

PUBLICATIONS UPDATE
Operating System/3 (OS/3)
FORTRAN
Programmer Reference
UP-8193 Rev. 1-C

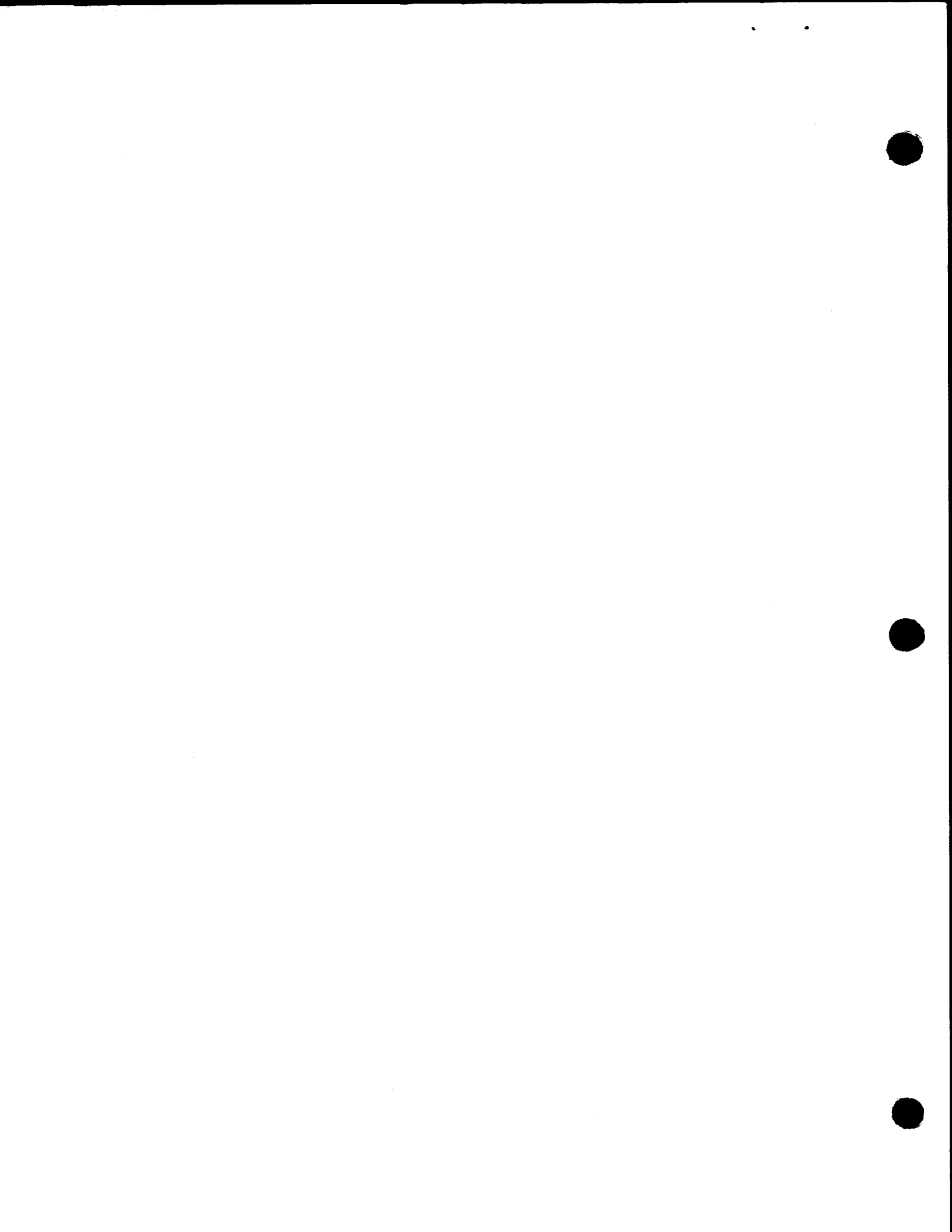
This Library Memo announces the release and availability of Updating Package C to "SPERRY UNIVAC FORTRAN Programmer Reference", UP-8193 Rev. 1.

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

- Specification of catalog file
- Expanded explanations of parameters

Copies of Updating Package C 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-8193 Rev. 1-C. To receive the complete manual, order UP-8193 Rev. 1.

<p>LIBRARY MEMO ONLY</p> <p>Mailing Lists BZ, CZ (less DE, GZ, HA) MZ, 18U, 19U, 20U, 21U, 75U and 76U</p>	<p>LIBRARY MEMO AND ASSOCIATED MATERIALS</p> <p>Mailing Lists DE, GZ, HA, 18, 19, 20, 21, 75 and 76 (Package C to UP-8193 Rev. 2, 7 pages plus Memo)</p>	<p>THIS SHEET IS</p> <p>Library Memo for UP-8193 Rev. 1-C</p> <p>RELEASE DATE: September, 1981</p>
--	--	---



**PUBLICATIONS
UPDATE**

Operating System/3 (OS/3)

FORTRAN

Programmer Reference

UP-8193 Rev. 1-B

This Library Memo announces the release and availability of Updating Package B to "SPERRY UNIVAC FORTRAN Programmer Reference", UP-8193 Rev. 1.

This update incorporates several corrections:

- Sources input cannot be accepted from tape
- A variable containing the value zero is acceptable in a DO control statement

Copies of Updating Package B are now available for requisitioning. Either the updating package alone, or the complete manual may be requisitioned by your local Sperry Univac Representative.

To receive the updating package alone, order UP-8193 Rev. 1-B.

To receive the complete manual, order UP-8193 Rev. 1.

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 B to UP-8193 Rev. 2, Covers and 9 pages plus Memo)	Library Memo RELEASE DATE: October, 1980



**PUBLICATIONS
UPDATE**

Operating System/3 (OS/3)

FORTRAN

Supplementary Reference

UP-8193 Rev. 1-A

This Library Memo announces the release and availability of updating Package A to "SPERRY Univac Operating System/3 (OS/3) FORTRAN Supplementary Reference", UP-8193 Rev. 1.

This Update includes minor corrections and modifies the formats of:

- UNIT macro instruction for card reader definition
- UNIT macro instruction for card punch definition

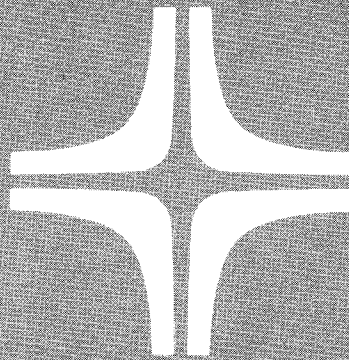
Copies of Updating Package A are now available for requisitioning. Either the updating package alone, or the complete manual with the updating package may be requisitioned by your local Sperry Univac Representative. To receive the updating package alone, order UP-8193 Rev. 1-A. To receive the complete manual, order UP-8193 Rev. 1.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists 18, 19, 20, 21, 75 and 76 (Package A to UP-8193 Rev. 1, 18 pages plus Memo)	Library Memo RELEASE DATE: June, 1979



FORTRAN

OS/3



Programmer Reference

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.

PAGE STATUS SUMMARY

ISSUE: Update C – UP-8193 Rev. 1
RELEASE: 7.1 Forward

Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level
Cover/Disclaimer		B						
PSS	1	C						
Contents	1 thru 6	Orig.						
1	1 2 thru 6	B Orig.						
2	1 thru 5	Orig.						
3	1 thru 5	Orig.						
4	1 thru 4 5 6, 7	Orig. B Orig.						
5	1 thru 12 13 14 thru 20	Orig. A Orig.						
6	1, 2 3 4 thru 7	Orig. A Orig.						
7	1 thru 5 6, 7 8 thru 21	Orig. A Orig.						
8	1, 2	Orig.						
9	1 thru 5	Orig.						
10	1, 2 3, 4 5 thru 7 8 9, 10	Orig. A Orig. A Orig.						
11	1 2, 3	B A						
12	1 thru 4 5 thru 12	C Orig.						
Appendix A	1 thru 4	Orig.						
User Comment Sheet								

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.



1. Introduction

1.1. GENERAL

This manual is intended to introduce the experienced FORTRAN programmer to the SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN. This manual supplements the general information concerning FORTRAN programming available in fundamentals of FORTRAN programmer reference, UP-7536 (current version).

1.2. OS/3 FORTRAN

OS/3 FORTRAN consists of:

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

The FORTRAN compiler accepts source programs, written in the FORTRAN language, which may reside in the control stream or in a source program library on disc. The output of the compiler must then be processed by the linker. (See system service programs user guide, UP-8062 (current version).) During this process, mathematical and I/O routines are taken from the FORTRAN system library and included in the executable program. User-defined procedures, if they are required, are also included during the linker processing. These procedures may be coded in FORTRAN or in some other language. The output of the linker is a load module which 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 loaded automatically by referencing a procedure in an overlay not currently in main storage. ←

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

During compilation, the compiler produces the following optional listings:

- a listing of the source program; each source statement is accompanied by compiler-generated diagnostics; for each diagnostic, the source statement is marked at the character for which the diagnostic is produced;

- a main storage map showing the allocation of the variables and arrays in the program; and
- the object code in the form of a pseudo-assembly language program.

Any of the listings may be suppressed by user options.

The compiler is self-initializing and any number of FORTRAN source programs may be processed by one call of the compiler by the job control program. 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.

1.2.1. Compatibility

OS/3 FORTRAN is an extended American National Standard Basic FORTRAN system. It is a compatible subset of IBM DOS/360 FORTRAN IV and SPERRY UNIVAC Operating System/7 (OS/7) FORTRAN IV, and it is a compatible superset of the IBM TOS/DOS Basic FORTRAN system.

1.2.2. OS/3 FORTRAN Extensions

OS/3 FORTRAN includes the following extensions to American National Standard Basic FORTRAN (X3.10-1966):

- Names may have up to six characters, and up to five digits may be specified for labels.
- Embedded comments are permitted.
- A double precision data type with D and G formats is provided.
- Arrays with three adjustable dimensions are possible.
- ASSIGN and assigned GO TO statements are provided.
- Optional arithmetic IF statement labels are permitted.
- PROGRAM statement is provided.
- Logical IF and executable END statements are provided.
- Generic reference to intrinsic and standard library functions.
- OVERFL, DVCHK, ERROR, ERROR1, SLITE, SLITET, SSWTCH, LOAD, FETCH, DUMP, and PDUMP subroutines are provided.
- Arguments and COMMON storage may be redefined by functions.
- Symbolic names may be typed by the IMPLICIT statement.
- Named COMMON blocks and an EXTERNAL statement are provided.
- End-of-file and error control are available in the READ statement.
- Format descriptors H, Z, A, T, and a literal descriptor are provided.
- An extended G edit capability exists for real and integer data types.

- Print carriage control and list-directed I/O are available.
- The direct access I/O statements DEFINE FILE, READ, WRITE, and FIND are available.
- A DATA statement is provided and block data subprograms may be named.
- Debugging aids include subscript checking, label trace, conditional compilation and formatted main storage dumps.

1.3. SOURCE PROGRAMS

General procedures to be followed in OS/3 FORTRAN programming are presented in the following paragraphs.

1.3.1. Character Set

The OS/3 FORTRAN 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 characters not shown in the table have no graphic equivalents in the OS/3 FORTRAN character set, but these characters can be stored internally and transmitted to and from card, tape, and disc storage.

Table 1-1. OS/3 FORTRAN Character Set

FORTRAN Character Set	
Alphanumerics	A through Z and \$ 0 through 9
FORTRAN Special Symbols	=, () + - * / . ' ;
Blank	written on the coding form as Δ or as a blank space
SPERRY UNIVAC 90/30 System Special Character Set*	
> < % ! : @ # ? _ (underscore) " (double quote) &	

*The special character set can change with the options selected for the system printer, with up to 127 characters available, depending on printer.

1.3.2. FORTRAN Statements

OS/3 FORTRAN statements are coded on the FORTRAN coding form in columns 1 through 72. All characters in a FORTRAN line are restricted to the FORTRAN character set, except in comments and literal constants where the special character set may be used. Columns 73 through 80 on the form are ignored by the compiler and can be used in any manner by the programmer. The information in these columns is printed in the source program listing.

Each FORTRAN statement is written in columns 7 through 72. The first line used for 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 or nonblank character in column 6 for each line that is a continuation. Therefore, a FORTRAN statement may consist of one initial line followed by any number of continuation lines. The capacity of the compiler to accept large statements is limited only by the amount of main storage available; the maximum capacity is achieved when long statements appear as early in the program as is practical.

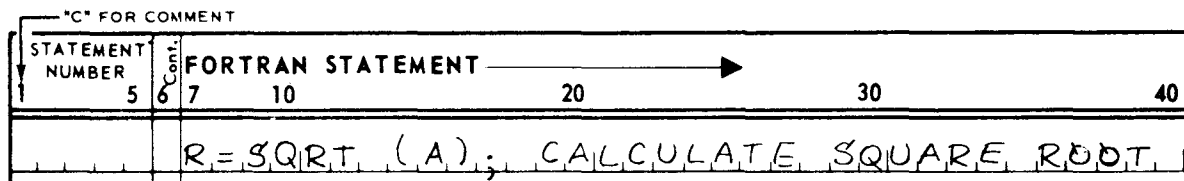
A statement label consists of one through five decimal digits in columns 1 through 5. The contents of these columns for continuation lines are ignored during program compilation (except for an X in column 1) but are shown on the program listing and may be used by the programmer. Leading zeros and embedded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

A special use of column 1 is indicated by an X coded there during program debugging (Section 9), and compilation (Section 11).

1.3.3. Comments

The compiler provides four methods of entering comments: columns 73 through 80 and columns 1 through 5 on continuation lines; the comment line; and embedded comments.

A comment line is indicated by the character C in column 1. Each comment line is shown on the program listing, but is ignored by the compiler. A semicolon in columns 7 through 71 in a FORTRAN line indicates that the information immediately following and written on the same line is to be treated as comments; for example:



A comment following a semicolon can be continued on a succeeding line by specifying a C in column 1. A comment following a C in column 1 may also be followed by a number of continuation lines specified by a nonzero or nonblank character in column 6 of each succeeding line.

Example:

↑

		DO 1100 I=1,9; BEGIN ITERATION
C		LOOP FOR THE CALCULATION OF
		CUBE ROOTS

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.

1.3.4. Symbolic Names

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

1.3.5. Source Statement Order

Table 1-2 shows the order in which the source statements of each program unit must be written.

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 or SUBROUTINE statement. Subprograms are headed by one of those statements. The term program unit is used for any main program or subprogram. All program units are terminated with an END statement.

Table 1-2. Source Statement Order

line 1	Program Declarators:		PROGRAM SUBROUTINE	FUNCTION BLOCK DATA
	COMMENT	FORMAT	IMPLICIT	
			COMMON DIMENSION DOUBLE PRECISION	INTEGER EXTERNAL REAL
			EQUIVALENCE	
			DEFINE FILE	DATA
	Arithmetic Assignment Arithmetic IF ASSIGN Assigned GO TO BACKSPACE CALL Computed GO TO Logical IF CONTINUE DO FIND ENDFILE PAUSE PRINT READ PUNCH RETURN REWIND STOP TRACE ON TRACE OFF Unconditional GO TO WRITE			
line n	END			

NOTE:

Vertical lines demarcate statements which may be intermixed; for example, FORMAT statements may appear anywhere between the program declarator (if used) and the END line.

Horizontal lines demarcate groups of statements which must be specified in the order shown. The dotted horizontal lines indicate that EQUIVALENCE statements must follow any of those specification statements which specify items to share storage; DATA statements must follow any specification statement that mentions an item to be initialized.

1.4. STATEMENT CONVENTIONS

Conventions used to illustrate FORTRAN statements throughout this manual 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.

2. Data Types

2.1. GENERAL

The data types available are integer, real, double precision, and literal. For additional information concerning data types in the SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN language, refer to fundamentals of FORTRAN programmer reference, UP-7536 (current version). For additional information on the hardware characteristics of the SPERRY UNIVAC 90/30 System affecting integer and real computation, see the processor programmer reference, UP-8052 (current version). Data types are categorized for manipulation by the FORTRAN program. Data may appear as constants, variables, or elements in an array. Each of these categories is explained in this section.

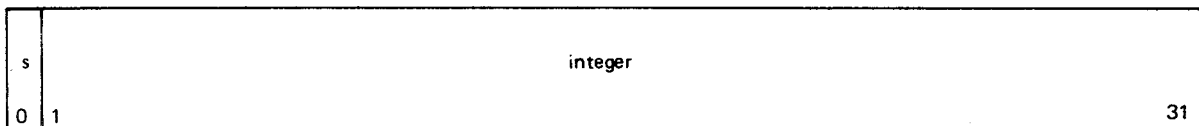
2.2. CONSTANTS

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

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.

