts the parenthesis and continues scanning.	Correct statement.
FC206	2	WARNING INSERTED MISSING COMMA	A comma necessary to syntax is missing. If syntax has been acceptable up to this point, the compiler inserts the comma and continues scanning. Commas will be inserted between multiple data lists in data statements, multiple equivalence groups, debug options or elements in an I/O list.	Correct list.

Table E-1. Compile-Time Diagnostic Messages (Part 19 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC207	2	WARNING INTEGER TOO BIG USED '7FFFFFFF'	An integer constant exceeds one full word. Binary bits were truncated on the left, and a maximum positive value of '7FFFFFFF' was substituted.	Examine constant for correctness. A precision or type change may be appropriate.
FC208	4	RECOVERED BAD SUBSCRIPT TO '1'	An array declarator or array element reference contains an invalid or unrecognizable expression as a subscript. The compiler substitutes the value '1' for the subscript and closes the current array. Scanning continues with the next array, if present.	Correct subscript.
FC209	4	RECOVERED MISSING FRACTION WIDTH TO 0	Fraction width in the format specification is missing. The compiler substitutes the value 0 and continues scanning.	Define format width.
FC210	4	RECOVERED MISSING SLASH TO BLANK COMMON	In a COMMON statement, either the slash delimiting the common block name is missing or the name itself is unrecognizable.	Put in missing delimiter or name.
FC211	4	RECOVERED AND RESTARTED SCAN AT THIS NAME	Recovery was made to the next valid name encountered after the bad data. The scan resumes at this point. All unidentifiable data prior to this point was ignored by the compiler.	Correct statement.
FC212	4	RECOVERED WRITE NO PAREN TO PRINT	A WRITE statement has a left parenthesis missing. The compiler therefore treats the statement as a PRINT statement.	Add missing delimiter.
FC213	6	SYNTAX UNRECOGNIZED	The compiler has not encountered any acceptable keywords or syntax and cannot identify the statement. Scanning is terminated.	Correct statement.
FC214	2	WARNING COMPLEX TYPE CONFLICT	The real and imaginary components of a complex constant do not agree in type. The compiler converts the component of lower type to the higher type.	Correct constant so that both components have the same type.

Table E-1. Compile-Time Diagnostic Messages (Part 20 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC215	2	WARNING CONSTANT TRUNCATED	For an integer constant, binary bits were truncated on the left, which may affect its sign. For a double precision constant, decimal digits on the right were discarded. For a single precision real constant, right hexadecimal digits from its preliminary internal representation were discarded.	Examine constant for correctness; a precision change may be appropriate.
FC216	4	WARNING FINAL PERIOD INSERTED	The final period that delimits a relational or logical operator is missing. The compiler inserts the missing delimiter and continues scanning.	Correct statement.
FC217	4	WARNING INITIAL PERIOD INSERTED	The initial period that delimits a relational or logical operator is missing. The compiler inserts the period if the following syntax appears correct.	Correct statement.
FC218	2	WARNING NOT AN IMPLIED DO	A parenthesized expression has been encountered in an I/O list. The compiler is anticipating an implied DO loop, but the format of the expression is not of the form (NAME (I), I = M ₁ , M ₂ , M ₃).	Correct parenthesized expression if it is intended to be an implied DO.
FC219	4	WARNING PERIOD CHANGED TO ASTERISK	The compiler has encountered a period which does not delimit a logical constant or a logical or relational operator. The compiler interprets the period as a misspunched asterisk and continues scanning.	Correct statement.
FC220	2	REAL EXPONENT DIGITS MISSING	The exponents on a real or double precision constant are missing.	Correct constant.
FC221	4	ERROR HEX TRUNCATED	A hexadecimal constant may be a maximum of 32 characters in length. Truncation occurs on the leftmost digits.	Correct or shorten hexadecimal constant.
FC222	2	IGNORED COMMA	An extraneous comma has been encountered in a define file list, an I/O list, or a format descriptor list.	Correct the statement by removing the comma or inserting the missing list item it delimits.

Table E-1. Compile-Time Diagnostic Messages (Part 21 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC223	2	IGNORED LABEL OUT OF RANGE AS DUMMY	A statement label has been found which exceeds the maximum value of 99999.	Correct label.
FC224	4	IGNORED UNRECOGNIZED	In scanning, the compiler has encountered invalid data which it cannot identify. Recovery is made to the next valid name found and all else in between is ignored.	Correct statement.
FC225	4	WARNING POSSIBLE MISSING ASTERISK	The compiler has encountered two apparently valid variable names or constants not separated by an operator, and assumes multiplication was intended.	Correct statement.
FC226	2	IGNORED BAD LENGTH SPECIFICATION	The length specification on implicit or explicit type statement is either an invalid constant or the value exceeds 32.	Correct length specification or decrease its value.
FC227	2	DO WARNING	The compiler has encountered a variable name of the form 'DO NNN NAME =' which looks like a DO statement.	Check statement.
FC228	2	MISSING APOSTROPHE	The closing apostrophe of a literal constant is missing.	Close literal string.
FC229	4	TRUNCATED LITERAL	A Hollerith constant or literal constant has exceeded the maximum length of 255 characters.	Shorten character string.
FC230	4	BAD LABEL	The statement label is either an invalid constant or it exceeds the maximum label value of '99999'.	Correct statement label.
FC901	8	COMPILER NEEDS MORE SPACE FOR THIS PROGRAM	The main memory is insufficient to compile this program unit.	Allocate more storage to the job, or segment the program into smaller units.

