decsystem10

FORTRAN IV PROGRAMMER'S REFERENCE MANUAL

digital

DEC-10-AFDO-D

decsystem10

FORTRAN IV PROGRAMMER'S REFERENCE MANUAL

The information in this document reflects the software as of Version 26 of the FORTRAN Compiler and Version 32 of the run-time operating system (LIB40).

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Printing March 1967
2nd Printing (Rev) November 1967
3rd Printing (Rev) September 1968
4th Printing April 1969
5th Printing June 1969
6th Printing September 1969
7th Printing (Rev) February 1970
Update Pages October 1970
Update Pages February 1971
Update Pages October 1971
Update Pages May 1972

Copyright © 1967, 1968, 1969, 1970, 1971, 1972 by Digital Equipment Corporation

The material in this manual is for information purposes and is subject to change without notice.

The following are trademarks of Digital Equipment Corporation, Maynard Massachusetts:

DEC	PDP
FLIP CHIP	FOCAL
DIGITAL	COMPUTER LAB

CONTENTS

Page

SECTION 1 THE PDP-10 FORTRAN LANGUAGE

CHAPTER 1 INTRODUCTION TO THE FORTRAN LANGUAGE

1.1	Line Format	1-1
1.1.1	Statement Number Field	1-1
1.1.2	Line Continuation Field	1-1
1.1.3	Statement Field	1-2
1.1.4	Comment Line	1-3
1.2	Character Set	1-3

CHAPTER 2 CONSTANTS, VARIABLES, AND EXPRESSIONS

2.1	Constants	2-1
2.1.1	Integer Constants	2-1
2.1.2	Real Constants	2-1
2.1.3	Double Precision Constants	2-2
2.1.4	Octal Constants	2-2
2.1.5	Complex Constants	2-2
2.1.6	Logical Constants	2-3
2.1.7	Literal Constants	2-3
2.2	Variables	2-4
2.2.1	Scalar Variables	2-4
2.2.2	Array Variables	2-4
2.3	Expressions	2-6
2.3.1	Numeric Expressions	2-6
2.3.2	Logical Expressions	2-8

CHAPTER 3 THE ARITHMETIC STATEMENT

3.1	General Description	3-1

CHAPTER 4 CONTROL STATEMENTS

4.1	GO TO Statement	4-1
4.1.1	Unconditional GO TO Statements	4-1
4.1.2	Computed GO TO Statements	4-1
4.1.3	Assigned GO TO Statement	4-2

		Page
4.2	IF Statement	4-2
4.2.1	Numerical IF Statements	4-3
4.2.2	Logical IF Statements	4-3
4.3	DO Statement	4-4
4.4	CONTINUE Statement	4-6
4.5	PAUSE Statement	4-6
4.6	STOP Statement	4-7
4.7	END Statement	4-7
CHAPTER 5	DATA TRANSMISSION STATEMENTS	
5.1	Nonexecutable Statements	5-1
5.1.1	FORMAT Statement	5-1
5.1.2	NAMELIST Statement	5-11
5.2	Data Transmission Statements	5-13
5.2.1	Input/Output Lists	5-14
5.2.2	Input/Output Records	5-15
5.2.3	PRINT Statement	5-15
5.2.4	PUNCH Statement	5-16
5.2.5	TYPE Statement	5-16
5.2.6	WRITE Statement	5-16
5.2.7	READ Statement	5-17
5.2.8	REREAD Statement	5-19
5.2.9	ACCEPT Statement	5-20
5.3	Device Control Statements	5-20
5.4	Encode and Decode Statements	5-21
CHAPTER 6	SPECIFICATION STATEMENTS	
6.1	Storage Specification Statements	6-2
6.1.1	DIMENSION Statement	6-2
6.1.2	COMMON Statement	6-4
6.1.3	EQUIVALENCE Statement	6-5
6.1.4	EQUIVALENCE and COMMON	6-6
6.2	Data Specification Statements	6-6
6.2.1	DATA Statement	6-6