An integer constant has the following representation in storage:



where:

s

Is the sign bit.

integer

Is the 31-bit binary integer with a maximum value of 2,147,483,647; if negative, the value is stored in twos complement form.

2.2.2. Real Constants

Real constants may be written as:

- a basic real constant which is an optionally signed string of up to seven significant digits with a decimal point preceding, embedded in, or following the string, such as +0000.1701.
- 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, 0000170.1E-03 is a valid real constant.
- an integer constant followed by a decimal exponent; if the integer portion exceeds the permitted seven digits, truncation of the excess rightmost digits results; +1701E-4 and 17010E-05 are valid real constants.

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

s	characteristic	fraction
0	1 7	8 31

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 located immediately to the left of bit 8.

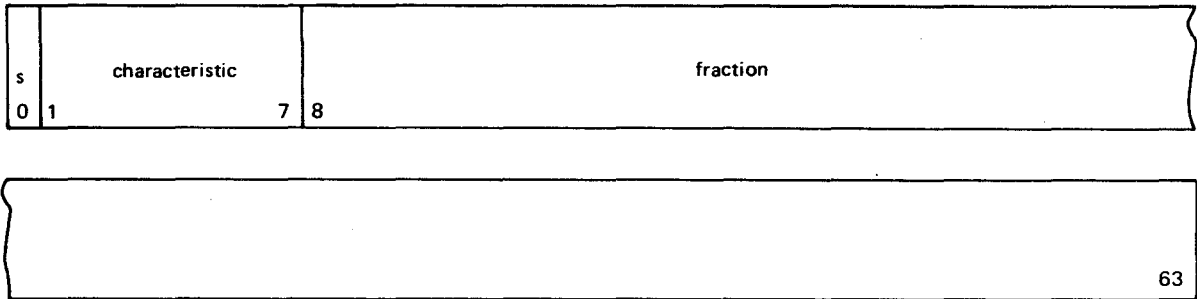
The maximum range for a real constant is from approximately 10^{-78} to 10^{75} or it may be 0.

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 double precision constant, which is an optionally signed string of 8 through 16 significant digits with a decimal point preceding, embedded in, or following the string, such as -180018101820.
- a basic real constant, a basic double precision 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; -.180018101820D12 is a valid double precision constant.

A double precision constant is stored like a real constant except that two words of main 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. Literal Constants

A literal constant consists of one or more characters from the SPERRY UNIVAC 90/30 System character set. Each character in a string requires one byte of main storage.

Two methods of writing literal constants are:

1. as a Hollerith constant in the form $wHc_1c_2...c_w$, where each c represents a constant character; or
2. as a character string enclosed in apostrophes: $'c_1c_2...c_n'$. If the apostrophe occurs in a string, it is represented by doubling that character.

The literal DO NOT would be represented by the Hollerith constant 6HDO NOT and the literal constant DON'T would be represented by 'DON''T' using the second method of writing literal constants.

2.2.5. Hexadecimal Constants

Hexadecimal constants are written as the letter Z followed by a string of hexadecimal digits; the hexadecimal digits and their equivalents are:

Hexadecimal Digit	Decimal Value	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
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 in DATA statements. Each hexadecimal digit occupies one half-byte of main storage. If the constant specifies more digits than can be stored in the associated variable, it is truncated on the left; if the constant specifies fewer digits, zeros are padded on the left.

2.3. VARIABLES

A variable is represented by a symbolic name (1.3.4) which identifies a single value. A data type is associated with a variable and there is both a standard and an optional length specification which determines the number of bytes assigned in main storage. The optional length specifications are shown in Table 2-1.

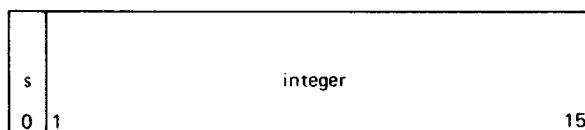
Table 2-1. Data Types and Optional Lengths

90/30 Standard Data Type	ANSI Standard Name	Length in Bytes	Optional Data Type	Length in Bytes
Integer*4	Integer	4	Integer*2	2
Real*4	Real	4	Real*8	8
Double Precision	Double Precision	8	None	

The type associated with a variable is determined by explicit type declaration statements (6.4.1), by the IMPLICIT statement (6.4.2), or by the variable named used. Names beginning with the letters I, J, K, L, M, or N are assumed to represent integer values; all names beginning with other letters are assumed to represent real values. Explicit type statements have the highest precedence and the IMPLICIT statement the next highest in this convention.

In this manual, to prevent confusion when the length can differ, the complete data type will appear: a reference to 8-byte real data will be written as real*8. Note that real*8 is the equivalent of double precision, but double precision variables have only a standard length. There is no variable type associated with literal data. The optional length described may be specified in either the explicit type statements or IMPLICIT statement.

The internal representation of the values of variables is identical to that described for constants, with the exception of integer*2 where there is no corresponding constant. The integer*2 variable occupies two bytes with the optional sign stored in the most significant bit:



The maximum value of an integer*2 variable is 32,767; note, however, that the hardware cannot provide overflow and underflow indications if this range is exceeded — numeric bits can be propagated into the sign position by arithmetic. For this reason, exercise discretion when using integer*2.

2.4. ARRAYS

An array is an ordered set of values. Each value is called an array element and the entire set is identified by a symbolic name known as an array name. An array is described by an array declarator which is explained in Section 6. An array can be declared as having a maximum of three 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 parentheses are subscripts. Each subscript must be an unsigned integer constant except when a dimension is adjustable. In this last case, the subscript 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:

$$n(s_1, s_2, s_3)$$

where:

n

Is the array name.

s

Is an integer expression of the form $c*v \pm k$, and both c and k are integer constants, and v is an integer variable. This provides seven kinds of subscript expressions:

- v k
- $v+k$ $v-k$
- $c*v$ $c*v+k$
- $c*v-k$

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 the first element are presented in Table 2-2. In the table, the first byte of the array is byte 0; the letters a , b , and c refer to the value of a subscript expression in an array element reference; the subscript expression corresponds to dimensions A , B , and C in the array declarator; the m is a multiplier determined by the number of bytes required for each array element.

Table 2-2. Relative Locations of Array Elements

Number of Dimensions	Declarator Form	Subscript Form	Relative Location of the 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)	$((a-1)+A*(b-1)+A*B*(c-1))*m$

Examples:

If an array declarator were AGO(17), if the element referenced is AGO(4), and if the elements are real types, 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 AGO(4) is the twelfth byte from the beginning of the array.

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



3. Expressions and Assignment Statements

3.1. GENERAL

This section discusses the use of expressions in SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN programming and describes the assignment statements. For more information, see fundamentals of FORTRAN programmer reference, UP-7536 (current version).

3.2. EXPRESSIONS

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

3.2.1. Arithmetic Expressions

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

3.2.2. Relational Expressions

A relational expression consists of two arithmetic expressions joined by a relational operator. This type of expression is actually a subset of logical expressions and is always evaluated as either TRUE or FALSE.

When mixed-mode arithmetic comparisons are made, the priorities of the data types are:

<u>Data Type</u>	<u>Priority</u>
real*8 (double precision)	1
real*4	2
integer*4	3
integer*2	4

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

3.2.3. Logical Expressions

A logical expression is:

- a relational expression,
- a logical or relational expression preceded by `.NOT.`, or
- two logical or relational expressions joined by `.AND.` or `.OR.`

3.2.4. Evaluation Order

An expression is evaluated according to the following rules:

1. Each operator has a priority as shown in Table 3-1. Operations are performed in this order of priority except when modified by the other rules.

Table 3-1. FORTRAN Operators and Evaluation Order

Operation	Operator	Order or 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

2. Begin with the leftmost operator.
3. The current operator is compared to the operator on its right if no parentheses intervene. If the priority of the current operator is greater than or equal to the priority of the next operator, the current operation is performed. Otherwise, the next operator becomes the current operator and this step is repeated using it as the basis for comparison.

4. Upon encountering the right end of an expression, remaining operations are performed from right to left.
5. Sequential exponentiation is performed from right to left. For example, $X^{**}Z^{**}Y$ is evaluated as $X^{**}(Z^{**}Y)$.
6. Expressions in parentheses are treated as single operands and are evaluated first, starting with the innermost parenthesized expression, before the left to right comparisons are continued.

3.2.5. Mixed-Mode Arithmetic

Mixed-mode arithmetic occurs when an operation is performed on two operands which are not the same type. The type and length of the result in such situations is shown in Table 3-2.

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

		First Operand Type			
		Integer*2	Integer*4	Real*4	Real*8 (Double Precision)
Second Operand Type	Integer*2	Integer*4	Integer*4	Real*4	Real*8
	Integer*4	Integer*4	Integer*4	Real*4	Real*8
	Real*4	Real*4	Real*4	Real*4	Real*8
	Real*8 (Double Precision)	Real*8	Real*8	Real*8	Real*8

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 or ERROR(i)

Used to set and test indicators.

See 5.6.3 for more information on these standard library subroutines.

3.3. ASSIGNMENT STATEMENTS

A value is assigned to a variable or an array element by executing an arithmetic assignment statement. This value is the current value until the variable or array element is redefined.

3.3.1. Arithmetic Assignment Statement

Format:

$v=e$

where:

v

Is any type variable name or an array element name.

e

Is any type arithmetic expression.

Description:

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

Table 3-3 demonstrates the conversion of the expression e to the data type of the receiving variable represented by v . The conversions are accomplished by intrinsic functions (6.1).

Table 3-3. Assignment Statement Conversions

		e			
		Integer*2	Integer*4	Real*4	Real*8 (Double Precision)
v	Integer*2	None	*	*	*
	Integer*4	**	None	IFIX(e)	IFIX(SNGL(e))
	Real*4	†	FLOAT(e)	None	SNGL(e)
	Real*8	†	DFLOAT(e)	DBLE(e)	None

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

**The sign is extended.

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

3.3.2. ASSIGN Statement

Format:

ASSIGN k TO i

where:

k

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

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 (4.6). Once the integer variable name has been assigned a value by the ASSIGN statement, it can then be used for no other purpose until it is redefined. For instance, it cannot be used in an arithmetic expression unless its value is redefined by an arithmetic assignment statement or a READ statement.



4. Control Statements

4.1. GENERAL

Control statements are executable instructions which modify the normal sequence of program execution. The control statements used in SPERRY UNIVAC Operating System/3 (OS/3) are identical in function to those described in the Control Statements section of fundamentals of FORTRAN programmer reference, UP-7536 (current version).

4.2. ARITHMETIC IF

Format:

IF (e) k_1, k_2, k_3

where:

e

Is any integer, real, or double precision expression.

k

Is a statement label in the same program unit.

Description:

The arithmetic IF control statement is used to transfer control to specified statements within the program depending on the evaluation of an arithmetic expression.

If the arithmetic expression value is negative, control is passed to statement labeled k_1 ; if 0, to the statement labeled k_2 ; and, if the value is positive, to the statement labeled k_3 . If any label is missing, control is passed to the next executable statement below the IF control statement when the conditions for the missing label are met. Trailing commas may be omitted from the control statement when labels are not specified.

When using the arithmetic IF control statement, remember that the internal representation of real and double precision values is an approximation. One of these value types could be stored as a nonzero approximation of zero.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Column	7	10	20	30
5		IF(I-1) 1,0,20			
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 executable 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 only when BETA is greater than 1.5.

4.3. LOGICAL IF

Format:

IF (e) s

where:

e

Is any logical expression.

s

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

Description:

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

Examples:

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

If both relational expressions (A.GT.B, C.LT.D) are TRUE, the GO TO control statement is executed and control passes to statement 20. If either expression is evaluated as FALSE, the GO TO statement is ignored and control passes to the following statement.

The WRITE statement in the example is executed if the value represented by A is greater than that represented by B. Otherwise, control passes to the next executable statement.

4.4. UNCONDITIONAL GO TO

Format:

GO TO k

where:

k

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

Description:

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

4.5. COMPUTED GO TO

Format:

GO TO (k₁,k₂,...,k_n), i

where:

k

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

i

Is an integer*4 variable, the value of which must be defined using an arithmetic assignment or READ statement before the execution of the computed GO TO control statement.

Description:

The computed GO TO control statement permits the transfer of control to a statement whose label occupies the position in the GO TO list which is equal to the value of i. For instance, if the value of i were 4, control would be transferred to the statement labeled with the fourth label in the list of labels in the computed GO TO control statement. If i, the integer variable, is negative, zero, or greater than the number of labels in the list, control is transferred to the next executable statement following GO TO control statement.

Example:

"C" FOR COMMENT		STATEMENT NUMBER		FORTRAN STATEMENT		
1	5	6	7	10	20	30
			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 is 3, control is transferred to statement 35; and so on. Any value other than 1 through 4 results in a transfer of control to the statement following the GO TO control statement.

4.6. ASSIGNED GO TO

Format:

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

where:

i

Is the name of an integer*4 variable.

k

Is a statement label of an executable statement within the same program unit; the list of labels is optional and may be omitted.

Description:

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

Example:

"C" FOR COMMENT		STATEMENT NUMBER		FORTRAN STATEMENT		
1	5	6	7	10	20	30
				GO TO K5, (10, 13, 15, 17, 18, 21)		

When the current value of the integer variable K5 matches one of the statement labels in parentheses, control is transferred to the statement with that label. The value of the integer variable could have been defined only by an ASSIGN statement (3.3.2).

4.7. DO

Format:

DO n i=m₁,m₂,m₃

where:

n

Is the statement label of the terminal statement of the DO loop.

i

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

m₁

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

m_2

Is the terminal parameter which 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 incremental parameter, the value of which is added to the value of control variable i after each execution of the DO loop and before the comparison of m_2 and the control variable; when this parameter is omitted, 1 is assumed.

Description:

A DO control statement initiates and controls the repeated execution of the group of statements within the DO range. The DO range extends from the first executable statement following the DO control statement to the terminal statement. The terminal statement is the FORTRAN statement following the DO control statement with the statement label specified within the DO control statement.

Either positive integer constants or integer variable names may be used as parameters for the DO control statement. A value of ZERO may be obtained by using a variable set to ZERO.

Example:

"C" FOR COMMENT	
STATEMENT NUMBER	FORTRAN STATEMENT
5	DO 12, K=2, 8, 3
6	:
12	CONTINUE
13	PRINT 100, K

The statements starting with the first executable statement following the DO control statement and ending with statement 12 are executed three times, with K having the values 2, 5, and 8. When statement 13 is executed, K is printed; at that time, its value is 8.

4.7.1. Transfers of Control From and To a Do Range

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

4.8. CONTINUE

Format:

CONTINUE

Description:

The CONTINUE control statement serves as a terminal statement of a DO range when the terminal statement would otherwise be a control statement. It produces no coding and may be used anywhere in the program, subject to the ordering shown in Table 1-2, without affecting the logic of program execution. When used as the terminal statement of a DO range, the CONTINUE control statement must have a statement label.

4.9. STOP

Formats:

STOP**STOP n**

where:

n

Is an unsigned integer constant of not more than four digits. (Although the compiler accepts values of five digits for compatibility purposes, the console displays only the four least significant digits.)

Description:

The STOP control statement terminates job step execution and returns control to the operating system, indicating the logical end of a program. When a STOP n control statement is executed, a message is produced at the system console. The STOP display also appears on the printer with the n value plus a count of the number of program check interrupts which occurred during program execution. No operator response is necessary.

4.10. PAUSE

Formats:

PAUSE**PAUSE n**

where:

n

Is an unsigned integer constant of not more than five digits.

Description:

The PAUSE control statement halts program execution and produces a message at the system console. The operator then has the choice of permitting the program to resume execution at the next executable statement or of terminating 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 have a statement label. When the END control statement is executed in a main program, it is interpreted as a STOP control statement and the display is identical to STOP. When the END control statement is executed in a subprogram, it is equivalent to a RETURN statement (5.4.1.2).

4.12. PROGRAM

Format:

PROGRAM s

where:

s

Is a 1- to 6-character name that is to be assigned to the object module produced by the compiler.

Description:

The PROGRAM control statement is optionally used to name a main program. When used, it must be the first statement present in the main program. If a PROGRAM statement is not present in a main program, the object module is assigned the name \$MAIN by default. When multiple main programs are being compiled in a single job, each must be assigned a unique name so that they may all be accessible to the linkage editor and librarian. Otherwise, only the last program compiled is accessible.



5. Functions and Subroutines

5.1. GENERAL

When a calculation or series of calculations is required repeatedly in a SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN program, the statements used to perform the calculations can be coded once as a procedure. This procedure can then be referenced each time the calculations are to be performed. Procedures, as explained here and described in fundamentals of FORTRAN programmer reference, UP-7536 (current version) may be categorized by:

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

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

Table 5-1. OS/3 FORTRAN Procedures

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

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

Functions always transfer values, but subroutines may or may not. When value transfers are made, they are accomplished by using arguments or COMMON. Arguments are included as part of the procedure definition; these are formal or dummy arguments. Arguments are also specified in the procedure reference; these are actual arguments. Substitutions are made by the FORTRAN compiler and a value is returned after the procedure is executed; the return of values may also be accomplished with arguments.