Table E-1. Compile-Time Diagnostic Messages (Part 22 of 22)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FC902	8	STACK OVERFLOW. SIMPLIFY THIS STATEMENT	The compiler control stack has become too large. The compilation cannot be completed.	Check program for excessively complex statements and simplify. If problem persists, submit a SUR.
FC903	8	I/O ERROR ON COMPILER'S WORK FILE	The scratch files the compiler uses have had a hardware error. It is impossible to complete compilation.	Make certain that the correct number of work files are allocated and available.
FC904	8	SOURCE PROGRAM NOT FOUND	Module name on the PARAM IN card was not found on the input disc file.	Display contents of input file to be certain the source program is there.
FC910	8	FORTRAN REQUIRES MICROLOGIC EXPANSION	The compiler requires the extended instruction set which is available only if the micrologic expansion has been loaded.	Load the 2K COS and a supervisor which supports floating-point features.
FC940	8	INTERNAL COMPILER ERROR IN PHASE N, CODE NN, CARD NUMBER NNNN.	As indicated. Submit a SUR.	<p>The code is for maintenance purposes only and is not meaningful to users. The card number is not always present, but can be useful, since the source code may be changed to get around the problem. The phase number may be useful in determining where the compiler is processing erroneously:</p> <ul style="list-style-type: none"> ■ Phase 1 reads the source program, performs syntax analysis and builds various tables for later use. ■ Phase 2 processes COMMON and EQUIVALENCE statements, performs preliminary storage allocation, processes and generates label tables, and prints the source program and any diagnostics. ■ Phase 3 processes subscripts, performs constant arithmetic, processes constants and argument lists, and symbolically expands statement function references. ■ Phase A performs storage allocation compilation and prints the related map. ■ Phase 5 generates preliminary code and processes common subexpressions. ■ Phase 6 generates the object module and executable code and prints it.





Appendix F. Run Time Modules

Table F-1. Extended FORTRAN Run-Time Modules (Part 1 of 10)

Module	CSECT or Entry Name	Function
FL\$ABS	FL\$ABS ABS DABS IABS JABS	Integer and real absolute value
FL\$ABTRM	FL\$ABTRM	Abnormal termination code
FL\$ARFOR	FL\$ARFOR	Array FORMAT processor
FL\$ASIN	FL\$ASIN ACOS ARCOS ARSIN ASIN	Arcsine/arccosine functions
FL\$ATAN	FL\$ATAN ATAN ATAN2	Arctangent functions
FL\$BCKSA	FL\$BCKSA	BACKSPACE processor
FL\$CABS	FL\$CABS CABS	Complex absolute value function
FL\$CBRT	FL\$CBRT CBRT	Cube root
FL\$CC\$	FL\$CC\$ FL\$CC FL\$CI FL\$CJ FL\$CR FL\$IC FL\$JC FL\$RC	Complex exponential functions: C**C C**I4 C**I2 C**R4 I4**C I2**C R4**C
FL\$CDABS	FL\$CDABS CDABS	Complex*16 absolute value function

Table F-1. Extended FORTRAN Run-Time Modules (Part 2 of 10)

Module	CSECT or Entry Name	Function
FL\$CDD\$	FL\$CDD\$ FL\$CD FL\$CDC FL\$CDD FL\$DC FL\$DCC FL\$DCD FL\$DCI FL\$DCJ FL\$DCR FL\$DDC FL\$IDC FL\$JDC FL\$RDC	Complex*16 exponential functions: C8**R8 C8**C16 C16**C16 R8**C8 C16**C8 C16**R8 C16**I4 C16**I2 C16**R4 R8**C16 I4**C16 I2**C16 R4**C16
FL\$CDEXP	FL\$CDEXP CDEXP	Complex*16 exponential functions
FL\$CDLOG	FL\$CDLOG CDLOG FL\$CDLG	Complex*16 logarithm function
FL\$CDMPY	FL\$CDMPY CDDVD# CDMPY#	Complex*16 multiply/divide
FL\$CDSIN	FL\$CDSIN CDCOS CDCOSH CDSIN CDSINH	Complex*16 sine/cosine and hyperbolic sine/cosine functions
FL\$CDSQT	FL\$CDSQT CDSQRT	Complex*16 square root function
FL\$CEXP	FL\$CEXP CEXP	Complex exponential function
FL\$CLNRW	FL\$CLNRW	File close routine (no rewind)
FL\$CLOG	FL\$CLOG CLOG	Complex logarithm function
FL\$CLOSE	FL\$CLOSE	Final file close
FL\$CMPLX	FL\$CMPLX CMPLX DCMPLX	Complex intrinsic functions: C8 C16
FL\$CMPY	FL\$CMPY CDVD# CMPY#	Complex multiply/divide
FL\$CNFLT	FL\$CNFLT	
FL\$COLUM	FL\$COLUM	

Table F-1. Extended FORTRAN Run-Time Module (Part 3 of 10)

