SPERRY UNIVAC COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

FORTRAN IV

Programmer Reference

UP-8474 Rev. 2

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) FORTRAN IV Programmer Reference", UP-8474 Rev. 2.

This manual now covers FORTRAN IV implemented on a 90/25, 90/30, 90/30B, or 90/40 system operating in a define the file (DTF), consolidated data management (CDM), or mixed mode environment.

Workstation support is also provided.

Other technical changes were made.

Additional copies may be ordered by your local Sperry Univac representative.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS:
Mailing Lists BZ, CZ (less DE, GZ, HA) MZ, 18U, 19U, 20U, 21U, 75U and 76U	Mailing Lists DE, GZ, HA, 18, 19, 20, 21, 75 and 76 (Covers and 289 pages)	Library Memo
		RELEASE DATE: October, 1980

SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

FORTRAN IV

Programmer Reference

UP-8474 Rev. 2-A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN IV Programmer Reference", UP-8474 Rev. 2.

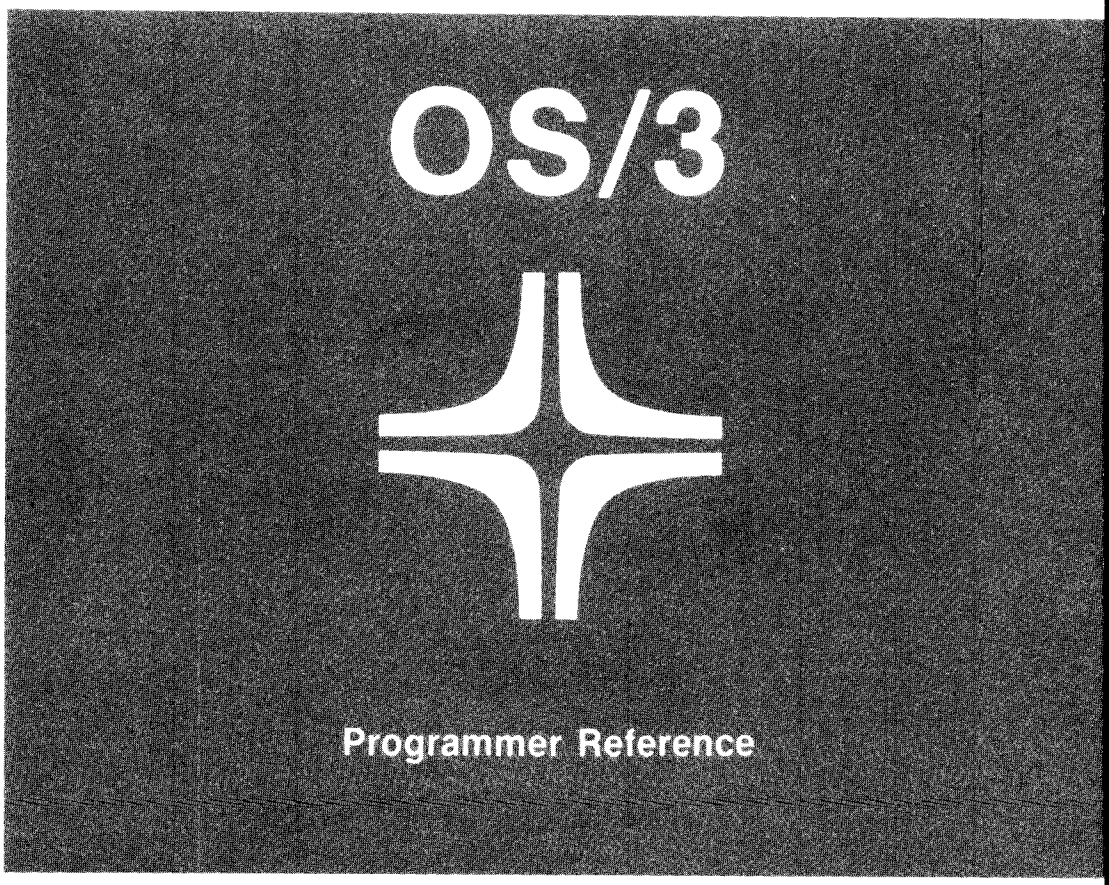
This update includes changes to the job control procedure for release 7.1:

- Specification of catalog files
- Expanded explanation of parameters

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-8474 Rev. 2-A. To receive the complete manual, order UP-8474 Rev. 2.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ (less DE, GZ, HA) MZ, 18U, 19U, 20U, 21U, 75U and 76U	Mailing Lists DE, GZ, HA, 18, 19, 20, 21, 75 and 76 (Package A to UP-8474 Rev. 2, 7 pages plus Memo)	Library Memo for UP-8474 Rev. 2-A
		RELEASE DATE: September, 1981

FORTRAN IV



Environment: 90/25, 30, 30B, 40 Systems

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.

©1980 — SPERRY CORPORATION

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: Update B – UP-8474 Rev. 2
RELEASE LEVEL: 8.0 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	B
Preface	1 2,3	Orig. B
Contents	1 thru 4 5 thru 7 8,9	Orig. B Orig.
PART 1	Title Page	Orig.
1	1 2 thru 9	B Orig.
2	1 thru 7	Orig.
PART 2	Title Page	Orig.
3	1 thru 6	Orig.
4	1 thru 6	Orig.
5	1 thru 28	Orig.
6	1,2 3 4 thru 8	Orig. B Orig.
7	1 thru 24 25,26 27	Orig. B Orig.
8	1 thru 3	Orig.
PART 3	Title Page	Orig.
9	1 2 3 4 4a 5	Orig. B Orig. B B* Orig.
10	1 thru 4	Orig.
11	1 2 thru 4 5 6 thru 15 16 17,18 19,20 21 thru 32	B Orig. B Orig. B Orig. B Orig.
12	1 thru 46	Orig.
13	1 thru 3 4 5	Orig. B Orig.
PART 4	Title Page	Orig.
Appendix A	1 thru 8	Orig.
Appendix B	1 2 thru 5	B Orig.
Appendix C	1 thru 7	Orig.
Appendix D	1 thru 5	Orig.
Appendix E	1,2 3 4 5 thru 7 8 thru 14	B A Orig. B Orig.
Appendix F	1 thru 15	Orig.
Appendix G	1 thru 14	Orig
Appendix H	1 thru 6	Orig.
Index	1 2 3 4,5 6 7 thru 10	Orig. B Orig. B Orig. B
User Comment Sheet		

*New pages

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 OS/3 FORTRAN IV. 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, UP-7536 (current version) also is 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 FORTRAN IV implementation for OS/3.

This manual is divided into the following parts:

- PART 1. FORTRAN IV PROGRAM STRUCTURE

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

- PART 2. FORTRAN STATEMENTS

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

- PART 3. COMPILE, DEBUG, AND EXECUTE PROCEDURES

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

- PART 4. APPENDIXES

  Provide additional information concerning:

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

Other current OS/3 publications, referenced in this manual, are useful to the programmer working with FORTRAN IV.

Consolidated Data Management (CDM) Environment

- System processor programmer reference, UP-8881

  Lists the hardware characteristics of integer/real arithmetic.

- Screen format services concepts and facilities, UP-8802

  Describes the screens and formats used on workstation terminals.

- Interactive services commands and facilities user guide/programmer reference, UP-8845

  Describes the commands and operating procedure for workstation terminals.

- Editor user guide, UP-8828

  Describes the interactive command language.

- System service programs (SSP) user guide, UP-8841

  Describes various system utilities (librarian and linkage editor among others).

- Consolidated data management macroinstructions user guide/programmer reference, UP-8826

  Describes the data management macroinstructions.

- System messages programmer/operator reference, UP-8076

  Lists and describes the system console messages issued during compilation by FORTRAN IV.

- Series 90 FORTRAN mathematical library programmer reference, UP-8029

  Lists the multiplication, division, and exponentiation library routines.

- Job control user guide, UP-8065

  Provides information on the format and usage of job control statements and linkage editor job control procedure call (jproc).

- Fundamentals of FORTRAN programmer reference, UP-7536

  Presents general information concerning FORTRAN.

Define the File (DTF) Environment

- Data management user guide, UP-8068

  Describes the data management macroinstructions.

- System service programs (SSP) user guide, UP-8062

  Describes various system utilities (librarian and linkage editor among others).

- 90/30 system processor programmer reference, UP-8052

  Lists the hardware characteristics of integer/real arithmetic.

- System messages programmer/operator reference, UP-8076

  Lists and describes the system console messages issued during compilation by FORTRAN IV.

- Series 90 FORTRAN mathematical library programmer reference, UP-8029

  Lists the multiplication, division, and exponentiation library routines.

- Job control user guide, UP-8065

  Provides information on the format and usage of job control statements and linkage editor job control procedure call (jproc).

- Fundamentals of FORTRAN programmer reference, UP-7536

  Presents general information concerning FORTRAN.

Contents

2. DATA TYPES

PART 2. FORTRAN STATEMENTS

3. EXPRESSIONS AND ASSIGNMENT STATEMENTS

4. CONTROL STATEMENTS

6. SPECIFICATION STATEMENTS

7. INPUT AND OUTPUT

TABLES

PART 1. FORTRAN IV PROGRAM STRUCTURE

PART I. FORTRAN IV PROGRAM STRUCTURE

1. Introduction

1.1. SCOPE AND PURPOSE

This Sperry Univac document describes FORTRAN IV operating in two environments: (1) the System 80 consolidated data management environment, and (2) the 90/30 define the file interface environment. When operating in the System 80 consolidated data management (CDM) environment, three changes in the run time procedure are required; when in the 90/30 define the file (DTF) interface environment, no changes are required.

The three changes required when operating in the System 80 CDM environment are:

- A new UNIT definition facility is required to define the devices and file characteristics.
- New devices to be supported must be defined.
- A new data access method is required. Called the consolidated access method (CAM), it is an enhancement of the OS/3 multiple indexed random access method (MIRAM).

The mathematical and I/O libraries used in System 80 CDM environment are identical with those supported in the 90/30 DTF environment. All statements are identical except that a new keyword SCREEN is added to the I/O statements to provide support for the workstation terminal.

Regardless of the operating environment, FORTRAN IV consists of the following components:

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

The FORTRAN IV 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 call the compiler, allocate scratch files, and perform other functions necessary for successful compilation. 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 FORTRAN library and included in the executable program.

User-defined procedures, if they are required, also are included during the link-edit. These procedures are coded in FORTRAN or in some other language (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.

During compilation, the compiler produces the following listings:

- A listing of the source program. For each diagnostic, the source statement is marked at the character for which the diagnostic is produced.
- An error listing that contains the diagnostic messages and associated severity codes. (See Appendix F.)
- A main storage map showing the addresses allocated to the variables and arrays in the program. An alphabetical and address sorted listing is optionally available.

Any of the listings may be suppressed by user options.

The FORTRAN IV compiler is self-initializing and up to 100 FORTRAN source programs can 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 FORTRAN IV language includes the American National Standard FORTRAN X3.9-1966 and the IBM System/360/370 DOS FORTRAN IV languages as subsets. Programs that conform to either of these specifications are accepted without change. FORTRAN IV is also highly compatible with SPERRY UNIVAC Series 70 FORTRAN.

1.1.2. Extensions

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

- Subscript expressions are integer or real arithmetic expressions (2.4.1).
- Arithmetic assignment statements are 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, DUMP, PDUMP, EXIT, FETCH, LOAD, and OPSYS (5.6.3).
- Arrays may have a maximum of seven dimensions (6.2.1).
- Dimension declarator subscripts are permitted in common storage (6.2.1).
- 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 is extended to cover integer and logical data (7.3.3.1.6).
- Z and T 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).
- Special I/O statements are provided to access the workstation. READ and WRITE commands to the workstation may be done as a unit record or full screen I/O (7.3.1) in a CDI environment.
- The specification of hexadecimal constants in DATA statements is permitted (8.2).
- The TRACE ON and TRACE OFF statements are provided (10.6).

The FORTRAN IV language also includes several extensions to IBM System/360/370 FORTRAN IV including:

- Embedded comments (1.2.3).
- Extended exponentiation (3.2).
- Optional statement labels on arithmetic IF statements (4.2).
- Logical IF, PAUSE, and STOP statements can be terminal statements of DO loops (4.7).
- An ABNORMAL statement is provided for optimal code generation (5.4.1.3).
- The mathematical library can be referenced by generic names (5.6).
- The ability to initialize variables and arrays in type statements (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 are provided (8.2).
- The BLOCK DATA statement contains an optional name for the subprogram (8.3.1).
- Blocked and buffered input/output is provided (Sections 11 and 12).
- Extended error recovery procedures are provided for the mathematical library (H.1.4).

1.2. SOURCE PROGRAMS

General procedures to be followed in FORTRAN programming are presented in subsections 1.2.1 through 1.4.

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 FORTRAN character set, but these characters can be stored internally and transmitted to and from card, tape, and disk storage.

Table 1—1. FORTRAN Character Set

FORTRAN character set	Alphabetics	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 extended character set can change with the options selected for the system printer (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. Columns 1 through 72 are 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.

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, although a warning diagnostic is produced. 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 (10.2).

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.

1.2.3. Comments

The compiler provides three methods of entering comments: columns 73–80 on any line, the comment line, and embedded comments. A comment line is indicated by the character C or * 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:

```
1       7
        R=SQRT(A); CALCULATE SQUARE ROOT
```

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

```
1       7
        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 EXCHANGES THE VALUES
       1SWAP      ; OF TWO REAL VARIABLES
       2(A,B)
```

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).

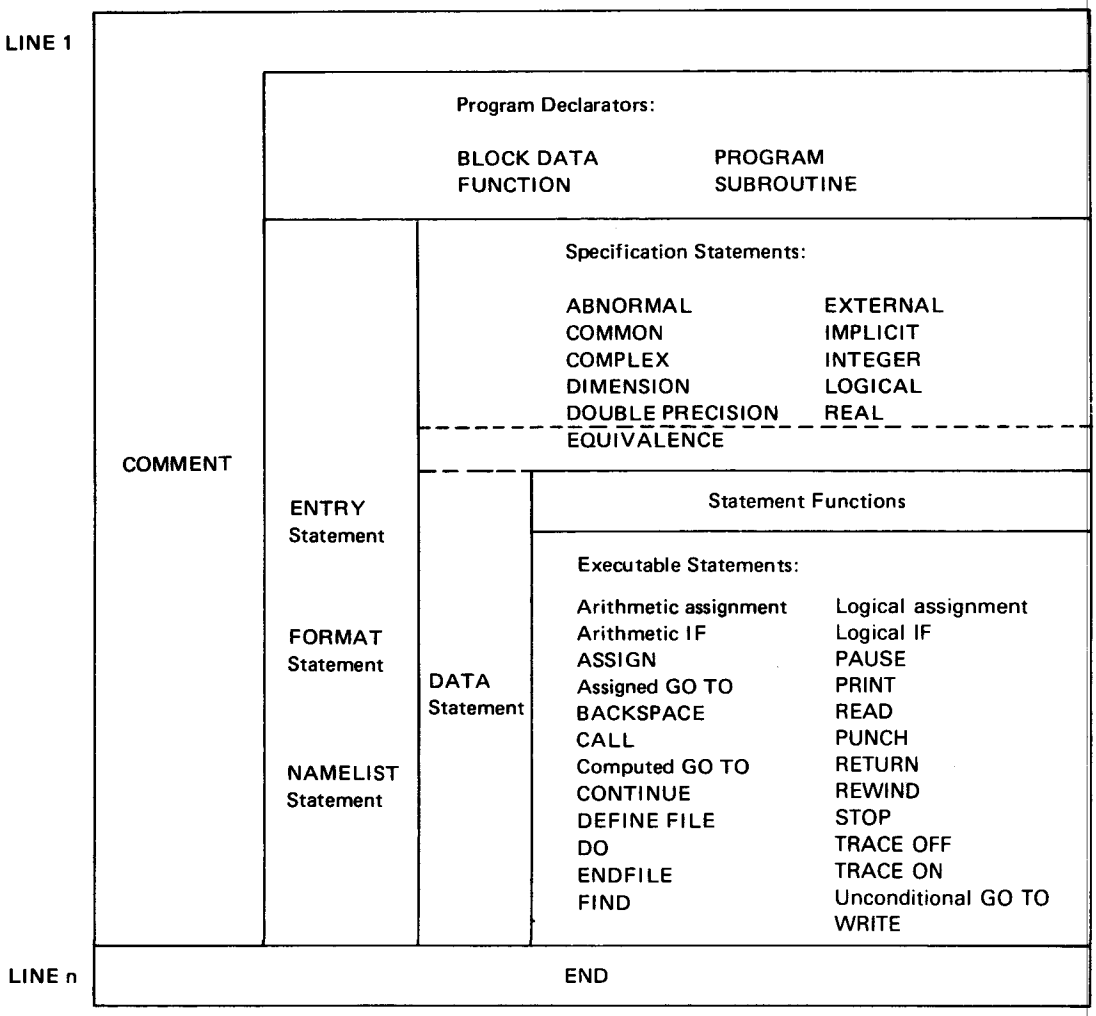
1.2.5. Source Statement Order

Figure 1-1 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.

A maximum of 100 FORTRAN program units may be processed by one compiler call. All program units are terminated with an END statement. The first statement of a main program may optionally be a PROGRAM statement.



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. The dotted horizontal lines indicate that EQUIVALENCE statements must follow any specification statements that specify items to share storage; DATA statements must follow any specification statements that reference items to be initialized.

Figure 1—1. Source Statement Order

1.3. SOURCE CODE GUIDELINES

The FORTRAN IV compiler performs most efficiently when the source statements are in the following order:

- IMPLICIT
- ABNORMAL
- EXTERNAL
- type statements
  - INTEGER
  - REAL
  - COMPLEX
  - LOGICAL
  - DOUBLE PRECISION
- DIMENSION
- COMMON
- EQUIVALENCE
- DATA statements
- Executable statements

The message:

FF850 PROGRAM REORDERED

indicates that efficiency has been lost.

Certain illegal FORTRAN DO loop constructions produce incorrect object-time results without any compile-time diagnostics. These include:

- Use of the induction variable after satisfaction of its DO loop.
- Changing the induction variable, increment, or maximum value during a DO loop.
- Branching into the range of a DO loop without having previously branched out of it.

These constructions are forbidden (see 4.7), although no explicit compile-time diagnostic is provided.

Some coding sequences or habits give rise to more efficient object code, such as the following:

- Use simple variables (not in COMMON or EQUIVALENCE) for frequently used induction variables and subscripts.
- Significantly faster code is generated for short DO loops that contain no branches outside the loop in which the induction variable is used only in subscripts.

- The best code is generated for subscripts in the form c*i±k, where i is an integer variable and c and k are optional integer constants.
- When a generalized subscript contains a constant term, write it at the righthand end (i.e., in the form i±k, where i is an integer expression and k is an integer constant).
- Use logical IF statements instead of arithmetic IF statements for conditional branching. The term IF (A.EQ.B) GO TO 10 is superior to IF (A–B) 11, 10, 11 followed by 11 CONTINUE.
- When possible, avoid the use of an EQUIVALENCE for any variable in a freqently used COMMON.
- When possible in multidimension arrays, use dimensions that equal exact powers of 2 as the leftmost dimension; i.e., an array (2, 10) is prefereable to an array (10, 2).

If the programmer writes logical expressions so that the truth or falsity of the expression can be determined early in left-to-right reading of the expression, time can be saved. For example, if L is usually false, then write

```
L .AND. A+B .GT. 25
```

instead of:

```
A+B .GT. 25 .AND. L
```

Although mixed arithmetic is a convenient feature of the FORTRAN language, too great a use of it may significantly slow down a program. For example, if a fixed-point variable occurs in many mixed expressions, you should create a floating-point variable that has the same value. This is easily done with an assignment statement of the form:

```
AI=I
```

1.4. STATEMENT CONVENTIONS

Conventions used to illustrate FORTRAN statements in Sections 1 through 9 are presented throughout these sections. Conventions for illustrating statements in assembler language in Sections 10, 11, and 12 and Appendixes D and E 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:

```
[FRECSIZE=(k                          )]
          {80 ; if FMODE=STD          }
          (160 ; if FMODE=BINARY      )
```

- 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 FORTRAN IV 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 programmer reference. Data types are categorized by their manipulation with 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 (2.2, 2.3, and 2.4, respectively). Additional information on the hardware characteristics of integer and real arithmetic is included in the discussion of the arithmetic section in the system processor programmer reference.

2.2. CONSTANTS

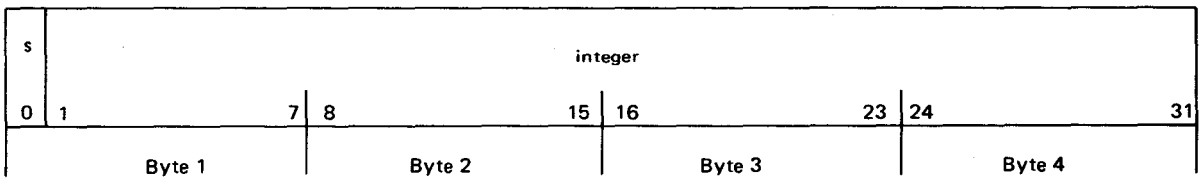
A constant is an arithmetic, logical, or literal value defined by its representation in the souce program. Once defined, a constant must not be redefined during program execution. An arithmetic constant is said to be signed if it is written with a plus or a minus sign, and an unsigned constant is treated 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:

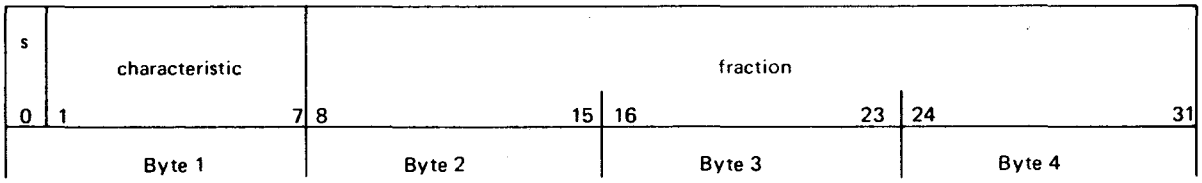
  170.1E-03

- 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.

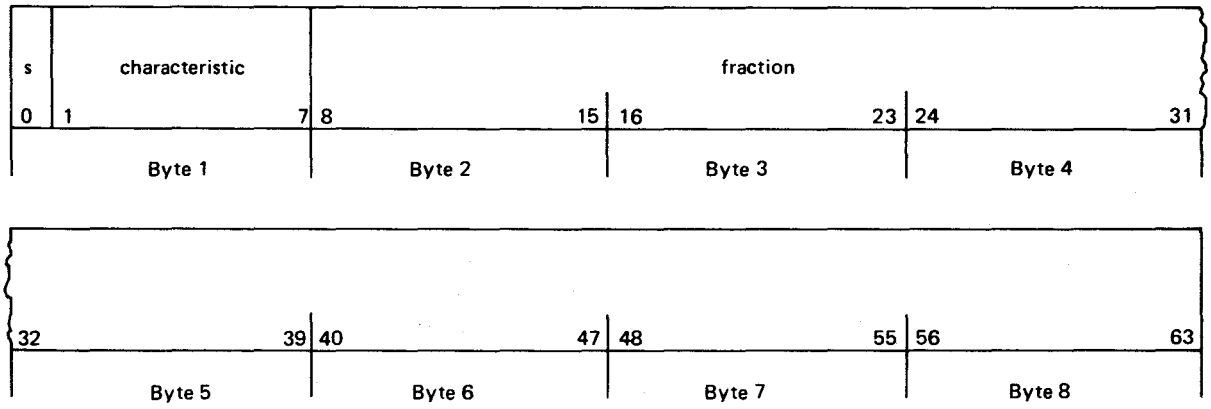
2.2.3. Double Precision Constants

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 may also be written as an optionally signed string of more than seven significant digits with a decimal point preceding, embedded in, or following the string.

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

(continued)

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 are 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 an internal representation of X'FF'.

2.2.7. Literal Constants

A literal constant consists of 1 to 255 characters from the FORTRAN character set (Table 1-1). Each character in the string requires one byte of storage. Two methods of writing literal constants are:

1. as a Hollerith constant in the form $wHc_1c_2...c_w$ where w is an unsigned integer constant and c represents a character; or

2. as a character string enclosed in apostrophes:$'c_1c_2...c_w'$ (if the apostrophe occurs in the string, it is represented by doubling that character).

The literal DO NOT is represented by 'DO NOT' or 6HDO NOT. The literal DON'T is 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 FORTRAN IV there is both a standard and an optional length specification that determines the number of bytes assigned in main storage (Table 2-1).

Table 2—1. Data Types and Optional Lengths

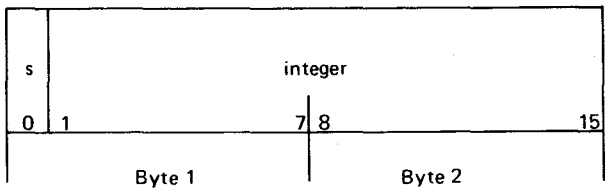
FORTRAN Name	Standard Data Type	Length in Bytes	Optional Data Type	Length in Bytes
Integer	Integer*4	4	Integer*2	2
Real	Real*4	4	Real*8	8
Double precision	Double precision	8	None	
Complex	Complex*8	8	Complex*16	16
Logical	Logical*4	4	Logical*1	1

The data 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.

To prevent confusion where the length can differ, the complete data type appears in this document; a reference to 16-byte complex data appears 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.

```
1       7
        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 zero; .TRUE. is nonzero and is usually X'FF'.

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 FORTRAN IV, 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 $(s_1, s_2, \ldots, s_n)$

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 this 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)+ ...+A*B*C*D*E*F*(g-1))*m

Examples:

If an array declarator is AGO(17), if the element referenced is AGO(4), and if the array is 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 is declared as AGO(9,10,11) and the element to be located is AGO(3,4,5), then the calculation is ((2)+9*(3)+9*10*(4))*4, or location 1556.

PART 2. FORTRAN STATEMENTS

PART 2. FORTRAN STATEMENTS

3. Expressions and Assignment Statements

3.1. GENERAL

This section discusses the use of expressions in FORTRAN IV programming and describes the assignment statements. For additional information, refer to the "FORTRAN Expressions" and "Assignment Statements" sections of the fundamentals of FORTRAN programmer reference.

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 3.2.1 through 3.2.7.

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 a .TRUE. or .FALSE. statement. 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 with the lowest 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 .NOT. operator; 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 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 these listed rules, the order in which operations are performed may be slightly affected by optimization. For example:

- Logical expressions are not always completely evaluated; once the value is known, evaluation ceases. Thus, for

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

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

Table 3—1. FORTRAN IV 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)

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 FORTRAN IV 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 Series 90 FORTRAN mathematical library programmer reference.

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 (described in 3.3.1) 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 expression for an arithmetic assignment statement or a logical expression for logical assignment statements.

e

Is any type of arithmetic expression for an arithmetic assignment statement or 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 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 low order three bytes of e are ignored. The conversions are accomplished by intrinsic functions (5.6.1).

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.

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. 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.

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.

***e is treated as an integer*4.

NOTE:

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

4. Control Statements

4.1. GENERAL

Control statements are executable statements that modify the normal sequence of program execution. The control statements used in FORTRAN IV are identical in function with those described in the "Control Statements" section of the fundamentals of FORTRAN programmer reference.

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 k_1 statement label; if the value is zero, k_2 receives control; and if the value is positive, k_3 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:

```
1    5  7
-----------------------
     5  IF(I-1)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:

```
1    5 7
       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,...,kn),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:

```
1      7
       GO TO (15,25,35,45), ITEM
```

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,(k1,k2,...,kn)
```

where:

i
: Is the name of a 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 the 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:

```
1       7
        GO TO K5, (10,13,15,17,18,21)
```

When the current value of the integer variable K5 is associated with one of the statement labels in parentheses, control is transferred to the statement with that label. The value of the integer variable can be 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 statement label associated with the value of the integer variable K5.

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.

m_1

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 m_2.

m_2

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 m_2, the DO control statement is satisfied and control passes out of the DO range.

m_3

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 m_2 and the control variable i. When this parameter is omitted, 1 is assumed to be the increment value.

Description:

The 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 action of the DO control block. A control statement can prevent the execution of the DO control block; for example, the loop

```
1    5 7
       DO 10 I=1,10
         .
         .
         .
    10 IF(A.GT.B)IF(C)20,30
```

may be executed 10 times, unless the condition

```
(A.GT.B).AND.(C.GT.0)
```

occurs, which prevents the execution of the DO control block and causes 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.

4.7.1. Transfers of Control to and from a DO Range

In FORTRAN IV 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 programmer reference.

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 Figure 1-1, 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 (13.3.3).

4.10. PAUSE

Formats:

```
PAUSE
PAUSE n
PAUSE '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 PAUSE control statement halts execution of the program and produces a display. 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 (5.4.1.2).

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 programmer reference, are 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 IV library.

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

Table 5—1. FORTRAN IV Procedures

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

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

where:

f
: Is the symbolic name used to identify the user-coded function in its function definition, or 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:

```
1      7
       CZ=CBRT(SUZU)+CARA+YAM
       MAICO=NORT**JAWA-INT(KS,ABL,R1)
```

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. This is because the function name is an integer variable name, and the value type returned by the statement function is determined by the function name.

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:

$$\texttt{CALL } s(a_1, a_2, \ldots, a_n)$$

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:

```
1       7
        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 using 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:

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

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.