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

Table 5-2. Actual 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
External procedure name	No	Yes	Yes

To use procedures in a FORTRAN program, they must be referenced as described in 5.2. The user-coded procedures must be defined as described in 5.3 for statement functions, and 5.4 for external functions and subroutines. Argument substitution, which applies to all procedures, user-coded or not, is explained in 5.5, and the library procedures supplied by Sperry Univac are described in 5.6.

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 with the general function reference. The function reference is used within an expression in a FORTRAN statement and has the form:

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

where:

f

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

a

Represents an actual argument; at least one is required.

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

Examples:

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

In the first example, 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. Although the actual arguments are both integer and real types, an integer value is returned to the referencing expression because the function name is implicitly integer.

5.2.2. Subroutine Reference (CALL Statement)

Format:

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

where:

s

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

a

Represents an actual argument. This argument list is optional and must be enclosed in parentheses when used.

Description:

All subroutines, whether written by the user or supplied with the compiler, are referenced with the CALL statement. It is used to transfer control to the subroutine specified by s. The maximum number of actual arguments permitted is 511; the allowed argument forms are shown in Table 5-2 for user-coded procedures and described in 5.6.3 for standard library subroutines.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
5					
		CALL	PGNUM		
		CALL	DVCHK	(INER)

Two subroutines are referenced by the calls in the example. In the first CALL statement, control is transferred to the subroutine PGNUM; no transfer of values is to take place, so no arguments are specified. When the next line is executed, the standard library subroutine DVCHK is called; the actual argument INER is specified.

5.3. STATEMENT FUNCTION DEFINITION

The user-coded functions are the external function and the statement function. The former is coded as a subprogram as described in 5.4, but the statement function is defined with only one FORTRAN statement.

Statement functions require at least one argument and return only one value to the referencing statement. They are referenced with the function reference previously described. After evaluation of the statement function, control is returned to the expression within the referencing statement.

The format of the statement function definition statement is:

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

where:

f
Is the symbolic name assigned to the statement function.

a
Is a dummy argument consisting of a variable name.

e
Is a limited arithmetic expression.

The statement function definition statement defines a function which 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 (Table 1-2).

A limited expression is an arithmetic expression which may not contain an array element reference or a reference to a statement function that is subsequently defined. For example,

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	LINE	7	10	20	30
5	6				
		QU (A) = 2.0*SQRT (A)			
		AVR (A,B,PNT) = A**B+QU (PNT)			

is permitted, but

		AVR (A,B,PNT) = A**B+QU (PNT)			
		QU (A) = 2.0*SQRT (A)			

is not.

The value returned by the statement function is typed by the statement function name. The name is typed according to the rules for variables described in 2.3, or it is typed by a type statement (6.4). 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.

Dummy argument names in the statement function definition may appear as variable names in the same program unit. A maximum of 511 arguments may be used in the definition.

5.4. SUBPROGRAM DEFINITION

The user-coded procedures defined as subprograms are the external function and the subroutine. The definitions are described in the following paragraphs.

5.4.1. External Functions

An external function is a user-coded function procedure requiring more than one FORTRAN statement for its definition, requiring at least one argument, referenced by a function reference, and returning at least one value to the referencing statement. After evaluation of an 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 procedure name, not by the data types of the arguments. The data type of the function name is decided by the first letter of the function name (2.3), by a type statement in the same program unit, or may be specified in the FUNCTION statement.

5.4.1.1. FUNCTION Statement

Format:

t FUNCTION f*s (a₁,a₂,...,a_n)

where:

t

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

f

Is the symbolic name used to identify the function; because system routines use a dollar sign as the third character of the function name, avoid these names to prevent conflict. The name must be assigned a value, using a READ or arithmetic assignment, in order to define the function value.

*s

Is an optional length specification allowing the use of the optional lengths for variables (2.3); this option may only be used when the data type option is used and the type specified is not DOUBLE PRECISION.

a

Is a dummy argument.

Description:

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

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	LINE	7	10	20	30
		INTEGER FUNCTION XX1*2 (A)			
		RETURN			
		END			
		FUNCTION YY1 (B, C, D, H)			
		RETURN			
		END			

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

5.4.1.2. RETURN Statement

Format:

RETURN

Description:

The RETURN statement causes control to be transferred from the subprogram used to define the external function or subroutine to the program unit which referenced the subprogram.

5.4.2. Subroutines

User-coded subroutines are procedures which, like external functions, are separately compiled as subprograms. Unlike external functions, 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 a SUBROUTINE statement (5.4.2.1);
- are referenced with a CALL statement (5.2.2); and
- return control to the first executable statement following the CALL statement.

Subroutines have a maximum of 511 arguments. The argument forms permitted are shown in Table 5-2.

5.4.2.1. SUBROUTINE Statement

Format:

SUBROUTINE s (a₁,a₂,...,a_n)

where:

s

Is a symbolic name identifying the subroutine. Avoid the use of the dollar sign as the third character of the subroutine name since this convention is used by system routines.

a

Is a dummy argument; this argument list is optional and, when included, it is enclosed in parentheses. Each argument may be a variable, array, or procedure name.

Description:

The SUBROUTINE statement defines the subroutine name and dummy arguments, and must be first statement of the subprogram.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	Col.	7	10	20	30	40
		:				
		SUBROUTINE MATADD(IN1, IN2, OUT, ROW, COL)				
		REAL IN1(1), IN2(1), OUT(1)				
		INTEGER ROW, COL				
		LIMIT = ROW*COL				
		DO 1, I=1, LIMIT				
1		OUT(I) = IN1(I)+IN2(I)				
		RETURN				
		END				
		:				

This subroutine might be referenced with a CALL such as:

```
CALL MATADD (A,B,C,10,10)
```

5.4.2.2. Subroutine RETURN Statement

The RETURN statement (5.4.1.2) is used to return control from a subroutine subprogram. Control is always returned to the first executable statement following the CALL statement.

5.5. ARGUMENT SUBSTITUTION

When a procedure is referenced, the actual arguments, if any, are substituted for the dummy arguments in the procedure receiving control. Two methods of argument substitution are provided:

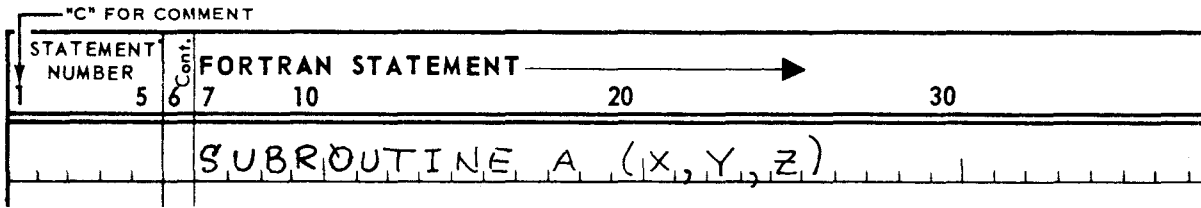
1. call by value, and
2. call by name (or call by address).

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 and FUNCTION statements are simple variables. All statement function arguments are called by value. For a procedure reference such as

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

and the procedure definition



the compiler generates a calling sequence for the CALL or function reference, and it generates a prologue for the SUBROUTINE or FUNCTION statement. The calling sequence consists of a transfer of control to the start of the procedure and a list of main storage addresses containing the actual arguments. The prologue contains instructions which perform the argument substitution. For the examples given, 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 in the procedure, an epilogue is executed. This reverses the substitutions and transmits the values of the dummy arguments to the calling program if they were redefined. For example, in the following, the subroutine on the left is treated as if it were written like the subroutine on the right.

<pre> SUBROUTINE A(B) . . . 100000 . 100001 RETURN </pre>	<pre> SUBROUTINE A B = actual argument ; PROLOGUE START GO TO 100001 ; PROLOGUE END actual argument = B ; EPILOGUE START RETURN ; EPILOGUE END CONTINUE . . . GO TO 100000 </pre>
---	--

Care must be taken that the unintended redefinition of variables with multiple associations does not take place. This can occur when variables are used in COMMON or in argument lists; for example:

```

:
:
CALL X(Y, Z, Y)
:
:
:
SUBROUTINE X (A1, A2, A3)
:
:
END

```

In the example, when the epilogue is executed, the values of the dummy arguments in the subroutine definition are returned to the actual arguments in sequence. First, the value of A1 is substituted for Y in the referencing program unit, then the value of A2 is substituted for Z, and finally, the value of A3 overlays the variable Y again. The initial substitution, $Y = A1$, is lost to the referencing program unit which may not be the programmer's intention. A similar problem can occur when two subprograms share COMMON storage and a COMMON variable occurs in the actual argument list.

5.5.2. Call by Name

The call-by-name method of argument substitution is the standard method of argument substitution when the dummy arguments in SUBROUTINE or FUNCTION statements are declared to be arrays or procedure names. In these cases, the prologue copies the address of the dummy 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 value of the array element for computational purposes. See 6.2.1 for additional information or array declarator processing. Procedure names are processed in the same fashion.

5.6. LIBRARY PROCEDURES

Three classes of procedures are available to the programmer as part of the FORTRAN library: intrinsic functions, standard library functions, and standard library subroutines.

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 (5.2.1) by the user's FORTRAN program. After control is transferred to the function and it is evaluated, control is returned to the expression containing the function reference.

Since the compiler provides a large number of intrinsic functions, generic name references are permitted. For example, the generic group ABS contains three members (ABS, IABS and DABS) which perform identical functions but differ in their argument types and function values. Therefore, if ABS is referenced with a double precision argument, the compiler will automatically generate a call to the proper member (DABS) to determine the absolute value.

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.

Like the intrinsic functions, the standard library functions may be referenced using the generic name; for example, if SIN is referenced using a double precision argument, the compiler will automatically reference DSIN.

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 DABS	Real*4 Integer*4 Double precision	Real*4 Integer*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.)	AMOD MOD DMOD	Real*4 Integer*4 Double precision	Real*4 Integer*4 Double precision
{ MAX } { MAX0 }	Select the largest value	≥ 2	AMAXO* AMAX1 { MAX } { MAX0 } MAX1* DMAX1	Integer*4 Real*4 Integer*4 Real*4 Double precision	Real*4 Real*4 Integer*4 Integer*4 Double precision
{ MIN } { MIN0 }	Select the smallest value	≥ 2	AMINO* AMIN1 { MIN } { MIN0 } MIN1* DMIN1	Integer*4 Real*4 Integer*4 Real*4 Double precision	Real*4 Real*4 Integer*4 Integer*4 Double precision
	Convert argument from integer to real or double precision	1	FLOAT* DFLOAT*	Integer*4 Integer*4	Real*4 Double precision
	Convert argument from real to integer	1	IFIX* HFIX*	Real*4 Real*4	Integer*4 Integer*2
SIGN	Replace the algebraic sign of the first argument with the sign of the second argument	2	SIGN ISIGN DSIGN	Real*4 Integer*4 Double precision	Real*4 Integer*4 Double precision

*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
DIM	Positive difference; subtract the smaller of the two arguments from the first argument	2	DIM IDIM DDIM	Real*4 Integer*4 Double precision	Real*4 Integer*4 Double precision
SNGL	Convert double precision to real	1	SNGL	Double precision	Real*4
DBLE	Convert from real to double precision	1	DBLE	Real*4	Double precision

Table 5-4. Standard Library Functions (Part 1 of 2)

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 $-1 \leq y \leq 1$
	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 $-1 \leq y \leq 1$
	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 $-M \leq y \leq M$
	{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) $-\pi \leq y \leq \pi$

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

General Operation	Generic Name	Member Name	Mathematical Definition	Argument			Function Value Type and Range
				Number	Type	Range	
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 $-M \leq y \leq M$
	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 $1 \leq y \leq M$
	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 $0 \leq y \leq M$
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 $y \geq -180.218$ $y \leq 174.673$
Common logarithm	{ ALOG10 LOG10 }	{ ALOG10 LOG10 }	$y = \log_{10} x$	1	real*4	$x > 0$	real*4 $y \geq -78.268$ $y \leq 75.859$
		DLOG10		1	real*8	$x > 0$	real*8 $y \geq -78.268$ $y \leq 75.859$
Square root	SQRT	SQRT	$y = \sqrt{x}$ or $y = x^{1/2}$	1	real*4	$x \geq 0$	real*4 $0 \leq y \leq M^{1/2}$
		DSQRT		1	real*8	$x \geq 0$	real*8 $0 \leq y \leq M^{1/2}$
Cube root	CBRT	CBRT	$y = x^{1/3}$	1	real*4	any real argument	real*4 $-M^{1/3} \leq y \leq M^{1/3}$
		DCBRT		1	real*8	any real argument	real*8 $-M^{1/3} \leq y \leq M^{1/3}$

NOTE:

$M = 16^{63}(1-16^{-6})$ for real*4 and $16^{63}(1-16^{-14})$ for real*8



5.6.3. Standard Library Subroutines

The standard library subroutines are subprogram procedures supplied with the compiler. Like user-coded subroutines, these procedures are accessed with the CALL statement (5.2.2) and control is returned to the referencing program unit at the first executable statement following the CALL statement. All of the standard library subroutines may be discarded, and the user may supply his own subroutine with any of the library subroutine names: OVERFL, ERROR, etc. Such routines must be included using an INCLUDE control card at the time the program is linked.

The standard library subroutines are described in the following paragraphs and summarized in Table 5-5.

■ Arithmetic Overflow and Underflow Test (OVERFL)

The overflow check subroutine informs the programmer when the result of a computation is not within the maximum or minimum magnitudes allowed for an integer, real, or double precision value. An overflow indicator is set to indicate either an overflow or an underflow condition, or both. This indicator remains set until it is reset. The actions the subroutine performs are:

— Overflow

If a computed result exceeds the maximum magnitude allowed, the overflow indicator is set and the computed result is set to the largest permitted value. Integer overflow is ignored and does not affect the results of the OVERFL subroutine.

— Underflow

If a nonzero result is less than the minimum magnitude allowed, the overflow indicator is set and the result is set to 0.

The state of the overflow indicator is tested and reset by executing the OVERFL subroutine with the following statement:

```
CALL OVERFL (i)
```

where:

i
Represents an integer*4 variable.

The variable is assigned a value of 1 if the test indicates an overflow condition, or if both an overflow and an underflow condition exist. The value is 2 if the test indicates neither, or 3 if only an underflow condition is detected.

The overflow indicator is automatically reset after execution of the OVERFL subroutine. Consequently, repeating the test yields a result of 2. Note that overflow and underflow can be tested separately.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT
STATEMENT NUMBER	Cont.	
5	6	7 10 20 30
		CALL OVERFL (I)
		GO TO (10, 20, 30), I
		...
10		PROCESS OVERFLOW OR UNDERFLOW AND OVERFLOW
		...
30		PROCESS UNDERFLOW ONLY
		...
20		

The overflow subroutine is called and the value is returned in I. If either an overflow condition or both an underflow and an overflow condition are found, I is set to 1 and the GO TO control statement transfers control to statement 10. If neither condition is found, I is 2 and control is transferred to 20. If only an underflow condition is found, I is 3 and statement 30 receives control for underflow processor.

■ Divide Check Subroutine (DVCHK)

The divide check subroutine can be used to determine if a division by 0 has occurred. If it has, the invalid divide indicator is set, and the quotient is set to 0. The state of the indicator is tested and reset by the DVCHK subroutine. The appropriate CALL statement is:

CALL DVCHK (i)

where:

i

Represents an integer*4 variable.

The integer variable is set to 1 if a division by 0 occurs; otherwise, it is 2.

Example:

		CALL DVCHK(I)
		GO TO (10, 20), I
10		STOP 27
20		CONTINUE

If a division by 0 was attempted, i equals 1 and program control is transferred to statement 10; otherwise, control goes to statement 20.

■ Error Indicator Test (ERROR)

This standard library subroutine tests an indicator to determine if a function error condition has occurred. Control is transferred to this subroutine by executing the following statement:

CALL ERROR (i)

where:

i
Represents an integer*4 variable.

This integer*4 variable is set to 1 if a function error condition exists, or 2, if no error exists. A subsequent call of the ERROR subroutine, prior to additional function references, always returns the value 2.