Module	CSECT or Entry Name	Function
FL\$CONJG	FL\$CONJG CONJG DCONJG	Conjugate intrinsic functions Single-precision conjugate function Double-precision conjugate function
FL\$CSIN	FL\$CSIN CCOS CCOSH CSIN CSINH	Complex sine/cosine and hyperbolic sine/cosine functions
FL\$CSQRT	FL\$CSQRT CSQRT	Complex square root function
FL\$DASIN	FL\$DASIN DACOS DARCOS DARSIN DASIN	Real*8 arcsine/arccosine functions
FL\$DATAN	FL\$DATAN DATAN DATAN2	Real*8 arctangent functions
FL\$DBLE	FL\$DBLE CDBLE DBLE	Single to double intrinsic functions: C8 to C16 R4 to R8
FL\$DBOUT	FL\$DBOUT FL\$DBCL FL\$DBOP FL\$FLSH FL\$STKR	Debug I/O routines
FL\$DCBRT	FL\$DCBRT DCBRT	Real*8 cube root function
FL\$DDPOW	FL\$DDPOW DEXP DEXP10 DLOG DLOG10 FL\$DD FL\$DEXP\$ FL\$DI FL\$DJ FL\$DLOG\$ FL\$DR FL\$ID FL\$JD FL\$RD FP\$DTD FP\$DTH FP\$DTI FP\$DTR FP\$HTD FP\$ITD FP\$RTD	Real*8 power functions R8**R8 R8**14 R8**12 R8**R4 14**R8 12**R8 R4**R8 R8**R8 (basic FORTRAN) R8**12 (basic FORTRAN) R8**14 (basic FORTRAN) R8**R4 (basic FORTRAN) 12**R8 (basic FORTRAN) 14**R8 (basic FORTRAN) R4**R8 (basic FORTRAN)

Table F-1. Extended FORTRAN Run-Time Module (Part 4 of 10)

Module	CSECT or Entry Name	Function
FL\$DEBUG	FL\$DEBUG FL\$DARI FL\$DBGUN FL\$DCHK FL\$DELI FL\$DINT FL\$DRTN FL\$DSBT FL\$DTON FL\$DTRC FL\$DUNT	Debug control routines: array INIT UNIT value SUBCHK array element INIT variable INIT RETURN SUBTRACE TRACE OFF TRACE ON TRACE UNIT
FL\$DEFIL	FL\$DEFIL	DEFINE FILE statement processor
FL\$DERF	FL\$DERF DERF DERFC	Real*8 error function
FL\$DFNDB	FL\$DFNDB	FIND statement processor
FL\$DGAMA	FL\$DGAMA DGAMMA DLGAMA	Real*8 distribution function
FL\$DHPER	FL\$DHPER DCOSH DSINH	Real*8 hyperbolic sine/cosine
FL\$DHYP	FL\$DHYP DTANH	Real*8 hyperbolic tangent
FL\$DIM	FL\$DIM DDIM DIM IDIM JDIM	Positive difference intrinsic functions
FL\$DMAX	FL\$DMAX DMAX1 DMIN1	Real*8 maximum/minimum intrinsic functions
FL\$DOPNA	FL\$DOPNA FL\$DFNDA	Direct access READ/WRITE processor Direct access FIND processor
FL\$DSIN	FL\$DSIN DCOS DSIN FL\$DCOS\$ FL\$DSIN\$	Real*8 sine/cosine function
FL\$DSQRT	FL\$DSQRT DSQRT	Real*8 square root function
FL\$DTAN	FL\$DTAN DCOT DCOTAN DTAN	Real*8 tangent/cotangent function

Table F-1. Extended FORTRAN Run-Time Module (Part 5 of 10)

Module	CSECT or Entry Name	Function
FL\$DUMP	FL\$DUMP DUMP FL\$DUMPD PDUMP	DUMP/PDUMP processor
FL\$DVCHK	FL\$DVCHK DVCHK	Divide check subroutine
FL\$EDTAI	FL\$EDTAI	A edit - input
FL\$EDTAO	FL\$EDTAO	A edit - output
FL\$EDTCI	FL\$EDTCI	Complex input
FL\$EDTCO	FL\$EDTCO	Complex output
FL\$EDTEO	FL\$EDTEO FL\$EDTDO	E edit - output D edit - output
FL\$EDTFI	FL\$EDTFI FL\$EDTEI FL\$EDTDI	F edit - input E edit - input D edit - input
FL\$EDTFO	FL\$EDTFO	F edit - output
FL\$EDTGI	FL\$EDTGI	G input
FL\$EDTGO	FL\$EDTGO	G output
FL\$EDTII	FL\$EDTII	I edit - input
FL\$EDTIO	FL\$EDTIO	I edit - output
FL\$EDTLI	FL\$EDTLI	L edit - input
FL\$EDTLO	FL\$EDTLO	L edit - output
FL\$EDTZI	FL\$EDTZI	Z edit - input
FL\$EDTZO	FL\$EDTZO	Z edit - output
FL\$ENDFA	FL\$ENDFA	ENDFILE processor
FL\$ERCTL	FL\$ERCTL FL\$SWTERR FL\$WTMSG	Error control and traceback routine
FL\$ERE	FL\$ERE	Syntax error CALL subroutine
FL\$ERF	FL\$ERF ERF ERFC	Real error function
FL\$ERRN	FL\$ERRN	Error message setup
FL\$ERTST	FL\$ERTST ERROR ERROR1	ERROR/ERROR1 subroutines

Table F-1. Extended FORTRAN Run-Time Module (Part 6 of 10)