		Page
6.2.2	BLOCK DATA Statement	6-8
6.3	Type Declaration Statements	6-8
6.3.1	IMPLICIT Statement	6-9
CHAPTER 7	SUBPROGRAM STATEMENTS	
7.1	Dummy Identifiers	7-1
7.2	Library Subprograms	7-1
7.3	Arithmetic Function Definition Statement	7-1
7.4	FUNCTION Subprograms	7-2
7.4.1	FUNCTION Statement	7-2
7.5	SUBROUTINE Subprograms	7-4
7.5.1	SUBROUTINE Statement	7-4
7.5.2	CALL Statement	7-5
7.5.3	RETURN Statement	7-5
7.6	BLOCK DATA Subprograms	7-6
7.6.1	BLOCK DATA Statement	7-6
7.7	EXTERNAL Statement	7-6
7.8	Summary of PDP-10 FORTRAN IV Statements	7-7
SECTION II	THE RUNTIME SYSTEM	
CHAPTER 8	THE FORTRAN IV LIBRARY - LIB40	
8.1	The FORTRAN Operating System	8-1
8.1.1	FORSE.	8-1
8.1.2	I/O Conversion Routines	8-2
8.1.3	FORTRAN UUOs	8-3
8.2	Science Library and FORTRAN Utility Subprograms	8-4
8.2.1	FORTRAN IV Library Functions	8-4
8.2.2	FORTRAN IV Library Subroutines	8-8
CHAPTER 9	SUBPROGRAM CALLING SEQUENCES	
9.1	Macro Subprograms Called by FORTRAN Main Programs	9-1
9.1.1	Calling Sequences	9-1
9.1.2	Returning of Answers	9-2

		Page
9.1.3	Use of Accumulators	9-2
9.1.4	Examples of Subprogram Linkage	9-2
9.2	Macro Main Programs Which Reference FORTRAN Subprograms	9-9
9.2.1	Calling Sequences	9-9
9.2.2	Returning of Answers	9-9
9.2.3	Example of Subprogram Linkage	9-10

CHAPTER 10 ACCUMULATOR CONVENTIONS FOR MAIN PROGRAMS AND SUBPROGRAMS

10.1	Locations	10-1
10.2	Accumulators	10-1
10.2.1	Accumulators 0 and 1	10-1
10.2.2	Accumulators 2 through 15	10-2
10.2.3	Accumulators 16 and 17	10-2
10.3	UUOs	10-2
10.4	Subprograms Called by JSA 16, Address	10-2
10.5	Subprograms Called by PUSHJ 17, Address	10-2
10.6	Subprograms Called by UUOs	10-3

CHAPTER 11 SWITCHES AND DIAGNOSTICS

11.1	FORTRAN Switches and Diagnostics	11-1

CHAPTER 12 FORTRAN USER PROGRAMMING

12.1	ASCII Character Set	12-1
12.2	PDP-10 Word Formats	12-2
12.3	FORTRAN Input/Output	12-3
12.3.1	Logical and Physical Peripheral Device Assignments	12-4
12.3.2	DECtape and Disk Usage	12-4
12.3.3	Magnetic Tape Usage	12-6
12.4	Random Access Programming	12-7
12.4.1	How to Use Random Access	12-8
12.4.2	Restrictions	12-8
12.4.3	Examples	12-9
12.5	PDP-10 Instruction Set	12-13

APPENDIX A THE SMALL FORTRAN IV COMPILER

ILLUSTRATIONS

		Page
1-1	Typical FORTRAN Coding Form	1-2
2-1	Array Storage	2-5
4-1	Nested DO Loops	4-5

TABLES

2-1	Types of Resultant Subexpressions	2-7
3-1	Allowed Assignment Statements	3-2
5-1	Magnitude of Internal Data	5-3
5-2	Numeric Field Codes	5-4
5-3	Device Control Statements	5-20
8-1	I/O Conversion Routine	8-2
8-2	FORTRAN UUOs	8-3
8-3	FORTRAN IV Library Functions	8-5
8-4	FORTRAN IV Library Subroutines	8-8
10-1	Accumulator Conventions for PDP-10 FORTRAN IV Compiler and Subprograms	10-3
11-1	FORTRAN Compiler Switch Options	11-1
11-2	FORTRAN Compiler Diagnostics (Command Errors)	11-2
11-3	FORTRAN Compiler Diagnostics (Compilation Errors)	11-3
11-4	FORTRAN Operating System Diagnostics (Execution Errors)	11-8
12-1	ASCII Character Set	12-1
12-2	PDP-10 FORTRAN IV Standard Peripheral Devices	12-3
12-3	Device Table for FORTRAN IV	12-5

PREFACE

This is a reference manual describing the specific statements and features of the FORTRAN IV language for the PDP-10. It is written for the experienced FORTRAN programmer who is interested in writing and running FORTRAN IV programs alone or in conjunction with MACRO-10 programs in the single-user or time-sharing environment. Familiarity with the basic concepts of FORTRAN programming on the part of the user is assumed. PDP-10 FORTRAN IV conforms to the requirements of the USA Standard FORTRAN.