■ Error Indicator Setting Subroutine (ERROR1)

This subroutine is used in conjunction with the ERROR subroutine; ERROR1 sets the function error indicator tested by ERROR. The call for the ERROR1 standard library subroutine is:

CALL ERROR1

Example:

"C" FOR COMMENT		STATEMENT NUMBER	FORTRAN STATEMENT
1	5	6	7 10 20 30 40
			: ;MAIN PROGRAM
			Z=XRAY (Q)
			CALL ERROR (I)
			GO TO (30,40),I
	40		CONTINUE
			: ;ERROR CONDITION PROCESSING
	30		...
			: ;FUNCTION DEFINITION
			FUNCTION XRAY (B)
			IF (B) 10,20,10
	20		CALL ERROR1
			RETURN
	10		CONTINUE
			: RETURN
			END

In the example, the external function XRAY is written with a test for errors incorporated. When an error is detected in the function subprogram, the ERROR1 subroutine is called and the function error indicator is set. In the main program unit, a test is made of that indicator after the evaluation of the XRAY function.

■ Indicator Setting Subroutine (SLITE)

The SLITE standard library routine sets or resets one or more of four indicators internal to the subroutine. This subroutine is used with the SLITET subroutine which tests the indicators. The SLITE subroutine is executed by using the statement:

CALL SLITE (e)

where:

e

Is an integer expression; its value determines the indicator settings made:

e = 0 resets all indicators.

e = 1, 2, 3, or 4 sets the corresponding indicator.

e = -1, -2, -3, or -4 resets the corresponding indicator.

■ Indicator Testing Subroutine (SLITET)

The SLITET subroutine tests the indicators controlled by the SLITE subroutine. The SLITET subroutine is executed by the statement.

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*4 variable name returning the results of the indicator test.

After execution of the SLITET standard library subroutine, if the value of the integer variable is 1, the tested indicator is set. If e is outside the range ($1 \leq e \leq 4$), i is set to 2 and the indicator is not set. Execution of the SLITET subroutine does not affect the indicator settings.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT		
STATEMENT NUMBER	LINE	START	END	END
5	6	7	10	20
		CALL SLITE(3)		
		:		
		:		
		CALL SLITET(3,5)		
		:		
		GO TO (10,20),J		
		:		
20		;SWITCH NOT SET		
		:		
10		;SWITCH IS SET		
		:		

■ Control Information Check (SSWTCH)

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

CALL SSWTCH (e,i)

where:

e

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

i

Is an integer*4 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. If it is not set, the value is 2. Execution of the SSWTCH subroutine does not alter the switch setting.

■ Main Storage Dump (DUMP and PDUMP)

These dump subroutines cause a dump or listing of main storage assigned to the FORTRAN program. The subroutines are described in 9.3.

■ Exit Subroutine (EXIT)

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

■ FETCH Subroutine

The FETCH subroutine loads an overlay phase but cannot be used to transfer control to a FORTRAN subroutine or function. The transfer address of the overlay phase must specify a FORTRAN main program or an assembly program which establishes its own cover. Processing in the calling program unit is not resumed. The CALL statement has the format:

CALL FETCH (s)

where:

s

Is a phase name which must be a double precision variable containing a phase name.

If an error occurs during the attempt to load the overlay, termination and an informational message results.

Example:

"C" FOR COMMENT		STATEMENT NUMBER		FORTRAN STATEMENT	
5	6	7	10	20	30
		:			
		:			
		DOUBLE PRECISION DNAME			
		DATA/DNAME/'PHASNM01' /			
		CALL FETCH (DNAME)			
		:			
		:			

■ LOAD Subroutine

The LOAD standard library subroutine loads subprogram overlays. Control is not transferred to the subprogram but is returned to the statement following the CALL statement requesting the overlay. If an error occurs during the attempt to load the overlay, termination and an informational message results. The format of the CALL statement is:

CALL LOAD (s)

where:

s

Is a phase name which is a double precision variable containing a phase name.

Table 5-5. Standard Library Subroutines

Subroutine	CALL Statement Format	Use
OVERFL	CALL OVERFL (i)	Tests for overflow and underflow.
DVCHK	CALL DVCHK (i)	Tests for invalid division.
ERROR	CALL ERROR (i)	Tests for function error conditions.
ERROR1	CALL ERROR1	Sets the function error indicator.
SLITE	CALL SLITE (e)	Sets the sense indicator(s) specified by e.
SLITET	CALL SLITET (e,i)	Tests specified sense indicator.
SSWTCH	CALL SSWTCH (e,i)	Tests the binary switch specified by e and returns a value in i.
DUMP	CALL DUMP (I)	Dumps main storage assigned to the program and terminates program.
PDUMP	CALL PDUMP (I)	Dumps main storage assigned to the program; program execution continues.
EXIT	CALL EXIT	Terminates the program.
FETCH	CALL FETCH (s)	Loads and transfers control to overlay specified by s.
LOAD	CALL LOAD (s)	Loads subprogram overlays.

6. Specification Statements

6.1. GENERAL

Specification statements in the SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN are nonexecutable statements used to describe program data and main storage allocation to the compiler. See fundamentals of FORTRAN programmer reference, UP-7536 (current version); these nonexecutable statements must be at the beginning of the program (Table 1-2).

6.2. ARRAY DECLARATION

An array is an ordered set of elements identified by a symbolic name and declared by an array declarator. An array may be declared in a DIMENSION statement, a COMMON statement, or in an explicit type statement.

6.2.1. Array Declarator

Format:

$$v (i_1, i_2, i_3)$$

where:

v

Is a symbolic name identifying the array.

i

Is a subscript consisting of an unsigned integer constant (or integer variable for adjustment dimensions). An integer variable used to declare an adjustable dimension must be a dummy argument; from one to three 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.

An array name XRRAY with 100 elements in three dimensions would be defined with the declarator:

$$\text{XRRAY (4,5,5)}$$

and the declarator:

$$\text{MRRAY (INX,INY)}$$

declares an array with adjustable dimensions. INX and INY are integer variables which define the size of the array when they are evaluated.

In the interest of efficiency, dummy arrays are processed at execution time in a special fashion. The procedure prologue (5.5.1) saves the subscripts in dimension declarators from the argument list. The prologue then derives a partial solution to the equation used to locate array elements (Table 2-2). Thereafter, the subscript calculations in the body of the procedure can be performed with relative ease. A side effect of this technique, however, is that it is impossible to redeclare array dimensions within a procedure; in the example

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

statement 10 cannot be made to behave as though the declarator were X(10,5).

6.3. DIMENSION STATEMENT

Format:

DIMENSION $v_1(i_1), v_2(i_2), v_3(i_3), \dots, v_n(i_n)$

where:

$v(i)$

Is an array declarator.

Description:

The DIMENSION specification statement provides a means of declaring any number of arrays. Both array names and dimensions are defined by the statement.

6.4. TYPE STATEMENTS

Two kinds of type statements can be used; the explicit type statements 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 yield integer values (FORTRAN name rule); all others are considered to be real. Note that external function procedure names may also be typed with their definition statements (5.4.1.1).

6.4.1. Explicit Type Statements

Format:

$t * s a_1 * s, a_2 * s, \dots, a_n * s$

where:

t

Is the type specification: INTEGER, REAL, or DOUBLE PRECISION.

a

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

***s**

Is an optional length specification; this may not be specified if the type is DOUBLE PRECISION.

Description:

An explicit type specification statement defines the data type of a symbolic name. The length associated with the type (either implicitly or by the *s option) applies to all names in the list unless specifically overridden by a length specification for the individual name.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT		
STATEMENT NUMBER	Column	7	10	30
5	6	REAL*8 A, B, C		
		INTEGER*2 I, J, K(12)		

In the first explicit type statement, the variables A, B, and C are all typed as real*8 (or double precision) due to the length specification. The second specification statement types I, J, and K as 2-byte integer and additionally declares K as an array of 12 elements.

6.4.2. IMPLICIT Statement

Format:

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

where:

t

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

***s**

Is the optional length specification; this may not be specified if the type is DOUBLE PRECISION.

a

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

Description:

The IMPLICIT specification statement permits the user to specify his own implicit type conventions for each program unit. The IMPLICIT statement types symbolic names by the first letter of their names; \$ is included as the last of the possible characters.

If \$ is to be included in a range specification, it must be last. The dollar sign indicates real data by the standard typing convention.

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

IMPLICIT statements may be preceded in the program unit only by SUBROUTINE, FUNCTION, or BLOCK DATA declarations. The IMPLICIT statements affect the typing of all names in the program, excluding intrinsic and double precision standard library functions (Tables 5-3 and 5-4). See also Tables 1-2 and 2-1.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Cont.	7	10	20	30
5	6	IMPLICIT REAL*8(A-D,F),			
		*INTEGER*2(N,Q,U,V), INTEGER(X-\$)			

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

- A through D are real*8 because of the IMPLICIT specification statement.
- E is real*4 implicitly and is unaffected by the IMPLICIT type statement.
- F is real*8 because of the IMPLICIT statement.
- G and H are real*4, and I through M are integer*4 implicitly.
- N is integer*2 because of the specifications in the IMPLICIT statement.
- O and P are real*4 implicitly.
- Q is integer*2 because of the IMPLICIT statement.
- R through T are real*4 because of the standard implicit conventions.
- U and V are integer*2 because of the IMPLICIT statement.

- W is real*4 implicitly.
- X through Z and \$ are integer*4 because of the IMPLICIT statement.

6.5. EQUIVALENCE STATEMENT

Format:

EQUIVALENCE (k₁), (k₂), ..., (k_n)

where:

k

Is a list of the form a₁, a₂, ..., a_m and each a is a variable name, an array element name, or an array name. Each name specified in the list shares assigned storage.

Description:

The EQUIVALENCE specification 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. Array declarators (in COMMON, DIMENSION, and type statements) must appear before the array element name is used in an EQUIVALENCE statement.

Program execution time is increased whenever a variable which does not have a proper boundary alignment is referenced. To achieve proper alignment, a variable must have an assigned main storage address which is an integral multiple of its length.

The first variable in each non-common EQUIVALENCE group is assigned a main storage address that is a multiple of 8. If erroneous boundaries are present in the equivalence group, the addresses in the group are increased successively by 2, 4, and 6 by the compiler in an attempt to correct the error.

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

6.6. COMMON STATEMENT

Format:

COMMON /x₁/a₁/.../x_n/a_n

where:

x

Is an optional symbolic name identifying the COMMON block.

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 which 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. Boundary error recovery is provided in the same manner as for the EQUIVALANCE statement but address adjustments are not attempted by the compiler.

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 lengths: double precision first, and then real and integer*4, and finally integer*2.

6.7. EXTERNAL STATEMENT

Format:

EXTERNAL v_1, v_2, \dots, v_n

where:

v
Is the name of an external function or subroutine.

Description:

When an external function or subroutine name is used as an actual argument to another procedure, the EXTERNAL specification statement must be used to identify these procedures.

If an intrinsic or library 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 by the compiler.

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

"C" FOR COMMENT		FORTRAN STATEMENT	
STATEMENT NUMBER	Column	7	30
1	5	6	20
10		CALL A	
20		CALL B(A)	

In the preceding example, no EXTERNAL statement is needed, but if statements 10 and 20 were reversed in sequence in the source program, the following statement would be needed:

"C" FOR COMMENT

STATEMENT NUMBER	Cont.	FORTRAN STATEMENT
5	6	7 10 20 30
		EXTERNAL A



7. Input and Output

7.1. GENERAL

This section describes the characteristics of the input/output (I/O) system and the SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN statements required for input and output control; see also fundamentals of FORTRAN programmer reference, UP-7536 (current version).

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 peripheral devices are assigned unit numbers within the user's system. The input and output devices which may be used are card readers and punches, printers, magnetic tape units and disc subsystems.

7.2. INPUT/OUTPUT LIST

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

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

where:

a

Is one of the following:

1. a simple I/O list which may be a variable, array element, or array name;
2. two simple I/O lists separated by a comma;
3. a simple I/O list in parentheses; or
4. a DO-implied list.

7.2.1. DO-Implied Lists

Format:

$$(k,d)$$

where:

k

Is an I/O list.

d

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

Description:

The DO-implied list allows the transfer of list elements in the sequence specified by the DO parameters.

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 the array AX are 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 used to determine the location of array elements.

7.3. SEQUENTIAL FILES

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

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

7.3.1. Unformatted I/O Statements

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

WRITE (u) k

READ (u [{ EOF } =I₁] [,ERR=I₂]) k

where:

u

Is an integer*4 constant or variable designating an I/O device.

EOF=I₁

Is an optional specification with I₁ denoting the label of the statement to receive control if an end-of-file condition occurs.

END=I₁

Is a specification which may be substituted for EOF=I₁.

ERR=I₂

Is an optional specification with I₂ denoting the statement to receive control if an error condition occurs.

k

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

Description:

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

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

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

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

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

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	Column	7	10	20	30	40
	5	6				
			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),			
			I(C, DUMMY(51))			
			WRITE (9) DUMMY			

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

7.3.2. Formatted READ/WRITE Statements

Formats:

$$\text{READ } (u, a \left[\begin{array}{l} \{ \text{EOF} \} \\ \{ \text{END} \} \end{array} = I_1 \right] \left[, \text{ERR} = I_2 \right]) k$$

where:

u

Is an integer*4 constant or variable designating an input or output device.

a

Is a label of a FORMAT statement or the asterisk (*) character (7.3.6).

EOF=I₁

Is an optional specification with I₁ denoting the label of the statement to receive control if an end-of-file condition occurs.

END=I₁

Is a specification which may be substituted for EOF=I₁.

ERR=I₂

Is an optional specification with I₂ denoting the statement to receive control if an error condition occurs.

k

Is an optional I/O list.

Description:

The formatted READ and WRITE statements initiate and control the transfer of formatted data between a designated peripheral device and main storage. Data is always converted from/to character strings on external media and the internal representations specified in 2.2 and 2.3. The presence of EOF=I or END=I indicates that if an end-of-file condition is encountered on input, the program is to branch to the label specified. EOF and END are interchangeable.

7.3.3. I/O Compatibility Statements

The following FORTRAN II statements are accepted by the compiler:

READ a,k

PUNCH a,k

PRINT a,k

where:

a

Is the statement label of a FORMAT statement or the asterisk (*) character (7.3.6).

k

Is an I/O list.

No unit specification is made with these input/output statements because none is necessary; the compiler addresses the appropriate device in the user's system configuration.

7.3.4. FORMAT Statement

Format:

a FORMAT (q₁t₁z₁t₂z₂...t_{n-1}t_nq₂)

where:

a

Is the statement 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 end of a logical record.

t

Is a field descriptor (7.3.4.1) or a group of field descriptors specifying the data conversion or the action to be performed.

z

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

Description:

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

Examples:

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

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, then describes a record with a 12-byte integer field, two blanks and another 12-byte integer field plus another blank record.

7.3.4.1. Field Descriptors

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

Table 7-1. *FORMAT Statement Field Descriptors*

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

LEGEND:

r = a repeat count, ≤ 32767

w = the field width, ≤ 255

s = the scale factor nP ($-128 \leq n \leq 127$)

d = decimal positions

c = character

p = character position in the external record ($0 < p < 255$)

The specifications within the field descriptors are explained in the following paragraphs and the input and output actions prescribed by the descriptors are described in 7.3.4.1.1 through 7.3.4.1.11.

■ Repeat Count

The repeat count allows a field descriptor to be repeated a maximum of 32,767 times and must be an unsigned integer constant. The field descriptor 5I3 is the same as I3,I3,I3,I3,I3.

■ Field Width

The field width specification is an unsigned integer constant indicating the number of character positions the data occupied (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 fundamentals of FORTRAN programmer reference, UP-7536 (current version).

■ Decimal Positions

The specification describes the number of digits to the right of the decimal point; if none exist, a 0 must be specified.

■ Character

Any character of the SPERRY UNIVAC 90/30 System character set is permissible.

■ Character Position

See 7.3.4.1.11.

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(X,15,F20.0) is X, 15, 15, F20.0, X, 15, 15, F20.0.

7.3.4.1.1. Integer Descriptor (rlw)

On input operations, if the value exceeds the range, only the least significant digits are stored (with the sign, if any). An integer, which consists of a signed integer constant where the positive sign is optional, may be preceded by or have embedded 0's or blanks. Blanks are interpreted as 0's.

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.

7.3.4.1.2. Real Descriptor – E Conversion (srEw.d)

On input, the external field consists of a string of digits optionally preceded by blanks or 0's preceded by an optional sign. Blanks are interpreted as 0's. The digit string may specify a decimal point which overrides the d specification in the descriptor. The digit string may be followed by exponent notation E or D followed by an optionally signed integer constant. If the integer constant is signed, the E or D may be omitted. If the number of significant digits exceeds the precision of the list element, the value will be rounded to the correct size. If the value exceeds the range, the maximum value will be substituted. If the value is too small for the range, a 0 will be substituted.

On output, the external field has the following format:

$$s_1 0.h_1 h_2 \dots h_d E s_2 e e$$

where:

s_1
Is the sign of the value, either blank or —.

h
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 seven or 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 0 character preceding the decimal point is deleted from the output form.
- If (w — d) is 5 and the value is positive, both the s_1 and the 0 characters preceding the decimal point are deleted from the output form.
- If neither of these conditions holds, the entire field is set to asterisks.