Module	CSECT or Entry Name	Function
FL\$FLOAT	FL\$FLOAT DFLOAT DHFLOT FLOAT HFLOAT	Float intrinsic functions
FL\$FORMT	FL\$FORMT	FORMAT processor
FL\$FTCH	FL\$FTCH FETCH	FETCH subroutine
FL\$GAMMA	FL\$GAMMA ALGAMA GAMMA	Real gamma function
FL\$GDIRI	FL\$GDIRI	List-directed input processor
FL\$GDIRO	FL\$GDIRO	List-directed output processor
FL\$GTMS3	FL\$GTMS3 FL\$GTMSG	OS/3 GET message processor
FL\$HXCVD	FL\$HXCVD	Binary to decimal conversion
FL\$HYPER	FL\$HYPER COSH SINH TANH	Real hyperbolic functions
FL\$IFIX	FL\$IFIX HFIX IFIX	Fix intrinsic function
FL\$IMAG	FL\$IMAG AIMAG DIMAG IMAG	Imaginary part of complex intrinsic function
FL\$INFL3	FL\$INFL3 FL\$INFL	Set file to input mode for OS/3
FL\$INITL	FL\$INITL	Program initialization routine
FL\$INT	FL\$INT AINT DINT IDINT INT	Integer intrinsic functions
FL\$IO	FL\$IO FL\$ERRCT FL\$FUNTB FL\$RERDB FL\$WORKA FL\$1 FL\$3 PRNTR PRNTRC	Standard I/O configuration

Table F-1. Extended FORTRAN Run-Time Module (Part 7 of 10)

Module	CSECT or Entry Name	Function
FL\$IOARA	FL\$IOARA	Dummy I/O common (basic FORTRAN)
FL\$IOCLS	FL\$IOCLS FL\$DCLSE FL\$FCLS FL\$NMLCL FL\$SCLSE	I/O statement termination direct access I/O formatted I/O namelist I/O sequential I/O
FL\$IOCOM	FL\$IOCOM FL\$ERBYT FL\$ERR FL\$GTSAV FL\$IFSAV FL\$IOSAV FL\$OFSAV FL\$ROSAV FL\$RWSAV FL\$SKADR FL\$TBSAV FL\$WTSAV	I/O control common error control routine
FL\$IOERR	FL\$IOERR	Data management fatal error processor
FL\$IOLST	FL\$IOLST FL\$IOLS	I/O list item processor
FL\$IOPEN	FL\$IOPEN FL\$BCKSP FL\$DFIND FL\$DOPEN FL\$ENDFL FL\$REWND FL\$SOPEN	I/O argument list processor for FIND statement for direct access I/O for sequential I/O
FL\$I01	FL\$I01 FL\$ERRCT FL\$FUNTB FL\$RERDB FL\$WORKA FL\$1 FL\$2 FL\$3 FL\$11 FL\$12 FORT11 FORT11C FORT11E FORT12 FORT12C FORT12E FORT2 FORT2C PRNTR PRNTRC	Alternate I/O configuration

Table F-1. Extended FORTRAN Run-Time Module (Part 8 of 10)

Module	CSECT or Entry Name	Function
FL\$IXPI	FL\$IXPI FL\$I1 FL\$IJ FL\$JI FL\$JJ FP\$HTH FP\$HTI FP\$ITH FP\$ITI	Integer power functions: I4**I4 I4**I2 I2**I4 I2**I2 I2**I2 (basic FORTRAN) I2**I4 (basic FORTRAN) I4**I2 (basic FORTRAN) I4**I4 (basic FORTRAN)
FL\$LOAD	FL\$LOAD LOAD OPSYS	LOAD and OPSYS subroutines
FL\$MAX	FL\$MAX AMAX0 AMAX1 AMIN0 AMIN1 JMAX0 JMIN0 MAX MAX0 MAX1 MIN MIN0 MIN1	Max/min intrinsic functions
FL\$MOD	FL\$MOD AMOD DMOD JMOD MOD	Modulo arithmetic intrinsic functions
FL\$NAMEI	FL\$NAMEI	NAMLIST input
FL\$NAMEO	FL\$NAMEO	NAMLIST output
FL\$OUTF3	FL\$OUTF3 FL\$OUTFL	Set file to output mode for OS/3
FL\$OVRFL	FL\$OVRFL OVERFL	Overflow subroutine
FL\$OVW70	FL\$OVW70 OVERFL	Series 70 overflow subroutine
FL\$POWER	FL\$POWER ALOG ALOG10 EXP EXP10 FL\$ALOG\$ FL\$EXP\$ FL\$IR FL\$JR FL\$RI FL\$RJ	Real *4 power functions I4**R4 I2**R4 R4**I4 R4**I2

Table F-1. Extended FORTRAN Run-Time Module (Part 9 of 10)

Module	CSECT or Entry Name	Function
FL\$POWER (cont)	FL\$RR FP\$EXP\$ FP\$HTR FP\$ITR FP\$RTH FP\$RTI FP\$RTR LOG LOG10	R4**R4 I2**R4 (basic FORTRAN) I4**R4 (basic FORTRAN) R4**I2 (basic FORTRAN) R4**I4 (basic FORTRAN) R4**R4 (basic FORTRAN)
FL\$READ	FL\$READ FL\$EOF FL\$ERROA	Input file processor
FL\$REAL	FL\$REAL DREAL REAL	Real part of complex intrinsic function
FL\$REOPN	FL\$REOPN	Reopen closed file
FL\$RWDA	FL\$RWDA	REWIND statement processor
FL\$SCNUM	FL\$SCNUM	
FL\$SIGN	FL\$SIGN DSIGN FL\$DSIGN FL\$ISIGN FL\$JSIGN ISIGN JSIGN SIGN	Sign intrinsic functions
FL\$SIN	FL\$SIN COS FL\$COS\$ FL\$SIN\$ SIN	Sine/cosine functions
FL\$SLITE	FL\$SLITE SLITE SLITET	SLITE/SLITET subroutines
FL\$SNGL	FL\$SNGL CSNGL SNGL	Single from double intrinsic functions
FL\$SOPNA	FL\$SOPNA	
FL\$SQRT	FL\$SQRT SQRT	Real*4 square root function
FL\$SSWTH	FL\$SSWTH SSWTH	System switch subroutines
FL\$STOP	FL\$STOP FL\$PAUSE	STOP/PAUSE processor