Description:

The statement function definition statement defines a function that may be referenced in a subsequent expression. The statement function definition statement must precede all executable statements in the program unit and must follow all specification statements (Figure 1-1). Note that a reference to another function is permitted, but if such a reference is made to a statement function, that statement function must have been previously defined in the program unit. For example, the following Example 1 is permitted but Example 2 is not:

Example 1:

```
1       7
        QU (A) = 2.Ø*SQRT (A)
        AVR (A,B,PNT) = A**B+QU (PNT)
```

Example 2:

```
        AVR (A,B,PNT) = A**B+QU (PNT)
        QU (A) = 2.Ø*SQRT (A)
```

The type of the value returned by the statement function is determined by the statement function name according to the rules for variables described in 2.3, or it's type if specified by a type statement (6.4) in the same program. Note that it is the function name, not the type of the arguments or of the expression, which determines the value type returned by the statement function, and that the function name cannot be referenced from other subprograms.

Dummy argument names in the statement function definition may appear as variable names in the same program unit. A maximum of 255 arguments may appear in the definition.

A limited expression is an arithmetic or logical expression that cannot contain a reference to another statement function unless that function was previously defined. If the statement function being defined appears in an external function or a subroutine, the expression cannot contain a symbolic name identical with an entry name in the same subprogram.

5.4. SUBPROGRAM DEFINITION

Of the three user-coded procedures – statement functions, external functions, and subroutines – the latter two are coded as subprograms.

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, 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(a_1, a_2, . . . , 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 LOGICAL.

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 (1.2.4). 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:

```
1       7
        INTEGER FUNCTION XX1*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, in the same program unit, the type of the external function name YY1 is specified 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 FORTRAN IV 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 f1, f2, . . . , fn
```

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 permited are shown in Table 5–2. Multiply entry into a subroutine is permitted (5.4.3). Subroutines are always considered to be abnormal.

5.4.2.1. SUBROUTINE Statement

```
SUBROUTINE s(a1,a2,...,an)
```

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.

The optional method of returning control from an external subroutine requires the use of the label parameter specification (signaled by an ampersand) in the actual argument list of the CALL statement, the use of an asterisk in the corresponding dummy argument in the SUBROUTINE statement, and the integer specification of the RETURN statement. If integer = n, the statement RETURN n causes control to be returned to the statement in the main program labeled with the nth label parameter in the actual argument list of the CALL statement.

Examples:

```
1       7
        .
        .
        .
        CALL SUB(X,&100,Y,&110,&120)
        .
        .
        .
        .
        .
        .
        SUBROUTINE SUB(A,*,B,*,*)
        .
        .
        .
        RETURN 2
        .
        .
        .
        RETURN
        .
        .
        .
        RETURN 3
        .
        .
        .
        RETURN 1
        END
```

The subroutine SUB is entered when the CALL statement is executed. Control is returned to differing parts of the calling program, depending on which RETURN statement is executed in the procedure definition.

If the first RETURN statement is executed, control is returned to the statement labeled 110 in the calling program. This occurs because the integer option of the RETURN statement is used and the value of the integer is 2; control is returned to the second label parameter in the CALL statement, &110.

If the second RETURN statement is executed, control returns to the executable statement immediately following the CALL statement; if the third is executed, control goes to statement 120; if the fourth, to statement 100.

When the value of the integer is less than 1 or greater than the number of asterisks, control is returned to the statement following the CALL statement.

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,...,an)
```

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.

ENTRY statements in a FUNCTION subprogram must define functions of the same type as that defined by the FUNCTION statement. Regardless of which ENTRY statement is used, the value of the function is defined by the last executed assignment of a value to the name of the function given in the FUNCTION statement. ENTRY-point names are never used to return results to the calling program.

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. An entry name must not occur anywhere in the subprogram except in the ENTRY statement.
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. Adjustable dimension arrays that appear in an ENTRY statement must also appear in the FUNCTION or SUBROUTINE statement.
9. The type of the entry name must match that of the function name.
10. An ENTRY statement that contains a dummy argument must occur before the argument is used unless the argument also occurs in the FUNCTION or SUBROUTINE statement.

Example:

```
1       7
        FUNCTION ENT1(X,Y,Z,A)
                 .
                 .
                 .
        ENTRY ENT2 (X)
                 .
                 .
                 .
        ENT1=Y / A + Z ** X
        RETURN
                 .
                 .
                 .
        END
```

The function may be called with four variables by the name ENT1, or with one variable by the name ENT2.

5.5. ARGUMENT SUBSTITUTION

When a procedure is called, the actual arguments, if any, are substituted for the dummy arguments in the procedure receiving control. FORTRAN IV provides three methods of argument substitution:

- Call by value
- Call by name (or address)
- Symbolic substitution

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; 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, subrountine 1 on the left is treated as though it were written like subroutine 2 on the right.

Subroutine 1

```
SUBROUTINE A(B)
   .
   .
   .
   .
   .
   .
   .
   .
   .
ENTRY C(D)
   .
   .
   .
   .
   .
   .
   .
   .
   .
RETURN
```

Subroutine 2

```
          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
            .
            .
            .
          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

          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 substitutuion 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.

5.6. LIBRARY PROCEDURES

The three classes of procedures 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, they are provided 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).

FORTRAN IV 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, FORTRAN IV 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
CABS	Determine the absolute value of the argument	1	CABS ① CDABS ①	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
{MAX, MAX0}	Select the largest value	⩾2	JMAX0 ① AMAX0 ② AMAX1 ① {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 ①} MIN1 ② 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 percision
	Convert argument from real to integer	1	IFIX ② HFIX ②	Real*4 Real*4	Integer*4 Integer*2

NOTES:

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

② This function is accessible only through its member name.

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

Generic Name	Use	Number Arguments	Member Function Name	Member Argument Type	Member Function Type
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

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.

5.6.2.1. Specification Statement Interaction

This section describes the effects of listing the name of an intrinsic or standard library function in a type statement in a FORTRAN IV compilation. An ABNORMAL or EXTERNAL statement causes no special effect.

Normally, FORTRAN IV uses a name in the form ILF#xx to reference a standard library function (or an intrinsic function being used as a subroutine argument). If the name of the standard library function appears in a type statement, this function will not be treated as a standard library function. It will be called via its actual name, and no warning diagnostics for its argument will be generated.

A type statement is useful when calling a routine other than the standard library routine. For example, if a REAL SQRT statement appears in a program, the FORTRAN IV compiler generates calls to SQRT rather than ILF#30. This permits linkage to a user's SQRT function.

A list of standard function names and their corresponding ILF#xx names is included in Appendix G.

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 $1 \leq y \leq M$
		DCOSH		1	real•8	$\|x\| < 175.366$	real•8 $1 \leq y \leq M$

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}	$y = \log_e x$ 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	$y=PVlog_e(z)$ PV is principle value, which means y_2 between $-\pi$ and π.	1	complex•8	$z \neq 0+0i$	complex•8 $y=y_1+y_2i$ $y_1 \geqslant -180.218$ $y_1 \leqslant 175.021$ $-\pi \leqslant y_2 \leqslant \pi$
		CDLOG		1	complex•16	$z \neq 0+0i$	complex•16 $y=y_1+y_2i$ $y_1 \geqslant -180.218$ $y_1 \leqslant 175.021$ $-\pi \leqslant y_2 \leqslant \pi$
Common logarithm	{ALOG10 / LOG10}	{ALOG10 / LOG10}	$y=\log_{10}x$	1	real•4	$x > 0$	real•4 $y \geqslant -78.268$ $y \leqslant 75.859$
		DLOG10		1	real•8	$x > 0$	real•8 $y \geqslant -78.268$ $y \leqslant 75.859$
Square root	SQRT	SQRT	$y=\sqrt{x}$ or $y=x^{1/2}$	1	real•4	$x \geqslant 0$	real•4 $0 \leqslant y \leqslant M^{1/2}$
		DSQRT		1	real•8	$x \geqslant 0$	real•8 $0 \leqslant y \leqslant M^{1/2}$
		CSQRT	$y=\sqrt{z}$ or $y=z^{1/2}$	1	complex•8	any complex argument	complex•8 $0 \leqslant y_1 \leqslant 1.0987\ (M^{1/2})$ $\lvert y_2 \rvert \leqslant 1.0987\ (M^{1/2})$
		CDSQRT		1	complex•16	any complex argument	complex•16 $0 \leqslant y_1 \leqslant 1.0987\ (M^{1/2})$ $\lvert y_2 \rvert \leqslant 1.0987\ (M^{1/2})$
Cube root	CBRT	CBRT	$y=x^{1/3}$	1	real•4	any real argument	real•4 $-M^{1/3} \leqslant y \leqslant M^{1/3}$
		DCBRT		1	real•8	any real argument	real•8 $-M^{1/3} \leqslant y \leqslant M^{1/3}$

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	$y = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-u^2} du$ $y = 1 - erf(x)$	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	$y = \log_e \Gamma(x)$ or $y = \log_e \int_0^\infty u^{x-1} e^{-u} du$	1	real•4	$x > 0$ and $x < 4.2913 \quad 10^{73}$	real•4 $-0.12149 \leq y \leq M$
		DLGAMA		1	real•8	$x > 0$ and $x < 4.2913 \quad 10^{73}$	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 may be included with an INCLUDE control card at the time the program is linked. Note that the library routine names all have a $ as the second character (e.g. O$ERFL, E$ROR).

The subroutines provided by FORTRAN IV are presented in 5.6.3.1 through 5.6.3.12 and summarized in Table 5-5.

5.6.3.1. Arithmetic Overflow and Underflow Test (OVERFL)

The overflow check subroutine, OVERFL, informs the program when computational results are not within the maximum or minimum magniturde 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:

- FORTRAN IV 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 FORTRAN IV 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.

5.6.3.2. 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 ±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 = A divide check has occurred. The indicator is reset.

2 = 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:

```
1    5 7
       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.

5.6.3.3. 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.

5 = If an ERR exit was taken because of an unrecoverable I/O error. No further references to the file are permitted.

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

5.6.3.4. 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:

```
1    5 7
      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 ERROR1
      RETURN
   10 CONTINUE
```

5.6.3.5. 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.

5.6.3.6. 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 $1 \leqslant e \leqslant 4$, then i is set to 2. Execution of the SLITET subroutine does not affect the indicator settings.

5.6.3.7. 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.

5.6.3.8. 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.

5.6.3.9. EXIT Subroutine

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

5.6.3.10. FETCH Subroutine

The FETCH subroutine loads a separately 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 an 8-character name enclosed in apostrophes, or a double precision or complex variable containing a load module name. The load module name must conform to the linkage editor LOADM parameter. See system services program manual, UP-8062 (current version).

Examples:

```
1      7
       .
       .
       .
       DOUBLE PRECISION DNAME/'LOADMXØØ'/
       CALL FETCH (DNAME)
       .
       .
       .
       CALL FETCH ('LOADMXØØ')
```

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

5.6.3.11. 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.

5.6.3.12. 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	Special Name
OVERFL	CALL OVERFL (i)	Tests for overflow or underflow.	O$ERFL
DVCHK	CALL DVCHK (i)	Tests for invalid division.	D$CHK
ERROR	CALL ERROR (i)	Tests for function or I/O error conditions.	E$ROR
ERROR1	CALL ERROR1	Sets the function error indicator.	E$ROR1
SLITE	CALL SLITE (e)	Sets the sense indicators specified.	S$ITE
SLITET	CALL SLITET (e,i)	Tests for the setting of specified sense indicators.	S$ITET
SSWTCH	CALL SSWTCH (e,i)	Tests the binary switch specified by the integer expression and returns a value in the integer variable name.	S$WTCH
DUMP	CALL DUMP (list)	Dumps main storage assigned to the program; program execution terminates.	D$MP
PDUMP	CALL PDUMP (list)	Dumps main storage assigned to the program; program execution continues.	P$UMP
EXIT	CALL EXIT	Terminates the program.	E$IT
FETCH	CALL FETCH (s)	Loads and transfers control to the overlay specified by the phase name.	F$TCH
LOAD	CALL LOAD (s)	Loads subprogram overlays and transfers control to the program statement after the CALL statement.	L$AD
OPSYS	CALL OPSYS ('LOAD', s)	Loads subprogram overlays and transfers control to the program statement after the CALL statement; equivalent to CALL LOAD statement.	O$SYS

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 programmer reference. All statements in this section are order dependent. Refer to Figure 1-1.

6.2. ARRAY DECLARATION

An array is an ordered set of elements identified by a symbolic name (1.2.4). 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:

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

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 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.

Example:

```
1    5 7
      DIMENSION B (5,1Ø)
      CALL A (B,5,1Ø)
           .
           .
           .
      SUBROUTINE A (X,I,J)
      DIMENSION X (I,J); DECLARES (5,1Ø)
    5 J=5
   1Ø I=1Ø
      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 $v_1(i_1), v_2(i_2), \ldots, v_n(i_n)$

where:

v(i)
Is an array declarator (6.2.1)

Description:

The DIMENSION statement declares arrays.

Examples:

```
    1    5 7
1.  DIMENSION INRAY (1Ø)
2.  DIMENSION ARRAY2 (1Ø)....ARRAY3 (1Ø,6,9)
```

1. This DIMENSION statement declares an integer array named INRAY, which has 10 elements. No initialization of the array is accomplished.

2. The second DIMENSION statement declares two arrays containing real data elements.

6.4. TYPE STATEMENTS

Two kinds of type statements can be used in FORTRAN IV; the explicit type statements INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGICAL; and the IMPLICIT type statement. In the absence of typing with these statements, symbolic names starting with the letters I, J, K, L, M, and N are considered to be integer*4 type (FORTRAN name rule); all others are considered to be real*4.

6.4.1. Explicit Type Statements

Format:

$t*s\ a_1\ *s/c_1/, a_2\ *s/c_2/, \ldots, a_n*s/cn/$

where:

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

a
: Is a variable name, an array name, an array declarator, or a function name.

c
: Is an optional list of constants used to initialize the immediately preceding variable name or array. When used to initialize an array, the /c/ may be a list, each element of which may be a c, or j*c when using the multiplier constant.

*s
: Is an optional length specification (2.3); may not be specified if the type is DOUBLE PRECISION.

Description:

An explicit type statement not only specifies the data type of a name but also contains initialization values when the /c/ option is used. Numeric initialization values are converted to the type of the corresponding variable or array, but note that truncation may occur.

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:

```
    I      7
1.  |      REAL LOAF,IOTA/5.2/,JOKE/7.5/,*MATRIX(3,4,5)/60*0.0/
2.  |      REAL*8 A,B,C
3.  |      REAL A,B*8,C
```

1. This statement specifies LOAF, IOTA, JOKE, and MATRIX as real types. In addition, 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.

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

3. This statement specifies A and C as real variables; B is double precision variable because of its length specification.

6.4.2. IMPLICIT Statement

Format:

IMPLICIT $t*s(a_1, a_2, \ldots, a_n), t*s(a_{n+1}-a_m, \ldots), \ldots$

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 specifies types of symbolic names by the first letter of the name, including the dollar sign.

$ may not be included in a range specification (two letters separated by a minus sign). It must be separate. The dollar sign indicates real data by standard typing conventions.

Symbolic names that start with a letter not covered by the IMPLICIT statement are typed according to the standard conventions. 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 (Figure 1-1).

Example:

```
1      7
       IMPLICIT REAL*8(A—D,F),LOGICAL(L),
      *INTEGER*2(N,Q,U,V),INTEGER(X-Z,$)
```

Explanation:

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;
- X and $ are implicitly typed as integer by the IMPLICIT statement; and
- Y and Z are real because of the standard convention.

6.5. EQUIVALENCE STATEMENT

Format:

EQUIVALENCE $(k_1),(k_2),\ldots,(k_n)$

where:

k

Is a list of the form $a_1,a_2,\ldots,a_n$ 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.

6.6. COMMON STATEMENT

Format:

COMMON /x_1/a_1/ . . . /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 x_1 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. FORTRAN IV inserts spaces to achieve proper alignment.

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:

```
1       7
        DIMENSION A(3)
        COMMON B,C,D
        EQUIVALENCE (A,D)
        COMMON E
```

The format 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 the following statement is an illegal extension of COMMON.

```
1       7
        EQUIVALENCE (A(3),B)
```

6.7. EXTERNAL STATEMENT

Format:

EXTERNAL $v_1, v_2, \ldots, v_n$

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:

```
1    5 7
  10    CALL A
 20     CALL B(A)
```

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

```
1       7
        EXTERNAL A
```

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 supersedes 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 FORTRAN IV statements required for input and output control. For further information, refer to the "Input/Output and FORMAT Statements" section of the fundamentals of FORTRAN programmer reference. Also see Section 11 in this manual that describes the usage of the data management system in the CDI environment and Section 12 that describes the usage in the DTF environment.

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 used in FORTRAN for sequential files include card reader, printer, card punch, magnetic tape, and disk subsystems used sequentially. Direct access processing also is possible with disk subsystems. The peripheral devices are assigned unit numbers within the user's system.

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:

$a_1, a_2, \ldots, a_n$

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.

The subscript expression of an array element may not reference a function.

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.

NOTE:

Section 11 discusses record length limitations with regard to various devices and the file access method in a CDI environment. Section 12 discusses record length in a DTF environment.

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 7.3.1. through 7.3.7. The FORMAT statement, used for editing values represented by character strings on the external media, also is described.

Files referenced with the standard statements are always treated as sequential, even when they reside on disk 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,SCREEN=b)k
WRITE (u)k
READ (u,END=label,SCREEN=b,ERR=label2)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.

SCREEN=b
Is an optional 8-character name enclosed in apostrophes, a double precision, or a complex variable containing the name of a full screen workstation format.

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

NOTES:

1. *The order of END (or EOF), ERR, and SCREEN specifications are interchangeable.*

2. *The SCREEN parameter name may also be specified on the USE job control statement which calls screen format services. Screen format service allows the user to set up a multiple line template that can be issued to the workstation terminals with one input or output imperative (full screen workstation) or a single line-by-line method. See the current versions of the screen formatting concepts and facilities user guide/programmer reference and the job control user guide.*

Description:

The unformatted I/O statements initiate and control the transfer of unformattted 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 specified in 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:

```
1      7
       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),
      1(DUMMY(51),C)
         .
         .
         .
       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, ERR, and SCREEN Clauses

In an unformatted READ statement, the END, ERR, and SCREEN clause may appear in any order after the unit designation. In a formatted READ statement, these clauses may appear in any order after the FORMAT designation. EOF is an alternate form for END and is identical in function in FORTRAN IV. 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 (5.6.3.3) are set. If the ERR parameter is not specified, the program is terminated with an informational message when a record cannot be processed.

The screen clause denotes the name of the full screen workstation terminal format where the input/output is to be applied (applicable to CDI environment only).

7.3.2. Formatted READ/WRITE Statements

Formats:

```
READ (u,a)k
READ (u,a,SCREEN=b)k
READ (u,a,EOF=label1)k
READ (u,a,END=label1)k
READ (u,a,ERR=label,SCREEN=b)k
READ (u,a,END=label1,ERR=label2)k
WRITE (u,a)k
WRITE (u,a,SCREEN=b)k
```

where:

u
Is a constant or an integer variable designating an input or output device.

a
Is an array name, 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.

SCREEN=b
Is an optional 8-character name enclosed in apostrophes, a double precision, or a complex variable containing the name of the full screen workstation format.

k
Is an optional I/O list.

NOTE:

The SCREEN parameter name may also be specified on the USE job control statement which calls the screen format services routine. See the job control user guide for details.

Description:

The formatted READ/WRITE statements initiate and control the transfer of formatted data between a designated peripheral device and main storage. Data is always converted from and to character strings on external media and the internal representation specified in 2.3. A COMPLEX variable or array element always requires two FORMAT editing codes. In a READ statement referencing a workstation terminal, any combination of the END, ERR and SCREEN clauses is permitted (CDI environment only).

7.3.2.1. I/O Compatibility Statements

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

```
READ a,k
PUNCH a,k
PRINT a,k
```

where:

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

k
Is an I/O list.

NOTE:

FORTRAN II unit specifications are required when executing with a user-defined I/O configuration. The FORTRAN system-supplied I/O module (FL$I01) provides the appropriate devices.

7.3.3. FORMAT Statement

Format:

```
l FORMAT (q1t1z1t2z2...tn-1zn-1tnq2)
```

where:

l
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 a blank (wX), Hollerith ($wHc_1c_2...c_w$) and literal ($'c_1c_2...c_w'$) 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 7.3.3.1. through 7.3.3.4.

Examples:

```
1    5 7
  100 FORMAT (' FIRST PAGE'/)
  110 FORMAT (///I12,2XI12/)
```

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. FORTRAN IV allows the descriptors listed in Table 7-1.

Table 7—1. FORMAT Statement Field Descriptors

Classification	Field Descriptor
Integer	rIw
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)	$wHc_1c_2...c_w$
Hexadecimal	rZw
Literal	$'c_1c_2...c_w'$
Blank	wX
Record Position	Tp

LEGEND:

r = a repeat count ($0<r\leq255$)

w = the field width ($0<w\leq255$)

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

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

c = character

p = character position in the external record ($0<p\leq32767$)

The specifications within the field descriptors are described in the following listing 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 programmer reference. Refer also to 7.3.3.1.13.

- 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 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,I5,I5,F10.0,2X,I5,I5,F10.0.

7.3.3.1.1. Integer Descriptor (rIw)

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 ±32,768 for integer*2 or ±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 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 is rounded to the correct size. If the value is too small or too large for the range, a zero is substituted.

On output, the external field has the following format:

$$s_1 \emptyset . n_1 n_2 \ldots n_d E s_2 e e$$

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 character 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:

$$s \; i_1 i_2 \ldots i_{w-d-1} * f_1 f_2 \ldots f_d$$

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.

Suffcient 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:

$s_1 \emptyset . n_1 n_2 \ldots n_d D s_2 e e$

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 the .FALSE. statement. 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 field in the format indicates the number of significant digits and must be greater than zero when editing real data. 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 specifes 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 ($whc_1c_2...c_w$)

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 ('$c_1c_2...c_w$')

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. A literal may not exceed 255 characters.

- 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 consolidatd 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.

```
1    5 7
       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)

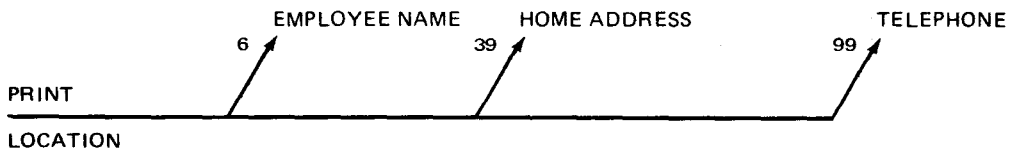
This descriptor omits the next w consecutive characters on input. On output, the blank descriptor skips w positions in the output record. At the time each output formatted record is started, it is filled with blanks.

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, 13 HEMPLOYEE△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.

```
1    5 7
       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 oP 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 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, '$h_1 ... h_n$', 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 Form of READ Statement

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 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 reedited using a different FORMAT statement. This order neither selects nor initiates action on a peripheral device.

The FORTRAN IV 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 (see Section 11 for CDI environment and Section 12 for DTF).

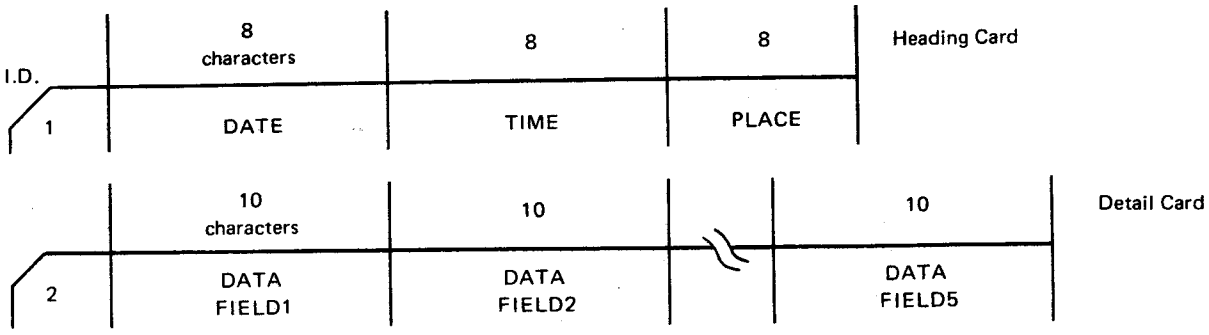
The reread feature is used when the program must determine the kind of information in a record. For instance, both header and detail records may be intermixed, and each kind of record may require different editing information in a FORMAT statement. After a READ order transfers a record to main storage, 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 statement such as:

```
READ (29,a)k
```

This would be 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:




```
1    5 7
      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 FORTRAN IV. Both classes process only formatted records, with the FORTRAN IV 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).

7.3.5.1. NAMELIST Statement

Format:

NAMELIST /n_1/$a_1, a_2, \ldots, a_n$/n_2/$a'_1, a'_2, \ldots, a'_n \ldots$

where:

n

Is a namelist name of from one to six characters, beginning with a letter and enclosed in slashes, used to identify the set of data names that follow.

a

Is a simple list of variables or arrays of any type representing the data to be transferred; array element names are not permitted.

Rules:

1. Once the namelist name is defined by its appearance in a NAMELIST statement, it cannot be redefined in any other statement and can appear only in I/O statements.

2. The list of variables and array names belonging to the specified namelist ends with the specification of another namelist name enclosed in slashes or with the end of the NAMELIST statement.

3. A variable name or array name may be associated with more than one namelist name.

Description:

The NAMELIST statement is a nonexecutable statement that permits formatted data transfer operations without either a FORMAT statement or a list of names in an I/O statement.

To use this statement, symbolic data set names are specified in the NAMELIST statement and also in the record of data to be transferred. No data type is implied by the data set name; for example, a NAMELIST statement specifying two sets of data may appear as:

```
1       7
NAMELIST/GRUP1/A,I,MATRIX/GRUP2/X,J
DIMENSION MATRIX (20)
```

GRUP1 contains the variable names A and I and the array MATRIX/GRUP2 contains the variable names X and J.

An I/O statement can specify a namelist name in place of the usual reference to a format specification. The name specified identifies the record to be transferred. Data in a record is preceded by a variable or array element name and an equals symbol. To ensure transfer of the correct data, the object program compares the data name associated with a namelist with those in the record.

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:

$$\&n\,a_1=c_1,\ a_2=c_2,$$
$$\vdots$$
$$a_n=c_n,\ \&END$$

where:

n
: Is a namelist name (a name identical with the name specified in the NAMELIST statement).

a
: Is a variable, an array element, or an 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 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. A 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 disk, 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:

```
1       7
        INTEGER E,F,G
        READ(U,*) A,B,C,D,E,F,G,H,I
```

```
12 14 /
17.23961727,12,2*,'HE''S'
```

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*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 disk.

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.

7.3.6.2. BACKSPACE Statement

Format:

```
BACKSPACE u
```

where:

u
Is an integer constant or variable designating a sequential file on tape or disk.

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 is 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 repositions the file; the file is open after a legal BACKSPACE statement.

The BACKSPACE statement cannot be used with:

- a file of blocked records;
- a file having two I/O areas; or
- a file having a work area (DTF environment only).

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 disk 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 listing shows instances where specific commands are prohibited or ignored.

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

W Indicates operation not permitted with double buffering.

In addition, since not all operations are permitted on all devices, the following listing shows prohibited combinations.

File type \ Operation	READ	WRITE	ENDFILE	BACKSPACE	REWIND
TAPE	*	*			
DISK	*	*			
CARD READ		P		P	P
CARD PUNCH	P				
PRINTER	P				
REREAD		P		P	P
WORKSTATION				P	P

*Prohibited when files are defined as input only or output only.

(See buffer allocation for tape and disk devices.)

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

7.3.8. File Screen Workstation I/O

In addition to line-by-line I/O, the user can reference the workstation terminal using pregenerated screen templates. These templates can be accessed via formatted or unformatted FORTRAN I/O statements. Data transmitted using editing features of screen format services is done via FORTRAN unformatted I/O statements. FORTRAN edited data should be transmitted as character inserts that are not re-edited by screen format services. It is the user's responsibility to coordinate the FORTRAN I/O statements with the screen format edit services. A complete discussion of screen format services can be found in screen formatting concepts and facilities user guide/programmer reference. A backspace or rewind is not permitted.

7.4. DIRECT ACCESS FILES

FORTRAN IV direct access statements are used to control disk subsystems. The term "direct access" refers to the ability of the disk to access a specified record of a file without accessing all preceding records. Disk 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:

$$\text{DEFINE FILE } u_1(r_1, m_1, x_1, v_1), u_2(r_2, m_2, x_2, v_2), \ldots, u_n(r_n, m_n, x_n, v_n)$$

where:

u
: Is a file identifier, an integer constant specifying a file, or a unit reference number.

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.

x
: Is one of three possible code letters to indicate an option of format control:

  L
  : Transfers either formatted or unformatted data, where the specification form determines the number of bytes.

  E
  : Transfers formatted data, where the specification for m determines the number of bytes.

  U
  : Transfers unformatted data, where the specification of m designates main storage units.

v

Is the associated variable for the file, which must be an unsubscripted integer*4 variable. After execution of a READ or WRITE statement, the variable is assigned a value in the range ($1 \leqslant v \leqslant r$) indicating the sequential position of the next record in the file; after execution of a FIND statement, it is assigned a value indicating the position of the desired record. It is not defined (i.e., set to a value) by the DEFINE FILE statement.