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

For input action, refer to the E conversion description (7.3.4.1.2). On output, the external field has the following format:

$$s_i i_1 i_2 \dots i_{w-d-1} . f_1 f_2 \dots f_d$$

where:

s Is the sign of the value, either blank or —.

i Is a digit within the integer portion of the output value.

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

Sufficient space must be provided for a minus sign if the value is negative. If the integer part of the value is positive (or 0), requiring more than (w — d — 1) character positions, or is negative, requiring more than (w — d — 2) character positions for its representation, then the E conversion is used instead of F conversion. Where neither F nor E conversions suffice to represent the value, the entire field is set to asterisks.

7.3.4.1.4. Double Precision Descriptor (srDw.d)

For input action, refer to the E conversion description. On output, the external field has the following format:

$$s_1 0 . h_1 h_2 \dots h_d D s_2 e e$$

Refer to E conversion output (7.3.4.1.2).

7.3.4.1.5. General Descriptor (srGw.d)

This descriptor provides the capabilities of the I, D, E, and F 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 and double precision data fields. For double precision data, the G descriptor is, in effect, the same as a D descriptor. For integer data, the G descriptor is interpreted as an I descriptor. The d editing information in the format may be omitted when transferring integer data; it is ignored when present.

7.3.4.1.6. Hollerith Descriptor – A Conversion (rAw)

This descriptor requires a corresponding variable name in the I/O list. The maximum number of characters that can be transmitted to a variable is equal to the length, in bytes, of the variable. A data field is transferred between storage and an external device according to the following rules:

- On input, if the descriptor specifies fewer than the maximum number of characters, the data field is transferred to main storage and left-justified; blanks are inserted in the remaining storage positions. If the descriptor specifies more than the maximum number of characters, only the rightmost characters of the data field are transferred to main storage. The remaining characters are skipped.
- On output, if the descriptor specifies fewer characters than can be represented in the variable type, the leftmost characters of the data field are transferred from main storage. If the descriptor specifies more characters than can be represented in the variable type, the data field, right-justified and preceded by blanks, is transferred from main storage to the external field.

7.3.4.1.7. Hollerith Descriptor – H Conversion (wHc₁c₂...c_w)

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

7.3.4.1.8. 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	Hexadecimal Digits
integer*2	4
integer*4	8
real*4	8
double precision	16

On input, the hexadecimal digits are stored two to a byte, right-justified and zero filled.

On output, the hexadecimal value is stored in the output field with preceding blanks.

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

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

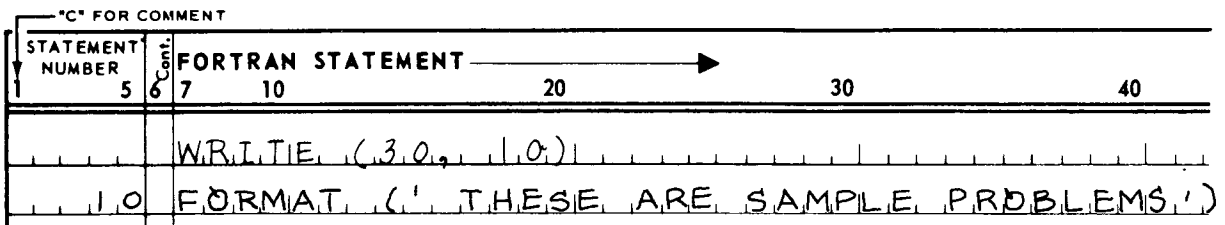
■ Input

The characters in the external field replace the literal data in the FORMAT specification in main storage. All characters are read exactly as they appear between the outermost apostrophes. All inner apostrophes are included in the count. For example, the FORMAT descriptor 'DON''T' causes the next six characters to be input. Each apostrophe in the input data field is treated as a separate character.

■ Output

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

For example, execution of the WRITE statement causes the following line to be printed: THESE ARE SAMPLE PROBLEMS.



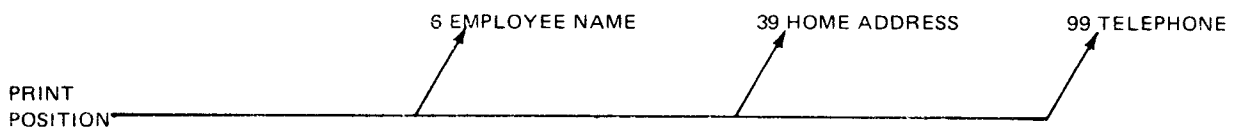
7.3.4.1.10. Blank Descriptor (wX)

This descriptor omits the next w consecutive characters on input or output. Output records are blank filled at the time they are started.

7.3.4.1.11. 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.

For example, the format specification (T7, 13HEMPLOYEEΔNAME, T100, 9HTELEPHONE, T40, 12HHOMEΔ 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 ten characters starting from position 20 of the record to be converted according to the F10.3 code and stored in Y, and the five characters starting from position 1 to be converted according to the F5.1 specification and stored in B.

"C" FOR COMMENT		FORTRAN STATEMENT			
STATEMENT NUMBER	Column	7	10	20	30
1	5				
	6				
		READ (3,2) Y, B			
2		FORMAT (T20, F10.3, T1, F5.1)			

7.3.4.2. Multiple Record Format Specification

The slash is both 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 remains 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 statement remain at the time it is scanned.

7.3.4.3. 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 character is described in Table 7-2.

Table 7-2. Carriage Control Conventions

Character	Meaning
0	2-line advance
+	No advance
1	Skip to top of next page
any other character, including blank	1-line advance

NOTE:

All actions take place before printing.

7.3.4.4. Format Interaction With the I/O List

During the execution of an I/O statement, the FORMAT specification is scanned from left to right. Editing codes of the form wH, 'h₁...h_n', wX, and Tp as well as slashes are interpreted and acted upon without reference to the I/O list. When any other editing code is encountered, one of two possible action is taken: if a list element remains to be transmitted, it is converted and transmitted, and the FORMAT scan continues; or 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: if level 2 parenthetical groupings exist, the FORMAT scan is resumed at the repeat count preceding the rightmost level 2 grouping; or the scan is resumed at the beginning of the FORMAT.

7.3.5. Reread

Format

```
READ (u,a) k
```

where:

u

Is a constant, or an integer*4 variable, designating the reread unit.

a

Is the statement label of a FORMAT statement.

k

Is an I/O list.

Description:

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

The FORTRAN library contains a unit table which 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 so designated.

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

```
READ (29,label) list
```

in conjunction with the desired FORMAT statement.

If an EOF label is specified and the previous read encountered an end of file, control is returned to the specified label. The reread may not, logically, follow a WRITE, BACKSPACE, REWIND, or ENDFILE statement. An unformatted record may not be reread.

7.3.6. List-Directed I/O

List-directed I/O statements are identical in concept to formatted READ and WRITE statements except for the lack of a specific FORMAT statement reference.

They are distinguished from other statements by the presence of the asterisk (*) character in place of the format reference, such 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 type 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 the list-directed formats, specifically in regard to the use of the comma, slash, and blank characters. List-directed output records are acceptable as list-directed input.

7.3.6.1. List-Directed Data Format

- Input

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

- blank (or a series of blanks)
- comma (preceded and followed by zero or more blanks)
- end-of-record
- slash (preceded by zero or more blanks)

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

```
r*constant
```

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

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

Example:

"C" FOR COMMENT		FORTRAN STATEMENT	
STATEMENT NUMBER	POSITION	START	END
5	6	7	10
		20	30
		INTEGER E, F, G	
		READ(4,*) A, B, C, D, E, F, G, H, I	

12 14 /

17.23961727, 12, 2*, 'HE'S'

After the READ 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

The output records will consist of a list of INTEGER and REAL constants, each separated by a comma. Output records will never contain repeat items (r*constant) or literals. The maximum precision commensurate with the list item will be represented. The format codes assumed are:

- G20.11 for REAL*8
- G16.7 for REAL*4
- I16 for INTEGER

7.3.7. Auxiliary I/O Statements

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

7.3.7.1. REWIND Statement

Format:

REWIND u

where:

u

Represents an integer*4 constant or variable designating a sequential file on tape or disc.

Description:

The REWIND statement positions the file at a point immediately preceding the first record.

7.3.7.2. BACKSPACE Statement

Format:

BACKSPACE u

where:

u

Is an integer*4 constant or variable designating a sequential file on tape.

Description:

The BACKSPACE statement activates the designated unit and causes a backspace of one 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. This statement should be used carefully when the file was written with list-directed I/O.

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.

NOTE:

Restrictions for the BACKSPACE statement are shown in 10.3.

7.3.7.3. ENDFILE Statement

Format:

ENDFILE u

where:

u

Is an integer*4 constant or variable designating an output device. The unit must specify card, tape, or sequential disk output files.

Description:

The ENDFILE statement terminates the file associated with the specified unit.

The ENDFILE statement writes file trailer labels based upon the last data transfer. If the last data transfer was a READ, no labels are written. If the last data transfer was a WRITE, file trailer labels are written and the tape is repositioned to a point between the last data block and the tapemark. BACKSPACE is not considered a data transfer — the processing mode is unaffected. The REWIND command sets the processing mode to WRITE following its repositioning operation.

7.3.8. Sequential File Considerations

The I/O statements may not be executed in arbitrary sequences. Table 7-3 shows instances where specific commands are prohibited or ignored.

Table 7-3. Ignored and Prohibited Operations Versus Specific Commands

Previous Operation	Current Operation				
	READ	WRITE	ENDFILE	BACKSPACE	REWIND
READ			I		
READ after EOF encountered	P	P	P		
WRITE	P				
ENDFILE	P	P	P		
BACKSPACE					I
REWIND				I (warning)	I
None				I (warning)	I

LEGEND:

I – Indicates an ignored operation.

P – Indicates a prohibited operation.

Further, not all operations are permitted on all devices. Table 7-4 shows prohibited combinations.

Table 7-4. Ignored and Prohibited Operations Versus File Type

File Type	Operation				
	READ	WRITE	ENDFILE	BACKSPACE	REWIND
Tape	*	*			
Disk	*	*		P	
Card reader		P	P	P	P
Card punch	P			P	P
Printer	P			P	P
Reread		P	P	P	P

LEGEND:

* – These operations may be prohibited when the files are defined as input only or output only. See 11.2 for further details.

P – Indicates a prohibited operation.

Finally, unformatted operations are prohibited with units designating the card reader, printer, card punch, or reread. Formatted and unformatted records can be intermixed freely on output tape and disk files, but it is a user responsibility to read these records in the same mode as they were written.

7.4. DIRECT ACCESS FILES

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

Files can be maintained on the disk storage as sequential files in the same manner as for magnetic tape units without using the direct accessing capability of the disk storage. For such files, the only statements required are the sequential I/O statements described in 7.3. For direct access (random) files, the DEFINE FILE statement identifies and describes its characteristics. A FIND statement causes positioning of an accessing head and, if executed earlier in the program prior to a READ statement, eliminates much of the delay caused by the mechanical movement of the accessing head, since the positioning operation is concurrent with program operation. The direct access I/O statements may transmit either formatted or unformatted data.

7.4.1. DEFINE FILE Statement

Format:

```
DEFINE FILE  $u_1(r_1, m_1, x_1, v_1), u_2(r_2, m_2, x_2, v_2), \dots, u_n(r_n, m_n, x_n, v_n)$ 
```

where:

- u** Is a file identifier, an integer constant specifying a file or unit reference number.
- r** Is an integer constant ≤ 65535 specifying the number of records in the file. ←
- m** Is an integer constant specifying the maximum size of the record in the file in terms of characters (bytes), main storage locations (bytes), or main storage units, depending on the designation chosen for x.
- x** Is one of three possible code letters to indicate the interpretation of m.
- v** Is an unsubscripted integer*4 variable, known as the associated variable of the file. After execution of a READ or WRITE statement, the variable is assigned a value in the range $1 \leq v \leq r$ 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.

Description:

A DEFINE FILE statement is executable and dynamically describes one or more files which 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 system. Thereafter, further definitions of previously defined units are ignored.

One of the following three letters must be placed in the position designated by x:

L

Transfer data as either formatted or unformatted; record size (m) designates bytes.

E

Transfer formatted data; record size (m) designates characters (bytes).

U

Transfer unformatted data; record size (m) designates main storage units (4-byte words). To calculate the record size in storage units, determine the total number of bytes required for all the items of the I/O list and divide this total by 4. If the quotient contains a remainder, round off the result to the next highest integer.

Example:

"C" FOR COMMENT			
STATEMENT NUMBER	5	6	7
FORTRAN STATEMENT			
		10	20
			30
		DEFINE FILE 3(100,120,L,FILE3)	
		15(98,20,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 is transferred; if not, unformatted data is transferred. File 5 contains 98 records of 80 bytes each.

7.4.2. Disk READ Statement

Format:

READ (u' record position,fmt,END=I) k

where:

u

Is a file identifier represented by an integer*4 constant or variable followed by an apostrophe.

record position

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

fmt

Is an optional specification indicating the label of a **FORMAT** statement, or it may be the asterisk (*) character.

I

Is the label of a statement to which control is to be transferred when the record position is outside the file boundaries. EOF and ERR are considered equivalent to END; the entire specification is optional.

k

Is an I/O list.

Example:

"C" FOR COMMENT		FORTRAN STATEMENT
STATEMENT NUMBER		
5	6	7 10 20 30 40 50
		DEFINE FILE 3(100,512,1,FILE3)
		FILE3=10
		READ(3,FILE3,87,END=110),A,B,C(11),I=1,30)
6	7	FORMAT(32F16.4)

The tenth record in file 3 is transferred to main storage when the READ statement is first executed. Each subsequent execution of the READ statement order transfers the next record in the file to main storage. 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 (16 x 32) 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 were:

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

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

7.4.3. Disk WRITE Statement

Format:

```
WRITE (f' r,fmt) k
```

where:

f

Is a file identifier represented by an integer*4 constant or variable followed by an apostrophe.

r

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

fmt

Is an optional FORMAT statement label or it may be the asterisk (*) character.

k

Is an I/O list.

Example:

"C" FOR COMMENT

STATEMENT NUMBER	Col.	FORTRAN STATEMENT
1	5	7 10 20 30
		DEFINE FILE 4(1150,35,L,FILE4)
		⋮
		DOUBLE PRECISION D
		⋮
		FILE4 = 2
		⋮
		WRITE (4' FILE4+1,2) I,R,D
		⋮
2		FORMAT (I8, F12.2, D15.5)

Thirty-five bytes (8 + 12 + 15) are transferred from storage to the third record in the file. The format specification indicates the number of bytes for the integer, real, and double precision information transferred. If the WRITE statement does not specify a format label, an unformatted WRITE is executed. In this case, 16 bytes are transferred.

Variable Name	Type	Number of Bytes
I	Integer	4
R	Real	4
D	Double Precision	8
		16 total

7.4.4. Disk FIND Statement

Format:

FIND(f' r)

where:

f

Is a file identifier represented by an integer*4 constant or variable and followed by an apostrophe.

r

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 track address specified by a file identifier and record number. 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.

Example:

"C" FOR COMMENT

STATEMENT NUMBER	FORTTRAN STATEMENT
5	
6	
7	10 → 20 → 30
	FIND (4' 20)
	⋮
	READ (4' 20) (A(I), I=1, 37, 3)

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 as it applies to SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN is described in this section; for more general information, refer to fundamentals of FORTRAN programmer reference, UP-7536 (current version).

8.2. DATA STATEMENT

Format:

```
DATA k1/h1/,k2/h2/,...,kn/hn /
```

where:

k

Is a list of variable names, array names, and array element names (with constant subscripts).

h

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, or specified array elements. None of these items should be in blank COMMON; they should be in labeled COMMON only if the DATA statement appears in a BLOCK DATA subprogram. A DATA statement must appear after any declarative (e.g., COMMON, DIMENSION, or type) affecting the variables to be initialized.

There must be a basic correspondence of type and a one-to-one correspondence of items between the variable and constant lists. The following relaxations to these rules are permitted:

- Real*4 and real*8 constants and variables may be intermixed freely. If a real*8 constant is associated with a real*4 variable, the least significant digits are truncated; if a real*4 constant is associated with a real*8 variable, it is padded with 0's in the least significant digits.
- Hollerith, literal, and hexadecimal constants may be associated with any variable type.
- If an array name appears as the last item in the variable list, the constant list is not required to completely fill the array. An array element name specifies only one constant value.

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 may contain only DATA, EQUIVALENCE, COMMON, DIMENSION, type, and IMPLICIT statements. The block data subprogram is headed by the BLOCK DATA statement. The order of statements is governed by the rules explained in Table 1-2.