Table F-1. Extended FORTRAN Run-Time Module (Part 10 of 10)

Module	CSECT or Entry Name	Function
FL\$STXIT	FL\$STXIT FL\$STXTA	STXIT control routine
FL\$TAN	FL\$TAN COT COTAN TAN	Real*4 tangent/contangent functions
FL\$UNFOR	FL\$UNFOR	Unformatted I/O processor
FL\$WRITE	FL\$WRITE	Output file processor

Appendix G. Subroutine Linkage

G.1. CALLING FORTRAN SUBPROGRAMS

All OS/3 language processors, including Extended FORTRAN, generate and expect standard subprogram linkages in their generated programs. These linkage conventions are defined in the supervisor user guide, UP-8075 (current version). In addition, special FORTRAN conventions and considerations are required (suggested) for successful operation of the run-time system.

G.1.1. Save Area

A FORTRAN subprogram requires a 72-byte, word-aligned save area, supplied by the calling program. Table G-1 illustrates the format of a save area.

Table G-1. Save Area Format (Part 1 of 2)

Word	Byte	Content
1	0	RESERVED FOR SYSTEM USE
2	4	SAVE AREA BACKWARD LINK ADDRESS
3	8	SAVE AREA FORWARD LINK ADDRESS
4	12	CALLING PROGRAM RETURN ADDRESS
5	16	CALLED PROGRAM ENTRY POINT ADDRESS
6	20	REGISTER 0
7	24	REGISTER 1
8	28	REGISTER 2
9	32	REGISTER 3
10	36	REGISTER 4
11	40	REGISTER 5
12	44	REGISTER 6
13	48	REGISTER 7

Table G-1. Save Area Format (Part 2 of 2)

Word	Byte	Content
14	52	REGISTER 8
15	56	REGISTER 9
16	60	REGISTER 10
17	64	REGISTER 11
18	68	REGISTER 12

NOTE:

Each word in the save area is aligned on a full-word boundary.

Word 1 of the save area contains the epilogue address for the entry point. Words 2 and 4 through 18 are initialized according to standard linking conventions.

During execution of a FORTRAN subprogram, register 13 contains the address of this program's save area. Word 2 contains the pointer to the save area supplied to the program. Just before returning to the calling program, register 13 is restored to the calling program's save area and a X'FF' is put into byte 12 of the save area as a termination indicator.

G.1.2. Required Entry Conditions

The following entry conditions are required:

- Register 13 must contain the save area address.
- Register 14 must contain the return address.
- Register 15 must contain the entry point address.
- If parameters are passed, register 1 must contain the address of a word-aligned parameter list. The compiler-generated program requires that the actual arguments specified in the parameter list conform in type and number with the dummy arguments. Each word in the parameter list contains the address of the actual argument. If the dummy argument is:
 - a simple variable, the parameter list contains the address of the actual value being passed;
 - an array name, the parameter list contains the address of the first element in the array;
 - a subprogram name, the parameter list contains the address of a word containing the address of the subprogram's entry point.

G.1.3. Exit Conditions

When a FORTRAN subprogram returns to the calling program, registers 2 through 14 are restored to their original contents and the contents of all call-by-value actual arguments are set to the value of the corresponding local dummy argument. If a subroutine is exiting, register 15 contains the K value of the RETURN K statement. A simple RETURN is equivalent to RETURN 0.

A function returns its value in a register depending on its type. The function types and corresponding registers are illustrated in Table G-2.

Table G-2. Function Types and Corresponding Registers

Function Type	Register Containing Value
{ INTEGER*2 } { INTEGER[*4] }	General register 0
{ REAL[*4] } { REAL*8,DOUBLE PRECISION }	Floating point register 0
{ COMPLEX[*8] } { COMPLEX*16 }	Real part in floating register 0 Imaginary part in floating register 2
{ LOGICAL*1 } { LOGICAL[*4] }	General register 0

NOTE:

Registers 0, 1, and 15 and all floating-point registers are not preserved over a subprogram reference.

G.1.4. Mathematical Library

The mathematical functions supplied by Extended FORTRAN are available to programs written in other languages. Tables 5-3 and 5-4 specify the functions available and Appendix F lists the actual modules containing these functions.

The mathematical library is entirely self-contained except for one external reference. If an error condition is possible, the function uses the first word of FL\$IOARA (I/O area) to get to an error control routine. The error routine FL\$ERR is required for Extended FORTRAN. If a FORTRAN-compiled program is also included in the executable module, FORTRAN automatically supplies the common area, FL\$IOARA. However, if FORTRAN-compiled subprograms are not included in the load module, the user must supply the FL\$IOARA module. An assembly language routine to accomplish this follows:

1	LABEL	ΔOPERATIONΔ	OPERAND	Δ
		10	16	
	FL\$IOARA	START	0	
		DC	A(FORTEERR)	
	FORTEERR	DIS	OH	
		USING	FORTEERR,15	
		DUMP		
		END		

More complicated routines may be substituted when required. However, the first word at FL\$IOARA must be an address constant containing the address of the processing routine.