Description:

A DEFINE FILE statement is executable, and it dynamically describes one or more files that may be referenced during program execution. At the start of execution of a FORTRAN program, all direct access units are considered to be undefined, and no READ, WRITE, or FIND references are permitted. When a DEFINE FILE is executed, the characteristics of one or more units are registered with the FORTRAN system, and the units are made available for use. Thereafter, further definitions of previously defined units are ignored.

The associated variable v should not be passed indiscriminately between subprograms or used for purposes other than a file pointer, since the compiler has no syntactic clues as to its usage when the DEFINE FILE statement is absent in a subprogram. When an associated variable must be transmitted to a subprogram, it should be passed in COMMON storage or, less preferably, associated with a dummy argument called by name.

To calculate the record size in storage units (when using the u specification for parameter x): determine the total number of bytes required for all the items of the I/O list, and divide this by 4. If the quotient is not an integer, round it to the next highest integer. There is no restriction on the transmission of multiple records by FORMAT/list interaction, but unformatted lists cannot specify more than one disk record.

Example:

```
1      7
       DEFINE FILE 3(100,120,L,FILE3),
      15(98,80,U,FILE5)
```

File 3 is composed of 100 records, the maximum size of which is 120 bytes. L indicates that the record size is specified in bytes. If the I/O statement contains a reference to a format, 120 bytes of formatted data are transferred; if unformatted data is transferred, File 5 contains 98 records, each 80 bytes in length.

7.4.2. Disk READ Statement

Format:

```
READ(u'p,a,ERR=label1,END=label2)k
```

where:

u

Is a file identifier represented by an integer constant or variable followed by an apostrophe.

p

Is an integer expression designating the position of the record in the file, which should be in the range ($1 \leqslant p \leqslant r$), where r is the number of records in the file.

a

Is an optional label of a FORMAT statement, an array name, or the character asterisk.

ERR=label$_1$
Optionally specifies the label of a statement to which control is to be transferred when an error condition occurs.

END=label$_2$
Optionally specifies the label of a statement to which control is to be transferred when an ENDFILE record is encountered, or when p is outside the file limits.

k
Is an I/O list.

Example:

```
1    5 7
       INTEGER FILE3/1/
       DEFINE FILE3(100,512,L,FILE3)
         .
         .
         .
       READ (3' FILE 3,87,ERR=110) A,B,(C(I),I=1,30)
    87 FORMAT (32F16:4)
```

The first record in file 3 is tranferred to main storage when the READ statement is first executed. Each subsequent execution of the READ statement transfers the next record in the file to main storage, unless the associated variable FILE3 is explicitly redefined. The descriptor 32F16.4 indicates that each unit of data consists of 16 bytes and 32 such units of data are to be transferred. Thus, the 512 bytes (16x32) of the record are transferred to main storage.

The slash in a FORMAT specification can control the starting point of data transfer in a file. If the FORMAT statement in the example is:

```
FORMAT(//32F16.4)
```

the first execution of the READ statement transfers the third record in the file; the second execution transfers the sixth record.

7.4.3. Disk WRITE Statement

Format:

```
WRITE(u'p,a)k
```

where:

u
Is a file identifier represented by an integer constant or variable 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, or the character asterisk.

k
Is an I/O list.

Example:

```
1    5 7
       DEFINE FILE 4(150,36,L,FILE4)
        .
        .
        .
       LOGICAL L
       DOUBLE PRECISION D
        .
        .
        .
       FILE4 = 2
        .
        .
        .
       WRITE (4' FILE4+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. Disk FIND Statement

Format:

```
FIND (u'p)
```

where:

u
: Is a file identifier represented by an integer constant or variable and 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 disk. This statement positions the access arms to a disk 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:

```
1       7
        FIND (4' 20)
          .
          .
          .
        READ (4' 20) A, B, C
```

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 FORTRAN IV programs is described in this section. For more information, refer to the fundamentals of FORTRAN programmer reference. See the type statements (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 $k_1/d_1/, k_2/d_2/, \ldots, k_n/d_n/$

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 should be in blank COMMON; they should be 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, hexidecimal, 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 corerect 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 on 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 lateral string exceeds the limit of the array or the 255-character literal limit.

Example:

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

This 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/
```

This produces:

```
ARR(1)  contains          'ABCD'
ARR(2)  contains          99.Ø
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/
```

This produces:

```
VAR       contains          17.Ø
ARR(1)    not initialized
ARR(2)    not initialized
ARR(3)    contains          'ABCD'
ARR(4)    contains          'EFGH'
ARR(5)    contains          'IJ△△'
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 subprogram is headed by the BLOCK DATA statement. The order of statements is governed by the rule shown in Figure 1-1.

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.

PART 3. COMPILE, DEBUG, AND EXECUTE PROCEDURES

PART 3: COMPILE, DEBUG, AND EXECUTE PROCEDURES

9. Compilation

9.1. GENERAL

The FORTRAN IV compiler accepts source programs from either a card file or a disk file. Card files are entered directly into the card reader along with the appropriate job control stream. Disk files are built by the system librarian and its job control stream is entered from a disk (filed job control stream) or from the card reader. When operating in the CDI environment, disk files may also be built from a workstation terminal using the facilities of the system editor. Appendix E contains compilation examples.

9.2. FORTRAN IV COMPILERS

The FORTRAN IV compilers are named FOR4 and FOR4L; both require one work file allocated in the job control stream. FOR4 requires 10800_{16} (X'10800') bytes of main storage plus space for the prologue; FOR4L requires 19000_{16} (X'19000') bytes of main storage plus space for the prologue. When FOR4 is executed, FOR4L will automatically be loaded if sufficient main storage was allocated for the job. No other use is made of additional storage. FOR4L contains significantly larger tables and workfile I/O buffers.

9.3. PARAMETER STATEMENT FORMAT

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

Format:

```
1
//      △PARAM△    n1=d1,n2=d2,...
```

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), and 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 1.4.

9.3.1. Compiler Arguments

A list of arguments provided by the FORTRAN IV compiler follows. Descriptions of the arguments follow the list.

Format:

LABEL	△OPERATION△	OPERAND
//	△PARAM△	OUT=filename,MAP=(S,A,L), LIN=filename,LST=option,OPT=(S,N,X,C,T), ERRFIL=module-name/lfdname IN=module-name/filename

Input Argument:

IN=module-name/filename
Specifies compilation of source programs residing in disk 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 specifed, a default name is assumed and can be described via the LIN argument.

The occurrence of an IN argument signals the end of the scanning for other PARAMs. Arguments following an IN argument on a given // PARAM card are ignored. Subsequent // PARAM statements may contain only IN arguments to allow for stacked compilations (see 9.4).

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 YRUN.

Map Argument:

MAP=(S,A,L)
Specifies the type of maps produced by the compiler. One or all options may be chosen. The options include:

S
Specifies object summary information, including module size and external subroutines called.

A
Specifies an alphabetical listing of the addresses assigned to variables, arrays, and statement labels.

L
Specifies a listing of the addresses assigned to variables, arrays, and statement labels in order by the storage locations assigned.

When a MAP argument is specified, it supercedes the maps selected by the LST argument. Also, when a MAP argument is specified, it is not necessary to specify LST=M.

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 file name. 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, and diagnostics.

S
Specifies, in addition to the N listing, the source code listing.

M
Specifies, in addition to the S listing, an object summary and a storage map showing the addresses assigned to variables and arrays. (Can be superseded by the MAP argument.)

If no LST PARAM is specified, the S option is assumed.

Options Argument:

`OPT=(S, N, C, T)`
Specifies compilation options.

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

S
Specifies that statement numbers will be inserted into the generated code as an aid to debugging. When S is specified, the size of the object program and its execution time can increase significantly.

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.

C
Specifies all references to array elements are to be checked to determine if they are outside the declared limits of the array.

T

Specifies that tracing of executed labels is requested. The compiler generates a special subroutine call at every label. A TRACE ON statement must occur in the program to activate tracing.

If only one OPT argument is specified, the parentheses are optional.

Error File Argument:

ERRFIL=module-name/lfdname

Specifies that an error file is created. This file allows the programmer to correct source code using the error file processor (EFP) instead of the output listing.

Module-name is a one to eight alphanumeric character identifier indicating the module name of an OS/3 MIRAM source library file. Lfdname is a one to eight alphanumeric character identifier indicating the LFD name of an OS/3 MIRAM source library file.

If no ERRFIL parameter is specified, the error file is not written.

9.4. STACKED COMPILATION

The FORTRAN IV compiler is capable of processing up to 100 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 proceded by compilation // PARAM statements. All FORTRAN IV compiler parameters are global and apply to all programs compiled. When a parameter is to be changed, the job control stream should be organized into two or more FORTRAN IV compilations, each containing the required parameters. For example:

```
1    5  7
// WORK1
// EXEC FOR4
// PARAM
/$
.
.              (one or more program units)
.
/*
// WORK1
// EXEC FOR4
// PARAM
/$
.
.              (one or more program units)
.
/*
```

When the source programs are on disk 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. Note that once FORTRAN IV encounters an IN parameter, no parameters, except other IN parameters, may occur.

When FORTRAN IV is doing a stacked compilation, the work file usage is cumulative. Thus, it may be desirable to do multiple executions of the compiler to reduce the work file requirements.

9.5. SOURCE CORRECTION FACILITY

When source programs reside on disk, 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 system librarian's module correcting (COR) function. Refer to the current version of the system service programs (SSP) user guide. 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:

```
1    5 7
// 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.

9.6. 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 jprocs is described in Appendix E. However, if you want to create your own job control stream, the following rules must be observed:

- The FORTRAN compiler requires one work file. The jproc WORK1 supplies this file.
- If the IN or OUT options are specified, the appropriate disk files must be defined.
- A printer device is required and must be defined by the // DVC 20 and the // LFD PRNTR job control statements.
- Because of the stacked compilation feature (9.4), 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 from the program is allowed; the source correction facility (9.5) or the stacked compilation feature (9.4) would mistake the data set for FORTRAN source code.

9.7. USE OF LARGER VERSION

Programs containing a large number of variables, arrays, statement labels, function references, etc., may require the FOR4L compiler to compile successfully. Some of the error messages that indicate that the larger version of the compiler is necessary are:

SYMBOL TABLE EXCEEDED

TOO MANY CONSTANT EXPRESSIONS

Recompile your program using the FOR4L compiler.

10. Debugging

10.1 GENERAL

Debugging aids are provided with the FORTRAN IV compiler. These debugging aids consist of the standard I/O statements (especially list-directed statements described in 7.3.5), conditional compilation, subscript checking, label tracing (TRACE ON and OFF), and the DUMP and PDUMP standard library subroutines for formatted main storage dumps.

10.2 CONDITIONAL COMPILATION

The compiler accepts the 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:

```
1    5 7
X       PRINT 1Ø, A, B, C
X   1Ø FORMAT (3F15.6)
```

This coding is 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. FORMATTED MAIN STORAGE DUMP

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

Format:

```
CALL p(u1, l1, f1, u2, l2, f2, . . . , un, ln, fn)
```

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 upper 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	f	Display Interpretation
0	Hexidecimal	5	Real*4
1	Logical*1	6	Real*8
2	Logical*4	7	Complex*8
3	Integer*2	8	Complex*16
4	Integer*4	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.

10.4 USE OF OPT=S

Initially a program unit should be compiled using the OPT=S option. This yields more diagnostics and makes debugging easier when the statement numbers are available. After the program unit has been debugged, it should be recompiled without the OPT=S option. This will eliminate the instructions that dynamically store the card sequence numbers.

When using the subscript check and label trace features, OPT=S supplies additional information and should always be specified.

10.5 SUBSCRIPT CHECKING

The FORTRAN IV compiler dynamically checks the value of the subscript by accepting a parameter that causes code to be generated before every subscripted reference.

The most common programming error is referencing outside the declared limits of an array. FORTRAN IV checks every subscript reference if the user specifies the OPT=C compilation parameter option. The following example shows the use of this feature.

Example:

```
1    5 7
// EXEC FOR4
// PARAM OPT=(S,C)
/$
        SUBROUTINE A
        DIMENSION X(10)
        I=20
        PRINT 100,X(I)
        STOP
  100   FORMAT (F20 3)
        END
/*
```

The following message is output to the diagnostic device during execution of the PRINT statement.

FL530 SUBSCRIPT 00000020 OUT OF RANGE ON CARD 0004 OF MODULE A

No recovery of the bad subscript is attempted. The number of times a subscript error is permitted and reported is controlled by the FARG option of the error control definition macro (ERRDEF).

The OPT=C option significantly increases the size and execution time of the program. After the program is debugged, it should be recompiled to eliminate the extra code.

The ON CARD field of the message requires the OPT=S option; otherwise, the field will be zeros.

10.6 LABEL TRACE

To follow the actual sequence of executed statements, FORTRAN IV has a trace facility. If the OPT=T parameter option is specified, the compiler generates a special subroutine call at every executable label. When the statement is executed, the following message is displayed on the diagnostic device if tracing is activated:

FLO50 CONTROL AT STATEMENT ____ ON ____ OF MODULE ____

To control the trace facility, two new statements are available (see 10.6.1 and 10.6.2).

The OPT=T option significantly increases the size and execution time of a program. After the program is debugged, it should be recompiled to eliminate the extra code.

The ON CARD field requires the OPT=S parameter option; otherwise, the field will be zeros.

10.6.1. TRACE ON Statement

Format:

```
TRACE ON
```

Description:

The TRACE ON statement enables the display of the TRACE message and is required to begin tracing. Tracing is disabled at the beginning of execution.

10.6.2. TRACE OFF Statement

Format:

```
TRACE OFF
```

Description:

The TRACE OFF statement disables the trace display. This permits the user to control the amount of output produced.

11. Consolidated Data Management (CDM) Execution Environment Configuration

11.1. CDM RELATIONSHIP

This section describes the interface between FORTRAN IV and consolidated data management (CDM), including:

- the relationship between unit numbers and external files;
- the kinds of devices supported;
- performance considerations such as record blocking and buffering; and
- system defaults (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 11.4.

The FORTRAN IV execution environment is consistent with the CDM concept of device independence. This means that substitution of similarly constructed files is possible without recompiling or reassembling. More discussion about device independence is presented in 11.3.

FORTRAN IV supports the following device classes:

- Disk and diskette
- Tape
- Workstation terminal
- Unit record (card reader, card punch, printer, and control stream input).

FORTRAN IV accesses these devices via the standard data management interfaces (CDIB and RIB structures). This interface is described in the common data management concepts user guide.

Before a FORTRAN IV program can be executed, a group of I/O subroutines must be incorporated to support the FORTRAN I/O statements and provide an interface to data management. These I/O subroutines are individually called by the FORTRAN IV compiler and automatically placed in the executable program by the linkage editor. In the executable program, the control module is the center of the entire I/O scheme, because it contains the following:

- A unit table consisting of one entry for each unit number specified in the user program. Each entry contains FORTRAN control data and the information needed to connect the unit number to the actual device.

- A work area for record processing.
- A buffer needed to support the REREAD feature.

An executable program may only contain one I/O control module. This module may be supplied either by FORTRAN IV or configured by the user. When supplied by FORTRAN IV, the configuration allows for standard unit numbers used to reference a printer, card punch, tape, card reader, and a reread unit (11.2). When supplied by the user, he may configure his own set of unit definitions by using the FORTRAN IV unit definition procedure (UNIT). The user-supplied unit numbers are associated with the physical device via a device assignment set (DVC-LFD) at program execution.

11.2. CDI-SUPPLIED CONFIGURATIONS

The following configurations are supplied for general use in simple applications. The unit numbers selected are industry standard. FORTRAN II I/O support is also included in these configurations.

- Control Module FL$IO

Unit	Notes
1	80 byte records; lfdname of FORT1; data can be reread; standard label if tape; can be cards in control stream if // LFD FORT1 is missing
3	Diagnostic device; lfdname of PRNTR; record size of 121; must be output device
5	Equivalent to unit 1
6	Equivalent to unit 3
29	Used to reread data from unit 1
READ	FORTRAN II READ statement (equivalent to unit 1)
PRINT	FORTRAN II PRINT statement (equivalent to unit 3)

- Control Module FL$IO1

Unit	Notes
1	80 byte records; lfdname of FORT1; data can be reread; standard label if tape; can be cards in control stream if // LFD FORT1 is missing
2	80 byte records; lfdname of FORT2
3	Diagnostic device; lfdname of PRNTR; record size of 121; must be output device
5	Equivalent to unit 1
6	Equivalent to unit 3
11	Fixed unblocked records; standard label if tape; lfdname of FORT11
12	Fixed unblocked records; standard label if tape; lfdname of FORT12
29	Used to reread data from unit 1
READ	FORTRAN II READ statement (equivalent to unit 1)
PRINT	FORTRAN II PRINT statement (equivalent to unit 3)
PUNCH	FORTRAN II PUNCH statement (equivalent to unit 2)

11.3. PROGRAMMER-DEFINED CONFIGURATIONS

The unit definition procedure is required to define the execution environment; namely, the logical units and their attributes. The attributes are optional and are used to define the features such as record and buffer sizes, record formats, file type, error recovery, etc. The UNIT definition procedure passes this information to data management at file open time.

When a new file is defined, data management supplies default values for any attributes not explicity specified. The combination of specified attributes and data management defaults defines the file characteristics. When accessing an exisiting file, the specified record size and record format attributes are compared to those specified when the file was originally created. Any incompatibilities between the two produce an error message and terminate the job step. All other specified attributes are accepted. If the record size and record format arguments were omitted from the UNIT definition procedure at file creation, then the default characteristics are used.

To achieve device independence, a unit identifier is the only argument required on the UNIT definition procedure. Since a file name is always required when accessing a file, a user-supplied logical file name may be specified or the UNIT definition procedure will default to a FORTRAN IV logical file name. In either case, the file name must be specified on the LFD job control statement for the device. At program execution time, providing the record sizes are compatible, the unit is connected to the device via data management.

The following examples shows some typical uses of device independence.

Example 1:

Let's suppose that we have a file that can be input from either a card reader, a tape, a disk, or a workstation. By specifiying the UNIT definition procedure with the file arguments FUNIT=1 and FFILEID=INFILE, device independence is achieved. At program execution time, by changing only the logical unit number on the DVC job control statement, the appropriate device is connected. Therefore, if a card reader is required, specify logical unit number 30 on the DVC job control statement. Later if a disk unit is required, specify 50. For a tape unit, specify 90 and for a workstation, specify 200. The lfdname INFILE specified in the FFILEID argument remains unchanged in the LFD job control statement.

Example 2:

Supposing that we have input file but we omit the logical file name; then UNIT definition procedure is coded UNIT FUNIT=10. FORTRAN IV supplies the default logical file name of FORT10. At program execution time, the lfdname of FORT10 would be specified on the LFD job control statement. This is yet another example of device independence whereby the // DVC... // LFD sequence can be changed at program execution time to have the appropriate device connected.

Example 3:

Again, let's suppose that the input file is using a card reader with the UNIT definition procedure of UNIT FUNIT=20,FFILEID=MASTER,FAUE=YES. The FAUE argument is for detection of mispunched cards and therefore unique to a card reader. Now, the input file is to reside on a disk (assuming the file was created prior in a previous job step). At program execution time, the logical unit number is changed to reflect a disk unit. Since the FAUE argument is unique to a card reader, it is ignored (no error message displayed) and processing continues.

Table 11-1 is a summary of the device types and their arguments. Looking at the table, there are eight arguments that are considered device independent. They are: FDIAGNOS, FFILEID, FOPTION, FRECSIZE, FREREAD, FSPOOLIN, FTYPEFLE, and FUNIT. At program execution time, the unit may be switched to another device by changing the logical unit number on the DVC job control statement, thus saving reassembling and recompiling. The remaining arguments are device oriented; however, when applied to a device not supporting the argument, it is ignored and no message is displayed.

There are only two arguments that may cause an OPEN error at program execution time. They are the FRECSIZE and FRECFORM arguments. The error occurs when the record size or record format is not compatible with the device change. A record format error could occur when one file is defined as fixed-unblocked, while at program execution the file is variable-unblocked.

In summary, any combination of arguments may be specified with any device type, with the exception of FRECSIZE and FRECFORM, without the need for reassembling or recompiling.

Table 11—1. FORTRAN IV Devices and Arguments

Argument	Disk	Tape	Unit record	Workstation	Equivalent	Reread
FAUE			*			
FBFSZ		*				
FBKNO		*				
FCHAR			*			
FCKPTREC		*				
FCLRW		*				
FCRDERR			*			
FDEVICE	*	*	*	*	*	*
FDIAGNOS	*	*	*	*		
FERROPT		*				
FEQUIV					*	
FFILABL	*					
FFILEID	*	*	*	*		
FIOOPT				*		
FLINCNTL				*		
FNUMBUF		*	*	*		
FOPRW		*				
FOPTION	*	*	*	*		
FRECFORM	*	*				
FRECSIZE	*	*	*	*		
FREREAD	*	*	*	*		
FSCREND				*		
FSPOOLIN	*	*	*	*		
FTRANS		*	*			
FTYPEFLE	*	*	*	*		
FUNIT	*	*	*	*	*	*
FVERIFY	*					

NOTE:

Only arguments applicable to the device type are recognized. All other arguments are ignored. If the record size or record format arguments are not compatible, they produce an OPEN error and terminate the job step.

If necessary, a unit can be defined as device dependent. This is done via a device type parameter on the UNIT definition procedure. Device dependency is discouraged because any device change requires reassembling a new UNIT definition. The UNIT definition procedure is called via an assembly language source module with the form:

```
1         10
-------------------------------------
name      START
          unit definition
          unit definition
                .
                .
                .
          unit definition_n
          unit termination
          error definition
          END
```

The elements of this assembly module are discussed in 11.3.1 through 11.3.5.

11.3.1. START Statement

The START statement, a subprogram declarator statement required by the assembler, is the first statement of the configuration definition.

Format:

```
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.

11.3.2. FORTRAN Unit Definition Procedure (UNIT)

Each file definition consists of a call on the FORTRAN unit definition procedure (UNIT), with optional arguments specifying characteristics of the file. Each argument consists of a file attribute name, an equal sign, and a particular characteristic of the file being defined. If an argument is not required it is omitted and the comma is deleted.

Consolidated data management (CDM) supplies the default file attributes whenever the target device is known (FDEVICE argument) and FORTRAN IV accepts these defaults. Attributes not relating to the assigned device are ignored.

When defining the file attributes with the UNIT definition procedure, the following syntactical differences between FORTRAN and assembly language should be remembered:

- In the assembler, the statement continuation character is required for line 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 following examples (11.3.2.1 through 11.3.2.6).

11.3.2.1. Unit Record Definition

The following devices are considered unit record devices: reader, punch, and printer. The unit record devices are defined by using the UNIT procedure call described in this format. The arguments may appear anywhere in the UNIT definition procedure while FUNIT is the only required argument. Following the format, descriptions of the UNIT arguments and a UNIT example are presented.

Format:

```
1         10    16
          UNIT  [FDEVICE=UNITREC] FUNIT={k     }
                                        {PRINT }
                                        {PUNCH }
                                        {READ  }

                [FFILEID={filename                          }]
                         {FORTk; if FUNIT=k                 }
                         {PRNTR; if FUNIT=PRINT             }
                         {PUNCH; if FUNIT=PUNCH             }
                         {READER; if FUNIT=READ             }

                [FNUMBUF={1}] [FTYPEFLE={INPUT; if FUNIT=READ                }]
                         {2}           {OUTPUT; if FUNIT=PRINT or PUNCH      }

                [FCHAR={OFF}] [FOPTION=YES] [FAUE=YES]
                       {ON }

               [FREREAD=YES] [FCRDERR=RETRY] [FRECSIZE=k]

                [FSPOOLIN=YES] [FTRANS=ASCII] [FDIAGNOS=YES]
                    or
                [FGETJCS=YES ]
```

Device Identification Argument:

FDEVICE=UNITREC
Specifies that this is a unit record device.

Unit Identifier Argument:

FUNIT=k
Specifies the unit identifier whose value is a unique integer constant in the range from $1 \leq k \leq 99$.

FUNIT=PRINT
Specifies PRINT as the unit identifier.

FUNIT=PUNCH
Specifies PUNCH as the unit identifier.

FUNIT=READ
Specifies READ as the unit identifier.

NOTE:

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 identifiers is not specified, the applicable device specified is used. The units are searched in the order 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≤k≤99. If this argument is omitted and FUNIT=k is specified, FORTk is assumed.

FFILEID=PRNTR
Specifies the file name as PRNTR. If this argument is omitted and FUNIT=PRINT is specified, PRNTR is assumed.

FFILEID=PUNCH
Specifies the file name as PUNCH. If this argument is omitted and FUNIT=PUNCH is specified, PUNCH is assumed.

FFILEID=READER
Specifies the file name as READER. If this argument is omitted and FUNIT=READ is specified, READER is assumed.

Buffer Allocation Argument:

A buffer pool is dynamically acquired via the supervisor DMEM macroinstruction. Once a unit is opened, one or two buffers from the pooled storage area are allocated. The size of each buffer is determined by the FBFSZ argument or from data management at open time. When the buffer pool is used, the work area cannot be used. (See the consolidated data management macro language user guide/programmer reference.)

FNUMBUF=1
Specifies one buffer to be allocated to the unit.

FNUMBUF=2
Specifies two buffers are allocated to the unit.

File Type Argument:

FTYPEFLE=INPUT
Specifies an input file. If this argument is omitted and FUNIT=READ is specified, INPUT is assumed. INPUT should be specified when the file is read but never written.

FTYPEFLE=OUTPUT
Specifies an output file. If this argument is omitted and FUNIT=PUNCH or PRINT is specified, OUTPUT is assumed. OUTPUT should be specified whenever the file is to be written but never read.

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 specified and the file is not allocated by job control, WRITE statements are ignored and the first READ reference causes an end-of-file condition.

A unit need not be declared as optional if the program logic does not reference 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 hold combination is detected.

Reread Argument:

FREREAD=YES
Specifies that the unit is to participate in the rearead feature (7.3.4).

The reread unit consists of a single buffer where each formatted input record is transferred. To save processor time, this data movement is inhibited unless specified.

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 disk for later transcription to a card punch).

Record Size Argument:

FRECSIZE=k
Specifies the logical record size (in bytes). When accessing an existing file, the value of this argument is compared to the record size specification of that file. Any incompatibilities produce OPEN errors. If FRECSIZE and FRECFORM are omitted from the UNIT definition procedure, the file is processed using the physical file information.

Spooled Card Input File Argument:

FSPOOLIN=YES
or
FGETJCS=YES
Specifies this unit will default to a spooled card input file via a GETCS when the lfdname declared in the FFILEID argument is not found.

The spoolin feature (FSPOOLIN=YES) can be applied to any device but cannot exceed 128 bytes. If the record size is omitted, 80 bytes is assumed.

Translate to ASCII Argument:

FTRANS=ASCII
Specifies that all incoming and outgoing records are translated to the ASCII character set.

Diagnostic Message Argument:

FDIAGNOS=YES
Specifies the current unit as the diagnostic device. If FRECSIZE is specified, its value must be 101 or greater. Debugging information may also be written to this device (10.3); not available for input files.

If multiple diagnostic devices are specified, messages are posted to the first diagnostic unit encountered.

If omitted, diagnostics are transmitted to the system log and either the system console or the initiating workstation terminal.

Example:

```
1         10                                                        72
                                                                    /
          UNIT   FUNIT=10,                                          X
                 FDIAGNOS=YES,                                      X
                 FTYPEFLE=OUTPUT
```

This unit procedure call specifies a device independent output file. The devices which may be allocated are: tape, disk, workstation, and unit record output. The device is accessed from the FORTRAN user program via the WRITE command to Unit 10. Diagnostics are posted to this unit during program execution. The file is an output file. Since no filename is specified (FFILEID), FORT10 becomes the default filename (FORT from the omitted FFILEID argument and 10 from the FUNIT argument). FORT10 must also appear as the filename on the LFD job control statement. Since the logical record size and buffer allocation were omitted, the default values are 121 and 1, respectively.

11.3.2.2. Tape File Definition

A single tape file is defined by using the UNIT definition procedure presented in this paragraph. The arguments may appear anywhere in the UNIT definition procedure but FUNIT is the only required argument. Following the format, descriptions of the UNIT arguments and a UNIT example are presented.

Format:

```
1         10      16
          UNIT  [FDEVICE=TAPE] FUNIT={k     }
                                     {READ  }
                                     {PUNCH }
                                     {PRINT }

               [FFILEID={filename                    }]
                        {FORTk; if FUNIT=k           }
                        {READER; if FUNIT=READ       }
                        {PUNCH; if FUNIT=PUNCH       }
                        {PRNTR; if FUNIT=PRINT       }

               [FTYPEFLE={WORK                                  }]
                         {INPUT; if FUNIT=READ                  }
                         {OUTPUT; if FUNIT=PRINT or PUNCH       }

               [FRECFORM={VARUNB}] [FNUMBUF={1}] [FRECSIZE=k]
                         {VARBLK}           {2}
                         {FIXUNB}
                         {FIXBLK}

               [FBFSZ={k                                  }]
                      {FRECSIZE; if FRECFORM=FIXUNB       }
                      {FRECSIZE+4; if FRECFORM=VARUNB     }
                      {FRECSIZE*4; if blocked             }

               [FREREAD=YES] [FDIAGNOS=YES] [FBKNO=YES]