8.3.1. BLOCK DATA Statement

Format:

BLOCK DATA [s]

where:

s

Is the name assigned to the block data subprogram.

Description:

The BLOCK DATA statement is optionally used to name a block data subprogram. When used, it must be the first statement present in the block data subprogram. When more than one block data subprogram is being compiled in a single job, each block data subprogram should be assigned a unique name so that they are all accessible to the linkage editor and librarian. Any block data subprogram compiled without the specification is assigned the name \$BLOCK by default. Therefore, only the last block data subprogram compiled is accessible to the linkage editor and librarian when multiple block data subprograms are compiled in a single job without using unique names.

9. Debugging

9.1. GENERAL

The SPERRY UNIVAC Operating System/3 (OS/3) FORTRAN compiler and library provide localized and specific diagnostics for syntax, argument, and I/O errors. In addition, debugging aids may be inserted into the source program to obtain label trace, subscript checking, conditional compilation, and storage dumps.

9.2. LABEL TRACE

The compiler enables the user to trace program flow by displaying the labels of executable statements as they are encountered during program execution when trace region parameters, the TRACE ON statement, and the TRACE OFF statement are used.

9.2.1. Trace Region Parameters

Format:

```
// PARAM TRACE = ALL
```

where:

ALL

Specifies the entire program unit as a trace region.

When TRACE = ALL is specified, all labels in all subprograms to be compiled are traced. The TRACE ON statement (9.2.2) must have been executed before trace output can be initiated.



Example:

"C" FOR COMMENT		FORTRAN STATEMENT				
STATEMENT NUMBER	6	7	10	20	30	40
11		PARAM TRACE=ALL				
		:				
		:				
5		IF (A) 10,20,30				
9999		A=B				
17		CALL Z(A)				

In the example, labels 5, 9999, and 17 will appear in the TRACE listing.

All the labels of executed FORTRAN statements beginning with the statement labeled 5 and ending with the statement labeled 17 are displayed if a TRACE ON statement is encountered in the program.

9.2.2. TRACE ON Statement

Format:

TRACE ON

Description:

The TRACE ON is an executable statement which enables the display of trace regions subsequently encountered during program execution. This statement must be present if trace information is to be displayed. It is ignored if no TRACE parameter is present.

Examples:

10		TRACE ON				
		IF (I.GT.17.AND.I.LT.47) TRACE ON				

9.2.3. TRACE OFF Statement

Format:

TRACE OFF

Description:

The TRACE OFF statement disables the display of label trace information until such time as another TRACE ON statement is executed. This permits the user to control the amount of trace information generated. The statement is treated as a CONTINUE if no TRACE parameter is present.

Examples:

"C" FOR COMMENT		FORTRAN STATEMENT
STATEMENT NUMBER	Cont.	
5	6	7 10 20 30
		TRACE OFF
		I.F. (A.GT.27.45) TRACE OFF

9.2.4. Trace Display

When a TRACE ON statement is executed, the line

```
program-unit-name TRACE LINE nnnnn ENABLED
```

is displayed, indicating that the TRACE ON statement at line nnnnn of the source program was executed. In a similar fashion,

```
program-unit-name TRACE LINE nnnnn DISABLED
```

is generated for a TRACE OFF statement. When a label in a trace region is encountered, the message

```
program-unit-name TRACE LABEL nnnnn
```

is generated immediately prior to the execution of the statement. The program-unit-name is \$MAIN for a main program, or is the FORTRAN subroutine or function name; it is shown only for the first messages generated for a given procedure.

9.3. FORMATTED MAIN STORAGE DUMPS

Format:

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

where:

- n**
Is either DUMP or PDUMP.
- u**
Is a variable or array element name which indicates the upper address boundary for the display.
- l**
Is a variable or array element name which indicates the lower address boundary for the display.
- f**
Is an integer constant 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 codes used for the format specification are:

f	Display Interpretation
0	hexadecimal
3	integer*2
4	integer*4
5	real*4
6	real*8
9	literal

The output of these subroutines is directed to the printer. The TRACE listing is double spaced, and a 120-character print line is required. For values other than those listed, f is treated as hexadecimal. The DUMP subroutine terminates the program after it is executed; PDUMP returns control to the first executable statement following the CALL.

An argument list must be present for DUMP or PDUMP.

9.4. CONDITIONAL COMPILATION

A statement may be optionally selected for compilation by inserting an X in position 1 of the statement line (see example). If the statement extends beyond one line the character X must also appear in position 1 of all the continuation lines. Statements with X in column 1 but without the parameter option activated are treated as comment lines.

Example:

"C" FOR COMMENT	
STATEMENT NUMBER	FORTRAN STATEMENT
1	
5	
6	
7	
10	
20	
30	
X	PRINT 10, A, B, C
X	10 FORMAT (3F15.6)

This capability is provided for printing intermediate results when debugging a program. When debugging is complete, the statements can remain as they are in the source program to be used again as necessary. (Refer to Section 11 for specifying the X parameter during compilation.)

9.5. SUBSCRIPT CHECKING

The compiler evaluates array element addresses without regard to the values of the variables used as subscripts. Thus, erroneous subscripts could result in storage accesses outside the bounds of an array. The compiler can, however, be forced to generate code which checks the final array element address to ensure the array boundaries are not violated. The parameter used to generate code for subscript checking is:

```
// PARAM SUBCHK = ALL
```

where:

ALL

Specifies that all array element addresses are to be checked.

Since the subscript checking feature is time consuming, it should be used for program debugging and not in production programs.



10. I/O Configuration

10.1. SYSTEM PROVISIONS

Every executable program must contain an input/output control module. A simple module is provided with the FORTRAN system. If necessary you may generate more complex modules.

The object module FP\$IO is provided in the system object library and is automatically included in the executable program by the linkage editor, unless you specifically include another I/O module. The module provided supports the following unit numbers:

- 1 – Card reader (GETCS)
- 3 – 120 character printer (plus a single-carriage control character)
- 5 – Equivalent to unit 1
- 6 – Equivalent to unit 3
- 29 – Reread unit

The LFD name FORT03 is required for the printer.

10.2. CONFIGURING AN I/O CONTROL MODULE

An I/O control module may be configured using an assembler language source module which contains only the following statements:

1	LABEL	△OPERATION△ 10 16	OPERAND △
	NAME	START	
		FUNTAB	
		UNIT	(1)
		UNIT	(2)
		.	
		.	
		.	
		UNIT	(n)
		FUNEND	
		END	

START is always the first directive. The name is the name which will be assigned to the generated object module. This name must be used on the linkage editor INCLUDE statement when specifically including the I/O module. FUNTAB, FUNEND and END are always required and must be in the sequence shown. Each UNIT macro instruction defines a single file corresponding to a FORTRAN unit number.

The following file types may be defined:

- Printer
- Card reader (GETCS)
- Card punch
- Magnetic tape
- Sequential disk file
- Direct access disk file
- Reread unit
- Equivalent file

The following is a sample of the assembler data for an I/O configuration:

1	LABEL	Δ OPERATION Δ	OPERAND	Δ	COMMENT
		10	16		
	1/5				
	MYID	START			
		FUNTAB			
		UNIT	FDEVICE=SPDOLIN, FUNIT=1		
		UNIT	FDEVICE=PRINTER, FUNIT=6, FNUMBUF=2		
		UNIT	FDEVICE=EQUIV1, FUNIT=3, FEQUIV=6		
		UNIT	FDEVICE=DISC, FUNIT=10, FRECSIZE=512		
		UNIT	FDEVICE=REREAD, FUNIT=29		
		FUNEND			
		END			
	1/6				

10.3. UNIT MACRO INSTRUCTION FORMATS AND PARAMETERS

The hardware configuration is the only limitation on the number and types of files that may be defined. However, only one card reader unit may be defined and at least one printer is required for routine diagnostics. Each file definition consists of a UNIT macro instruction and the necessary keyword parameters specifying the file characteristics. Every file required for the execution of your job must be defined in this way. The following paragraphs describe the keyword parameters that may be used with the UNIT macro instruction that defines each type of file.

10.3.1. Card Reader Definition

Only one spooled card input file is permitted for a given application.

Format:

1	10	16
	UNIT	FDEVICE=SPOOLIN,FUNIT=k [,FRECSIZE=k]

FDEVICE Keyword Parameter:

FDEVICE=SPOOLIN

Specifies that this is a spooled card input file.

FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number.

FRECSIZE Keyword Parameter:

FRECSIZE=k

Specifies that the record size is 1-128 bytes.

10.3.2. Printer File Definition

At least one printer file must be defined.

Format:

1	10	16
	UNIT	FDEVICE=PRINTER,FUNIT=k [,FNUMBUF=2] [,FRECSIZE=k]

FDEVICE Keyword Parameter:

FDEVICE=PRINTER

Specifies that this is a printer file.

FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number that must be the same as that specified on the LFD job control statement defining this file. For example, FUNIT=01 requires a // LFD FORT01 job control statement, and FUNIT=02 requires a // LFD FORT02 job control statement.

FNUMBUF Keyword Parameter:

FNUMBUF=2

Optionally allocates two unique buffers to the unit for faster performance. In the absence of this specification, all I/O is performed in demand mode.

FRECSIZE Keyword Parameter:

FRECSIZE=k

Specifies the record size in bytes. The default is 121, which will accommodate a 120-position printer with a carriage control character.

10.3.3. Card Punch Definition

The card punch or the 8413 diskette output file is described by the following UNIT macro instruction.

Format:

1	10	16
	UNIT	FDEVICE=CARDOUT,FUNIT=k [,FNUMBUF=2] [,FCRDERR=RETRY] [,FRECSIZE=k]



FDEVICE Keyword Parameter:

FDEVICE=CARDOUT

Specifies that this is a single card or 8413 diskette output file.

FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number that must be the same as that specified on the LFD job control statement defining this file. For example, FUNIT=01 requires a // LFD FORT01 job control statement, and FUNIT=02 requires a // LFD FORT02 job control statement.

FNUMBUF Keyword Parameter:

FNUMBUF=2

Optionally allocates two unique buffers to the unit for faster performance. In the absence of this specification, all I/O is performed in demand mode.

FCRDERR Keyword Parameter:

FCRDERR=RETRY

Causes the inclusion of optional device error recovery procedures. If omitted or if recovery fails on retry, the program is terminated when a device error occurs.



FRECSIZE Keyword Parameter:

FRECSIZE=k

Specifies that the record size is 1-128 bytes.



10.3.4. Tape File Definition

Each tape file required for the execution of your job must be described by the following UNIT macro instruction. Records must be variable length and unblocked.

Format:

1	10	16
	UNIT	FDEVICE=TAPE, FUNIT=k [,FTYPEFLE= { INPUT OUTPUT }] [,FNUMBUF=2] WORK [,FBKSZ=k] [,FBKNO=YES] [,FERROPT= { IGNORE } SKIP }] [,FFILABL= { STD } NO }] [,FCKPT=YES] [,FOPTION=YES]

FDEVICE Keyword Parameter:

FDEVICE=TAPE

Specifies that this is a tape file.

FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number that must be the same as that specified on the LFD job control statement defining this file. For example, FUNIT=01 requires a // LFD FORT01 job control statement, and FUNIT=02 requires a // LFD FORT02 job control statement.

FTYPEFLE Keyword Parameter:

FTYPEFLE=INPUT

Specifies an input file. If specified, the BACKSPACE command cannot be issued to this unit.

FTYPEFLE=OUTPUT

Specifies an output file. If specified, the BACKSPACE command cannot be issued to this unit.

FTYPEFLE=WORK

Specifies a work file and should be specified if the tape is to be read and written.

FNUMBUF Keyword Parameter:

FNUMBUF=2

Optionally allocates two unique buffers to the unit for faster performance. In the absence of this specification, all I/O is performed in demand mode. If specified, the BACKSPACE command cannot be issued to this unit.



FBKSZ Keyword Parameter:

FBKSZ=k

Specifies the block size for variable-length unblocked records. The formula for a tape file is:

18.LE.k.LE.32767

The default value is 263.

FBKNO Keyword Parameter:

FBKNO=YES

Causes optional tape block numbers to be written on output and checked on input.

FERROPT Keyword Parameter:

FERROPT=IGNORE

Specifies that, when parity errors occur or improper lengths are detected on an input data block, the block is to be processed as valid.

FERROPT=SKIP

Specifies that an erroneous record is to be bypassed and the next block is to be read.

If omitted, the system terminates the program when parity errors occur.

FFILABL Keyword Parameter:

FFILABL=NO

Specifies that the tape file is unlabeled.

FFILABL=STD

Specifies that the tape file contains system standard labels.

If omitted, the default value is NO.

FCKPT Keyword Parameter:

FCKPT=YES

Specifies that the input tape file contains OS/3 checkpoint dumps.

FOPTION Keyword Parameter:

FOPTION=YES

Indicates that the file is not a logical requirement for every execution of the program. If the file is not defined with an LFD job control statement, the first READ reference to the file will return an end-of-file condition.



10.3.5. Sequential Disk File Definition

Each sequential disk file required for the execution of your job must be described by the following UNIT macro instruction. Records must be variable length and unblocked.

Format:

1	10	16
	UNIT	FDEVICE=SDISC, FUNIT=k [,FTYPEFLE={ INPUT OUTPUT WORK }] [,FNUMBUF=2] [,FBKSZ=k] [,FERROPT={ IGNORE SKIP }] [,FOPTION=YES] [,FVERIFY=YES]

FDEVICE Keyword Parameter:

FDEVICE=SDISC

Specifies a sequential disc file with variable-length unblocked records.

FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number that must be the same as that specified on the LFD job control statement defining this file. For example, FUNIT=01 requires a // LFD FORT01 job control statement, and FUNIT=02 requires a // LFD FORT02 job control statement.

FTYPEFLE Keyword Parameter:

FTYPEFLE=INPUT

Specifies an input file. This will save storage if no writes occur.

FTYPEFLE=OUTPUT

Specifies an output file.

FTYPEFLE=WORK

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

FNUMBUF Keyword Parameter:

FNUMBUF=2

Optionally allocates two unique buffers to the unit for faster performance. In the absence of this specification, all I/O is performed in demand mode.



FBKSZ Keyword Parameter:

FBKSZ=k

Specifies the block size for variable-length unblocked records. The formula for a disk file is:

$$9.LE.k.LE.7294$$

The default value is 256. A formatted record may not exceed k-8 bytes.

FERROPT Keyword Parameter:

FERROPT=IGNORE

Specifies that, when parity errors occur or improper lengths are detected on an input data block, the block is to be processed as valid.

FERROPT=SKIP

Specifies that an erroneous record is to be bypassed and the next block is to be read.

If omitted, the system terminates the program when parity errors occur.

FOPTION Keyword Parameter:

FOPTION=YES

Indicates that the file is not a logical requirement for every execution of the program. If the file is not defined with an LFD job control statement, the first READ reference to the file will return an end-of-file condition.

FVERIFY Keyword Parameter:

FVERIFY=YES

Specifies that the system is to check parity after every disk write. This additional security degrades performance.

10.3.6. Direct Access Disk File Definition

Any direct access disk file required for the execution of your job must be described by the following UNIT macro instruction. Records are fixed length and unblocked.

Format:

1	10	16
UNIT	FDEVICE=DISC, FUNIT=k [,FRECSIZE=k] [,FTYPEFLE= { INPUT } { OUTPUT }] [,FVERIFY=YES]	

FDEVICE Keyword Parameter:

FDEVICE=DISC

Specifies a direct access disk file.



FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number that must be the same as that specified on the LFD job control statement defining this file. For example, FUNIT=01 requires a // LFD FORT01 job control statement, and FUNIT=02 requires a // LFD FORT02 job control statement.

FRECSIZE Keyword Parameter:

FRECSIZE=k

Specifies the record size in bytes.

If omitted, the default value is 256. The actual record defined by the DEFINE FILE statement must be less than or equal to the value specified for FRECSIZE.

FTYPEFLE Keyword Parameter:

FTYPEFLE=INPUT

Specifies an input file. This will save storage if no writes occur.

FTYPEFLE=OUTPUT

Specifies an output file.

FVERIFY Keyword Parameter:

FVERIFY=YES

Specifies that the system is to check parity after every disk write. This additional security degrades performance.

10.3.7. Reread Unit Definition

Any reread unit required for the execution of your job must be described by the following UNIT macro instruction.

Format:

1	10	16
	UNIT	FDEVICE=REREAD, FUNIT=k

FDEVICE Keyword Parameter:

FDEVICE=REREAD

Identifies a reread unit.

FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number that must be the same as that specified on the LFD job control statement defining this file. For example, FUNIT=01 requires a // LFD FORT01 job control statement, and FUNIT=02 requires a // LFD FORT02 job control statement.