G.1.5. Compiled Subprograms

Other language programs may use FORTRAN-compiled subprograms. FORTRAN subprograms assume availability of a complete FORTRAN run-time library for support. However, if only a subprogram is used and the FORTRAN library support is not available, then OVERFL, DVCHK, and orderly termination on fatal errors are not normally supported.

To use the complete FORTRAN run-time library, the PARAM option STX=YES is available. A subprogram compiled with this option calls the FL\$INITL routine to provide full support.

The FL\$INITL routine uses both the program check and abnormal termination island code services of the OS/3 supervisor. In addition, the program mask bits in the PSW are set to allow exponent overflow and underflow interrupts. A complete FORTRAN I/O environment is required for printing diagnostic information.

A subprogram may use FORTRAN I/O to process data files. However, a FORTRAN STOP statement must be used to terminate job step processing. An ENDFILE statement should be used for any active sequential I/O unit before the final exit from the FORTRAN routine. The ENDFILE statement ensures that the file is closed by data management with all I/O activity completed.

G.2. CALLING FROM FORTRAN PROGRAMS

When a FORTRAN-compiled program references a subprogram:

- register 1 contains the address of a parameter list;
- register 13 contains the address of an 18-word save area;
- register 14 contains the return address; and
- register 15 contains the subprogram's entry address.

The four bytes at the address in register 14 is a NOP with the FORTRAN source line number in hexadecimal as the second half word. An equivalent assembly language calling sequence would be as follows:

1	LABEL	△OPERATION△ 10	16	OPERAND	△
		LIA		R13, SAVEAREA	
		LIA		R1, PARMLST	
		L		R15, =V(PROGNAME)	
		BALR		R14, R15	
		NOP		X'(1,linenumber)'	

G.2.1. Parameter List Formats

If the subprogram reference has an actual argument list, register 1 contains the address of the parameter list. The parameter list is a sequence of words containing the addresses of the actual arguments. The last word in the parameter list is identified with bit 0 of the first byte of the word set to 1.

If the actual argument is a variable, array element reference, or constant, the parameter list points to the appropriate location containing the value. An actual argument that is an array name is equivalent to passing the first element in the array. Label arguments (G.2.2) are not passed in the parameter list.

NOTE:

Logical constants are always passed as LOGICAL*4 values and integer constants are always passed as INTEGER*4 values.

G.2.2. Label Arguments

Labels may be passed in an actual argument list in a CALL statement. When labels occur, they are not explicitly passed in the parameter list, but immediately following the CALL statement, they are converted to a form similar to a computed GOTO. Upon return, the compiler expects register 15 to contain a value indicating how to process the labels. For n labels, if register 15 contains a value i with $1 \leq i \leq n$, control passes to the statement at the ith label. Any other values in register 15 cause control to pass to the next sequential statement.

G.2.3. Conventions

A FORTRAN-compiled program assumes that registers 2 through 14 are not modified during a subprogram reference. Register 13 contains a save area address for the called subprogram to save any needed registers. The called subprograms should conform to the standard usage of this save area through normal linkage conventions. Words 1 and 2 must not be modified; they contain required FORTRAN system information. Registers 0, 1, 15, and the four floating-point registers may be modified by the called program.

If the subprogram is a function, it must return a value. The location of this value is specified in Table G-2.

G.3. TRACEBACK INTERFACE

When the FORTRAN run-time system prints a diagnostic, a traceback of the current subprogram linkage is attempted. Beginning with the current save area, indicated by register 13, the traceback routine uses the backward link, word 2, of each save area to determine the sequence of calls and then prints this information. Observing the following conventions will avoid any possible problems with the traceback routines.

1. During subprogram execution, point register 13 to a local save area. This ensures a correct beginning for the traceback.
2. Fill the backward link address, word 2, in every save area with the appropriate address. The main program must have a zero in this field.
3. The traceback routine assumes the module name is located 72 bytes from the beginning of a save area. The form is a 1-byte length, followed by one to eight bytes containing the module name. It would appear as follows in assembly language:

1	LABEL	Δ OPERATION Δ	OPERAND	Δ
		10	16	
	MYMDDI	START		
		:		
		:		
	SAVE AREA	DC	18F'0'	72 bytes
		DC	X'6'	length
		DC	CL5'MYMDD'	name
		:		
		:		



4. The traceback routine assumes the entry name is located four bytes after the entry point. The form is a 1-byte length, followed by one to six bytes containing the entry name. In assembly language, the normal entry point is coded as follows:

1	LABEL	ΔOPERATIONΔ	OPERAND	Δ
		10	16	
	MYMODI	START		
		:		
		:		
		ENTRY	MYENT	
		USING	MYENT, R15	
	MYENT	B	*+6	
		DC	X'5'	
		DC	CL5'MYENT'	

5. The traceback routine assumes that a half-word line number is located two bytes after the return point (specified in register 14). Refer to G.2 for an assembly language example of this.