               [FERROPT={IGNORE}] [FFILABL={STD}] [FCKPTREC=YES]
                        {SKIP  }          {NO  }

               [FCLRW={RWD   }] [FOPRW=NORWD] [FOPTION=YES]
                      {NORWD }
                      {UNLOAD}

               [FSPOOLIN=YES] [FTRANS=ASCII]
                    or
               [FGETJCS=YES]
```

Device Identification Argument:

FDEVICE=TAPE
Specifies this is a tape file.

Unit Identifier Argument:

FUNIT=k
Specifies the unit identifier whose value is an unique integer constant in the range from $1 \leq k \leq 99$.

FUNIT=READ
Specifies READ as the unit identifier.

FUNIT=PUNCH
Specifies PUNCH as the unit identifier.

FUNIT=PRINT
Specifies PRINT as the unit identifier.

File Name Argument:

FFILEID=filename
Specifies a 1- to 7-character FORTRAN style symbolic name (filename).

FFILEID=FORTk
Specifies the filename as FORTk, where $1 \leqslant k \leqslant 99$. If this argument is omitted and FUNIT=k is specified, FORTk is assumed.

FFILEID=READER
Specifies the file name as READER. If this argument is omitted and FUNIT=READER is specified, READER is assumed.

FFILEID=PUNCH
Specifies the file name as PUNCH. If this argument is omitted and FUNIT=PUNCH is specified, PUNCH is assumed.

FFILEID=PRNTR
Specifies the file name as PRNTR. If the argument is omitted and FUNIT=PRINT is specified, PRINT is assumed.

File Type Argument:

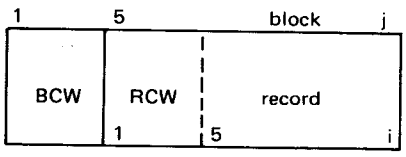
FTYPEFLE=WORK
Specifies a work file. WORK should be specified if the tape is to be read and written. WORK files are limited to a single volume (reel).

FTYPEFLE=INPUT
Specifies an input file. If this argument is omitted and FUNIT=READ is specified, INPUT is assumed. INPUT should be specified if the tape is to be read but never written.

FTYPEFLE=OUTPUT
Specifies an output file. If this argument is omitted and FUNIT=PUNCH or PRINT is specified, OUTPUT is assumed. OUTPUT should be specified if the tape is to be written but not read.

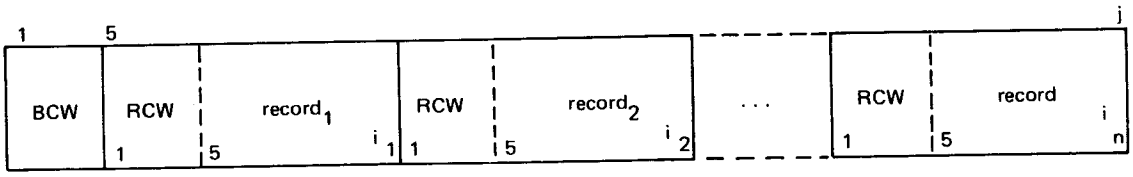
Record Format Argument:

FRECFORM=VARUNB
Specifies variable length unblocked records.



FRECFORM=VARBLK

Specifies variable length blocked records.



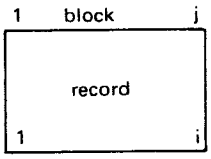
For both unblocked and block records, i specifies record size, j specifies block size, BCW specifies a data management block control word, and RCW specifies a data management record control word.

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 FORTRAN IV 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.

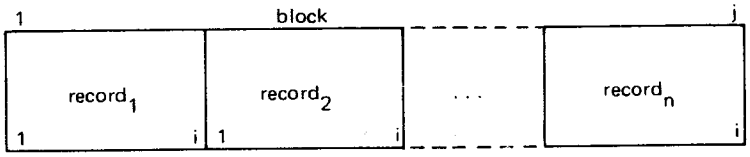
FRECFORM=FIXUNB

Specifies fixed length unblocked records.



FRECFORM=FIXBLK

Specifies fixed length blocked records.



For both unblocked and blocked records, i is the size specified for the FRECSIZE argument, and j is the size specified for the FBFSZ argument. For unblocked records, i and j must be equal. For block 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. In other words, when a FORMAT processes a record, it cannot request more than "recsize" bytes of data.

Buffer Allocation Argument:

A buffer pool is dynamically acquired via the supervisor DMEM macroinstruction. Once a unit is opened, one or two buffers from the pooled storage area are allocated. The size of each buffer is determined by the FBFSZ argument or from data management at open time. (See the consolidated data management macro language user guide/programmer reference.)

FNUMBUF=1
Specifies one buffer to be allocated to the unit.

FNUMBUF=2
Specifies two buffers to be allocated to the unit.

Record Size Argument:

FRECSIZE=k
Specifies the logical record size (in bytes). When accessing an existing file, the value of this argument is compared to the record size specification on that file. Any incompatibilities produce OPEN errors. If FRECSIZE and FRECFORM are omitted from the UNIT definition procedure, the file is processed using the physical file information.

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 $18 \leqslant k \leqslant 32767$.

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 times 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. File containing blocked records cannot be backspaced.

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 where each formatted input record is transferred. To save processor time, this data movement is inhibited unless specified.

Diagnostic Messages Argument:

FDIAGNOS=YES
Specifies the current tape unit as the diagnostic device. If FRECSIZE is specified, its value must be 101 or greater. Debugging information may also be written to this device.

If multiple diagnostic devices are specified, FORTRAN IV will post messages to the first diagnostic unit encountered; not available for input files. If omitted, diagnostics are transmitted to the system log and either the system console or the initiating workstation terminal.

Block Numbering Argument:

FBKNO=YES
Specifies that sequence numbers 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. If omitted, no block numbering occurs.

Device Error Processing Argument:

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. 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.

NOTES:

1. *SKIP and IGNORE should be used with discretion, since device position may be lost for unformatted files and NAMELISTs.*

2. *When the problem program receives control at the ERR label, the ERROR subroutine (5.6.3.3) 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 on a list-directed statement, which always implies loss of position.*

3. *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).*

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:

FCKPTREC=YES
Specifies that checkpoint blocks contained on an input tape will be bypassed. If omitted , the blocks are read as data records. Argument is ignored when applied to an output file.

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 Arguments:

FOPTION=YES
Specifies an optional unit; a unit not always required during program execution. When specified and the file is not allocated by job control, WRITE statements are ignored and the first READ reference causes an end-of-file condition. A unit need not be declared as optional if the program logic does not reference the unit.

Spooled Card Input File Argument:

FSPOOLIN=YES
or
FGETJCS=YES
Specifies this unit will default to a spooled card input file via a GETCS when the lfdname for the unit does not appear in the job control stream.

The spoolin feature can be applied to any device but cannot exceed 128 bytes. If the record size is omitted, 80 bytes is assumed.

Translate to ASCII Argument:

FTRANS=ASCII
Specifies that all incoming and outgoing records are translated to the ASCII character sets.

Example:

```
1         10    16                                                    72
          UNIT  FDEVICE=TAPE,                                         X
                FUNIT=7,                                              X
                FCKPTREC=YES,                                         X
                FRECSIZE=100,                                         X
                FTYPEFLE=WORK,                                        X
                FRECFORM=FIXBLK
```

This unit procedure call specifies a tape device (FDEVICE=TAPE). The unit number is 7 (FUNIT=7); checkpoint records are bypassed (FCKPTREC=YES); the record size is 100 bytes (FRECSIZE=100); it is a work file (FTYPEFLE=WORK); and fixed-length blocked records are used (FRECFORM=FIXBLK).

The following defaults are assumed:

- FORT7 becomes the default filename (FORT from the omitted FFIELD argument and 7 from the FUNIT argument).
- One input/output buffer is allocated since the FNUMBUF argument is omitted.
- The buffer size defaults to 104 bytes based on the record size argument (buffer size must be equal to or greater than the record size).

11.3.2.3. Disk File Definition

A disk file is defined by using the UNIT definition procedure presented in this paragraph. A MIRAM file is the only file type supported in the CDM environment. The arguments may appear anywhere in the UNIT definition procedure while FUNIT is the only required argument. Following the format, descriptions of the UNIT arguments and a unit example are presented.

Format:

```
1         10    16
          UNIT  [FDEVICE=DISK]
                FUNIT={k      }
                      {READ   }
                      {PUNCH  }
                      {PRINT  }
                [FFILEID={filename                    }]
                         {FORTk;if FUNIT=k            }
                         {READER;if FUNIT=READ        }
                         {PUNCH;if FUNIT=PUNCH        }
                         {PRNTR;if FUNIT=PRINT        }
                [FTYPEFLE={WORK                                  }] [FBFSZ=k]
                          {INPUT;if FUNIT=READ                   }
                          {OUTPUT;if FUNIT=PUNCH or PRINT        }
                [FRECFORM={VARUNB}] [FRECSIZE=k]
                          {FIXUNB}
                [FOPTION=YES] [FVERIFY=YES]
                [FDIAGNOS=YES] [FSPOOLIN=YES] [FREREAD=YES]
                               [     or     ]
                               [FGETJCS=YES ]
```

Device Identification Argument:

FDEVICE=DISK
Specifies that this is a disk file.

Unit Identifier Argument:

FUNIT=k
Specifies a unique integer in the range $1 \leqslant k \leqslant 99$.

FUNIT=READ
Specifies READ as the unit identifier.

FUNIT=PUNCH
Specifies PUNCH as the unit identifier.

FUNIT=PRINT
Specifies PRINT 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, PUNCH, and PRINT are provided for reference by the FORTRAN II statements READ, PUNCH, and PRINT, 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 $1 \leqslant k \leqslant 99$. If this argument is omitted and FUNIT=k is specified, FORTk is assumed.

FFILEID=READER
Specifies the file name as READER. If this argument is omitted and FUNIT=READ is specified, READER is assumed.

FFILEID=PUNCH
Specifies the file name as PUNCH. If this argument is omitted and FUNIT=PUNCH is specified, PUNCH is assumed.

FFILEID=PRNTR
Specifies the file name as PRNTR. If this argument is omitted and FUNIT=PRINT is specified, PRNTR is assumed.

File Type Argument:

FTYPEFLE=WORK
Specifies a work file. WORK should be specified if the disk is to be read and written.

FTYPEFLE=INPUT
Specifies an input file. If this argument is omitted and FUNIT=READ is specified, INPUT is assumed. INPUT should be specified if the disk is to be read but never written.

FTYPEFLE=OUTPUT

Specifies an output file. If this argument is omitted and FUNIT=PUNCH or PRINT is specified, OUTPUT is assumed. OUTPUT should be specified if the disk is to be written but not read.

Buffer Size Argument:

FBFSZ=k

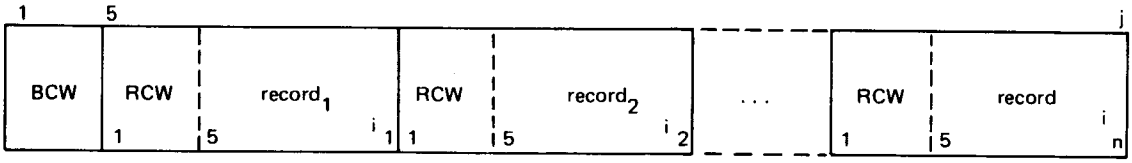
Specifies the size of the input/output area used in processing the file records. The size must be a positive value greater than or equal to the record size. System overhead is reduced if the buffer size is an integer multiple of the record size.

Record Format Argument:

FRECFORM=VARUNB

Specifies that the records are variable and unblocked.

Variable-length unblocked records



where:

i
: Specifies record size

j
: Specifies block size

BCW
: Specifies a data management block control word

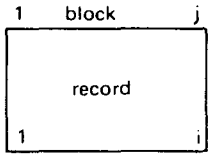
RCW
: Specifies a data management record control word

The FORMAT statement (7.3.3) may not specify a record greater than i–4. For unformatted input/output records, no size limitation exists because large FORTRAN IV records are automatically segmented into multiple data management records via the record control words identifying the start, middle, and the end segment of the I/O list.

The block and record control words are controlled by FORTRAN IV and data management and are not accessible with the FORTRAN IV language. The FBFSZ and FRECSIZE arguments are interpreted as maximums; shorter records are accepted and are generated, if possible, to save space on the external file and reduce channel contention for main storage access.

FRECFORM=FIXUNB
Specifies that the records are fixed and unblocked.

Fixed-length unblocked records



where:

i
Specifies FRECSIZE argument

j
Specifies FBFSZ argument

The block size (j) and the record size (i) must be equal. The FORMAT statement may not require more than i character positions. In an unformatted I/O list, no more than i bytes may be required for a record. When a FORMAT statement processes a record, it cannot request more than "recsize" bytes of data.

Record Size Argument:

FRECSIZE=k
Specifies the logical record size (in bytes). When accessing an existing file, the value is compared to the record size specification of that file. Any incompatibilities produce OPEN errors. If FRECSIZE and FRECFORM are omitted from the UNIT definition procedure, the file is processed using the physical file information. If the file has never been written to, then a 255-byte logical record size and fixed, unblocked records are assumed. ←

Optional Units Argument:

FOPTION=YES
Specifies an optional unit, a unit not always required during program execution. When specified and the file is not allocated by job control, WRITE statements are ignored and the first READ reference causes an end-of-file condition. A unit need not be declared as optional if the program logic does not reference the unit.

Write Verification Argument:

FVERIFY=YES
Specifies that all WRITE statements cause the data to be automatically read back to ensure proper recording on the disk surface.

This increased reliability necessarily causes some performance degradation.

Diagnostic Messages Argument:

FDIAGNOS=YES
Specifies the current disk unit as the diagnostic device. If FRECSIZE is specified, its value must be 101 or greater. Debugging information may also be written to this device. If multiple diagnostic devices are specified, the FORTRAN system will post messages to the first diagnostic unit encountered. This argument is not available for input files. If omitted, diagnostics are transmitted to the system log and either the system console or the initiating workstation terminal.

Spooled Card Input File Argument:

FSPOOLIN=YES
or
FGETJCS=YES
Specifies that this unit will default to a spooled card input file via a GETCS when the lfdname declared in the FFILEID is not found.

The spoolin feature can be applied to any device but cannot exceed 128 bytes. If the record size is omitted, 80 bytes are assumed.

Reread Argument:

FREREAD=YES
Specifies this unit is to participate in the reread feature (7.3.4). The reread unit consists of a single buffer where each formatted input record is transferred. To conserve processor time, this data movement is inhibited unless specified.

Example:

```
1         10    16                                                    72
                                                                  SS
          UNIT  FUNIT=35,                                             X
                FDEVICE=DISK,                                         X
                FFILEID=PAYROL,                                       X
                FTYPEFLE=INPUT
```

This UNIT procedure call specifies a disk file (FDEVICE=DISK). The unit number is 35 (FUNIT=35), the file name is PAYROL, and it is an input file.

The following defaults are assumed by CDM if the file has never been written to:

- The record format (FRECFORM argument) is fixed and unblocked.
- The record size (FRECSIZE argument) is 255 bytes.

11.3.2.4. Workstation Unit Definition

The workstation unit is defined by using the UNIT definition procedure presented in this paragraph. The workstation terminal supports single line input and output (similar to a card reader) and full screen input and output provided by the screen format services. The arguments may appear anywhere in the UNIT definition procedure but FUNIT is the only required argument. Following the format, descriptions of the UNIT arguments and a UNIT example are presented.

Format.

```
1         10        16
          UNIT  [FDEVICE=WORKSTN] FUNIT=(k
                                         {READ }
                                         {PUNCH}
                                         {PRINT})

                [FFILEID=(filename                          )] [FSCREND=(WRAP    )]
                         {FORTk;if FUNIT=k                  }          {SCROLL  }
                         {READER;if FUNIT=READ              }          {NEWPAGE }
                         {PUNCH;if FUNIT=PUNCH              }
                         {PRNTR;if FUNIT=PRINT              }

                [FTYPEFLE=(WORK;                                    )]
                          {INPUT;if FUNIT=READ                      }
                          {OUTPUT;if FUNIT=PUNCH or PRINT           }

                [FIOOPT=YES] [FLINCNTL=YES] [FNUMBUF=(1)]
                                            [        {2}]

                [FOPTION=YES] [FRECSIZE=k] [REREAD=YES]

                [FSPOOLIN=YES] [FDAIGNOS=YES]
                [      or    ]
                [FGETJCS=YES ]
```

Device Identification Argument:

FDEVICE=WORKSTN
Specifies that this is a workstation terminal device.

Unit Identifier Argument:

FUNIT=k
Specifies the unit identifier whose value is a unique integer constant in the range from $1 \leqslant k \leqslant 99$.

FUNIT=READ
Specifies READ as the unit identifier.

FUNIT=PUNCH
Specifies PUNCH as the unit identifier.

FUNIT=PRINT
Specifies PRINT as the unit identifier.

NOTE:

The identifiers READ, PUNCH, and PRINT are provided for reference by FORTRAN II statements READ, PUNCH, and PRINT, respectively, since statements contain no specific unit identification. When a FORTRAN II statement is executed and one of these identifiers is not specified, 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 can 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≤k≤99. If this argument is omitted and FUNIT=k is specified, FORTk is assumed.

FFILEID=READER
Specifies the file name as READER. If this argument is omitted and FUNIT=READ is specified, READER is assumed.

FFILEID=PUNCH
Specifies the file name as PUNCH. If this argument is omitted and FUNIT=PUNCH is specified, PUNCH is assumed.

FFILEID=PRNTR
Specifies the file name as PRNTR. If this argument is omitted and FUNIT=PRINT is specified, PRNTR is assumed.

END of Screen Display Options Argument:

FSCREND=WRAP
Specifies when an end-of-screen is reached; the remainder of the display continues at the top of the screen, which is the leftmost position on the first line. No clearing takes place.

FSCREND=SCROLL
Specifies when an end-of-screen is reached; the screen "scrolls up" to contain the remainder of the display. Scrolling is when the entire screen display moves up one line at a time until the display is completed. No clearing takes place.

FSCREND=NEWPAGE
Specifies when the end-of-screen is reached; the screen is cleared and the remainder of the display continues at the top of the screen origin position, which is the leftmost position on the first line.

File Type Argument:

FTYPEFLE=WORK
Specifies a work file. WORK should be specified if the screen is to be read and written.

FTYPEFLE=INPUT
Specifies an input file. If this argument is omitted and FUNIT=READ is specified, INPUT is assumed. INPUT should be specified if the screen is to be read but never written.

FTYPEFLE=OUTPUT
Specifies an output file. If this argument is omitted and FUNIT=PUNCH or PRINT is specified, OUTPUT is assumed. OUTPUT should be specified if the screen is to be written but not read.

Input/Output Buffer Specification Argument:

FIOOPT=YES
Specifies one buffer is used exclusively for input and one buffer for output operations. This argument is activated when double buffering (FNUMBUF=2) has been specified.

Output Line Control Argument:

FLINCNTL=YES
Allows the workstation terminal, when accessed in single line output mode (WSAM), to perform like a printer. The control characters used to position the output record on the screen are identical to those for the printer (see Table 7-2).

Buffer Allocation Argument:

A buffer pool is dynamically acquired via the squence DMEM macroinstruction. Once a unit is opened, one or two buffers from the pooled storage area are allocated. The size of each buffer is determined by the FBFSZ argument or from data management. (See the consolidated data management macro language user guide/programmer reference.)

FNUMBUF=1
Specifies one buffer to be allocated to the unit.

FNUMBUF=2
Specifies two buffers to be allocated to the unit.

Record Size Argument:

FRECSIZE=k
Specifies the logical size (in bytes). When accessing an existing file, the value is compared to the record size specification on that file. Any incompatibilities produce OPEN errors. If FRECSIZE and FRECFORM are omitted from the UNIT definition procedure, the file is processed using the physical file information.

Reread Argument:

FREREAD=YES
Specifies this unit is to participate in the reread feature (7.3.4). The reread unit consists of a single buffer where each formatted input record is transferred. To conserve processor time, this data movement is inhibited unless specified.

Spooled Card Input File Argument:

FSPOOLIN=YES
or
FGETJCS=YES
Specifies this unit will default to a spooled card input file via a GETCS when the lfdname declared in the FFILEID argument is not found.

The spoolin feature (FSPOOLIN) can be applied to any device but cannot exceed 128 bytes. If the record size is omitted, 80 bytes is assumed.

Diagnostic Messages Argument:

FDIAGNOS=YES
Specifies the current unit as the diagnostic device. If FRECSIZE is specified, its value must be 101 or greater. Debugging information may also be written to this device (10.3).

If multiple diagnostic devices are specified, FORTRAN IV will post the messages to the first diagnostic unit encountered. If omitted, diagnostics are transmitted to the system log and either to the system console or to the initiating workstation terminal.

Example:

```
1         10    16                                                    72
-----------------------------------------------------------------------
          UNIT  FUNIT=20,                                             X
                FFILEID=SEEIT,                                        X
                FDEVICE=WORKSTN,                                      X
                FTYPEFLE=OUTPUT,                                      X
                FSCREND=NEWPAGE,                                      X
                FLINCNTL=YES
```

This UNIT procedure call specifies a workstation terminal (FDEVICE=WORKSTN). FORTRAN IV recognizes that this is a workstation terminal via its unique arguments. The unit number is 20 (FUNIT=20); the file name is SEEIT (FFILEID=SEEIT); and it is an output file (FTYPEFLE=OUTPUT). Since it is a workstation terminal, when the end of screen is reached, the screen is cleared and the remaining information to be displayed starts on the first line at the leftmost position (FSCREND=NEWPAGE). Also, the workstation terminal will function as a printer using the same control characters as in the FORMAT statement in our source program (FLINCNTL=YES).

11.3.2.5. 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, a 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 $1 \leqslant k \leqslant 99$.

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:

```
1         10        16                                                  72
                                                                       X
          UNIT   FDEVICE=REREAD,
                 FUNIT=29
```

The reread unit is defined to be unit 29.

11.3.2.6. 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 a FORTRAN IV 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
                 FUNIT={k     }
                       {READ  }
                       {PRINT }
                       {PUNCH }
                 FEQUIV={k     }
                        {READ  }
                        {PRINT }
                        {PUNCH }
```

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:

```
1         10      16                                                    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.

```
1        10    16                                                72
---------------------------------------------------------------------
         UNIT  FDEVICE=EQUIV,                                      X
               FUNIT=1,                                            X
               FEQUIV=2
                 .
                 .
                 .
         UNIT  FDEVICE=EQUIV,                                      X
               FUNIT=2,                                            X
               FEQUIV=1
```

11.3.3. 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 (and reread buffer) when all record sizes are available; and
- posts warning messages for missing diagnostic unit and imcompatibility of reread units, and FORTRAN II I/O.

Format:

```
1        10    16
--------------------------------
         FUNEND [MAXREC=record-size]
```

where:

MAXREC=record-size
: Increases processing performance by bypassing the dynamic storage allocation mechanism. The record size is the size of the largest record being processed using this input/output configuration. When specified, static allocation of the work area and the reread buffer is automatically performed by the FUNEND processor.

If the FDIAGNOS argument is omitted, the diagnostics are sent to the system log and either the system console or the initiating workstation terminal. If the REREAD=YES argument is specified and no reread unit is defined, unit 29 is provided as the reread unit. This default is only supplied if unit 29 has not been previously defined.

11.3.4. Error Environment Definition Procedure (ERRDEF)

During the execution of the object program, the 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 mesages to be produced.

The seven error classes include program, arithmetic, argument, alignment, read, and data errors, explained in the definitions following the format, 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[FROG=((i|ALL|1),(j|ALL|1))]
                 [FARITH=((i|ALL),(j|ALL|10))]
                 [FARG=((i|ALL),(j|ALL|10))]
                 [FUNDFLO=YES]
                 [FALIGN=((i|ALL),(j|ALL|10))]
                 [FREAD=((i|ALL),(j|ALL|10))]
                 [FDATA=((i|ALL),(j|ALL|10))]
                 [FERROPT=((NONE|READ|READ,DATA|READ,UNREC|READ,DATA,UNREC|DATA|DATA,UNREC|UNREC))]
```

where:

i

Is a positive integer constant less than 32,768, with i≥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≤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 5–4) 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.

This argument also controls the acceptance and printing of subscript checking. (See 10.5.)

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 or 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 FORTRAN IV 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 contol 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:

```
1         10        16                                                  72
          ERRDEF  FPROG=(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.

11.3.5. 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 the FUNEND procedure call.

11.4. TYPICAL CONFIGURATION EXAMPLE

The following coding example defines a typical configuration that can be used by a FORTRAN IV program:

```
     1         10    16                                                  72
1.   // JOB FUNDEFS
2.   // DVC 20 // LFD PRNTR
3.   // DVC 50 // VOL D99999 // LBL INPUTF // LFD DATA
4.   // DVC 40 // LFD PUNCH
5.   // DVC 200 // LFD STN
6.   // WORK1
     // WORK2
7.   // EXEC ASM
8.   /$
9.   USERID    START
10.            UNIT   FUNIT=PRINT,                                      X
11.                   FRECSIZE=121
12.            UNIT   FUNIT=63,                                         X
13.                   FFILEID=DATA,                                     X
14.                   FTYPEFLE=INPUT
15.            UNIT   FDEVICE=UNITREC,                                  X
16.                   FUNIT=PUNCH,                                      X
17.                   FRECSIZE=80,                                      X
18.                   FCRDERR=RETRY
19.            UNIT   FUNIT=3,FFILEID=STN,                              X
20.                   FSCREND=NEWPAGE
21.            FUNEND
22.            ERRDEF
23.            END
24.  /*
25.  /&
26.  // FIN
```

Explanations:

1. Indicates the job name FUNDEFS.
2. Indicates the device assignment set for the printer.
3. Indicates the device assignment set for the disk.
4. Indicates the device assignment set for the card punch.
5. Indicates the device assignment set for the workstation.
6. Specifies two assembler work files.
7. Begins the assembler execution.
8. Specifies the start of data to the assembler.
9. Specifies the start of the execution module.
10. Defines the FORTRAN II print file since no FFILEID argument is specified. FORTRAN IV defaults to PRNTR (see line 2).

11. Defines the record size.

12. Starts the definition for unit 63.

13. Indicates a disk file since the filename DATA is the same as the lfdname associated with the disk device (see line 3).

14. Indicates an input file.

15. Starts the definition for a unit record device.

16. Defines the device as PUNCH.

17. Indicates the logical record size of 80 bytes.

18. Indicates retry error recovery is attempted.

19. Starts the definition for a device (in this instance, a workstation since the filename STN is the same as the lfdname for DVC job control statement, 200).

20. Indicates when the workstation end-of-screen is reached; the screen is cleared and the remaining screen display starts at the leftmost position on the first line.

21. Terminates the UNIT definition procedures.

22. Creates the error control table in the executable program.

23. Terminates the source program.

24.
25. } Indicates end-of-job.
26.

12. Define the File (DTF) Execution Environment Configuration

12.1. DATA MANAGEMENT INTERFACE

This section describes the interface between FORTRAN IV and the data management system in a DTF environment 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 also are described. An example of a complete execution environment is given in 12.3.6.

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 FORTRAN IV library (12.2). For more complex programs, the control module must be configured, using the FORTRAN IV unit definition procedures (UNITs). Only one control module can exist in an executable program.

12.2. DTF-SUPPLIED CONFIGURATIONS

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

- Control Module FL$IO

Unit	Device	Notes
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

Unit	Device	Notes
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

12.3. PROGRAMMER-DEFINED CONFIGURATIONS

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

```
                                                    Continuation
1          10                                                  72
name       START
           file initialization
           file definition1
           file definition2
                  .
                  .
                  .
           file definitionn
           file termination
           error definition
           END
```

Each element of the assembly module is discussed in detail in 12.3.1 through 12.3.7. For an explanation of the statement conventions applicable to this section refer to 1.4.

12.3.1. File Definition Conventions

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

12.3.1.1. 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 FORTRAN system assumes the card length to be 80, unless the user specifies otherwise.

File support provided by FORTRAN IV is largely device independent. The user need not be concerned with whether the device is a UNISERVO I-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.

12.3.1.2. Record and Block Sizes

Record and block sizes are specified by the two optional arguments FRECSIZE and FBKSZ. 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 disk devices and must always be greater than or equal to the record size.

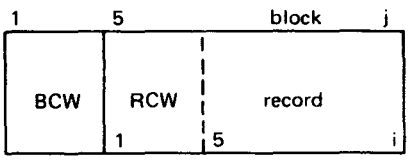
12.3.1.3. Record Formats

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.

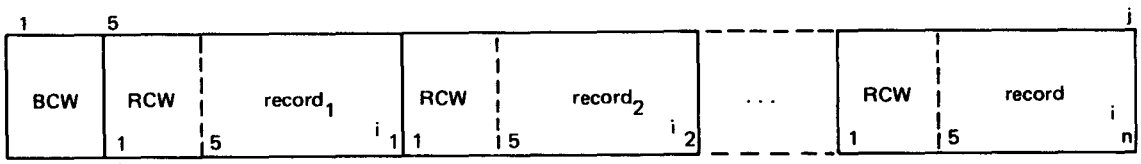
- Variable-Length Records

  Formats for variable-length unblocked and blocked records follow.

  a. Unblocked Records (VARUNB)



  b. 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.

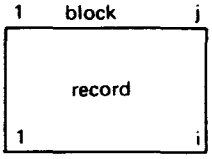
The FORMAT statement (7.3.3) may not specify a record larger than i-4 for variable-length records. For unformatted input/output, no size limitation exists, since large FORTRAN records are automatically segmented into multiple data management records, using the record control words to identify beginning,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

The BCW and RCW are controlled by FORTRAN IV 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.

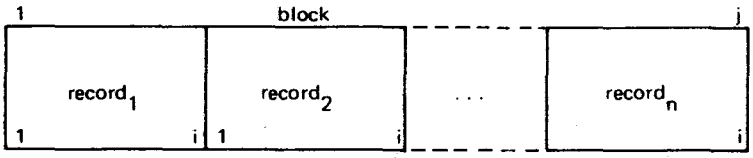
- Fixed-Length Records

Formats for fixed-length unblocked and blocked records follow.

a. Unblocked Records (FIXUNB)



b. Blocked Records (FIXBLK)



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 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.

12.3.1.4. 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 descretion, 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. On output this 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 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 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. A work area cannot be used if buffer pooling is selected.

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 disk 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. BACKSPACE requires this configuration. There is no overlap between computational and I/O activity.

- One buffer, work area, no buffer pooling

  This is the usual FORTRAN IV 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.

12.3.1.5. 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 disk 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.

12.3.2. 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 FUNTAB procedure call.

12.3.3. FORTRAN Initialization Procedure (FUNTAB)

The FUNTAB statement follows the START statement and precedes all other statements. It initializes Basic FORTRAN, FORTRAN IV, or Extended FORTRAN parameters needed by statements which follow. To initialize the FORTRAN IV parameters, the FUNTAB statement is coded as follows:

Format:

```
1         10     16
          FUNTAB SYS=FOR
```

NOTE:

Omitting the SYS=FOR operand from the FUNTAB call initializes basic FORTRAN parameters.

12.3.4. 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. In FORTRAN, it is required in lines 2 though n in column 6.
- No embedded blanks are permitted, and all continuation lines must start in column 16 (as illustated in following examples).

Format:

```
                                                              Continuation
1        10     16                                                      72
---------------------------------------------------------------------------
         UNIT   n1=c1,                                                   X
                n2=c2,                                                   X
                 .                                                       .
                 .                                                       .
                 .                                                       X
                nn=cn
```

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.

12.3.4.1. Printer File Definition

A single printer file is defined by using the UNIT procedure call shown in the following format. Listed in order of relative importance and utility are the arguments that may appear on this UNIT procedure call. Descriptions of the UNIT arguments and a UNIT example also are presented. Work areas and buffer pooling are not supported for printers. The default number of buffers is 2.

Format:

```
UNIT   FDEVICE=PRINTER
       FUNIT={k    }
             {PRINT}
             {PUNCH}
       [FFILEID={filename                  }]
       [        {FORTk; if FUNIT=k         }]
       [        {PRNTR; if FUNIT=PRINT     }]
       [        {PUNCH; if FUNIT=PUNCH     }]
       [FRECSIZE={k  }]
       [         {121}]
       [FNUMBUF={1}]
       [        {2}]
       [FDIAGNOS=YES]
       [FPRINTOV={SKIP  }]
       [         {NOSKIP}]
       [FCHAR={OFF}]
       [      {ON }]
       [FOPTION=YES]
```

Device Identification Argument:

FDEVICE=PRINTER
: Specifies this is a printer file.

Unit Identifier Argument:

FUNIT=k
Specifies a unique integer constant (k) in the range 1≤k≤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 to 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≤k≤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.

FILEID=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={k / 121}
Specifies a positive integer constant (k), in the range 1≤k≤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 FORTRAN IV 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 (103). 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:

```
1         10    16                                                  72
          UNIT  FDEVICE=PRINTER,                                    X
                FUNIT=10,                                           X
                FRECSIZE=101,                                       X
                FDIAGNOS=YES,                                       X
                FPRINTOV=NOSKIP
```

A printer is defined (unit 10) with 100 printable characters per line. It is also to be used for diagnostic purposes. No automatic forms overflow is to take place; 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.

12.3.4.2. Card Input File Definition

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 disk file before the executable program is activated. When a card image is requested by the program, the operating system reads the card image from disk 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 12.3.4.2.1 and 12.3.4.2.2.

12.3.4.2.1. Spooled Card Input File Definition

A spooled card input file is defined by using the UNIT procedure call. Following the format is a listing, in order of 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={k   }
                      {READ}
                [FREREAD=YES]
                [FBKSZ={k   }]
                       {400 }
                [FBUFPOOL=YES]
                [FRECSIZE={k  }]
                          {80 }
```

Device Identification Argument:

FDEVICE=SPOOLIN
Specifies that this is a spooled card input file.

Unit Identifier Argument:

FUNIT=k
Specifies a unique integer constant in the range $1 \leqslant k \leqslant 99$.

FUNIT=READ
Specifies READ as the unit identifier.

A maximum of 102 unique unit identifiers (values 1 to 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={k / **400**}
Specifies a positive integer constant (k) this is an integral multiple of FRECSIZE. A large multiple of FRECSIZE reduces operating system overhead. The default block size is the largest integral multiple of FRECSIZE that is less than or equal to 400. If a number is specified that is not an integral multiple of FRECSIZE, the block size is rounded downward to the nearest multiple.

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 be 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 FORTRAN IV 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:

```
1         10    16                                                  72
                                                                    5 5
          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.

12.3.4.2.2. Data Management Card Input File Definition

A single data management card or 8413 diskette input file is defined by using the UNIT procedure call in this format. Following the format is a listing, in the order of 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 $1 \leq k \leq 99$.

FUNIT=READ
Specifies READ as the unit identifier.

A maximum of 102 unique unit identifiers (values 1 to 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 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.

FRECSIZE=$\left\{\begin{matrix} k \\ \underline{80} \end{matrix}\right\}$
Specifies a positive integer constant (k) in the range $1 \leqslant k \geqslant 128$.

If 96-column cards are to be read, 96 must be specified. If an 8413 diskette is to be read, the record size must correspond to that actually recorded on the device. If this argument is omitted, 80 is the default record size.

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 block size for an 8413 diskette.

FBKSZ={k / FRECSIZE}

Specifies a positive integer constant (k≤1024) that should be an integral multiple of FRECSIZE. A large multiple of FRECSIZE reduces operating overhead. The default block size is FRECSIZE. 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. Operator intervention is sought when an illegal hole combination is detected.

Example:

```
1         10    16                                                    72
          UNIT  FDEVICE=CARDIN,                                       X
                FUNIT=READ,                                           X
                FNUMBUF=2,                                            X
                FRECSIZE=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.

12.3.4.3. Card Output File Definition

A single card or 8413 diskette output file is defined by using the UNIT procedure calls presented in this format. Following the format is a listing, in the order of 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 to 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 $1 \leqslant k \leqslant 99$. 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 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:

FRECSIZE={k / 80}

Specifies a positive integer constant (k) in the range 1≤k≤128.

If this argument is omitted, 80 is the default record size.

Block Size Argument:

This argument specifies the block size for an 8413 diskette.

FBKSZ={k / FRECSIZE}

Specifes a positive integer constant (k≤1024) that should be an integral multiple of FRECSIZE. A large multiple of FRECSIZE reduces operating overhead. The default block size is FRECSIZE. 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 disk 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 the job control statement, WRITE statements are effectively ignored.

A unit need not be declared optional if the logic of the program does not cause a reference to the unit.

Example:

```
1         10    16                                                    72
          UNIT  FDEVICE=CARDOUT,                                      X
                FUNIT=32,                                             X
                FBUFPOOL=YES,                                         X
                FCRDERR=RETRY
```

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.

12.2.4.4. Tape File Definition

A single tape file is defined by using the UNIT procedure call presented in this format. Following the format is a listing, in the order of 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 }
                [FNUMBUF={1}]
                         {2}
                [FWORKA={YES;if FNUMBUF=1}]
                        {NO ;if FNUMBUF=2}
                [FBUFPOOL=YES]
                [FRECSIZE={k  }]
                          {508}
                [FBKSZ={k                             }]
                       {FRECSIZE;if FRECFORM=FIXUNB   }
                       {FRECSIZE+4;if FRECFORM=VARUNB }
                       {FRECSIZE*4;otherwise          }
                [FREREAD=YES]
                [FDIAGNOS=YES]
                [FBKNO=YES]
                [FERROPT={IGNORE}]
                         {SKIP  }
                [FCKPT=YES]
                [FFILABL={STD}]
                         {NO }
                [FCKPTREC=YES]
                [FCLRW={RWD   }]
                       {NORWD }
                       {UNLOAD}
                [FOPRW=NORWD]
                [FOPTION=YES]
```

Device Identification Argument:

FDEVICE=TAPE
Specifies that this is a tape file;

Unit Identifier Argument:

FUNIT=k
SDpecifies 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 to 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. BACKSPACE is not allowed if this option is specified.

FRECFORM=FIXUNB
Specifies fixed-length unblocked records.

FRECFORM=FIXBLK
Specifies fixed-length blocked records, BACKSPACE is not allowed if this option is specified.

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 $14 \leq k \leq 32727$ if variable records are specified. If this argument is omitted, 508 is the default record size.

FORTRAN IV 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 $18 \leqslant k \leqslant 32767$.

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:

```
1         10        16                                                  72
                    FRECFORM=VARBLK,                                    X
                    FRECSIZE=1800
```

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 (10.3). This argument is not available for input files.

The FORTRAN IV 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 statment; 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 (5.6.3.3) 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 lengths 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 (12.3.4.6). Refer also to the ERRDEF procedure (12.3.6).

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:

```
1         10    16                                                  72
          UNIT  FDEVICE=TAPE,                                       X
                FUNIT=7,                                            X
                FTYPEFILE=INPUT,                                    X
                FRECFORM=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.

Because a work area is requested, BACKSPACE will not be allowed.

12.3.4.5. Files on Disk

FORTRAN supports three separate types of files on disk: sequential disk, direct access disk, and combined sequential-direct disk.

- SAM

  Sequential disk uses the disk sequential access method (SAM) and uses a file similar to a tape file. Only sequential I/O statements are allowed.

- DAM

  Direct access disk uses the direct access method (DAM) and creates a file of fixed-size records referenced by relative record number only.

- MIRAM

  Combined sequential direct disk uses the multi-indexed random access method (MIRAM).

Because of its flexibility and effeciency the MIRAM file is recommended for most applications.

12.3.4.5.1 Sequential Disk File Definition

A single sequential disk file is defined by using the UNIT procedure call presented in this format. Following the format is a listing, in the order of 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 disk files are conceptually identical with tape files. Most arguments are treated identically with tape file arguments.

Format:

```
1         10      16
          UNIT    FDEVICE=SDISC
                  FUNIT=(k
                         {READ
                         (PUNCH)
                  [FSECTOR={NO }]
                           {YES}
                  [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}
                  [FNUMBUF={1}]
                           {2}
                  [FBUFPOOL=YES]
                  [FWORKA={YES;if FNUMBUF=1}]
                          {NO ;if FNUMBUF=2}
                  [FRECSIZE={k   }]
                            {508}
                  [FBKSZ=(k                                   )]
                         {FRECSIZE;if FRECFORM=FIXUNB         }
                         {FRECSIZE+4;if FRECFORM=VARUNB       }
                         {FRECSIZE*4;otherwise                }
                  [FREREAD=YES]
                  [FDIAGNOS=YES]
                  [FERROPT={IGNORE}]
                           {SKIP  }
                  [FRECERR=YES]
                  [FOPTION=YES]
                  [FVERIFY=YES]
```

Device Identification Argument:

FDEVICE=SDISC
Specifies that this is a sequential disk 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 to 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 disk is expected (e.g., an 8416). FSECTOR parameter is valid for all file types: input, output, or work. When specified, the FORTRAN IV 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 disk is expected. This will conserve space.

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 the argument is not specified and FUNIT=k has been specified, WORK is the FTYPEFLE default. FTYPEFLE=WORK should be specified if the disk 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 disk 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 disk 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 disks 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:

FRECSIZE={k / **508**}
Specifies the record size as a postive integer constant (k). If this argument is omitted, 508 is the default record size. See the specific disk 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 $3 \leqslant k \leqslant 3625$. The upper limit can be increased to 7294 bytes for SPERRY UNIVAC 8414/8424/8425 Disk Drive Units and to 13030 bytes for SPERRY UNIVAC 8430 Disk 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:

```
1         10    16
-------------------------------
                FRECFORM=VARBLK
                FRECSIZE=1800
```

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 (10.3). This argument is not available for input files.

The FORTRAN IV 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 (5.6.3.3) 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 (12.3.4.6). Refer also to the ERRDEF procedure (12.3.6).

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 disk is expected (e.g., an 8416 or 8418). FSECTOR parameter is valid for all file types: input, output, or work. The FORTRAN IV 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 disk is expected. This will conserve space.

Write Verification Argument:

FVERIFY=YES
Specifies that all WRITE statements cause the data to be automatically read back to ensure proper recording on the disk surface.

This increased reliability necessarily causes some performance degradation.

Example:

```
1         10    16                                                    72
                                                                  55
          UNIT  FDEVICE=SDISC,                                        X
                FUNIT=9,                                              X
                FTYPEFLE=OUTPUT,                                      X
                FRECSIZE=1000,                                        X
                FVERIFY=YES
```

This FUNDEF procedure call specifies a sequential disk 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.

12.3.4.5.2. Direct Access Disk File Definition

A direct access disk file is defined by using the UNIT procedure call presented in this format. Following the format is a listing, in the order of relative importance and utility, of the arguments that may appear on the FUNDEF procedure call. Following the listing, descriptions of UNIT arguments, programming considerations, and a UNIT example are presented.

Direct access disk file options are treated identically with sequential disk file options. Records are fixed length and unblocked.

Format:

```
1         10      16
          UNIT    FDEVICE=DISC
                  FUNIT=k
                  [FSECTOR={NO  }]
                           {YES }
                  [FFILEID={filename                }]
                           {FORTk;where k=FUNIT     }
                  [FTYPEFLE={INPUT }]
                            {OUTPUT}
                  [FBUFPOOL=YES]
                  [FRECSIZE={k   }]
                            {512 }
                  [FREREAD=YES]
                  [FRECERR=YES]
                  [FVERIFY=YES]
```

Device Identification Argument:

FDEVICE=DISC
Specifies that this is a direct access disk file.

Unit Identifier Argument:

FUNIT=k
Specifies a unique integer constant (k) in the range $1 \leqslant k \leqslant 99$.

A maximum of 102 unique unit identifiers (values 1 to 99 and READ, PRINT and PUNCH) may be specified by a control module.

Sector Processing Argument:

FSECTOR=YES
Specifies that processing on a sectorized disk is expected (e.g., an 8416 or 8418) FSECTOR parameter is valid for all file tapes: input, output, or work. The FORTRAN IV 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 disk is expected. This will conserve space.

File Name Argument:

FFILEID=filename
Specifies a 1- to 7-character FORTRAN style symbolic name.

FFILEID=FORTk
Specifies the file name as FORTk, where $1 \leqslant k \leqslant 99$. 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.

NOTE:

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 disk 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 (12.3.4.6). Refer also to the ERRDEF procedure (12.3.6).

Write Verification Argument:

FVERIFY=YES
Specifies that all WRITE statements cause the data to be automatically read back to ensure proper recording on the disk 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 FORTRAN system thereafter enforces the record size specified on the DEFINE FILE statement, but always transfers records of FRECSIZE bytes to and from the disk.

Example:

```
1         10    16                                                  72
                                                                    S S
          UNIT  FDEVICE=DISC,                                       X
                FUNIT=13,                                           X
                FTYPEFLE=INPUT,                                     X
                FRECSIZE=348
```

This UNIT procedure call specifies a direct access disk 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.

12.3.4.5.3. Combined Disk Files

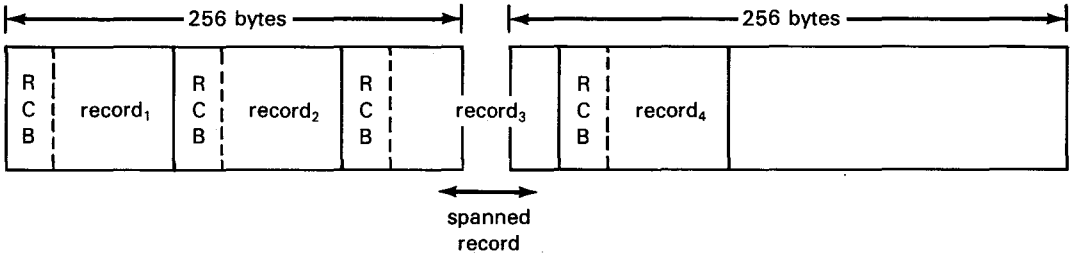
MIRAM disk files may be created and accessed as either sequential or direct access files. Since the type of access is not included in the file definition, the same definition may be used for both. Within each FORTRAN program, however, only one access method may be used for a file.

If direct access is chosen, the FORTRAN user must use the proper I/O statements including DEFINE FILE (7.4.1). The direct access FIND statement is ignored because it serves no purpose for MIRAM files. A read of a record that was not written causes a fatal error.

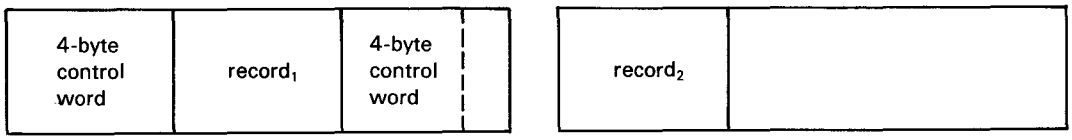
12.3.4.5.3.1. Record Formats for MIRAM Disk Files

Data records are maintained in fixed-length slots. All records in a MIRAM file are created with a record control byte (RCB). For fixed-length records, slot size is one byte greater than record size because of the RCB. For variable-length records, the slots are equal to the maximum record size plus a 4-byte overhead.

- Fixed-Length Records (FIX)



- Variable-Length Records (VAR)



By recording the data records in consecutive order and associating a relative record slot number with each record, files may be accessed by either sequential or relative record techniques.

The FORTRAN user does not specify a parameter for buffer size. The size is determined in the unit procedure as follows:

- If the slot length is less than or equal to 256 bytes and is evenly divisible into 256, the buffer size is 256.
- If the slot length is greater than 256 and a multiple of 256, the buffer size equals the slot length.
- Otherwise, the buffer size is calculated by adding 255 to the slot length and rounding up to the next multiple of 256.

A work area is always allocated and its size is equal to the slot length.

12.3.4.5.3.2. MIRAM Disk File Definition

A MIRAM disk file is defined by using the UNIT procedure call presented in this format. The listing that follows gives the arguments, in order of relative importance and utility, that may appear on the UNIT procedure call. Following the listing are descriptions of UNIT arguments and a UNIT example.

Format:

```
1         10    16
          UNIT  FDEVICE=MIDISC
                FUNIT={k     }
                      {READ  }
                      {PUNCH }
                [FFILEID= {filename                    }]
                          {FORTk;if FUNIT=k            }
                          {READER;if FUNIT=READ        }
                          {PUNCH;if FUNIT=PUNCH        }
                [FTYPEFLE= {WORK;if FUNIT=k          }]
                           {INPUT;if FUNIT READ      }
                           {OUTPUT;if FUNIT=PUNCH    }
                [FRECFORM={FIX}]
                          {VAR}
                [FBUFPOOL=YES]
                [FRECSIZE= {k                        }]
                           {255;if FRECFORM=FIX      }
                           {252;if FRECFORM=VAR      }
                [FREREAD=YES]
                [FRECERR=YES]
                [FOPTION=YES]
                [FVERIFY=YES]
```

Device Identification Argument:

FDEVICE=MIDISC
Specifies that this is a MIRAM disk file.

Unit Identifier Argument:

FUNIT=k
Specifies a unique integer constant (k) in the range $1 \leqslant k \leqslant 99$.

FUNIT=READ
Specifies READ as the unit identifier.

A maximum of 102 unique unit identifiers (values 1 to 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 $1 \leqslant k \leqslant 99$. 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
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 disk 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 disk 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 disk is to be written but never read.

Record Format Argument:

FRECFORM=FIX
Specifies fixed-length records.

FRECFORM=VAR
Specifies variable-length records.

Buffer Pooling Argument:

FBUFPOOL=YES
Specifies that buffer pooling is to be used.

The buffers are to be 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 and on output means the execution of an ENDFILE statement.

Record Size Argument:

FRECSIZE=$\left(\begin{matrix}k\\255\\252\end{matrix}\right\}$

Specifies the record size as a positive constant (k). If the argument is omitted and FRECFORM=FIX, 255 is the default record size. If FRECFORM=VAR, 252 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.

Devcie Error Processing Argument:

FRECERR=YES

Specifies that formatted records in blocks with parity errors or wrong length 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 (12.3.4.6). Refer also to the ERRDEF procedure (12.3.6).

Optional Units Argument:

FOPTION=YES

Specifies an optional unit (i.e., 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 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.

Write Verification Argument:

FVERIFY=YES

Specifies that all WRITE statements cause the data to be automatically read back to ensure proper recording on the disk surface.

This increased reliability necessarily causes some performance degradation.

Example:

```
1         10        16                                                  72
          UNIT   FDEVICE=MIDISC,                                        X
                 FUNIT=8,                                               X
                 FRECFORM=VAR,                                          X
                 FRECSIZE=300
```

This FUNDEF procedure call specifies a MIRAM disk file (unit 8) intended for input and output of variable-length records with a maximum length of 300 bytes and a record slot length of 304 bytes.

The defaults are:

- file name is FORT8;
- file is a work file available for input and output;
- one buffer allocated;
- no buffer pooling;
- no reread facility (i.e., records are not available in reread buffer after ERR returns);
- not an optional unit; and
- no verification

A work area is automatically allocated and the buffer size is 768 bytes. If the FORTRAN user wants to access the file using direct access, a define file statement must be included in the FORTRAN program. Otherwise, sequential access is assumed.

12.3.4.6. Reread Unit Definition

The reread unit is defined by using the UNIT procedure call presented in this format. Following the format is a listing, in the order of 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 $1 \leqslant k \leqslant 99$.

FUNIT=READ
Specifies READ as the unit identifier.

A maximum of 102 unique unit identifiers (values 1 to 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:

```
1         10        16                                                  72
                                                                        X
          UNIT   FDEVICE=REREAD,
                 FUNIT=29
```

The reread unit is defined to be unit 29.

12.3.4.7. Equivalent Unit Definition

An equivalent unit is defined by using the UNIT procedure call presented in this format. Following the format is a listing, in the order of 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 a FORTRAN IV 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
                 FUNIT=(k      )
                       (READ   )
                       (PRINT  )
                       (PUNCH  )
                 FEQUIV=(k     )
                        (READ  )
                        (PRINT )
                        (PUNCH )
```

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 $1 \leqslant K \leqslant 99$.

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 to 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 $1 \leqslant k \leqslant 99$.

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:

```
1         10    16                                                  72
                                                                    S S
          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.

Example:

Circular equivalence, such as the following, is not permitted.

```
1         10    16                                                    72
          UNIT  FDEVICE=EQUIV,                                        X
                FUNIT=1,                                              X
                FEQUIV=2
                  .
                  .
                  .
          UNIT  FDEVICE=EQUIV,                                        X
                FUNIT=2,                                              X
                FEQUIV=1
```

12.3.5. 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.

```
1         10    16                                                    72
          UNIT  FDEVICE=PRINT,                                        X
                FUNIT=PRINT,                                          X
                FDIAGNOS=YES,                                         X
                FRECSIZE=101
```

Format:

```
1         10    16
          FUNEND
```

12.3.6. Error Environment Definition Procedure (ERRDEF)

During the execution of the object program, the 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 [FPROG=((i|ALL|1),(j|ALL|1))]
                 [FARITH=((i|ALL),(j|ALL|10))]
                 [FARG=((i|ALL),(j|ALL|10))]
                 [FUNDFLO=YES]
                 [FALIGN=((i|ALL),(j|ALL|10))]
                 [FREAD=((i|ALL),(j|ALL|10))]
                 [FDATA=((i|ALL),(j|ALL|10))]
                 [FERROPT=((NONE|READ|READ,DATA|READ,UNREC|READ,DATA,UNREC|DATA|DATA,UNREC|UNREC))]
```

where:

i

Is a positive integer constant less than 32,768, with i≥k that specifies 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≤i that specifies 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 terminatnion 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 5–4) 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.

This argument also controls the acceptance and printing of subscript checking. (See 10.5.)

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 FORTRAN IV default is to pass control to the ERR label only for parity or wrong length errors. The ERROR subroutine (5.6.3.3) 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:

```
1         10    16                                                  72
          ERRDEF FPROG=(100,100),                                    X
                 FALIGN=(ALL,50),                                    X
                 FERROPT=(DATA)
```

This ERRDEF procedure call indicates that if a program error occurs, the error may be acepted 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.

12.3.7. 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    16
          END
```

The END statement, coded as shown, follows the ERRDEF procedure call, or the FUNEND procedure call.

Example:

An example of a complete execution environment follows.

```
1         10    16                                                    72
MYIO      START
          FUNTAB SYS=FOR
          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
```

13. Program Collection and Execution

13.1. GENERAL

Before a set of program units can be executed, they must be collected and the necessary FORTRAN supporting routines made available to them. The linkage editor performs this task.

After the program units are link edited, all physical devices required for execution are assigned via job control statements.

13.2. LINK EDITING FORTRAN PROGRAMS

Several special interfaces of the linkage editor are used by FORTRAN IV 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 OS/3 system service programs (SSP) user guide can be used only if they do not conflict with requirements of FORTRAN IV. Also, the linkage editor jproc call described in the current version of the OS/3 job control user guide shows an easy method for executing the linkage editor.

13.2.1. FORTRAN IV 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 that may need to be supplied before a FORTRAN program is executable.

A complete list of functions and services supplied with FORTRAN IV are included in Appendix G.

Because of the special conventions used in generating references to the FORTRAN standard library subroutines (5.6.3), a user program that attempts to override the supplied routine must:

1. specifically include the module; and

2. equate the special name to the real name.

For example, if the program used the following coding to call a special error subroutine:

```
1      7
-------------------------------------------------------------------
          .
          .
          .
       CALL ERROR(I,K,L)
          .
          .
          .
       SUBROUTINE ERROR(I1,J1,K1)
          .
          .
          .
```

then the control stream to link the load modules of the program would require the following coding:

```
1          10     16
-------------------------------------------------------------------
              .
              .
              .
           INCLUDE ERROR,input file
E$ROR EQU ERROR
              .
              .
              .
```

13.2.2. 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 linkage editor provides overlay and region segmentation methods to assist in creating smaller executable load modules.

Programs compiled by the FORTRAN IV compiler reference each subprogram with the automatic overlay feature of the 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$ERCTL (FD$IOCOM, FD$ABTRM, FD$ERCTL for DTF)
  - The main program of the execution
  - The module FF#MPI
- 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.

13.2.3. Linkage Editor Output

The executable module created by the linkage editor is placed in the system file, YRUN. The program may be executed directly from this file, or it may be saved with the system librarian.

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 linkage editor can be executed either via the LINK job control procedure call statement (jproc) or using the conventional device assignment set and EXEC statement. It is recommended that LINK jproc be used since it requires less coding. For example:

- jproc Method

```
//TEST LINK $MAIN,MYIO
```

- Conventional Method

```
// WORK1
// EXEC LNKEDT
/$
LOADM TEST
INCLUDE $MAIN
INCLUDE MYIO
/*
```

Both methods cause the FORTRAN IV program ($MAIN) and the execution module (MYIO) to be linked to an executable program (TEST). All supporting FORTRAN IV run-time modules needed by $MAIN and MYIO programs are automatically included into TEST from the system object module library (YOBJ).

13.3. EXECUTION OF FORTRAN PROGRAMS IN A CDI ENVIRONMENT

The FORTRAN IV compiler uses the operating system and the common data interface to execute its compiled programs. The following information describes the various interfaces that FORTRAN IV requires.

13.3.1. CDI 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 that 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 that allocate the device for this job or job step. When a device is not used during an execution of a program, the device need not be assigned.

13.3.2. CDM Pause Messages

If a PAUSE statement is executed, the text of the PAUSE message is displayed on the initiating workstation terminal, if present, or else 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.

13.3.3. CDM Diagnostic Messages

The FORTRAN run-time system has many diagnostic messages that may be displayed during execution. These messages are output to the FORTRAN unit assigned for diagnostic information (FDIAGNOS=YES). If no diagnostic device is specified, the messages are simultaneously written to the system log and the initiating workstation terminal. In lieu of an initiating workstation terminal, the messages are sent to the system console.

The amount of information output by the FORTRAN run-time system may be controlled by the error definition procedure (ERRDEF). 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 copy printer. For a complete list of run-time diagnostics, refer to the system messages programmer/operator reference.

Diagnostic messages that can be generated during compilation are listed and described in Appendix F.

13.4. EXECUTION OF FORTRAN PROGRAMS IN A DTF ENVIRONMENT

The FORTRAN IV compiler uses the operating system and the data management system to execute its compiled programs. The following information describes the various interfaces that FORTRAN IV requires.

13.4.1. DTF 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 that 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 argument.

The FORTRAN diagnostic device must always be allocated to the executing program. In all FORTRAN IV 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.

13.4.2. DTF 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.

13.4.3. DTF Diagnostic Messages

The FORTRAN run-time system has many diagnostic messages that may be displayed during execution. These messages are output to the FORTRAN unit assigned for diagnostic information (FDIAGNOS=YES).