↓

10.3.8. Equivalent Unit Definition

The function of an equivalent unit is to provide another reference number of a file. For example, an input file might be referenced with both a Basic FORTRAN statement with a unit number and a statement that implies the special name READ. An equivalent unit can be used to resolve conflicts of this type. The equivalent unit is described by the following UNIT macro instruction.

Format:

1	10	16
	UNIT	FDEVICE=EQUIV, FUNIT=k, FEQUIV=j

FDEVICE Keyword Parameter:

FDEVICE=EQUIV

Specifies that one unit is referenced to another unit.

FUNIT Keyword Parameter:

FUNIT=k

Is a unique 1- or 2-digit decimal integer unit number that must be the same as that specified on the LFD job control statement defining this file. For example, FUNIT=01 requires a // LFD FORT01 job control statement, and FUNIT=02 requires a // LFD FORT02 job control statement.

FEQUIV Keyword Parameter:

FEQUIV=j

Specifies a 1- or 2-digit decimal integer unit number that was previously defined in the I/O configuration on a FUNIT keyword parameter. When the equivalent unit is referenced, device action takes place on this unit.

↑

11. Compilation

11.1. GENERAL

The FORTRAN compiler is named FORT. It requires one work file and 5C00₁₆ bytes of main storage plus space for the prologue. If allocated, additional storage is used to increase the compiler capacity.

The compiler requires the extended micrologic feature. A message:

COMPILER REQUIRES 2K COS

is displayed on the printer and on the operator console if the extended micrologic feature is not present and the job is canceled with an error code of 610.

11.2. COMPILATION DIRECTIVES

Compilation directives are specified on PARAM job control statements. The format of a PARAM statement is:

```
//ΔPARAMΔspec1 , ..., specn
```

The following list of directives shows the directive format and provides a brief explanation of each specification.

■ LST=k

k

Is the sum of the following options:

- 1 – source code listing
- 2 – diagnostic listing
- 4 – storage allocation map
- 8 – object code listing

The default is LST=7.

■ IN=MNAME[/FNAME]

MNAME

Is the name of the source module to be compiled

FNAME

Is the optional filename corresponding to the LFD name for the disc file where the source resides. The default filename is \$Y\$SRC.

■ OUT=(FILENAME)

OUT=NO

NO

Indicates that no object module is to be generated; filename in parenthesis is the LFD name of the file where the generated object module will be placed. The default filename is \$Y\$RUN.

■ TRACE=ALL

See 9.2.1 for a description of this parameter.

■ X

See 9.4 for a description of this parameter.

■ SUBCHK=ALL

See 9.5 for a description of this parameter.

11.3. OBJECT PROGRAM STRUCTURE

The object programs produced by the compiler have a formal structure, as shown in Figure 11-1.

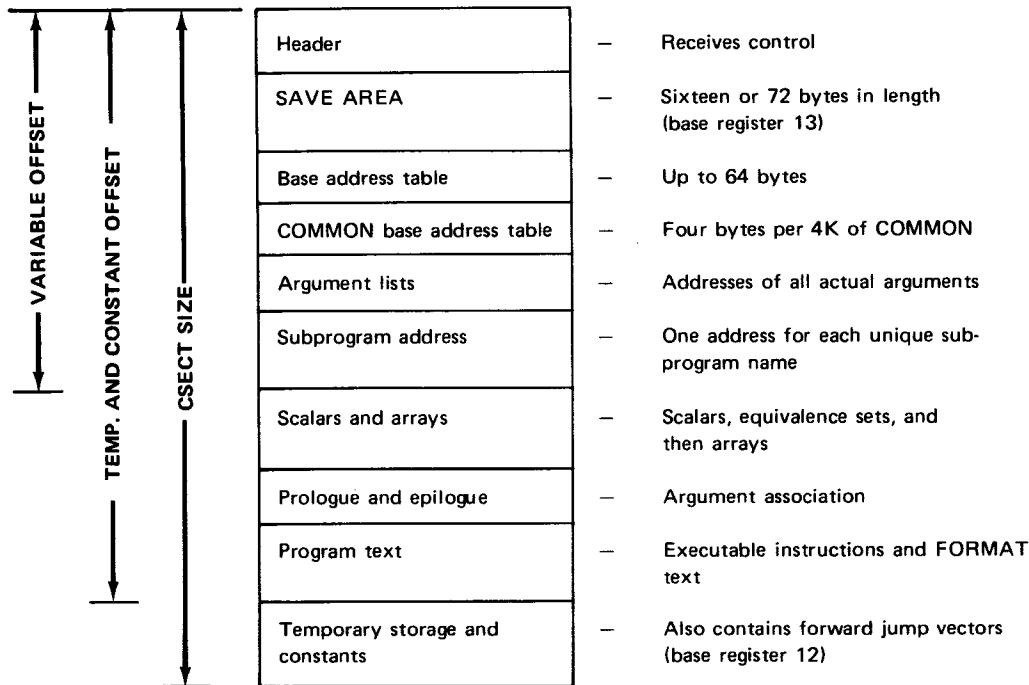


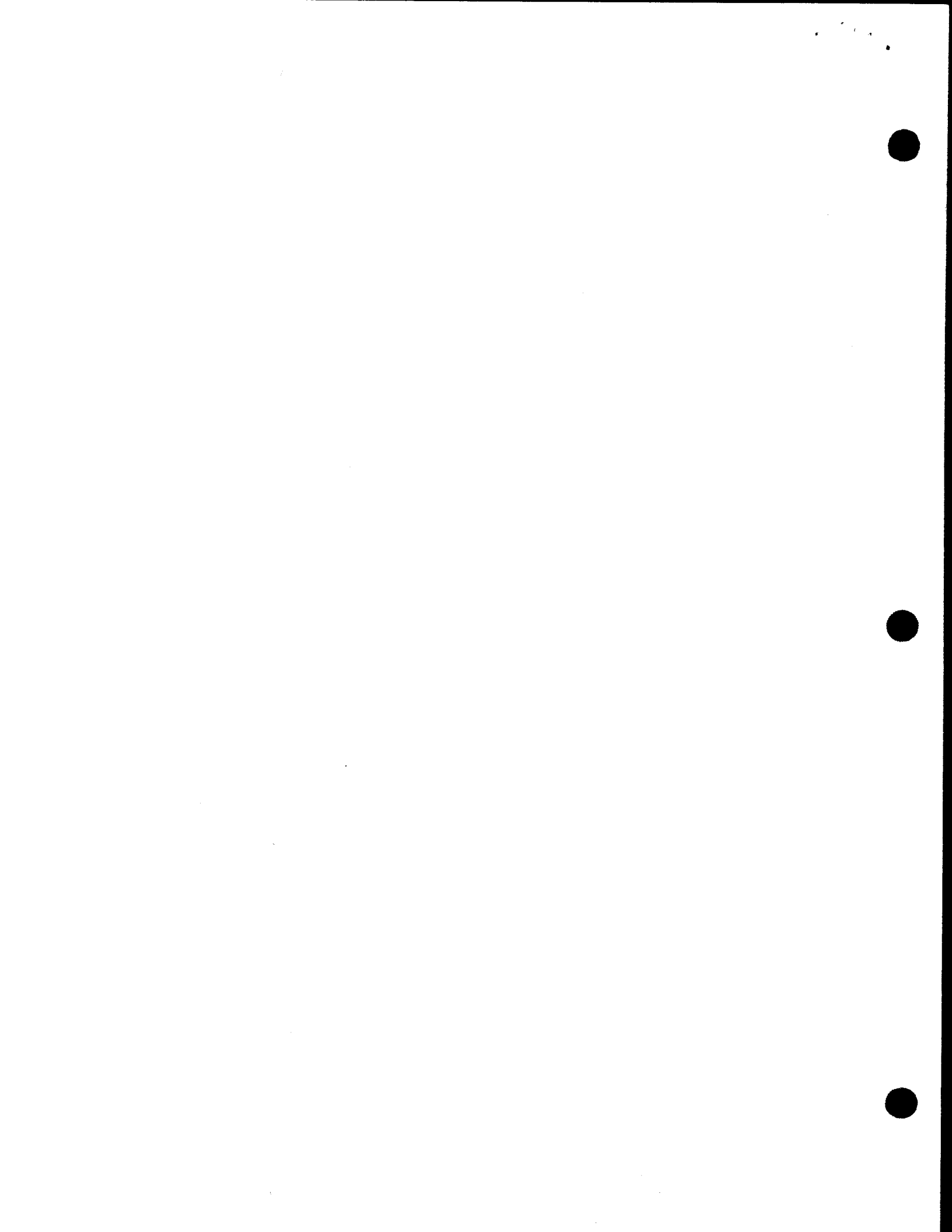
Figure 11-1. FORTRAN Control Section (CSECT) Structure

In addition to the just-cited CSECT, the compiler generates one additional CSECT for each COMMON BLOCK. For BLOCK DATA subprograms, only the COMMON CSECTs are logically produced.

11.4. CAPACITY OF THE COMPILER

The design of the compiler and its addressing environment cause limitations on the size of program units which can be successfully processed. The capacity is sufficiently generous for most large programs. The capacities are:

- COMMON blocks (blank or named) can contain up to 65,532 bytes.
- The FORTRAN CSECT can contain up to 65,532 bytes.



12. Compile, Link, and Execute Procedures

12.1. JOB CONTROL PROCEDURES

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

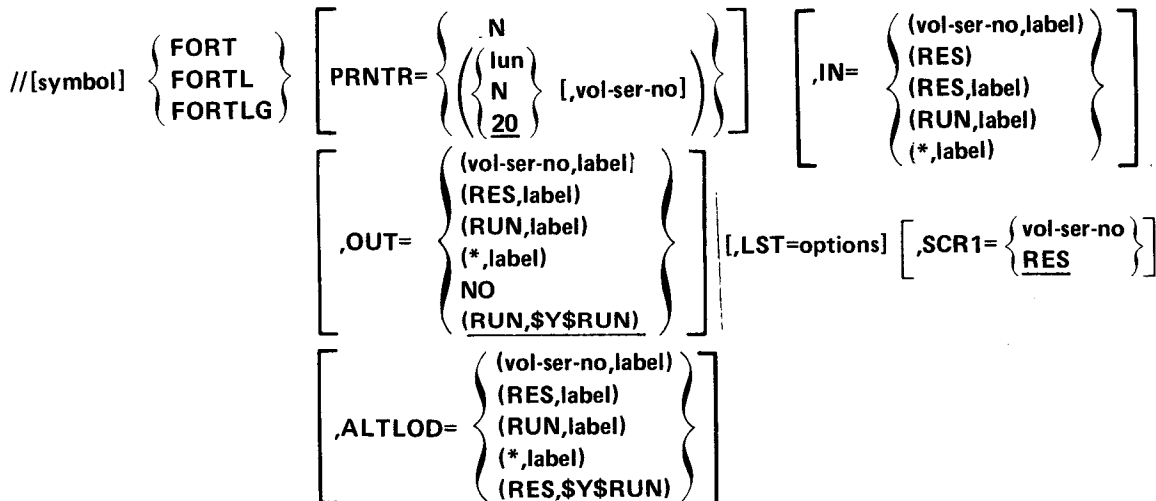
- Input – source library
- Output – object library
- PARAM control statements defining the format of the compiler listing
- Automatically link and/or execute the program

The input may be embedded data cards (/ \$, source deck, /*) immediately after the FORT procedure call, or in a module from the library defined by the IN parameter. Specifying a module and IN parameter option results in the appropriate DVC-LFD control statement sequence with an LFD name, INFPUT, and the PARAM control statement, PARAM IN=module-name/INFPUT.

The object code is written in \$Y\$RUN by default, but a specific output library can be specified by the OUT parameter. This results in the appropriate DVC-LFD control statement sequence with an LFD name, OUTFPUT, and the PARAM control statement, PARAM OUT=OUTFPUT.

The ALTLOD parameter generates the necessary DVC-LFD control statements with an LFD name, ALTLOD, and the appropriate EXEC control statement to load and execute the FORTRAN compiler from a private library other than \$Y\$LOD.

Format:



Label:

symbol

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

Operation:

FORT

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

FORTL

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

FORTLG

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

Keyword Parameter PRNTR:

$$PRNTR = \left\{ \left(\begin{array}{c} N \\ \left(\begin{array}{c} lun \\ N \\ \underline{20} \end{array} \right) \end{array} \right) [,vol-ser-no] \right\}$$

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

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

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

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

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

Keyword Parameter IN:

$$IN = \left\{ \begin{array}{l} (vol-ser-no, label) \\ (RES) \\ (RES, label) \\ (RUN, label) \\ (*, label) \end{array} \right\}$$

Specifies the input file definition, to which the PARAM IN control statement connects. If omitted, the source input is assumed to be embedded data cards (/ \$, source deck, /*).

$$IN = (vol-ser-no, label)$$

Specifies the volume serial number (vol-ser-no) and the file identifier (label) where the source input is located.

$$IN = (RES)$$

Specifies that the source input is located on the SYSRES device in \$Y\$SRC.

IN=(RES,label)

Specifies that the source input is located on the SYSRES device, but the file identifier (label) is user specified, not \$Y\$SRC.

IN=(RUN,label)

Specifies that the source input is located on the job's \$Y\$RUN file with the file identifier (label) specified by the user.

IN=(*,label)

Specifies that the source input is located on a catalog file identified by the file identifier (label).

Keyword Parameter OUT:

$$\text{OUT} = \left\{ \begin{array}{l} (\text{vol-ser-no, label}) \\ (\text{RES, label}) \\ (\text{RUN, label}) \\ (*, \text{label}) \\ \text{NO} \\ (\text{RUN}\$Y\$RUN) \end{array} \right\}$$

Specifies the output file definition to which the PARAM OUT control statement connects. If omitted, the object code is placed into the job's \$Y\$RUN file.

OUT=(vol-ser-no,label)

Specifies the volume serial number (vol-ser-no) and the file identifier (label) where the object code is to be placed.

OUT=(RES,label)

Specifies that the object code is to be placed on the SYSRES device, within the file specified by the label parameter.

OUT=(RUN,label)

Specifies that the object code is to be placed on the job's \$Y\$RUN file identified by a user specified file identifier (label).

OUT=(*,label)

Specifies that the object code is to be placed on a catalog file identified by the file identifier (label).

OUT=NO

Specifies that there is to be no object code outputted.

Keyword Parameter LST:

LST=options

Specifies the format of the compiler listing. The LST options are listed in 11.2.

Keyword Parameter SCR1:

$$\text{SCR1} = \left\{ \begin{array}{l} \text{vol-ser-no} \\ \underline{\text{RES}} \end{array} \right\}$$

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:



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

Specifies the volume serial number (vol-ser-no) and the file identifier (label) of an alternate load library that contains the FORTRAN compiler. If omitted, the compiler is loaded from \$Y\$RUN.

ALTLOD=(RES,label)

Specifies that the alternate load library is located on the job's SYSRES device, in the file identified by the file identifier (label).

ALTLOD=(RUN,label)

Specifies that the alternate load library is located on the job's \$Y\$RUN file with the file identifier (label) specified by the user.

ALTLOD=(* ,label)

Specifies that the alternate load library is located on a catalog file identified by the file identifier (label).



Example 1a:

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

1	LABEL	△OPERATION△	OPERAND	△
		10 16		
1	/ / JOB FRTRN1A			
2	/ / FORT			
3	/ \$			
4	.			
5	SOURCE DECK			
6	.			
7	/ *			

Line	Explanation
1	Indicates that the number of the job is FRTRN1A.
2	Indicates the name of the procedure being called (FORT). No keyword parameters specifying special options for this compilation are used.
3	Indicates start of data.
4-6	Represents the source deck to be compiled.
7	Indicates end of data.

Example 1b:

The basic form generates the following control stream:

1	LABEL	△OPERATION△ 10 16	OPERAND	△
1	// JOB	FRTRN1B		
2	// DVC	20	// LFD	PRNTR
3	// DVC	RES		
4	// EXT	ST, C ₁ , 3, C _Y L, 1		
5	// LBL	\$SRC1	// LFD	\$SRC1
6	// EXEC	FORT		
7	/\$			
8				
9	SOURCE	DECK		
10				
11	/*			

Line Explanation

- 1 Indicates that the name of the job is FRTRN1B.
- 2 Indicates the default logical unit number and LFD name of the printer.
- 3-5 Indicates that the work file needed for compiling is, by default, on the SYSRES device, has both a file label and LFD name of \$SRC1, and uses the sequential access technique; that allocation is contiguous; that three cylinders are allocated for the secondary increment; and that one cylinder is allocated for the first extent.
- 6 Loads the FORTRAN compiler from \$Y\$LOD.
- 7 Indicates start of data.
- 8-10 Represents the source deck to be compiled.
- 11 Indicates end of data.