Index

Term	Reference	Page	Term	Reference	Page
A					
ABNORMAL statement	5.4.1.3	5-7	ASSIGN statement	3.3.2	3-4
Argument substitution			Assigned GO TO statement	4.6	4-4
call by name	5.5.2	5-13	Assignment statements		
call by value	5.5.1	5-13	arithmetic and logical	3.3.1	3-5
description	5.5	5-12	conversion	Table 3-3	3-5
symbolic	5.5.3	5-14	description	3.3	3-4
Arguments			AT statement	10.4.1	10-3
compiler	9.2.1	9-2	B		
description	5.1	5-2	BACKSPACE auxiliary I/O statement	7.3.6.2	7-21
forms	Table 5-2	5-2	Binary arguments		
UNIT	See UNIT arguments.		card input	C.3	C-2
Arithmetic assignment statements	3.3.1	3-5	card output	C.4	C-3
Arithmetic expressions	3.2.1	3-1	Blank descriptor	7.3.3.1.11	7-12
	3.2.6	3-3	BLOCK DATA statement	8.3.1	8-3
Arithmetic IF statement	4.1	4-1	Block sizes	11.3.1.2	11-3
Arithmetic, mixed mode	3.2.5	3-3	Buffer allocation	11.3.1.4	11-4
Arithmetic operations			C		
implementation	3.2.7	3-4	Call by name, argument substitution	5.5.2	5-13
user checks	3.2.6	3-3	Call by value, argument substitution	5.5.1	5-13
Arithmetic underflow and overflow	5.6.3	5-23	CALL statement		
Arrays			description	5.2.2	5-3
declaration	6.2	6-1	standard library subroutines	5.6.3	5-23
declarator	6.2.1	6-1			
description	2.4	2-6			
element position location	2.4.2	2-7			
element reference	2.4.1	2-6			
ASCII character set arguments	C.3	C-2			
	C.4	C-3			

Term	Reference	Page	Term	Reference	Page
Calling from FORTRAN programs			Complex constants	2.2.5	2—4
conventions	G.2.3	G—5	Computed GO TO statement	4.5	4—3
description	G.2	G—4	Conditional compilation	10.2	10—1
label arguments	G.2.2	G—5	Configurations		
parameter list formats	G.2.1	G—4	file definition	11.3.1	11—2
Card input files			programmer-defined	11.3	11—2
arguments	Table B—3	B—2	supplied	11.2	11—1
data management	11.3.4.2.2	11—12	Constants		
description	11.3.4.2	11—10	complex	2.2.5	2—4
spooled	11.3.4.2.1	11—10	double precision	2.2.3	2—3
	Table B—2	B—2	hexadecimal	2.2.4	2—3
Card output files			integer	2.2.1	2—1
arguments	Table B—4	B—3	literal	2.2.7	2—5
definition	11.3.4.3	11—16	logical	2.2.6	2—4
Card punch options	C.4	C—3	real	2.2.2	2—2
Card reader options	C.3	C—3	CONTINUE statement	4.8	4—6
Carriage control conventions	7.3.3.3	7—13	Control information check	5.6.3	5—27
	Table 7—2	7—14	Control statements		
Character set			CONTINUE statement	4.8	4—6
description	1.2.1	1—2	DO loops	4.7	4—4
EBCDIC	Table A—1	A—1	END statement	4.11	4—7
printer graphics	A.2	A—3	GO TO statement, assigned	4.6	4—4
source program and input data	A.1	A—1	GO TO statement, computed	4.5	4—3
Coding form	1.2.2	1—4	GO TO statement, unconditional	4.4	4—3
Collection, program	Section 12		IF statement, arithmetic	4.2	4—1
Comments	1.2.3	1—4	IF statement, logical	4.3	4—2
COMMON statement			PAUSE statement	4.10	4—7
description	6.6	6—6	STOP statement	4.9	4—6
interaction with EQUIVALENCE statement	6.6.1	6—7			
Compatibility	1.1.1	1—2	D		
Compilation			D format	7.3.3.1.4	7—10
arguments	9.2.1	9—2	Data initialization		
conditional	10.2	10—1	BLOCK DATA statement	8.3.1	8—3
description	9.1	9—1	block data subprogram	8.3	8—3
PARAM options	D.5	D—14	DATA statement	8.2	8—1
PARAM statement	9.2	9—1	Data management		
source correction facility	9.4	9—6	additional devices	C.8	C—5
stacked	9.3	9—5	card input file definition	11.3.4.2	11—10
Compile-assemble-link-execute, sample	D.4	D—11	interface	11.1	11—1
Compile-link-execute, sample	D.2	D—9	DATA statement	8.2	8—1
Compile-time diagnostics	Table E—1	E—2	Data types		
Compiled subprograms	G.1.5	G—4	description	2.1	2—1
			constants	2.2	2—1
			variables	2.3	2—5
			arrays	2.4	2—6