The amount of information output by the FORTRAN run-time system may be controlled by the error definition procedure (ERRDEF). 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 copy printer. For a complete list of run-time diagnostics, refer to the system messages programmer/operator reference.

Diagnostic messages that can be generated during compilation are listed and described in Appendix F.

PART 4. APPENDIXES

PART A. APPENDIXES

Appendix A. Character Set

A.1. SOURCE PROGRAM AND INPUT DATA CHARACTERS

Table A-1 shows the EBCDIC input character set for FORTRAN IV. Binary bit positions 1 through 3, along with their hexadecimal equivalents, are read at the left of the grid. Binary bit positions 4 through 7, along with their hexadecimal equivalents, are read from left to right at the top of the grid. The point at which these coordinates intersect represents the value of the corresponding EBCDIC graphic character as it appears on the keyboard of a SPERRY UNIVAC 1700 Series keypunch.

When two hexadecimal or binary number coordinates intersect showing no equivalent EBCDIC graphic character, this means the SPERRY UNIVAC 1004 Card Processor changes these hexadecimal or binary values to the character blank (hexadecimal 40). For example, the hexadecimal value C1 is encoded with the keyboard character A; the hexadecimal value 77 produces a blank.

Table A-2 lists EBCDIC graphic characters, their Hollerith punched card code equivalents, and decimal, hexadecimal, and binary equivalents.

A.2. PRINTER GRAPHICS

Many different printer devices are supported, each with subtly different character sets. The character sets vary in size from 16 to 94 characters to accommodate differing national languages and the needs of various applications. Internal representations of the character set may differ due to translations performed by using a load code, a translation table within the printer control unit.

There is a difference between the EBCDIC graphic character set for card punched input and EBCDIC printer graphics because output graphic character sets vary according to printer models.

Table A-3 shows a representative character set and its internal hexadecimal representation. Special characters and lowercase alphabetics may differ due to the printer model, the features installed, and the load code in use. These features should be checked to ensure availability, and the table should be updated to reflect installation usages.

Table A—1. EBCDIC Input Graphic Character Set

HEXA-DECIMAL →		4567															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
↓	0123	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000																
1	0001																
2	0010																
3	0011																
4	0100	(Space)										¢	.	<	(	+	|
5	0101	&										!	$	*	)	;	¬
6	0110	(minus)	/										(comma)	%	(underline)	>	?
7	0111											:	#	@	(apostrophe)	=	(quote mark)
8	1000																
9	1001																
A	1010																
B	1011																
C	1100	{	A	B	C	D	E	F	G	H	I						
D	1101	}	J	K	L	M	N	O	P	Q	R						
E	1110			S	T	U	V	W	X	Y	Z						
F	1111	0	1	2	3	4	5	6	7	8	9						

Bit Positions: 0 1 2 3 4 5 6 7

Weight: 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

Table A—2 lists EBCDIC graphic characters, their Hollerith punched card code equivalents, and decimal, hexadecimal, and binary equivalents.

Table A—2. EBCDIC/Hollerith Cross-Reference Table (Part 1 of 5)

EBCDIC				
Decimal	Hexa-deci-mal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code
0	00	0000 0000		12-0-9-8-1
1	01	0000 0001		12-9-1
2	02	0000 0010		12-9-2
3	03	0000 0011		12-9-3
4	04	0000 0100		12-9-4
5	05	0000 0101		12-9-5
6	06	0000 0110		12-9-6
7	07	0000 0111		12-9-7
8	08	0000 1000		12-9-8
9	09	0000 1001		12-9-8-1
10	0A	0000 1010		12-9-8-2
11	0B	0000 1011		12-9-8-3
12	0C	0000 1100		12-9-8-4
13	0D	0000 1101		12-9-8-5
14	0E	0000 1110		12-9-8-6
15	0F	0000 1111		12-9-8-7
16	10	0001 0000		12-11-9-8-1
17	11	0001 0001		11-9-1
18	12	0001 0010		11-9-2
19	13	0001 0011		11-9-3
20	14	0001 0100		11-9-4
21	15	0001 0101		11-9-5
22	16	0001 0110		11-9-6
23	17	0001 0111		11-9-7
24	18	0001 1000		11-9-8
25	19	0001 1001		11-9-8-1
26	1A	0001 1010		11-9-8-2
27	1B	0001 1011		11-9-8-3
28	1C	0001 1100		11-9-8-4
29	1D	0001 1101		11-9-8-5
30	1E	0001 1110		11-9-8-6
31	1F	0001 1111		11-9-8-7
32	20	0010 0000		11-0-9-8-1
33	21	0010 0001		0-9-1
34	22	0010 0010		0-9-2
35	23	0010 0011		0-9-3
36	24	0010 0100		0-9-4
37	25	0010 0101		0-9-5
38	26	0010 0110		0-9-6
39	27	0010 0111		0-9-7
40	28	0010 1000		0-9-8
41	29	0010 1001		0-9-8-1
42	2A	0010 1010		0-9-8-2
43	2B	0010 1011		0-9-8-3
44	2C	0010 1100		0-9-8-4
45	2D	0010 1101		0-9-8-5
46	2E	0010 1110		0-9-8-6
47	2F	0010 1111		0-9-8-7
48	30	0011 0000		12-11-0-9-8-1
49	31	0011 0001		9-1
50	32	0011 0010		9-2
51	33	0011 0011		9-3
52	34	0011 0100		9-4
53	35	0011 0101		9-5
54	36	0011 0110		9-6

Table A—2. EBCDIC/Hollerith Cross-Reference Table (Part 2 of 5)

EBCDIC				
Decimal	Hexa-deci-mal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code
55	37	0011 0111		9-7
56	38	0011 1000		9-8
57	39	0011 1001		9-8-1
58	3A	0011 1010		9-8-2
59	3B	0011 1011		9-8-3
60	3C	0011 1100		9-8-4
61	3D	0011 1101		9-8-5
62	3E	0011 1110		9-8-6
63	3F	0011 1111		9-8-7
64	40	0100 0000	SP	No punches
65	41	0100 0001		12-0-9-1
66	42	0100 0010		12-0-9-2
67	43	0100 0011		12-0-9-3
68	44	0100 0100		12-0-9-4
69	45	0100 0101		12-0-9-5
70	46	0100 0110		12-0-9-6
71	47	0100 0111		12-0-9-7
72	48	0100 1000		12-0-9-8
73	49	0100 1001		12-8-1
74	4A	0100 1010	[	12-8-2
75	4B	0100 1011	.	12-8-3
76	4C	0100 1100	<	12-8-4
77	4D	0100 1101	(	12-8-5
78	4E	0100 1110	+	12-8-6
79	4F	0100 1111	!	12-8-7
80	50	0101 0000	&	12
81	51	0101 0001		12-11-9-1
82	52	0101 0010		12-11-9-2
83	53	0101 0011		12-11-9-3
84	54	0101 0100		12-11-9-4
85	55	0101 0101		12-11-9-5
86	56	0101 0110		12-11-9-6
87	57	0101 0111		12-11-9-7
88	58	0101 1000		12-11-9-8
89	59	0101 1001		11-8-1
90	5A	0101 1010	]	11-8-2
91	5B	0101 1011	$	11-8-3
92	5C	0101 1100	*	11-8-4
93	5D	0101 1101	)	11-8-5
94	5E	0101 1110	;	11-8-6
95	5F	0101 1111	^	11-8-7
96	60	0110 0000	-	11
97	61	0110 0001	/	0-1
98	62	0110 0010		11-0-9-2
99	63	0110 0011		11-0-9-3
100	64	0110 0100		11-0-9-4
101	65	0110 0101		11-0-9-5
102	66	0110 0110		11-0-9-6
103	67	0110 0111		11-0-9-7
104	68	0110 1000		11-0-9-8
105	69	0110 1001		0-8-1
106	6A	0110 1010	¦	12-11
107	6B	0110 1011	,	0-8-3
108	6C	0110 1100	%	0-8-4
109	6D	0110 1101	—	0-8-5

Table A—2. EBCDIC/Hollerith Cross-Reference Table (Part 3 of 5)

EBCDIC				
Decimal	Hexa-deci-mal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code
110	6E	0110 1110	>	0-8-6
111	6F	0110 1111	?	0-8-7
112	70	0111 0000		12-11-0
113	71	0111 0001		12-11-0-9-1
114	72	0111 0010		12-11-0-9-2
115	73	0111 0011		12-11-0-9-3
116	74	0111 0100		12-11-0-9-4
117	75	0111 0101		12-11-0-9-5
118	76	0111 0110		12-11-0-9-6
119	77	0111 0111		12-11-0-9-7
120	78	0111 1000		12-11-0-9-8
121	79	0111 1001	`	8-1
122	7A	0111 1010	:	8-2
123	7B	0111 1011	#	8-3
124	7C	0111 1100	@	8-4
125	7D	0111 1101	'	8-5
126	7E	0111 1110	=	8-6
127	7F	0111 1111	"	8-7
128	80	1000 0000		12-0-8-1
129	81	1000 0001	a	12-0-1
130	82	1000 0010	b	12-0-2
131	83	1000 0011	c	12-0-3
132	84	1000 0100	d	12-0-4
133	85	1000 0101	e	12-0-5
134	86	1000 0110	f	12-0-6
135	87	1000 0111	g	12-0-7
136	88	1000 1000	h	12-0-8
137	89	1000 1001	i	12-0-9
138	8A	1000 1010		12-0-8-2
139	8B	1000 1011		12-0-8-3
140	8C	1000 1100		12-0-8-4
141	8D	1000 1101		12-0-8-5
142	8E	1000 1110		12-0-8-6
143	8F	1000 1111		12-0-8-7
144	90	1001 0000		12-11-8-1
145	91	1001 0001	j	12-11-1
146	92	1001 0010	k	12-11-2
147	93	1001 0011	l	12-11-3
148	94	1001 0100	m	12-11-4
149	95	1001 0101	n	12-11-5
150	96	1001 0110	o	12-11-6
151	97	1001 0111	p	12-11-7
152	98	1001 1000	q	12-11-8
153	99	1001 1001	r	12-11-9
154	9A	1001 1010		12-11-8-2
155	9B	1001 1011		12-11-8-3
156	9C	1001 1100		12-11-8-4
157	9D	1001 1101		12-11-8-5
158	9E	1001 1110		12-11-8-6
159	9F	1001 1111		12-11-8-7

Table A—2. EBCDIC/Hollerith Cross-Reference Table (Part 4 of 5)

EBCDIC				
Decimal	Hexa-deci-mal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code
160	A0	1010 0000		11-0-8-1
161	A1	1010 0001	~	11-0-1
162	A2	1010 0010	s	11-0-2
163	A3	1010 0011	t	11-0-3
164	A4	1010 0100	u	11-0-4
165	A5	1010 0101	v	11-0-5
166	A6	1010 0110	w	11-0-6
167	A7	1010 0111	x	11-0-7
168	A8	1010 1000	y	11-0-8
169	A9	1010 1001	z	11-0-9
170	AA	1010 1010		11-0-8-2
171	AB	1010 1011		11-0-8-3
172	AC	1010 1100		11-0-8-4
173	AD	1010 1101		11-0-8-5
174	AE	1010 1110		11-0-8-6
175	AF	1010 1111		11-0-8-7
176	B0	1011 0000		12-11-0-8-1
177	B1	1011 0001		12-11-0-1
178	B2	1011 0010		12-11-0-2
179	B3	1011 0011		12-11-0-3
180	B4	1011 0100		12-11-0-4
181	B5	1011 0101		12-11-0-5
182	B6	1011 0110		12-11-0-6
183	B7	1011 0111		12-11-0-7
184	B8	1011 1000		12-11-0-8
185	B9	1011 1001		12-11-0-9
186	BA	1011 1010		12-11-0-8-2
187	BB	1011 1011		12-11-0-8-3
188	BC	1011 1100		12-11-0-8-4
189	BD	1011 1101		12-11-0-8-5
190	BE	1011 1110		12-11-0-8-6
191	BF	1011 1111		12-11-0-8-7
192	C0	1100 0000	{	12-0
193	C1	1100 0001	A	12-1
194	C2	1100 0010	B	12-2
195	C3	1100 0011	C	12-3
196	C4	1100 0100	D	12-4
197	C5	1100 0101	E	12-5
198	C6	1100 0110	F	12-6
199	C7	1100 0111	G	12-7
200	C8	1100 1000	H	12-8
201	C9	1100 1001	I	12-9
202	CA	1100 1010		12-0-9-8-2
203	CB	1100 1011		12-0-9-8-3
204	CC	1100 1100		12-0-9-8-4
205	CD	1100 1101		12-0-9-8-5
206	CE	1100 1110		12-0-9-8-6
207	CF	1100 1111		12-0-9-8-7
208	D0	1101 0000	}	11-0
209	D1	1101 0001	J	11-1

Table A—2. EBCDIC/Hollerith Cross-Reference Table (Part 5 of 5)

EBCDIC				
Decimal	Hexa-deci-mal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code
210	D2	1101 0010	K	11-2
211	D3	1101 0011	L	11-3
212	D4	1101 0100	M	11-4
213	D5	1101 0101	N	11-5
214	D6	1101 0110	O	11-6
215	D7	1101 0111	P	11-7
216	D8	1101 1000	Q	11-8
217	D9	1101 1001	R	11-9
218	DA	1101 1010		12-11-9-8-2
219	DB	1101 1011		12-11-9-8-3
220	DC	1101 1100		12-11-9-8-4
221	DD	1101 1101		12-11-9-8-5
222	DE	1101 1110		12-11-9-8-6
223	DF	1101 1111		12-11-9-8-7
224	E0	1110 0000	\	0-8-2
225	E1	1110 0001		11-0-9-1
226	E2	1110 0010	S	0-2
227	E3	1110 0011	T	0-3
228	E4	1110 0100	U	0-4
229	E5	1110 0101	V	0-5
230	E6	1110 0110	W	0-6
231	E7	1110 0111	X	0-7
232	E8	1110 1000	Y	0-8
233	E9	1110 1001	Z	0-9
234	EA	1110 1010		11-0-9-8-2
235	EB	1110 1011		11-0-9-8-3
236	EC	1110 1100		11-0-9-8-4
237	ED	1110 1101		11-0-9-8-5
238	EE	1110 1110		11-0-9-8-6
239	EF	1110 1111		11-0-9-8-7
240	F0	1111 0000	0	0
241	F1	1111 0001	1	1
242	F2	1111 0010	2	2
243	F3	1111 0011	3	3
244	F4	1111 0100	4	4
245	F5	1111 0101	5	5
246	F6	1111 0110	6	6
247	F7	1111 0111	7	7
248	F8	1111 1000	8	8
249	F9	1111 1001	9	9
250	FA	1111 1010		12-11-0-9-8-2
251	FB	1111 1011		12-11-0-9-8-3
252	FC	1111 1100		12-11-0-9-8-4
253	FD	1111 1101		12-11-0-9-8-5
254	FE	1111 1110		12-11-0-9-8-6
255	FF	1111 1111		12-11-0-9-8-7

Table A—3. Representative EBCDIC Output Graphic Character Set

HEXA-DECIMAL →		4567 →															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
↓	0123	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000																
1	0001																
2	0010																
3	0011																
4	0100	(Space)										[	.	<	(	+	!
5	0101	&										]	$	*	)	;	Λ
6	0110	¯ (minus)	/									¦	, (comma)	%	_ (underline)	>	?
7	0111										' (single quote)	:	#	@	' (apostrophe)	=	" (quote mark)
8	1000		a	b	c	d	e	f	g	h	i						
9	1001		j	k	l	m	n	o	p	q	r						
A	1010			s	t	u	v	w	x	y	z						
B	1011																
C	1100	{	A	B	C	D	E	F	G	H	I						
D	1101	}	J	K	L	M	N	O	P	Q	R						
E	1110	\		S	T	U	V	W	X	Y	Z						
F	1111	0	1	2	3	4	5	6	7	8	9						

Bit Positions: 0 1 2 3 4 5 6 7

Weight: 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

Appendix B. Summary of CDM UNIT Options

Summaries of UNIT arguments and the types of files they define are presented in Tables B-1 through B-6.

Table B—1. Summary of UNIT Arguments for Unit Record

Argument	Use
FAUE=YES	Specifies cards with an illegal hold combination in a column be bypassed and not sent to the program.
FBFSZ=k	Specifies buffer size.
FCHAR= {OFF / ON}	Specifies printer action for illegal characters. OFF causes blank substitution; ON causes program termination.
FCRDERR=RETRY	Specifies error recovery coding is included in the executable program.
FDEVICE=UNITREC	Specifies this is a unit record device.
FDIAGNOS=YES	Specifies the unit as a diagnostic device.
FFILEID= {filename / FORTk; if FUNIT=k / PRNTR; if FUNIT=PRINT / PUNCH; if FUNIT=PUNCH / READER= if FUNIT=READ}	Specifies job control file reference name (LFD). Defaults to PRNTR, PUNCH, and READER taken only for FORTRAN II FUNIT.
FNUMBUF= {1 / 2}	Specifies the number of I/O buffers.
FOPTION=YES	Specifies a file not logically required. If the file is not allocated, output is ignored.
FREREAD=YES	Specifies a copy of each formatted input record is transferred to the reread buffer.
FSPOOLIN=YES or FGETJCS=YES	Specifies this unit defaults to a spooled card input file via a GETCS when the lfdname declared in the FFILEID agreement is not found.
FTRANS=ASCII	Specifies all incoming and outgoing records are translated to the ASCII character set.
FUNIT= {k / PRINT / PUNCH / READ}	Specifies the FORTRAN IV unit reference number or FORTRAN II statement reference.

Table B—2. Summary of UNIT Arguments for a Tape File

Argument	Use
FBFSZ= {k / FRECSIZE; if FRECFORM=FIXUNB / FRECSIZE+4; if FRECFORM=VARUNB / FRECSIZE*4; if blocked}	Specifies the size of the unit buffer. FORTRAN IV will supply a default buffer size only if record size and record format are specified.
FBKNO=YES	Specifies output tape blocks are sequentially numbered and input tape blocks are checked.
FCKPTREC=YES	Specifies that checkpoint blocks are bypassed.
FCLRW= {RWD / NORWD / UNLOAD}	Specifies positioning at end of program execution for input and output tapes.
FDEVICE=TAPE	Specifies device type to be used for the file.
FDIAGNOS=YES	Specifies the unit as a diagnostic device.
FERROPT= {IGNORE / SKIP}	Specifies the action for device errors. IGNORE and SKIP disable the ERR clause for parity/length.
FFILABL= {STD / NO}	Specifies standard or missing labels on magnetic tape.
FNUMBUF= {1 / 2}	Specifies the number of I/O buffers.
FOPRW=NORWD	Specifies the rewind is disabled at first reference to tape file.
FOPTION=YES	Specifies a file is not logically required. If the file is not allocated, output is ignored, and input causes an end return.
FRECFORM= {VARUNB / VARBLK / FIXUNB / FIXBLK}	Specifies records as variable or fixed and blocked or unblocked.
FRECSIZE=k	Specifies logical record size. Taken as a maximum for variable records.
FREREAD=YES	Specifies a copy of each formatted input record is transferred to the reread buffer.
FSPOOLIN=YES or FGETJCS=YES	Specifies this unit will default to a spooled card input file via a GETCS when the lfdname declared in the FFILEID argument is not found.
FTRANS=YES	Specifies all incoming and outgoing records are translated to the ASCII character set.
FTYPEFLE= {WORK; if FUNIT=k / INPUT; if FUNIT=READ / OUTPUT; if FUNIT=PUNCH}	Specifies the level of data management support.
FUNIT= {k / READ / PUNCH / PRINT}	Specifies FORTRAN IV unit reference number of FORTRAN II statement reference.

Table B—3. Summary of UNIT Arguments for a Disk File

Argument	Use
FDEVICE=DISK	Specifies device type to be used for this file.
FDIAGNOS=YES	Specifies the unit as a diagnostic device.
FFILEID= (filename / FORTk; if FUNIT=k / READER; if FUNIT=READ / PUNCH; if FUNIT=PUNCH / PRNTR; if FUNIT=PRINT)	Specifies job control file reference name (LFD). Defaults READER, PUNCH, and PRNTR taken only for FORTRAN II FUNIT.
FOPTION=YES	Specifies a file not logically required. If the file is not allocated, output is ignored and input causes an end return.
FRECFORM= (VARUNB / FIXUNB)	Specifies records as variable or fixed. No blocking.
FRECSIZE=k	Specifies the logical record size. Taken as maximum for variable records.
FREREAD=YES	Specifies a copy of each formatted input record is transferred to reread buffer.
FSPOOLIN=YES or FGETJCS=YES	Specifies this unit will default to a spooled card input file via a GETCS when the lfdname declared in the FFILEID argument is not found.
FTYPEFLE= (WORK; if FUNIT=k / INPUT; if FUNIT=READ / OUTPUT; if FUNIT=PUNCH or FUNIT=PRINT)	Specifies level of data management support.
FUNIT= (k / READ / PUNCH / PRINT)	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.
FVERIFY=YES	Specifies a reread of each written block to ensure proper parity.

Table B—4. Summary of UNIT Arguments for a Workstation

Argument	Use
FDEVICE=WORKSTN	Specifies device to be used for a workstation.
FUNIT= {k / READ / PUNCH / PRINT}	Specifies the FORTRAN IV unit reference number or FORTRAN II statement reference.
FFILEID= {filename / FORTk; if FUNIT=k / READER; if FUNIT=READ / PUNCH; if FUNIT=PUNCH / PRNTR; if FUNIT=PRINT}	Specifies the job control file reference name (LFD). Defaults to READER, PUNCH, and PRNTR taken only for FORTRAN II FUNIT.
FSCREND={WRAP / SCROLL / NEWPAGE}	Specifies action taken when end of screen display is reached.
FTYPEFLE={WORK / INPUT; if FUNIT=READ / OUTPUT; if FUNIT=PUNCH or PRINT}	Specifies level of data management support.
FIOOPT=YES	Specifies one buffer is used exclusively for input and one buffer for output. This is the effect only when FNUMBUF=2 is specified.
FLINCNTL=YES	Indicates the workstation will function as a printer.
FNUMBUF={1 / 2}	Specifies the number of input/output buffers.
FOPTION=YES	Specifies a file not logically required. If file is not allocated, output is ignored and input causes an end return.
FRECSIZE=k	Specifies the logical record size.
FREREAD=YES	Specifies a copy of each formatted input record be transferred to the reread buffer.
FSPOOLIN=YES or FGETJCS=YES	Specifies this unit will default to a spooled card input file via a GETCS when the lfdname declared in the FFILEID argument is not found.
FDIAGNOS=YES	Specifies diagnostic messages are posted to this device.

Table B—5. Summary of UNIT Arguments for Reread Unit

Argument	Use
FDEVICE=REREAD	Specifies device to be used for the file.
FUNIT= {k / READ}	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.

Table B—6. 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 FORTRAN IV unit reference number or FORTRAN II statement reference.
FEQUIV= {k / READ / PRINT / PUNCH}	Specifies unit to be activated.

Appendix C. Summary of DTF UNIT Options

Summaries of UNIT arguments and the types of files they define are presented in Tables C-1 through C-10.

Table C—1. Summary of UNIT Arguments for Printer File

Argument	Use
FDEVICE=PRINTER	Specifies the device to be used for the file.
FUNIT={k / PRINT / PUNCH}	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.
[FFILEID={filename / FORTK; if FUNIT = k / PRNTR; if FUNIT = PRINT / PUNCH; if FUNIT = PUNCH}]	Specifies job control file reference name (LFD). Defaults PRNTR and PUNCH taken only for FORTRAN II FUNIT.
[FRECSIZE={k / 121}]	Specifies logical record size.
[FNUMBUF={1 / 2}]	Specifies number of input/output buffers.
[FDIAGNOS=YES]	Specifies the unit as the diagnostic device.
[FPRINTOV={SKIP / NOSKIP}]	Specifies printer action when bottom of page is encountered.
[FCHAR={OFF / ON}]	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 C—2. Summary of UNIT Arguments for Spooled Card Input File

Argument	Use
FDEVICE=SPOOLIN	Specifies the device to be used for the file.
FUNIT={k / READ}	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.
[FREREAD=YES]	Specifies that a copy of each formatted input record is transferred to the reread buffer.
[FBKSZ={k / 400;}]	Specifies size of unit buffer.
[FBUFPOOL=YES]	Specifies that the buffer for the unit is pooled.
[FRECSIZE={k / 80}]	Specifies card size read.

Table C—3. Summary of UNIT Arguments for Card Input File

Argument	Use
FDEVICE=CARDIN	Specifies device used for the file.
FUNIT={k / READ}	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.
[FFILEID={filename / FORTk; if FUNIT=k / READER; if FUNIT=READ}]	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 transferred to the reread buffer.
[FBUFPOOL=YES]	Specifies that buffers for the unit are 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 processed in a work area rather than in the buffer.
[FRECSIZE={k / 80}]	Specifies logical record size.
[FSTUB={51 / 66}]	Specifies that cards shorter than 80 columns are 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 mispunched cards are ignored.
[FBKSZ={k / FRECSIZE}]	Specifies a buffer size for multiple record input for the 8413 diskette only.

Table C—4. Summary of UNIT Arguments for Card Output File

Argument	Use
FDEVICE=CARDOUT	Specifies device used for the file.
FUNIT={k / PUNCH}	Specifies FORTRAN IV 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 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 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 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 a buffer size for multiple record input for the 8413 diskette only.

Table C—5. Summary of UNIT Arguments for Tape File (Part 1 of 2)

Argument	Use
FDEVICE=TAPE	Specifies device used for the file.
FUNIT={k / READ / PUNCH}	Specifies FORTRAN IV 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 processed in a work area rather than in the buffer.

Table C—5. Summary of UNIT Arguments for Tape File (Part 2 of 2)

Argument	Use
[FBUFPOOL=YES]	Specifies that buffers for the unit are 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 transferred to the reread buffer.
[FDIAGNOS=YES]	Specifies the unit as the diagnostic device
[FBKNO=YES]	Specifies that output tape blocks are sequentially numbered and input tape blocks are 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 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 C—6. Summary of UNIT Arguments for Sequential Disk Files

Argument	Use
FDEVICE=SDISC	Specifies device used for the file.
FUNIT={k / READ / PUNCH}	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.
[FSECTOR={NO / YES}]	Specifies processing on a sectorized disk expected.
[FFILEID= filename / FORTk; if FUNIT=k / READ; 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.
[FBUFPOOL=YES]	Specifies that buffers for the unit are pooled, or shared, with all other units so specified.
[FNUMBUF={1 / 2}]	Specifies number of input/output buffers.
[FWORKA={YES; if FNUMBUF=1 / NO; if FNUMBUF=2}]	Specifies that logical records are processed in a work area rather than in the buffer.
[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 transferred to the reread buffer.
[FDIAGNOS=YES]	Specifies the unit as the diagnostic device.
[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 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 C—7. Summary of UNIT Arguments for Direct Access Disk Files

Argument	Use
FDEVICE=DISC	Specifies device used for the file.
FUNIT=k	Specifies FORTRAN IV unit reference number.
[FSECTOR={NO / YES}]	Specifies processing on a sectorized disk expected.
[FFILEID={filename / FORTk; where k=FUNIT}]	Specifies job control file reference name (LFD).
[FTYPEFLE={INPUT / OUTPUT}]	Specifies level of data management support.
[FBUFPOOL=YES]	Specifies that buffers for the unit are 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 moved to the reread buffer.
[FREREAD=YES]	Specifies that a copy of each formatted input record is transferred to the reread buffer.
[FVERIFY=YES]	Specifies a reread of each written block to ensure proper parity.

Table C—8. Summary of UNIT Arguments for Reread Unit

Argument	Use
FDEVICE=REREAD	Specifies device used for the file.
FUNIT={k / READ}	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.

Table C—9. Summary of UNIT Arguments for Equivalent Unit

Argument	Use
FDEVICE=EQUIV	Specifies device used for the file.
FUNIT={k / READ / PRINT / PUNCH}	Specifies FORTRAN IV unit reference number or FORTRAN II statement reference.
FEQUIV={k / READ / PRINT / PUNCH}	Specifies the unit to be activated.

Table C—10. Summary of UNIT Arguments for MIRAM Disk Files

Argument	Use
FDEVICE=MIDISC	Specifies device used for the file.
FUNIT={k / READ / PUNCH}	Specifies FORTRAN IV 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={WORK; if FUNIT=k / INPUT; if FUNIT=READ / OUTPUT; if FUNIT=PUNCH}]	Specifies level of data management support.
[FRECFORM={FIX / VAR}]	Specifies records as variable or fixed.
[FBUFPOOL=YES]	Specifies that buffers for the unit are pooled, or shared, with all other units so specified.
[FRECSIZE={k / 255; if FRECFORM=FIX / 252; if FRECFORM=VAR}]	Specifies logical record size. Taken as maximum for variable records.
[FREREAD=YES]	Specifies that a copy of each formatted input record is to be transferred to the reread buffer.
[FRECERR=YES]	Specifies that records with bad parity or wrong length are 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.

Appendix D. Additional UNIT Options in DTF Environment

D.1. GENERAL

Additional options for execution environment configuration not presented in Section 12 are descibed 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 programmer reference. For an explanation of the statement conventions applicable to this appendix, refer to 1.4.

D.2. CARD READER OPTIONS

Additional options for the card reader are described.