Example 2a:

The following example illustrates the use of a FORT procedure call statement that defines all the keyword parameters:

1	LABEL	△OPERATION△ 10 16	OPERAND	72
1	// JOB	FRTRN2A		
2	// PROG	NM FORT	PRNTR=21, IN=(DSC1, U\$SCR),	X
3	//		OUT=(DSC2, U\$OBJ), LST=12,	X
4	//		SCR1=DSC2, ALTL0D=(DSC3, ALTLODLIB)	
5	/\$			

- | Line | Explanation |
|------|--|
| 1 | Indicates that the name of the job is FRTRN2A. |
| 2 | Indicates the name of the procedure being called (FORT). The source module name is PROGDM. The logical unit number of the printer is 21, and the input file has a volume serial number of DSC1, with a file label of U\$SRC. |
| 3 | Indicates that the output file volume serial number is DSC2, with a file label of U\$OBJ. The format of the compiler listing is supplied by the LST parameter. |
| 4 | Indicates that the work file needed for compiling has a volume serial number of DSC2. The FORTRAN compiler is located on the device with a volume serial number of DSC3 in the file labeled ALTLODLIB. |
| 5 | End of job. |

Example 2b:

By using the keyword parameters in example 2a, the following control stream is generated.

1	LABEL	△OPERATION△	OPERAND	△
		10	16	
1	/// JOB	FRTRN2A		
2	/// DVC	21	/// LFD	PRNTR
3	/// DVC	50	/// VOL	DSC1
4	/// LBL	U\$SRC	/// LFD	INFPUT
5	/// DVC	51	/// VOL	DSC2
6	/// LBL	U\$OBJ	/// LFD	OUTFPUT
7	/// DVC	51	/// VOL	DSC2
8	/// EXT	ST, C, 3,	CYL, 1	
9	/// LBL	\$SCRI	/// LFD	\$SCRI
10	/// DVC	52	/// VOL	DSC3
11	/// LBL	ALTLODLIB	/// LFD	ALTLOD
12	/// EXEC	FORT,	ALTLOD	
13	/// PARAM	IN=	PROGDM/INFPUT	
14	/// PARAM	OUT=	OUTFPUT	
15	/// PARAM	LST=	2	
16	/&			

<u>Line</u>	<u>Explanation</u>
1	Indicates that the name of the job is FRTRN2B.
2	Indicates that the printer is to be assigned to the logical unit number 21, with an LFD name of PRNTR. This was obtained from line 2 in example 2a.
3	Indicates that the input file volume serial number is DSC1. This was obtained from the IN parameter of line 2 in example 2a. It is assigned to the device with a logical unit number of 50, which was the first available number in the range of 50-54.
4	Indicates that the input file is labeled U\$SRC with an LFD name of ININPUT. This was obtained from the IN parameter of line 2 in example 2a.
5	Indicates that the output file volume serial number is DSC2. This was obtained from the OUT parameter of line 3 in example 2a. It is assigned to the device with a logical unit number of 51, which was the next available number in the range of 50-54. Logical unit number 50 was already assigned to the device with a volume serial number of DSC1 (line 3).
6	Indicates that the output file is labeled U\$OBJ with an LFD name of OUTFPUT. This was obtained from the OUT parameter of line 3 in example 2a.
7-9	Indicates the work file for the compiler has a volume serial number of DSC2. Because this volume serial number was already used, this work file uses the same device logical unit number of 51. This work file has both a file label and LFD name of \$SCR1 and uses the sequential access technique; allocation is contiguous; three cylinders are allocated for the secondary increment; and one cylinder is allocated for the first extent. This was obtained from line 4 in example 2a.
10	Indicates that the alternate load library for the compiler has a volume serial number of DSC3. It is assigned to the device with a logical unit number of 52, which was the next available number in the range of 50-54. This was obtained from the ALTLOD parameter of line 4 in example 2a.
11	Indicates that the alternate load library has a label of ALTLODLIB with an LFD name of ALTLOD. This was obtained from the ALTLOD parameter of line 4 in example 2a.
12	Loads the FORTRAN compiler from the file labeled ALTLOD.
13-15	PARAM control statements, which identify the processing options for the FORTRAN compiler. These are generated in the following manner: Line 13 - The filename ININPUT is generated automatically when the IN parameter is specified. The module name PROGNM is generated from the label field in line 2 of example 2a. Line 14 - The filename OUTFPUT is generated automatically when the OUT parameter is used. Line 15 - Indicates that diagnostic error messages are to be listed. This was obtained from the LST parameter in line 3 of example 2a.
16	End of job.

Optional formats of the FORT procedure call statement generate the necessary job control statements to compile and link a FORTRAN program (FORTL) and to compile, link, and execute a FORTRAN program (FORTLG). The keyword parameters of FORT also apply to FORTL and FORTLG.

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

1	LABEL	ΔOPERATIONΔ	OPERAND	Δ
		10	16	
1.	// JOB	FRTRN3A		
2.	// FORTL			
3.	/ \$			
4.				
5.		SOURCE DECK		
6.	/ *			
7.	/ \$			
8.		LOADM	X.MPL3A	
9.	/ *			

Line	Explanation
1	Indicates the job name is FRTRN3A.
2	Indicates the name of the procedure being called (FORTL).
3	Indicates compiler start of data.
4-5	Represents source deck to be compiled.
6	Indicates compiler end of data.
7	Indicates linkage editor start of data.
8	Indicates name of load module generated by linkage editor (if omitted, default=LNKLOD).
9	Indicates linkage editor end of data.

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

1	LABEL	Δ OPERATION Δ	OPERAND	Δ
		10	16	
1.	/I	JOB, FRTRN3B		
2.	/I	FORTLG		
3.	/I	\$		
4.		•		
5.		SOURCE DECK		
6.		•		
7.	/I	*		
8.	/I	\$		
9.		LOADM	XMPL3B	
10.	/I	*		
11.	/I	\$		
12.		•		
13.		EXECUTION DATA		
14.		•		
15.	/I	*		

Line	Explanation
1	Indicates the job name is FRTRN3B.
2	Indicates the name of the procedure being called (FORTLG).
3	Indicates compiler start of data.
4-6	Represents source deck to be compiled.
7	Indicates compiler end of data.
8	Indicates linkage editor start of data.
9	Represents input to linkage editor (e.g., specific includes, etc).
10	Indicates linkage editor end of data.
11	Indicates start of data for execution time input.
12-14	Represents card input for execution.
15	Indicates end of data for execution.

NOTES:

1. When either the FORTL or FORTLG procedure call statements are used, the OUT parameter cannot be used.
2. The FORTLG procedure call statement cannot be used when generating with the shared data management feature. Instead, use the FORTL procedure call statement and include a separate EXEC statement to execute the load module.



12.2. DISK COMPILATION PROCEDURE

The following example shows the job control statements needed for disk compilation.

1	LABEL	ΔOPERATIONΔ	OPERAND	Δ
		10	16	
	// JOB NAME			
	//PROGNM	FORT	IN=(DISK,SRCE)	
	/&			

This job stream compiles the program named PROGNM in the file, SRCE, on a disk labeled DISK.

12.3. LINKING PROCEDURES

Sample job control streams for the linkage editor may be found in the OS/3 system service programs user guide, UP-8062 (current version). In the absence of a specific INCLUDE for a particular I/O configuration (e.g., INCLUDE MYIO), the system-provided object module, FP\$IO, will be included automatically in the executable program. In addition, the following points should be noted:

- All external references generated by the compiler are V-cons. The automatic overlay feature of the linker cannot be used if the program also contains a CALL LOAD or CALL FETCH.
- The associated variable of a DAM file is updated after every reference to the file. Therefore, it must be addressable in every path containing a reference to the file; it is suggested that it be placed in common for this reason.
- Mathematical library routines may be placed in an overlay phase; I/O library routines should always remain in the root phase.

12.4. EXECUTION PROCEDURES

All files used by the program must be defined with a DVC...LFD sequence of job control statements which appear in the job control stream prior to the execution of the program. The LFD name is FORTnn, where nn is the unit number.

12.4.1. Diagnostics

All diagnostics are directed to the first printer unit defined. There are four classes of messages:

- TERMINATION/PAUSE:

STOP and PAUSE print the identifying number first; STOP and EXIT then print the number of program check interrupts which have occurred during execution. PAUSE requires a GO or an EOJ response from the operator. GO means to continue processing; EOJ means to terminate execution.

■ MATHEMATICAL LIBRARY:

The mathematical library delivers a warning message when arguments are out of range and provides a substitute for the function value. The substitute value is always zero except in the following cases:

SIN/COS	—	SQRT(2.0)/2.0
TAN/COTAN	—	X.GT. 82E5, 1.0 X.EQ. 0, 7.2E75
ALOG/ALOG10	—	X.EQ. 0, 5.3E-79 X.LT. 0, ALOG(ABS(X))
EXP/EXP10	—	for X.GT. 174.6, 7.2E75
SQRT	—	for X.LT. 0, SQRT(ABS(X))
SINH	—	for X.LT. -175.3, 5.3E-79
SINH/COSH	—	for X.GT. 175.3, 7.2E75

The message format is:

F\$nn text

where nn digits identify the message. Some messages are merely warnings and program execution continues. A severe error is followed by a traceback and the program is terminated. See system messages manual, UP-8076 (current version) for the list of diagnostic messages. ←

12.4.2. Dumps ←

In the event of an abnormal termination, variables may be located in the memory dump by using the following procedure:

■ Common Variables

Add the address of the variable given in the compilation listing to the address allocated to the common block by the linker to yield the address for the most significant (left-most) byte of the variable.

■ Local Variables

Add the address of the variable, the value specified by 'VARIABLE OFFSET' in the compilation, and the address allocated to the object module by the linker.

■ Array Elements

Use one of these procedures to locate the first element. Table 2-2 shows how to locate any array element thereafter.



Sometimes, the reason a program terminates is not clear even with the diagnostic, traceback, and dump information. In such cases, the debugging aids (Section 9) should be used to isolate the problem areas. The most common error resulting in worthless or destroyed information occurs when a value is stored outside an array.

For example:

"C" FOR COMMENT	
STATEMENT NUMBER	FORTRAN STATEMENT
5	
6	
7	D I M E N S I O N A (5)
8	I = 1 0
9	A (I) = 0
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

This will cause unpredictable results. To diagnose these situations, the program should be recompiled with the SUBCHK (subscript checking) option specified.



Appendix A. Compile Time Error Messages

A.1. COMPILER TERMINATION ERRORS

If an error occurs during compilation that results in a termination of the compiler, an error code indicating the cause is printed out. Table A-1 lists the error codes, the causes, and the corrective actions that may be taken.

Table A-1. Compiler Termination Error Codes

Error Code	Cause	Corrective Action
1	Statement too long	Simplify statement
2	Statement too recursive	Simplify statement
3	Up to 511 of each of the following permitted: scalar variables arrays entities in all COMMON statements entities in EQUIVALENCE statements statement labels names in type statements unique REAL constants unique INTEGER constants unique DOUBLE PRECISION constants unique subprograms called arithmetic statement functions arguments for all subroutines and functions	Simplify the program
4	Insufficient storage for the compiler	Recompile specifying more main storage
5	Compiler error	Submit a software user report (SUR)
6	Object program longer than 65,535 bytes	Simplify programs; shorten arrays
7	Too many argument lists or calls	Simplify the program
8	Compiler error	Submit a software user report (SUR)

For faster response when submitting an SUR, include the listing, a dump, and a source module that re-creates the error.

↓

A.2. INITIAL SCAN ERRORS

Table A—2 lists the error messages that can be produced during the initial scan by the compiler.

Table A—2. Initial Scan Error Messages (Part 1 of 2)

Message Text	Problem Description	Corrective Action
ALLOCATION	This message has several causes: <ol style="list-style-type: none"> 1. The names of dummy arguments have appeared illegally in an EQUIVALENCE, COMMON, or DATA statement. 2. Program attempting to allocate numerical constants to improper variable boundaries. 3. Program attempting to save a single numerical constant in an array. 	<ol style="list-style-type: none"> 1. Correct or remove dummy arguments from these statements. 2. Correct statement. 3. Correct statement.
BLOCK DATA ONLY	Variables in labeled COMMON can be initialized only in a BLOCK DATA subprogram.	Move DATA statement to a BLOCK DATA subroutine.
DATA COUNT	Data items are not in a one-to-one correspondence with the list items on a DATA statement.	Correct the DATA statement.
DATA SIZE	A literal data item is too long for its corresponding list item on a DATA statement.	Correct the DATA statement.
DATA TYPE	A violation of the basic correspondence of type rule between a data item and a list item has occurred on a DATA statement.	Correct the DATA statement.
FORMAT ID NOT ARRAY	A nonarray name has been referenced as a FORMAT.	Specify dimensions for the FORMAT name.
ID CONFLICT	A conflicting usage for a symbolic name in this statement has occurred previously. An example is a variable or array name that appears later as a subprogram name in a CALL statement.	Correct the conflicting usage.
ILLEGAL DO CLOSE	Improper DO loop structure is indicated. Problem such as ending a DO loop with an explicit or implicit branching statement may have occurred.	Correct the DO loop.
ILLEGAL LABEL	Program attempting to branch to a FORMAT or other nonexecutable statement.	Correct the branching statement.

↑

Table A-2. Initial Scan Error Messages (Part 2 of 2)

Message Text	Problem Description	Corrective Action
LENGTH	An illegal numerical length was specified in a TYPE statement. The only legal length specifications are 2, 4, and 8 depending on variable type.	Correct the TYPE statement.
NUMBER	An illegal number has been encountered in this statement. An example is a hexadecimal constant with more than 16 digits.	Correct the number in error.
MISSING COMMA	A comma syntactically necessary to the preceding statement was not found.	Supply the missing comma.
MULTIPLE DEFINED LABEL	The same label has occurred previously on another statement.	Correct the conflicting label usage.
ORDER	The flagged statement is out of order. Common examples are given by placing specification statements (DIMENSION, etc.) physically after executable statements (IF, etc.) in the program.	Correct the statement order in the program.
RANGE	An IMPLICIT statement has a range of letters specified backwards	Correct the letter range of the IMPLICIT statement.
SUBSCRIPT RANGE	A constant subscript of an array is outside the range implied by the array's declaration.	Correct the subscript in error.
SUBSCRIPTS	<ol style="list-style-type: none"> 1. Subscripting is illegal for non-array FORTRAN names. 2. Illegal subscripting has been encountered, e.g., a negative subscript. 	<ol style="list-style-type: none"> 1. Use an array declarator before subscripting. 2. Correct subscript in error.
SYNTAX	A serious statement error. An example would be starting a variable name with a nonletter character.	Correct statement's formal grammar.
TYPE CONFLICT	A real or integer term or expression has occurred on one side or the other of an .AND. or .OR. operator in a logical IF statement.	Correct the logical IF statement.
UNDIMENSIONED	An undimensioned variable has been used in this statement as though it were dimensioned. An open parenthesis follows directly after a variable name.	Correct statement or use an array declarator for the name in error.
UNRECOGNIZABLE STATEMENT	Serious statement error. Possible misspelled FORTRAN keyword.	Correct spelling errors, if any; otherwise, correct statement.

↓
A.3. INITIAL SCAN WARNING MESSAGES

Table A—3 lists the warning messages that can be produced during the initial scan by the compiler.

Table A—3. Initial Scan Warning Messages

Message Text	Problem Description	Corrective Action
ARGUMENT NUMBER CONFLICT	A breach of the argument number identity rule between definition and reference.	Correct program to conform to the argument number identity rule.
ARGUMENT TYPE CONFLICT	A breach of the argument type identity rule between definition and reference.	Correct program to conform to the argument type identity rule.
MISSING LABEL	The statement needs a label in order to be referenced.	Supply the missing label.
REAL*4 FUNCTION ASSUMED	The REAL*4 function is defaulted to in case a required generic function is not supported. An example of such an occurrence would be passing an INTEGER*2 argument to the generic function ABS.	Alter program to compensate for missing generic function if REAL*4 function is not required.
SYMBOL TOO LONG	All characters beyond the sixth in a FORTRAN name are ignored.	Shorten the symbolic name.

↑

Comments concerning this manual may be made in the space provided below. Please fill in the requested information.

System: _____

Manual Title: _____

UP No: _____ Revision No: _____ Update: _____

Name of User: _____

Address of User: _____

Comments:

NOTE: DO NOT USE THIS FORM TO ORDER MANUALS.


FOLD

FIRST CLASS
PERMIT NO. 21
BLUE BELL, PA.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

SPERRY  **UNIVAC**

P.O. BOX 500
BLUE BELL, PA.
19422

ATTN: SYSTEMS PUBLICATIONS DEPT.

CUT

FOLD

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

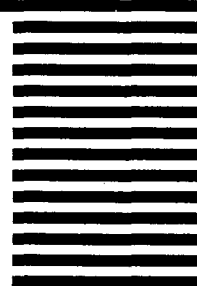
FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424



CUT

FOLD

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