Term	Reference	Page	Term	Reference	Page
Error environment definition procedure &ERRDEF?	11.3.6	11—39	double precision	7.3.3.1.4	7—10
Error indicator set subroutine	5.6.3	5—25	general	7.3.3.1.6	7—10
Error indicator test subroutine	5.6.3	5—25	hexadecimal	7.3.3.1.9	7—11
ERROR subroutine call statement	5.6.3	5—25	Hollerith &A conversion?	7.3.3.1.7	7—10
ERROR1 subroutine call statement	5.6.3	5—25	Hollerith &H conversion?	7.3.3.1.8	7—10
Evaluation order, expressions	3.2.4 Table 3—1	3—2 3—3	integer	7.3.3.1.1	7—8
Execution environment			literal	7.3.3.1.10	7—11
configurators supplied	11.2	11—1	logical	7.3.3.1.5	7—10
data management interface	11.1	11—1	real &E conversion?	7.3.3.1.2	7—9
error environment definition	11.3.6	11—39	real &F conversion?	7.3.3.1.3	7—9
file definition conventions	11.3.1	11—2	record position	7.3.3.1.12	7—12
programmer-defined configurations	11.3	11—2	File definition		
START statement initialization &FUNTAB?	11.3.2 11.3.3	11—6 11—6	card output	11.3.4.3	11—16
unit definition	11.3.4	11—6	data management card input	11.3.4.2.2	11—12
unit definition termination &FUNEND?	11.3.5	11—39	direct access disc	11.3.4.6	11—32
Exit conditions, subprogram	G.1.3	G—2	printer	11.3.4.1	11—7
EXIT subroutine	5.6.3	5—27	sequential disc	11.3.4.5	11—26
Explicit type statement	6.4.1	6—3	spooled card input	11.3.4.2.1	11—10
Expressions			tape	11.3.4.4	11—19
arithmetic	3.2.1	3—1	File definition conventions		
evaluation order	3.2.4 Table 3—1	3—2 3—3	buffer allocation	11.3.1.4	11—4
logical	3.2.3	3—2	device type	11.3.1.1	11—3
relational	3.2.2	3—1	file type	11.3.1.5	11—5
Extensions	1.1.2	1—2	record and block sizes	11.3.1.2	11—3
External functions			record formats	11.3.1.3	11—3
ABNORMAL statement	5.4.1.3	5—7	File type	11.3.1.5	11—5
description	5.4.1	5—6	FIND statement	7.4.4	7—27
FUNCTION statement	5.4.1.1	5—6	FOR call statement	D.1	D—1
RETURN statement	5.4.1.2	5—7	FORMAT statement		
EXTERNAL statement	6.7	6—8	carriage control conventions	7.3.3.3	7—13
			description	Table 7—2	7—14
			field descriptors	7.3.3	7—6
			interaction with I/O list	7.3.3.1	7—7
			multiple record format specification	Table 7—1	7—7
			permissible associations of list items	7.3.3.4	7—14
			Formatted READ/WRITE statements	7.3.3.2	7—13
			FRECSIZE	Table 7—3	7—15
			Function reference	7.3.2	7—4
			FUNCTION statement	11.3.4.1	11—7
			Function subprograms, multiple entry	5.2.1	5—3
				5.4.1.1	5—6
				5.4.3	5—11
FETCH subroutine call statement	5.6.3	5—28			
Field descriptors, FORMAT statement					
blank	7.3.3.1.11	7—12			
description	7.3.3.1	7—7			
	Table 7—1	7—7			

F

Term	Reference	Page	Term	Reference	Page
Functions			list	7.2	7-1
description	5.1	5-2	list-directed	7.3.5	7-16
external	5.4.1	5-6	sequential files	7.3	7-2
intrinsic	5.6.1	5-15		7.3.7	7-22
	Table 5-3	5-15	statements	See I/O statements.	
standard library	5.6.2	5-17	Integer constants	2.2.1	2-1
	Table 5-4	5-18	Integer descriptor	7.3.3.1.1	7-8
statement, definition	5.3	5-4	Intrinsic functions	5.6.1	5-15
FUNEND unit definition termination procedure	11.3.5	11-39	Table 5-3	5-15	
FUNTAB initialization procedure	11.3.3	11-6	I/O list, format interaction	7.3.3.4	7-14
			Table 7-3	7-15	
			I/O statements		
G			BACKSPACE	7.3.6.2	7-21
General field descriptor (G)	7.3.3.1.6	7-10	compatibility	7.3.2.1	7-5
GO TO statement			DEFINE FILE	7.4.1	7-23
assigned	4.6	4-4	disc FIND	7.4.4	7-27
computed	4.5	4-3	disc READ	7.4.2	7-24
unconditional	4.4	4-3	disc WRITE	7.4.3	7-26
Graphics, printer	A.2	A-3	ENDFILE	7.3.6.3	7-21
			FORMAT	7.3.3	7-6
			NAMELIST	7.3.5.1	7-17
			READ	7.3.1	7-3
				7.3.2	7-4
				7.3.4	7-15
			REWIND	7.3.6.1	7-20
			WRITE	7.3.1	7-3
				7.3.2	7-4
			I/O unit module	12.3.1	12-3
			J		
			Job control procedure	D.1	D-1
			Job control stream	9.5	9-6
			Job stream examples	Appendix D	
			L		
			Label arguments	G.2.1	G-4
			Library procedures		
			description	5.6	5-14
			intrinsic functions	5.6.1	5-15
				Table 5-3	5-15
			standard library functions	5.6.2	5-17
				Table 5-4	5-18
			standard library subroutines	5.6.3	5-23
				Table 5-5	5-29
i					
IF statement					
arithmetic	4.2	4-1			
logical	4.3	4-2			
IMPLICIT statement	6.4.2	6-4			
Indicator setting subroutine (SLITE)	5.6.3	5-26			
Indicator testing subroutine (SLITET)	5.6.3	5-27			
Initialization procedure (FUNTAB)	11.3.3	11-6			
Input files, card	See card input files.				
Input/output					
description	7.1	7-1			
direct access files	7.4	7-23			

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.)

(Revision No.)

(Update No.)

Comments:

Cut along line.

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424



CUT

FOLD

Comments concerning this manual may be made in the space provided below. Please fill in the requested information.

System: _____

Manual Title: _____

UP No: _____ Revision No: _____ Update: _____

Name of User: _____

Address of User: _____

Comments:

CUT


FOLD

FIRST CLASS
PERMIT NO. 21
BLUE BELL, PA.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

SPERRY  **UNIVAC**

P.O. BOX 500
BLUE BELL, PA.
19422

ATTN: SYSTEMS PUBLICATIONS DEPT.

CUT

FOLD