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 2^5 through 2^0 of byte 1; holes 4 through 9 are mapped onto bits 2^5 through 2^0 of byte 2, etc. Bits 2^7 and 2^6 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; it is the accepted standard for information interchange.

D.3. CARD PUNCH OPTIONS

Additional options for the card punch are specified.

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 2^5 through 2^0 of byte 1; holes 4 through 9 are mapped onto bits 2^5 through 2^0 of byte 2, etc. Bytes 2^7 and 2^6 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.

D.4. TAPE FILE OPTIONS

Additional options for magnetic tape are specified.

User Header and Trailer Label 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 12, 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.

- FBUFOFF

  FBUFOFF=k
  Specifies that a block length field of 0 to 99 bytes is to be prefixed on each block. FBUFOFF may be specified only if FASCII=YES has been specified. A value of 0 to 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, FBUFOFF 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 FBUFOFF=4 have been specified.

D.5. SEQUENTIAL DISK FILE OPTION

This is an additional option for sequential disk processing.

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.

D.6. DIRECT ACCESS DISK FILE OPTIONS

Additional options for direct access disk are specified.

- 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.

D.7. ADDITIONAL DATA MANAGEMENT DEVICES

Files for sequential devices supported by data management but not presented in Section 12, such as optical document readers, paper tape, etc., are defined by using the following UNIT procedure call. A listing, in the order of 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}]
                           {FORTk   }
                 [ FRECFORM= {VARUNB}]
                 [           {VARBLK}
                 [           {FIXUNB}
                 [           {FIXBLK}]
                 [ FRECSIZE= {k    }]
                 [           {508  }]
                 [FREREAD=YES]
```

Device Identification Argument:

FDEVICE=DMS
Specifies that this file is for a sequential device supported by data management.

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 to 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) macroinstruction labeled with the file name must be provided. An EXTRN statement is automatically generated for the label specified in FFILEID.

Record Formats Arguments:

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={k / 508}
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 consits 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 E. FORTRAN Sample Job Streams

E.1. JOB CONTROL PROCEDURE

The FOR4 procedure call statement generates the necessary job control statements to compile a FORTRAN IV program. Optionally, the procedure call statement can generate job control statements that specify the following:

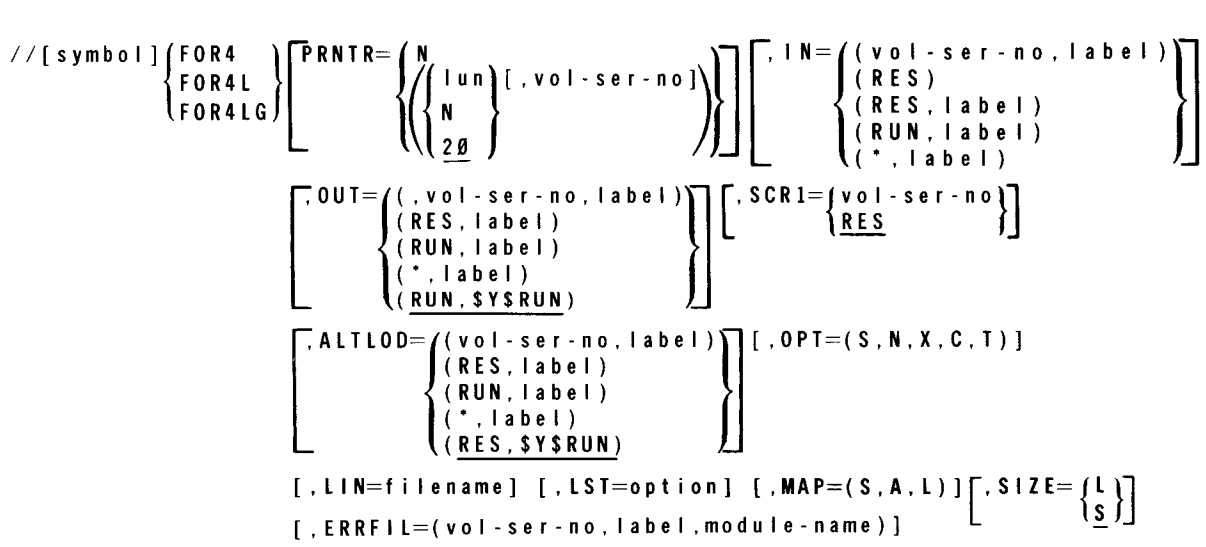
- input - source library;
- output - object library;
- PARAM control statements defining the compiler processing logic; and
- automatically link and/or execute the program.

The input may be embedded data cards (/$, source deck, /*) immediately after the FOR4 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 may be written in YRUN 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 YLOD.

Format:



Label:

symbol
Specifies the 1- to 6-character source module name; only used when the IN parameter is also used.

Operation:

FOR4
This form of the procedure call statement is used to compile a FORTRAN IV source program.

FOR4L
This form of the procedure call statement is used to compile a FORTRAN IV source program and link-edit the object modules (see Note 1).

FOR4LG
This form of the procedure call statement is used to compile a FORTRAN IV source program, link-edit the object modules, and execute the load module (see Notes 1, 2, and 3).

NOTES:

1. *Linkage control cards or program data are not allowed with this form of the procedure call statement.*
2. *The FOR4LG procedure call statement cannot be used when operating with the shared code data management feature. Instead, use the FOR4L procedure call statement and provide a separate EXEC statement to execute the load module.*
3. *Device assignment sets must be specified prior to the jproc.*

Keyword parameter PRNTR:

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

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.

If omitted, 20 is assumed.

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 YSRC.

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 YRUN 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 YRUN 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 identified by the label parameter.

OUT=(RUN,label)
Specifies that the object code is to be located on the job's YRUN 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 YRUN.

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 FORTRAN IV 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 YRUN 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=(S, N, X, C, T)
Specifies one or all of the following compilation options.

S
Specifies that statement numbers will be inserted into the generated code as an aid to debugging. When S is specified, the size of the object program and its execution time can increase significantly.

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.

C
Specifies all references to array elements are to be checked to see if they are outside the declared limits of the array.

T
Specifies that tracing of executed labels is requested. The compiler generates a special subroutine call at every label. A TRACE ON must occur in the program to activate tracing.

If only one OPT argument is specified, the parentheses are optional.

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, and diagnostics.

S
Specifies, in addition to the N listing, the source code listing.

M
Specifies, in addition to the S listing, an object summary and a storage map showing the addresses assigned to variables and arrays. (Can be superseded by the MAP parameter.)

If no LST parameter is specified, the S option is assumed.

Keyword Parameter MAP:

MAP=(S, A, L)

Specifies the type of maps produced by the compiler. One or all of the following options may be chosen.

S

Specifies object summary information, including module size and external subroutines called.

A

Specifies an alphabetical listing of the addresses assigned to variables, arrays, and statement labels.

L

Specifies a listing of the addresses assigned to variables, arrays, and statement labels in order by the storage locations assigned.

When a MAP argument is specified, it supersedes the maps selected by the LST parameter. Also, when a MAP argument is specified, it is not necessary to specify LST=M.

Keyword Parameter SIZE:

SIZE={L / S}

Specifies the size of the FORTRAN IV compiler to be used.

L

Specifies the large version.

S

Specifies the small version.

If omitted, S is assumed.

Keyword Parameter ERRFIL:

This parameter specifies that error diagnostic messages are written to a file that is accessed by the error file processor. When you specify this parameter, error records are created for every error generated by the compiler.

ERRFIL=(vol-ser-no,label,module-name)

The vol-ser-no specifies the volume serial number of the file. The label specifies the file identifier (name of the file that the module is placed into). The module-name is the name of the module that is referenced by the error file processor.

If omitted, the error file is not created.

Example:

The following example illustrates the use of the FOR4 procedure call statement in its basic form:

```
      1         10
1.  // JOB FRTRN1A
2.  // FOR4
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 (FOR4). 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.

E.2. SAMPLE COMPILE-LINK-EXECUTE

The following job control streams illustrate a simple compilation from cards, linking the program EX1, and executing the bound program TEST1 from YRUN. (TEST1 is the new name for EX1.)

Example A shows a job control stream using the conventional method, that is, the EXEC statements with their supporting device assignment sets. Example B shows the jproc method, which is more efficient because it requires less coding, thereby reducing the possibility of mistakes. Both examples produce the same result in either a CDI or DTF environment.

Example A:

```
     1         10     16
1.  // JOB EXAMPLE1
2.  // DVC 20 // LFD PRNTR PRINTER FOR ALL PROCESSORS
3.  // WORK1              ONE WORK FILE
4.  // EXEC FOR4          BEGIN COMPILATION
5.  // PARAM              PARAMETERS(AS NEEDED)
6.  /$
          PROGRAM  EX1
         (program body)
7.        END
8.  /*                    END COMPILATION
9.  // WORK1              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 FORTRAN IV compiler execution
4	Begins compilation (FOR4 or FOR4L)

Line	Explanation
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
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

Example B:

```
     1          10
1.  // JOB EXAMPLE1
2.  // FOR4
3.  /$
        .  \
        .   } program  EX1
        .   } source card deck
        .  /  END
4.  /*
5.  //TEST1 LINKG EX1
6.  /&
```

Line	Explanation
1	Indicates the job name, EXAMPLE1
2	Indicates the FORTRAN jproc; allocates a printer and a work file and starts compilation
3	Start of data to the compiler (source program)
4	End of source data
5	Indicates the linkage editor jproc; allocates a work file, uses the input source program EX1 and names the new load module TEST1
6	End of job

E.3. SOURCE FROM DISK LIBRARY - STACKED COMPILATION

This job control stream represents the source module from the disk library for a stacked compilation. Source programs on disk files are identified using a librarian module name. Each source module consists of one or more FORTRAN program units.

Example:

```
     1         10     16
1.  // JOB EXAMPLE2
2.  // DVC 20  // LFD PRNTR
3.  // WORK1
4.  // DVC 50 // VOL DISC00 // LBL FORSOURCE  // LFD INPUT
5.  // EXEC FOR4
6.  // PARAM IN=MODULE1/INPUT
7.  // PARAM IN=MODULE2/INPUT
8.          .
            .
9.          .
10. // PARAM IN=MODULEn/INPUT
11. /&
```

Line	Explanation
1	Indicates job name, EXAMPLE2
2	Indicates the printer device number
3	Specifies one work file for the FORTRAN IV compiler
4	Specifies that the file, FORSOURCE, on disk DISC00, device 50, is the INPUT file.
5	Begins compilation (FOR4)
6-7	Identifies the first and second source program 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

E.4. COMPILE, ASSEMBLE, LINK, AND EXECUTE

This example shows the user-specified execution environment, control stream input, and print and tape output in a DTF environment. For a CDI environment, every statement is the same except that the FUNTAB is omitted.

```
     1         10    16                                                  72
1.   // JOB EXAMPLE3
2.   // FOR4
3.   /$
        .
        .    source
        .    program deck
        .    ($MAIN)
4.   /*
5.   // ASM LST=N
6.   /$
7.   MYIO      START
8.             FUNTAB  SYS=FOR
9.             UNIT  FDEVICE=PRINTER,                                   X
10.                  FUNIT=1
11.            UNIT  FDEVICE=SPOOLIN,                                   X
12.                  FUNIT=READ,                                        X
13.                  FBKSZ=80
14.            UNIT  FDEVICE=TAPE,                                      X
15.                  FUNIT=10,                                          X
16.                  FFILEID=XYZ,                                       X
17.                  FRECFORM=FIXBLK,                                   X
18.                  FRECSIZE=256
19.            FUNEND
20.            EJECT
21.            ERRDEF
22.            END
23.  /*
24.  //TEST3 LINKG $MAIN,MYIO
25.  // DVC 21 // LFD FORT1
26.  // DVC 40 // VOL TAPE00 // LFD XYZ
27.  /$
        .
        .    control
        .    stream
        .    input
28.  /*
29.  /&
```

Line	Explanation
1	Indicates job name, EXAMPLE3
2	Indicates the FORTRAN job control procedure (jproc), FOR4
3	Start of data to compiler (source program – $MAIN)
4	End of data
5	Indicates assembler jproc, ASM; no cross-reference listing is produced (LST=N)

Line	Explanation
6	Start of data to the assembler
7	Start of execution environment module (MYIO)
8	Initiates file for FORTRAN IV
9	Defines first file (UNIT definition procedure) specifying a printer file
10	Specifies printer unit number 1
11-13	Defines second file (UNIT definition procedure) specifying a spooled input file
12	Identifies the reader as input device
13	Indicates input block size
14-18	Defines third file (UNIT definition procedure) specifying a tape file
15	Specifies tape unit number 10
16	Indicates the tape filename, XYZ
17	Specifies fixed-length blocked records
18	Specifies a tape record length of 256 bytes
19	Terminates UNIT procedure calls
20	File termination
21	Includes library table of error information in executable program
22	Terminates source program
23	End of data to assembler
24	Indicates the linkage editor jproc; names the new load module as TEST3 which includes the modules $MAIN (FORTRAN source program) and MYIO (assembler source program) and executes the module
25-26	Connects devices assigned before execution of TEST3 to the FORTRAN unit table via their LFD names
27	Start of spoolin data
28	End of spoolin data
29	End of job

NOTE:

The default FFILEID for the printer is FORT1.

E.5. COMPILATIONS WITH PARAMETER OPTIONS

The following example shows the use of special parameter options associated with the FOR4 jproc in either the CDI or DTF environment.

Example:

```
      1         10     16
1.  // JOB EXAMPLE4
2.  // FOR4  MAP=(S),OPT=(X)
3.  /$
        .
        .   program deck
        .
4.  /*
5.  //TEST4 LINKG $MAIN
6.  /&
```

Line	Explanation
1	Indicates job name, EXAMPLE4
2	Indicates the FORTRAN jproc, FOR4; an object summary is produced and all cards with an X in column 1 are accepted for compilation as FORTRAN statements.
3	Start of data to compiler (source program-$MAIN)
4	End of data
5	Indicates the linkage editor jproc, LINKG; names the new load module as TEST4 which is the source program $MAIN, and executes the program.
6	End of job

E.6. COMPILATION FROM A WORKSTATION TERMINAL

In the CDI environment, compilation can originate from a workstation terminal. The following example shows a typical compilation.

```
     1         10     16
1.  LOGON SYSPUBS,6944,DOIT
2.  /EDT
      .
      .  FORTRAN IV source program
      .
    @ WRITE INPUT1
    @ HALT
3.  RV JC$BUILD
      .
      .  job control screen
      .
4.  RV COMPIL
5.  LOGOFF
```

Line	Explanation
1	Connects the workstation terminal to the host system and defines the user, his account number, and password.
2	Calls the system editor. The FORTRAN IV source program immediately follows /EDT command. After the last FORTRAN IV source statement, the @WRITE command is issued to save the source program in a library. The @HALT command ends the EDT session.
3	Calls the system build command. This command writes the job control stream to the system job control stream library file (YJCS). This job control stream defines the system resources that the source program requires.
4	Calls the job control stream from YJCS. This step compiles the source program.
5	Ends the workstation terminal session.

NOTE:

For more information about workstation terminal input, see the workstation user guide and the editor user guide.

E.7. EXECUTION FROM A WORKSTATION USING A SCREEN FORMAT

The following example shows the job control stream that will execute a FORTRAN IV loadable program using screen format services.

Example:

```
     1         10    16
1.  LOGON STSPUBS,6944,DOIT
2.  RU JC$BUILD
3.  // JOB APPLIC
4.  // DVC 200
5.  // USE SFS,MYFILE
6.  // DVC 50
    // VOL D00028
    // LBL DISK-FORMAT-FILE
    // LFD MYFILE
7.  // DVC 20
    // LFD LIST
8.  // DVC 51  // VOL D00027
    // LBL APPLIC.FILE
    // LFD APPL1
9.  // EXEC FORT01
10. LOGOFF
```

Line	Explanation
1	Connects the workstation terminal to the system and also identifies the user.
2	Builds the job control stream to be used by the loadable FORTRAN IV program.
3	Identifies the job name of the filed job control stream.
4	Defines the workstation terminal.
5	Calls the screen format services routine and specifies the screen format file that the executable program is using. The screen parameter must specify a screen residing on the file (7.3.1).
6	Defines the file where the screen format resides.
7	Defines the print file to be used by the executable program.
8	Defines the disk file to be used by the executable program.
9	Calls the executable program.
10	Disconnects the workstation terminal from the system.

E-8. CREATE AND COMPILE FROM A WORKSTATION USING EDT

You use the following commands at the workstation to create and compile your program:

```
      1         10    16                                          72
1.  LOGON user-id
2.  /EDT
      .
      .  FORTRAN IV source program
      .
    @WRITE INPUT1
    // JOB COMPIL
    // FOR4,IN=INPUT1
    /&
    @WRITE.$Y$JCS
    @HALT
3.  RV COMPIL
4.  LOGOFF
```

Line	Explanation
1	Connects the workstation terminal to the system and also identifies the user.
2	Calls the system editor (EDT).
3	Compiles the program.
4	Disconnects the workstation terminal from the system.

Appendix F. Compile-Time Diagnostic Messages

All messages produced by the FORTRAN IV compiler are printed on one or more pages containing the title ERROR LISTING as a header line. Separate error listings are produced for each program module processed.

Two classes of messages may be produced–severity code 1 or severity code 2. Severity code 1 messages denote a condition that, while not entirely incorrect, is not fully correct and should be examined and possibly changed by the user. Severity code 2 messages indicate an error that inhibits correct program execution. A severity code 2 error sets the leftmost bit of the UPSI byte, which may be tested via the job control statement, // SKIP target-label,1. The severity code of each error is printed immediately to the left of the message in the error listing.

In most cases, the card causing the error (message), is itself printed on the error listing, followed by a line containing a pointer to the place at which the error occurs and also containing the severity code and the error message. However, it is not always possible to print the card at fault. In that case, only the severity code and the message are printed. For example, in some cases parts of the compilation process are shared by the various passes of the compiler (e.g., constant evaluation), so the card at fault may no longer be available at the time the error is detected. Also, sometimes no one card is at fault, as in the case of the possible interrelationship between COMMON, DIMENSION, EQUIVALENCE, and DATA statements.

Table F–1 lists and describes the FORTRAN IV compile-time diagnostics. All messages are prefixed by severity code shown in the Severity Code column where:

1 = warning

2 = serious

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. The messages are presented in alphabetical order by severity code.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 1 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
1	ARRAY MISSING SUBSCRIPTS	A variable known to be an array is referenced without subscripts.	Correct source program.
1	"," ASSUMED BEFORE TYPE	In an IMPLICIT statement, commas must separate type specifications.	Correct statement.
1	BLOCK DATA STMT ENDS BADLY	Extraneous field detected on BLOCK DATA statement.	Correct BLOCK DATA statement.
1	DIM DECLARED BEFORE	The dimensions of this array have been declared previously.	Delete declarator. Do not delete the array name if a type or COMMON statement.
1	END STMT MISSING	Compiler encountered another program unit before an END statement.	Insert END statement before next program unit.
1	EXTRANEOUS LABEL	Label is not accepted as the destination point of a control transfer.	Warning only.
1	EXTRANEOUS OPTR	Extraneous operator detected after end of valid statement.	Correct statement.
1	FIELD AFTER END OF STMT	Extraneous field detected after end of valid statement.	Correct statement.
1	GO TO W/O ASSIGN ON CARD nnnn	An assigned GO TO on card nnnn has been encountered without an ASSIGN statement in the program.	Provide an ASSIGN statement or change the GO TO.
1	"," IGNORED	Extraneous comma in a COMMON statement ignored.	Correct COMMON statement.
1	ILLEGAL CHARACTER	Character not a member of the FORTRAN character set.	Probable keypunch error. Correct error. Correct statement.
1	ILLEGAL ELEMENT	In an EQUIVALENCE statement, an attempt was made to subscript a nonarray.	Dimension array or correct statement.
1	"*" ILLEGAL HERE	Optional length specification illegal for DOUBLE PRECISION.	Remove optional length.
1	ILLEGAL LABEL	The statement label is either an invalid constant or it exceeds the maximum label value of 99999.	Correct statement label.
1	ILLEGAL RANGE	A statement label has been found which exceeds the maximum value of 99999.	Correct label.
1	ILLEGAL STMT FORM	FORTRAN statement not in valid format.	Correct statement.
1	ILLEGAL STMT IN LOG IF	A DO or logical IF cannot be the object statement of a logical IF.	Correct statement.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 2 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
1	ILLEGAL USE OF STD FCN	Improper use of or wrong number of arguments in standard library function.	Insert a TYPE statement to declare function as users, or correct function. reference.
1	INTEGER>2**31	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.
1	LABEL IN CONT.	Characters appeared in label field of continuation card.	Warning only.
1	LENGTH ILLEGAL HERE	Optional length illegal for DOUBLE PRECISION.	Remove optional length.
1	LITERAL TOO BIG	A Hollerith constant or literal constant has exceeded the maximum length of 255 characters.	Shorten character string.
1	LITERAL TOO SHORT	Zero length literal constant encountered.	Correct constant.
1	"," MISSING	Comma missing in assigned GO TO (before branch list).	Warning. Insert comma if desired.
1	MISSING ")" OR "),"?	An EQUIVALENCE set is missing its closing parenthesis.	Correct EQUIVALENCE statement.
1	MISSING SIZE	Optional size in a type statement is missing, but the '*' was detected.	Remove '*' or insert size.
1	MISSING "TO" IN ASSIGN	Keyword TO missing from ASSIGN statement.	Correct ASSIGN statement.
1	NAME TRUNCATED	Symbolic name cannot exceed six characters.	Shorten name to six or less characters.
1	NULL EXPONENT	The exponents on a real or double precision constant are missing.	Correct constant.
1	NUMBER > 16**63	Magnitude of real constant out of range (positive).	Correct program.
1	NUMBER < 16**-65	Magnitude of real constant out of range (negative).	Correct program.
1	PREVIOUS "," IGNORED	Previous comma in a COMMON or type statement has been ignored.	Correct statement.
1	PROG ENDS BADLY	STOP statement missing.	Warning.
1	SHOULD HAVE INTEGER HERE	Variable should be of type integer.	Correct statement.
1	SHOULD HAVE LABEL AFTER BRANCHES	Label missing on statement immediately following an unconditional branch.	Correct program logic.
1	"," SHOULD PRECEDE	A comma should precede the name in a DATA statement.	Correct DATA statement.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 3 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
1	STMT ENDS BADLY	Extraneous field detected on END statement.	Correct END statement.
1	"STOP" COMPILED HERE	RETURN statement occurs in a main program. STOP statement substituted.	Change RETURN to STOP.
1	TOO FEW SUBSCRIPTS	Array element reference contains fewer subscripts than the array declarator.	Correct either the declarator or the reference.
1	TRANSFER ENDS DO LOOP	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.
2	ADJ DIM IN COMMON OR MAIN PROG	Adjustable dimensions are prohibited in main programs.	Correct program logic.
2	#ARGS + ENTRY > 398 or COMMONS > 85	As stated.	Break up program.
2	ARITH CONST & LOG VAR	DATA statement processor cannot perform a meaningful conversion.	Correct DATA statement.
2	ARITH OPND & LOG OPTR	A nonlogical was used in formation of logical expression.	Correct expression.
2	ARRAY ILLEGAL IN STMT FCN	A statement function cannot reference an array.	Correct statement function definition.
2	BADLY NESTED DO	A DO loop must be completely enclosed by any surrounding DO loop. See fundamentals of FORTRAN programmer reference, UP-7536 (current version).	Correct structure of loops.
2	BINARY OPTR STARTS SUBEXP	+, *, /, or ** begins an arithmetic expression.	Correct expression.
2	COMMONS: TOO MANY; OR TOO BIG.	Internal table overflow processing COMMON statement.	Try large compiler or change program logic.
2	COMMON W/EQUIV.: TOO MANY; OR TOO BIG.	Internal table overflow processing COMMON statement.	Try large compiler or change program logic.
2	COMPLEX COMPARAND	Operators GT, GE, EQ, NE, LT, LE cannot be used with complex entries.	Correct expression.
2	COMPLEX OPND IN IF	Complex expression not permitted in arithmetic IF.	Correct program.
2	DATA TABLE EXCEEDED	An implied-do in a DATA statement is too complex.	Break up the DATA statement.
2	DIM 2**24	Value of a dimension must be less than 2**24.	Correct statement.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 4 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
2	DUPLICATE END REFERENCE	Two END clauses in one statement.	Probably intended to be an ERR clause.
2	DUPLICATE ERR REFERENCE	Two ERR clauses.	Correct statement. Probably should be an END clause.
2	EQUIV TO ITSELF	A variable has been made equivalent to itself indirectly.	Correct EQUIVALENCE statement.
2	EQUIV TOO EXTENSIVE	Excessive number of equivalence sets in the program unit. The compiler can address only 65K in the equivalence table.	Reduce number of equivalence groups.
2	EXTRANEOUS ","	Comma encountered in expression analysis.	Correct statement.
2	FUNCTION NOT ASSIGNED A VALUE	Function name does not take on a value.	Correct function.
2	GO TO LIST TOO BIG	List for computed GO TO or assigned GO TO has more than 127 entries.	Break up computed GO TO so that each list is less than 127 entries.
2	IF ENDS BADLY	Error detected in processing the labels in an arithmetic IF.	Correct labels for arithmetic IF.
2	ILLEGAL ARRAY DECLARATOR	A dimension declarator must be a constant or integer variable.	Correct subscript.
2	ILLEGAL ASSIGN	Label of ASSIGN must be an executable statement label.	Correct statement.
2	ILLEGAL BLOCK SYMBOL	Illegal common block name.	Correct COMMON statement.
2	ILLEGAL COMMON STMT	Unrecoverable error in COMMON statement.	Correct COMMON statement.
2	ILLEGAL COMPLEX CONSTANT	A component of the constant is not an acceptable real constant.	Correct statement.
2	ILLEGAL DATA	Unrecoverable error in DATA statement.	Correct DATA statement.
2	ILLEGAL DATA SET REFERENCE NO.	A data set reference number must be $\geqslant 1$ and $\leqslant 99$.	Correct statement.
2	ILLEGAL DATA STMT	NAME is not a variable or array name, or is a dummy argument, or a blank common.	Correct statement.
2	ILLEGAL DMY ARG	A dummy argument list may consist only of simple variable names which are unique in the list.	Correct list.
2	ILLEGAL DO	Bad DO statement	Correct DO statement.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 5 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
2	ILLEGAL EQUIV GROUP	Equivalence set is attempting to distort the structure of an array which demands the contiguity of successive array elements.	Correct EQUIVALENCE statement.
2	ILLEGAL FORM OF I/O STMT	Error detected in READ, WRITE, PRINT, or PUNCH statement.	Correct statement.
2	ILLEGAL IMPLIED DO	Error in implied DO in I/O list or DATA statement.	Correct statement.
2	ILLEGAL IN ARITH EXP	.NOT. is illegal in arithmetic expression.	Correct expression.
2	ILLEGAL IN BRANCH LIST	Illegal symbol in GO TO list.	Correct statement.
2	ILLEGAL IN GO TO	Error detected in processing the object of a GO TO statement.	Correct statement.
2	ILLEGAL IN SUBSCRIPTS	Complex and logical expressions may not be used as subscripts. A subscript expression may only be integer or real.	Correct subscript.
2	ILLEGAL I/O LIST	The symbolic name indicated by the marker cannot appear in an I/O list.	Correct statement.
2	ILLEGAL LENGTH FOR TYPE	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.
2	ILLEGAL NO. FOR LABEL	Error in processing label designating DO loop terminator.	Fix label in DO statement.
2	ILLEGAL NUMBER OF ARGS	A standard library function has been referenced with an incorrect number of arguments.	Correct function reference.
2	ILLEGAL OPND IN STMT	Illegal operand in CALL, REWIND, ENDFILE, or BACKSPACE statement.	Correct statement.
2	ILLEGAL SUBSCRIPTS	Illegal specification of subscripts in a DATA statement.	Correct DATA statement.
2	ILLEGAL SYMBOL AFTER BRANCHES	Error in name after branch list in computed GO TO.	Correct statement.
2	ILLEGAL SYMBOL BEFORE BRANCHES	Error detected identifying beginning of branch list for assigned GO TO.	Correct assigned GO TO.
2	ILLEGAL TO HAVE INITIAL VALUES	Common variables may only be initialized by BLOCK DATA.	Change program logic.
2	ILLEGAL TYPE STMT	Unrecoverable error in a type statement.	Correct statement.
2	INCONSISTENT USE OF LABEL	An attempt was made to branch to a format label.	Correct statement.
2	"=" INSIDE EXP	An equal sign detected inside.	Correct statement.
2	LIST ILLEGAL FOR NAMELIST I/O	The variable list must appear in the NAMELIST statement and not in the READ/WRITE statement.	Correct both statements.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 6 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
2	LITERAL CONST OR &NO. ILLEGAL HERE	Literal constant or &label detected outside of a call statement.	Correct statement.
2	LOCAL ARRAYS TOO LARGE > 2**20	Storage necessary for local arrays is too large.	Decrease array storage necessary.
2	LOG COMPARAND	Operators GT, GE, EQ, NE, LT, LE cannot be used with logical entries.	Correct expression.
2	LOG CONST & ARITH VAR	DATA statement processor cannot perform a meaningful conversion.	Correct DATA statement.
2	LOG OPND & ARITH OPTR	Logical primary detected in arithmetic expression.	Correct expression.
2	")" MISSING	Right parenthesis missing or redundant; left parenthesis present.	Add or delete a parenthesis.
2	MISSING OPND IN STMT	Invalid unit operand in REWIND, ENDFILE, or BACKSPACE statement.	Correct statement.
2	MODES MIXED OVER "="	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.
2	<name> -ADJ. DIM. -MUST BE INTEGER①	Variable must be of type integer.	Correct statement.
2	<name> EQUIV. TO COMMON - INITIALIZED①	A variable made equivalent to a COMMON variable can only be initialized in a BLOCK DATA statement.	Correct statement.
2	<name> EXTENDS COMMON BLOCK NEGATIVELY①	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.
2	<name> IN COMMON TWICE; DUE TO EQUIV.①	Incorrect equivalence set has caused <name> to have two different locations in common.	Correct EQUIVALENCE statement.
2	<name> IN INCONSISTENT EQUIVALENCES.①	Equivalence set is attempting to distort the structure of an array.	Correct EQUIVALENCE statement.
2	<name> MISALIGNED DUE TO EQUIVALENCE①	Variable specified is an improper main storage boundary.	EQUIVALENCE statements should be reorganized.
2	NEED "(" IN ASSIGNED GO TO	Open parenthesis missing after comma in assigned GO TO.	Correct statement.
2	NEED INTEGER	Relative record number in direct access I/O statement must be integer.	Correct statement.
2	NEED INTEGER FOR ADJ DIM	Variable must be of type integer.	Correct statement.
2	NEED INT VAR IN COMP GO TO	Variable for a computed GO TO must be a simple variable of type integer*4.	Correct program.

① The character string <name> is replaced by the name of the variable in error, at the time the message is printed.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 7 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
2	NEED INT*4 IN ASSIGNED GO TO	Variable in assigned GO TO must be of type integer*4.	Correct program.
2	NEED SIMPLE VAR IN COMP GO TO	Variable for a computed GO TO must be a simple variable of type integer*4.	Correct program.
2	NESTED LOG IF	A logical IF cannot be the object statement of a logical IF.	Correct statement.
2	NO. CONST < NO. VAR	DATA statement has more variables than constants.	Correct DATA statement.
2	NO. DIM > 7	An array may have from one to seven dimensions.	Correct statement.
2	NO. DIM > DECLARED	In an EQUIVALENCE statement, the number of dimensions of an array exceeds its declaration.	Correct statement.
2	.NOT. AFTER SUBEXP	.NOT. ends an expression. .NOT. can only begin an expression.	Correct statement.
2	NO. VAR < NO. CONST	More constants than variables appear in the DATA statement.	Correct DATA statement.
2	OPTR ENDS SUBEXP	+, –, *, /, or ** ends an arithmetic expression.	Correct statement.
2	OPTR MISSING	An arithmetic or logical operator was expected where indicated.	Correct statement.
2	"&" OR "." ILLEGAL HERE	Invalid label after "&", or invalid number after ".".	Correct statement.
2	PROGRAM OVERFLOWS SVCT REGION	Object program has become too large.	Break up program.
2	REL AFTER REL	Two consecutive relationals (.NOT., .OR., .AND.) appear in a statement.	Correct statement.
2	SIMPLE VAR. AREA > 32664 BYTES	Too many simple variables in program unit.	Break program unit into smaller units and recompile or use large compiler.
2	STMT ENDS BADLY	Extraneous field encountered on a declarator statement.	Correct statement.
2	STMT ILLEGAL IN BLOCK DATA	A block data subprogram may only contain specification and data initialization statements.	Remove statement.
2	STMT NOT IN THIS FORTRAN	Acceptable keyword not implemented in this compiler.	Remove statement from program.
2	STMT TOO COMPLEX	FORTRAN statement too complicated to complete compilation.	Simplify statement.

Table F—1. FORTRAN IV Compile-Time Diagnostic Messages (Part 8 of 8)

Severity Code	Diagnostic Message	Explanation	
		Reason	Recovery
2	SUBSCRIPT ERROR ON CARD nnnn.	There is an error in the subscripts for an implied DO loop in the DATA statement for card nnnn. Either the subscripts are outside the array boundaries, or a variable subscript is not an indication variable.	Correct DATA statement or array dimensions.
2	SYMBOL TABLE EXCEEDED	Too many symbolic names in program unit.	Shorten program unit or try large compiler.
2	TOO MANY ")"	No corresponding left parenthesis for marked delimiter.	Check statement.
2	TOO MANY CONSTANT EXPRESSIONS	As stated.	Try larger compiler or break up program.
2	TOO MANY CONTINUATION CARDS	FORMAT statement is too large.	Break up statement.
2	TOO MANY EQUIVALENCES' CAN'T PROCESS.	Excessive number of equivalence sets in the program unit.	Reduce number of equivalence groups.
2	TOO MANY EXTERNAL SYMBOLS	Program is calling too many subroutines.	Break up program.
2	TOO MANY NESTED DO STMT	More than 15 nested DO statements.	Change program logic.
2	TOO MANY SUBSCRIPTS	Maximum number of subscripts for an array is seven.	Change program logic.
2	TOO MANY VAR IN ( )	Table overflow in equivalence processing.	Try large compiler, or break up program.
2	UNABLE TO EVALUATE CONSTANT	An unrecoverable error, such as overflow, has occurred trying to evaluate a constant expression.	Correct statement.
2	UNCLASSIFIABLE STMT	Statement cannot be classified because of misspelled keyword or unrecognizable syntax.	Correct statement.
2	UNDEFINED LABEL nnnnn	Transfer of control to nonexecutable statement not permitted.	Change label.

When the current usage of a name conflicts with the previous usage of the same name, the FORTRAN IV compiler produces a diagnostic error message in the form:

text-1 ILLEGAL text-2

where text-1 and text-2 may take on any one of several values. The possible values for text-1 and text-2 with an explanation for each are listed alphabetically by text value, as follows:

- Values for text-1 with explanation

ADJ'ING DIM.
An adjusting dimension variable

ARRAY
An array

COM. ARY
An array that is in a common

COM. VAR.
A variable in a common

DMY. ARG.
A dummy argument of a subprogram

DMY. ARRAY
A dummy argument of a subprogram that is determined to be an array

DMY. FCN.
A dummy argument of a subprogram that the compiler has determined is a function

DMY. SBR.
A dummy argument that the compiler has determined is a subroutine

DMY. SUBPRG.
A dummy argument that the compiler has determined is a subprogram (function or subroutine).

DMY. VAR.
A nonarray dummy argument of a subprogram

DUP. ARG.
Duplicate arguments

ENTRY
A name that has appeared on an ENTRY statement

EQUIV. ARRAY
An array that has occurred on an EQUIVALENCE statement

EQUIV. VAR
A variable that has occurred on an EQUIVALENCE statement

EXT. FCN.
A function that has appeared on an EXTERNAL statement

INIT. ARRAY
An array that has been initialized by a DATA statement or by a type statement

INIT. VAR.
A variable that has been initialized by a DATA statement or by a type statement

NAMELIST
A NAMELIST name

NON-COM. ARY.
An array not in any common

NON-COM. VAR.
A variable not in any common

NON-INT.
A noninteger variable

NON-INT*4
A noninteger*4 variable

REPETITION
A name encountered more than once

STD.FCN.
A FORTRAN standard library subroutine

STD. SBR.
A FORTRAN standard library subroutine

STMT. FCN.
A statement function

SUBPRG
A subprogram (subroutine or function)

THIS FCN.
A name the compiler has typed a function

THIS NAME
A name to which no attributes have been determined

THIS SBR.
A name the compiler has typed a subroutine

TOOLATE.
A use of a name occurred after conflicting use

TYPED VAR.
A variable that has occurred on a type statement

VAR.
A simple variable

- Values for text-2 with explanation

AS ADJ'ING DIM.
When the compiler expects an adjusting dimension variable

AS ASSIGN VAR.
When the compiler expects an ASSIGN variable

AS CALLED SBR.
When the compiler expects a SUBROUTINE name

AS COM. BLOCK
When the compiler expects the name of a common block

AS DMY. ARG.
When the compiler expects a dummy argument of a subprogram

AS ENTRY ARG.
When the compiler expects an argument on an ENTRY statement

AS ENTRY STMT.
On an ENTRY statement

AS FMT.
When the compiler expects a FORMAT statement

AS FMT.; NAMLST
When the compiler is expecting either a FORMAT array or a NAMELIST name

AS INT. VAR.
When the compiler expects an integer variable

AS NAMELIST
When the compiler expects a NAMELIST name

AS STMT. FCN.
When the compiler expects a statement function

FOR DMY. ARGS.
Within the dummy argument list for a subprogram

IN ABNORMAL
In an ABNORMAL statement

IN COM. STMT.
In a COMMON statement

IN DATA STMT.
In a DATA statement

IN DIM. STMT.
In a DIMENSION statement

IN I/O LIST
In the list portion of an I/O statement

IN EQUIV. STMT.
In an EQUIVALENCE statement

IN EXP
Within an expression (logical or arithmetic)

IN EXTERNAL
In an EXTERNAL statement

IN TYPE STMT.
On a type statement

ON LEFT
On the left-hand side of the equals sign

Example 1:

TOOLATE. ILLEGAL AS STMT. FCN.

This error indicates that the compiler has detected a statement that looks like a statement function, but occurs too late in the program unit to be a statement function. This error occurs most frequently when a name is used as an array but is never dimensioned.

Example 2:

REPETITION ILLEGAL IN TYPE STMT.

This error indicates that the name pointed to by the error pointer has already occurred on a type statement.

The FORTRAN IV compiler operation-type diagnostic error messages are listed and described in Table F-2. The severity code has the same values and meanings as shown in Table F-1; it separates the message number and the message text. A blank severity code indicates a serious error.

Table F—2. Operation-Type Diagnostic Messages (Part 1 of 2)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FF800	1	UPDATE 'nnnnnnnn' IS OUT OF SEQUENCE	An out-of-sequence source correction card detected ('nnnnnnnn' is sequence number)	Fix the source correction deck
FF801	2	SOURCE LIBRARY ERROR ON 'PARAM IN=mod/lfd', OPEN ERROR.	Source file specified by LFD-name 'lfd' could not be opened	Check for a missing // LFD lfd
FF802	2	SOURCE LIBRARY ERROR ON 'PARAM IN=mod/lfd', MODULE NOT FOUND.	Source module 'mod' on the file specified by LFD-name 'lfd' could not be found	Check for incorrect spelling of module or LFD-name
FF803	2	SOURCE LIBRARY ERROR ON 'PARAM IN=mod/lfd', DIRECTORY READ ERROR.	A SAT read error occurred while reading the directory of the source file	Check disk drive and/or pack
FF804	2	SOURCE LIBRARY ERROR ON 'PARAM IN=mod/lfd', MODULE IS EMPTY.	Module 'mod' on the source file specified by LFD-name 'lfd' is empty	Re-create source module
FF850		PROGRAM REORDERED	This source module has caused the FORTRAN IV compiler to call its reordering algorithm. Some degradation in compiler performance may occur.	Reordering can be avoided if the specification statements precede the first DATA statement, statement function definition, or executable statement, and if they occur in the following order: IMPLICIT, ABNORMAL, EXTERNAL, type statements (INTEGER, REAL, COMPLEX, LOGICAL, DOUBLE PRECISION) DIMENSION, COMMON, and EQUIVALENCE
FF851		REORDERING FAILED FOR nnnnnn	The program nnnnnn was too complicated to reorder mechanically. Program terminated.	See FF850.
FF900		PROGRAM CHECK IN COMPILER, JOB CANCELED.	A program check has occurred during the compilation	Submit a Software User Report (SUR), including a dump, all compilation printout, and a source listing

Table F—2. Operation-Type Diagnostic Messages (Part 2 of 2)

Message Number	Severity Code	Diagnostic Message	Explanation	
			Reason	Recovery
FF901		COMPILER WORK FILE ERROR nn, JOB CANCELED.	An error has occurred on the compiler work file (WORK1) where nn is: 06 open failed (WORK1 may be missing) 09 attempted to compile more than 100 modules 10 SAT write error 11 SAT read error 12 SAT read or write error	If nn is not one of these values, submit a Software User Report (SUR) with a dump
FF902		SOURCE LIBRARY ERROR nn, 'PARAM IN=mod/lfd', JOB CANCELED.	An unrecoverable error has occurred in processing source module 'mod' from the file 'lfd'. Error code nn has the values: 02 library format error 04 SAT read error 05 GETCS error on update cards 07 SEQ card error	If nn is not one of these error codes, submit a Software User Report (SUR) with a dump
FF903		ERROR PROCESSING OUTPUT FILE, JOB CANCELED.	An unrecoverable error has occurred in writing the object module	If PARAM OUT= was specified, ensure that its LFD was defined
FF904		NO SOURCE FILE: END-OF-FILE ON SPOOLIN, JOB CANCELED	The compiler has no input (no card input or PARAM IN= specified)	Provide input by using a PARAM IN= or by providing a card deck
FF905		PRNTR ERROR, JOB CANCELED.	An unrecoverable error has occurred on compiler's printer file PRNTR. This message is written only to the operator's console.	Make sure that // LFD PRNTR is included in the control stream
FF910		FORTRAN REQUIRES MICRO-LOGIC EXPANSION, JOB CANCELED.	The compiler requires the extended instruction set which is available only if the micro-logic expansion has been loaded	Load the 2K COS and a supervisor that supports floating-point procedures

Appendix G. Run-Time Modules

G.1. FORTRAN RUN-TIME MODULES

Table G-1 provides a list of run-time modules with a description of their function and the entires in each module. For additional information about mathematical routines, see Table 5-4.

G.2. FORTRAN IV STANDARD LIBRARY FUNCTION NAMES

The FORTRAN IV compiler generates external references to special names for calls upon standard library functions. These special names consist of ILF# followed by two characters.

For example, when a FORTRAN program calls for the library SQRT function, FORTRAN IV generates an external reference for the ILF#30 FORTRAN IV name. Table G-2 lists special external reference names and their standard library functions.

Table G—1. FORTRAN IV Run-Time Modules (Part 1 of 12)

Module	CSECT or Entry Name	Function
FD$ABTRM	FL$ABTRM	Abnormal termination code for DTF
FD$CLOSE	FL$CLOSE	Final file close for DTF
FD$DEFIL	FL$DEFIL	Define FILE statement processor for DTF
FD$EDTFI	FL$EDTFI FL$EDTEI FL$EDTDI	F edit – Input for DTF E edit – Input for DTF D edit – Input for DTF
FD$EDTII	FL$EDTII	I edit – Input for DTF
FD$EDTZI	FL$EDTZI	Z edit – Input for DTF
FD$ERCTL	FL$ERCTL FL$WTERR FL$WTMSG	Error control and traceback routine for DTF
FD$ERRN	FL$ERRN	Error message setup for DTF
FD$GTMS3	FL$GTMS3 FL$GTMSG	OS/3 GET message processor for DTF

Table G—1. FORTRAN IV Run-Time Modules (Part 2 of 12)

Module	CSECT or Entry Name	Function
FD$INFL3	FL$INFL3 FL$INFL	Set file to input mode for DTF
FD$INITL	FL$INITL	Program initialization routine for DTF
FD$IO	FL$IO FL$ERRCT FL$FUNTB FL$RERDB FL$WORKA FL$1 FL$3 PRNTR PRNTRC	Standard I/O configuration for DTF
FD$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 for DTF
FD$IOPEN	FL$IOPEN FL$BCKSP FL$DFIND FL$DOPEN FL$ENDFL FL$REWND FL$SOPEN	I/O argument list processor for DTF For BACKSPACE statement For FIND statement For direct access I/O For ENDFILE statement For REWIND statement For SEQUENTIAL I/O
FD$IO1	FL$IO1 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 for DTF

Table G—1. FORTRAN IV Run-Time Modules (Part 3 of 12)

Module	CSECT or Entry Name	Function
FD$OUTF3	FL$OUTF3 FL$OUTFL	Set file to output mode for DTF
FD$READ	FL$READ FL$EOF FL$ERROA	Input file processor for DTF
FD$STOP	FL$STOP FL$PAUSE	STOP/PAUSE processor for DTF
FD$WRITE	FL$WRITE	Output file processor for DTF
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
FLCC	FLCC 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 G—1. FORTRAN IV Run-Time Modules (Part 4 of 12)

Module	CSECT or Entry Name	Function
FLCDD	FLCDD 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 G—1. FORTRAN IV Run-Time Modules (Part 5 of 12)

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**I4 R8**I2 R8**R4 I4**R8 I2**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 G—1. FORTRAN IV Run-Time Modules (Part 6 of 12)

Module	CSECT or Entry Name	Function
FL$DEBUG	FL$DEBUG FL$DARI FL$DBGUN FL$DCHK FL$DELI FL$DINT FL$DRTN FL$DSBT FL$DTOF 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$DHPER DCOSH DSINH	Real*8 hyperbolic sine/cosine
FL$DHYPT	FL$DHYPT DTANH	Real*8 hyperbolic tangent
FL$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 G—1. FORTRAN IV Run-Time Modules (Part 7 of 12)

Module	CSECT or Entry Name	Function
FL$DUMP	FL$DUMP D$MP FL$DUMPD P$UMP	DUMP/PDUMP processor
FL$DVCHK	FL$DVCHK D$CHK	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$ERCTL FL$WTERR 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$ERTST E$ROR E$ROR1	ERROR/ERROR1 subroutines

Table G—1. FORTRAN IV Run-Time Modules (Part 8 of 12)

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 F$TCH	FETCH subroutine
FL$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 G—1. FORTRAN IV Run-Time Modules (Part 9 of 12)

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$IO1	FL$IO1 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 G—1. FORTRAN IV Run-Time Modules (Part 10 of 12)

Module	CSECT or Entry Name	Function
FL$IXPI	FL$IXPI	Integer power functions:
	FL$II	I4**I4
	FL$IJ	I4**I2
	FL$JI	I2**I4
	FL$JJ	I2**I2
	FP$HTH	12**12 (Basic FORTRAN)
	FP$HTI	12**14 (Basic FORTRAN)
	FP$ITH	14**12 (Basic FORTRAN)
	FP$ITI	14**14 (Basic FORTRAN)
FL$LOAD	FL$LOAD	LOAD and OPSYS subroutines
	L$AD	
	O$SYS	
FL$MAX	FL$MAX	Max/min intrinsic functions
	AMAX0	
	AMAX1	
	AMIN0	
	AMIN1	
	JMAX0	
	JMIN0	
	MAX	
	MAX0	
	MAX1	
	MIN	
	MIN0	
	MIN1	
FL$MOD	FL$MOD	Modulo arithmetic intrinsic functions
	AMOD	
	DMOD	
	JMOD	
	MOD	
FL$NAMEI	FL$NAMEI	NAMELIST input
FL$NAMEO	FL$NAMEO	NAMELIST output
FL$OUTF3	FL$OUTF3	Set file to output mode for system
	FL$OUTFL	
FL$OVRFL	FL$OVRFL	Overflow subroutine
	O$ERFL	
FL$OVW70	FL$OVW70	Series 70 overflow subroutine
	O$ERFL	
FL$POWER	FL$POWER	Real*4 power functions
	ALOG	
	ALOG10	
	EXP	
	EXP10	
	FL$ALOG$	
	FLEXP	
	FL$IR	I4**R4
	FL$JR	I2**R4
	FL$RI	R4**I4
	FL$RJ	R4**I2

Table G—1. FORTRAN IV Run-Time Modules (Part 11 of 12)

Module	CSECT or Entry Name	Function
FL$POWER (cont)	FL$RR	R4**R4
	FPEXP	
	FP$HTR	12**R4 (Basic FORTRAN)
	FP$ITR	14**R4 (Basic FORTRAN)
	FP$RTH	R4**12 (Basic FORTRAN)
	FP$RTI	R4**14 (Basic FORTRAN)
	FP$RTR	R4**R4 (Basic FORTRAN)
	LOG	
	LOG10	
FL$READ	FL$READ	Input file processor
	FL$EOF	
	FL$ERROA	
FL$REAL	FL$REAL	Real part of complex intrinsic function
	DREAL	
	REAL	
FL$REOPN	FL$REOPN	Reopen closed file
FL$RWNDA	FL$RWNDA	REWIND statement processor
FL$SCNUM	FL$SCNUM	
FL$SIGN	FL$SIGN	Sign intrinsic functions
	DSIGN	
	FL$DSIGN	
	FL$ISIGN	
	FL$JSIGN	
	ISIGN	
	JSIGN	
	SIGN	
FL$SIN	FL$SIN	Sine/cosine functions
	COS	
	FLCOS	
	FLSIN	
	SIN	
FL$SLITE	FL$SLITE	SLITE/SLITET subroutines
	S$ITE	
	S$ITET	
FL$SNGL	FL$SNGL	Single from double intrinsic functions
	CSNGL	
	SNGL	
FL$SOPNA	FL$SOPNA	
FL$SQRT	FL$SQRT	Real*4 square root function
	SQRT	
FL$SSWTH	FL$SSWTH	System switch subroutines
	S$WTCH	
FL$STOP	FL$STOP	STOP/PAUSE processor
	FL$PAUSE	

Table G—1. FORTRAN IV Run-Time Modules (Part 12 of 12)

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$WRITE	Output file processor
FF$AG	ILF#AG ILF#CG	FORTRAN IV assigned GO TO FORTRAN IV computed GO TO
FF$IC	ILF#IC	FORTRAN IV I/O interface
FF$MPI	ILF#MPI ILF#BUG	FORTRAN IV initialization FORTRAN IV diagnostics
FF$NL	ILF#NL	FORTRAN IV namelist interface
FF$PA	ILF#PA	FORTRAN IV PAUSE
FF$OPNA		Checks operation sequences
FF$SCHK	ILF#SCHK ILF#F0 ILF#F1 ILF#F2 ILF#F3 ILF#F4	FORTRAN IV subscript check processor 1 byte arrays 2 byte arrays 4 byte arrays 8 byte arrays 16 byte arrays
FF$TR	ILF#TR	FORTRAN IV label tracing
FF$XD	ILF#XD ILF#XV	FORTRAN IV complex division FORTRAN IV real/complex
FF$XM	ILF#XM	FORTRAN IV complex multiply

Table G—2. FORTRAN IV Standard Library Function Names (Part 1 of 2)

Name	Standard Library Function
ILF#00	EXP
ILF#01	DEXP
ILF#02	CEXP
ILF#03	CDEXP
ILF#04	ALOG
ILF#05	DLOG
ILF#06	CLOG
ILF#07	CDLOG
ILF#08	ALOG10
ILF#09	DLOG10
ILF#10	ATAN
ILF#11	DATAN
ILF#12	SIN
ILF#13	DSIN
ILF#14	CSIN
ILF#15	CDSIN
ILF#16	JMAX0
ILF#17	MAX0
ILF#18	AMAX1
ILF#19	DMAX1
ILF#20	JMIN0
ILF#21	MIN0
ILF#22	AMIN1
ILF#23	DMIN1
ILF#24	COS
ILF#25	DCOS
ILF#26	CCOS
ILF#27	CDCOS
ILF#28	TANH
ILF#29	DTANH
ILF#30	SQRT
ILF#31	DSQRT
ILF#32	CSQRT
ILF#33	CDSQRT
ILF#34	MOD
ILF#35	JMOD
ILF#36	AMOD
ILF#37	DMOD
ILF#38	CABS
ILF#39	CDABS
ILF#40	ABS
ILF#41	IABS
ILF#42	JABS
ILF#43	DABS
ILF#44	ATAN2
ILF#45	DATAN2
ILF#46	INT
ILF#47	IDINT
ILF#48	AINT
ILF#49	DINT

Name	Standard Library Function
ILF#50	AMAX0
ILF#51	MAX1
ILF#52	AMIN0
ILF#53	MIN1
ILF#54	FLOAT
ILF#55	DFLOAT
ILF#56	IFIX
ILF#57	HFIX
ILF#58	SIGN
ILF#59	ISIGN
ILF#60	DSIGN
ILF#61	DIM
ILF#62	IDIM
ILF#63	SNGL
ILF#64	CSNGL
ILF#65	REAL
ILF#66	DREAL
ILF#67	AIMAG
ILF#68	DIMAG
ILF#69	DBLE
ILF#70	CDBLE
ILF#71	CMPLX
ILF#72	DCMPLX
ILF#73	CDNJG
ILF#74	DCONJG
ILF#75	DDIM
ILF#76	JSIGN
ILF#77	JDIM
ILF#78	HFLOAT
ILF#79	DHFLOT
ILF#80	SINH
ILF#80	COSH
ILF#82	DSINH
ILF#83	DCASH
ILF#84	ASIN
ILF#85	ACOS
ILF#86	DASIN
ILF#87	DACOS
ILF#88	TAN
ILF#89	COTAN
ILF#90	DTAN
ILF#91	DCOTAN
ILF#92	ERF
ILF#93	ERFC
ILF#94	DERF
ILF#95	DERFC
ILF#96	GAMMA
ILF#97	ALGAMA
ILF#98	DGAMMA
ILF#99	DLGAMA

Table G—2. FORTRAN IV Standard Library Function Names (Part 2 of 2)

Name	Standard Library Function
ILF#A0	CSINH
ILF#A1	CDSINH
ILF#A2	CCOSH
ILF#A3	CDCOSH
ILF#A4	EXP10
ILF#A5	DEXP10
ILF#A6	CBRT
ILF#A7	DCBRT

Appendix H. Subroutine Linkage

H.1. CALLING FORTRAN SUBPROGRAMS

All language processors, including FORTRAN IV, generate and expect standard subprogram linkages in their generated programs. These linkage conventions are defined in the supervisor user guide. In addition, special FORTRAN conventions and considerations are required (suggested) for successful operation of the run-time system.

H.1.1. Save Area

A FORTRAN subprogram requires a 72-byte, word-aligned save area, supplied by the calling program. Table H-1 illustrates the format of a save area. Word 1 of the save area is reserved for system use. 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.

Table H—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

Table H—1. Save Area Format (Part 2 of 2)

Word	Byte	Content
11	40	REGISTER 5
12	44	REGISTER 6
13	48	REGISTER 7
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.

H.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.

FORTRAN IV requires that integer*2 simple variables and array arguments contain the actual address minus 2. This permits integer constants to be passed as integer*2 arguments.

H.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 H–2.

Table H—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.

H.1.4. Mathematical Library

The mathematical functions supplied by FORTRAN IV are available to programs written in other languages. Tables 5–3 and 5–4 specify the functions available and Appendix G 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 FORTRAN IV. 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 is:

```
1         10      16
FL$IOARA  START   0
          DC      A(FORTERR)
FORTERR   DS      0H
          USING   FORTERR,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.

H.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.

The FL$INITL routine uses both the program check and abnormal termination island code services of the 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.

H.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         10      16
          LA      R13,SAVEAREA
          LA      R1,PARMLST
          L       R15,=V(PROGNAME)
          BALR    R14,R15
          NOP     X'(linenumber)'
```

H.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 (H.2.2) are not passed in the parameter list. For integer*2 variables and arrays, the parameter list points to two bytes before the location containing the value(s).

NOTE:

*Logical constants are always passed as logical*4 values and integer constants are always passed as integer*4 values.*

H.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 GO TO. 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 $1 \leqslant i \leqslant n$, control passes to the statement at the ith label. Any other values in register 15 cause control to pass to the next sequential statement.

H.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 H-2.

H.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.

```
1         10      16
MYMOD     START
            .
            .
            .
SAVEAREA  DC      18F'Ø'                72 bytes
          DC      X'6'                  length
          DC      CL5'MYMOD'            name
            .
            .
            .
```

3. 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         10      16
MYMOD     START
            .
            .
            .
          ENTRY  MYENT
          USING  MYENT,R15
MYENT     B      *+6
          DC     X'5'
          DC     CL5'MYENT'
```

4. The traceback routine assumes that a half-word line number is located two bytes after the return point (specified in register 14). Refer to H.2 for an assembly language example of this.

Index

D

SPERRY UNIVAC

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:

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

USER COMMENT SHEET

Comments concerning the content, style, and usefulness of this manual may be made in the space provided below. Please fill in the requested information.

This User Comment Sheet will not normally lead to a reply to the originator. Requests for copies of manuals, lists of manuals, pricing information, etc. must be made through your Series 1100 site manager, to your Sperry Univac representative, or to the Sperry Univac office serving your locality. Software problems should be submitted on a Software User Report (SUR) form UD1–745. Questions of a technical nature regarding either the manual or the software should be submitted on a Technical Question (question/answer) form UD1–1195. These forms are available through your Sperry Univac representative.

Customer Name: _____ System Type: _____

Title of Manual: _____

UP No.: _____ Revision No.: _____ Update: _____

Name of User: _____ Date: _____

Address of User: _____

Comments: Give page and paragraph reference where appropriate.

Please rate this manual.	Good	Adequate	Not Adequate
Organization of the text			
Clarity of the text			
Adequacy of coverage			
Examples			
Cross references			
Tables			
Illustrations			
Index			
Appearance			

YOUR COMMENTS, PLEASE - - - -

This manual is part of a library that serves as a source of information for personnel using SPERRY UNIVAC® systems. Space is provided on the opposite side of this form for your comments concerning the usefulness of the information presented. Each comment will be carefully reviewed by the persons responsible for writing and publishing this manual. All comments and suggestions become the property of Sperry Univac.

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

SPERRY UNIVAC

ATTN: Systems Support
1100 Systems Publications
M.S. 4533

P.O. Box 43942
St. Paul, Minnesota 55164

FOLD

CUT

SPERRY UNIVAC

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

SPERRY UNIVAC

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