The FORTRAN compiler translates source programs written in the FORTRAN IV language into the machine language of the PDP-10. This translated version of the FORTRAN program exists as a retrievable, relocatable binary file on some storage device. All relocatable binary filenames have the extension .REL if they reside on a directory-oriented device (disk or DECtape). Binary files may also be created by the MACRO-10 assembler (see Chapter 9)[1].

In order for the FORTRAN program to be processed, the Linking Loader must load the relocatable binary file into core memory. Also loaded are any relocatable binary files found in the FORTRAN library (LIB40) which are necessary for the program's execution. Within the FORTRAN source program, the library files may be called explicitly, such as SIN, in the statement

```
X = SIN(Y)
```

or implicitly, such as FLOUT., the floating-point to ASCII conversion routine, which is implied in the following statements.

```
      PRINT    3,X
3     FORMAT(1X,F4.2)
```

A FORTRAN main program and its FORTRAN and/or MACRO-10 subprograms may be compiled or assembled separately and then linked together by the Linking Loader at load time. The core image may then be saved on a storage device. When saved on a directory storage device, these files have the extension .SAV in a multiprogramming Monitor system and .SVE in a single-user Monitor system.

The Time-Sharing Monitors act as the interface between the user and the computer so that all users are protected from one another and appear to have system resources available to themselves. Several user programs are loaded into core at once and the Time-Sharing Monitors schedule each program to run for a certain length of time. All Monitors direct data flow between I/O devices and user programs, making the programs device independent, and overlap I/O operations concurrently with computations.

In a multiprogramming system, all jobs reside in core and the scheduler decides which of these jobs should run. In a swapping system, jobs can exist on an external storage device (usually disk) as well as in core. The scheduler

[1] For further information on the MACRO-10 assembler, see the MACRO-10 ASSEMBLER manual, DEC-10-AMZC-D.

decides not only which job is to run but also when a job is to be swapped out onto the disk or brought back into core.

The number of users that can be handled by a given size time-sharing configuration is further increased by using the reentrant user-programming capability. This means that a sequence of instructions may be entered by more than one user job at a time. Therefore, a single copy of a reentrant program may be shared by a number of users at the same time to increase system economy. The FORTRAN compiler and operating system are both reentrant.

SECTION I

The PDP-10 FORTRAN IV Language

The seven chapters of this section deal with the PDP-10 FORTRAN IV language. Included in these chapters are the language elements of FORTRAN IV and the five categories of FORTRAN IV statements (arithmetic, control, input/output, specification, and subprogram).

CHAPTER 1

INTRODUCTION TO THE FORTRAN LANGUAGE

The term FORTRAN IV (FORmula TRANslation) is used interchangeably to designate both the FORTRAN IV language and the FORTRAN IV translator or compiler. The FORTRAN IV language is composed of mathematical-form statements constructed in accordance with precisely formulated rules. FORTRAN IV programs consist of meaningful sequences of FORTRAN statements intended to direct the computer to perform the specified operations and computations.

The FORTRAN IV compiler is itself a computer program that examines FORTRAN IV statements and tells the computer how to translate the statements into machine language. The compiler runs in a minimum of 9K of core. The program written in FORTRAN IV language is called the source program. The resultant machine language program is called the object program. Digital's small FORTRAN compiler, which runs in 5.5K of core, is virtually identical to the larger compiler, except for differences explained in Appendix 2. Operating procedures and diagnostic messages for both compilers are explained in the PDP-10 System Users Guide (DEC-10-NGCC-D).

1.1 LINE FORMAT

Each line of a FORTRAN program consists of three fields: statement number field, line continuation field, and statement field. A typical FORTRAN program is shown in Figure 1-1.

1.1.1 Statement Number Field

A statement number consists of from one to five digits in columns 1-5. Leading zeros and all blanks in this field are ignored. Statement numbers may be in any order and must be unique. Any statement referenced by another statement must have a statement number. For source programs prepared on a teletypewriter, a horizontal tab may be used to skip to the statement field with from 0 through 5 characters in the label field. This is the only place a tab is not treated as a space.

1.1.2 Line Continuation Field

If a FORTRAN statement is so large that it cannot conveniently fit into one statement field, the statement fields of up to 19 additional lines may be used to specify the complete statement. Any line which is not continued, or the first line of a sequence of continued lines, must have a blank or zero in column 6. Continuation lines must

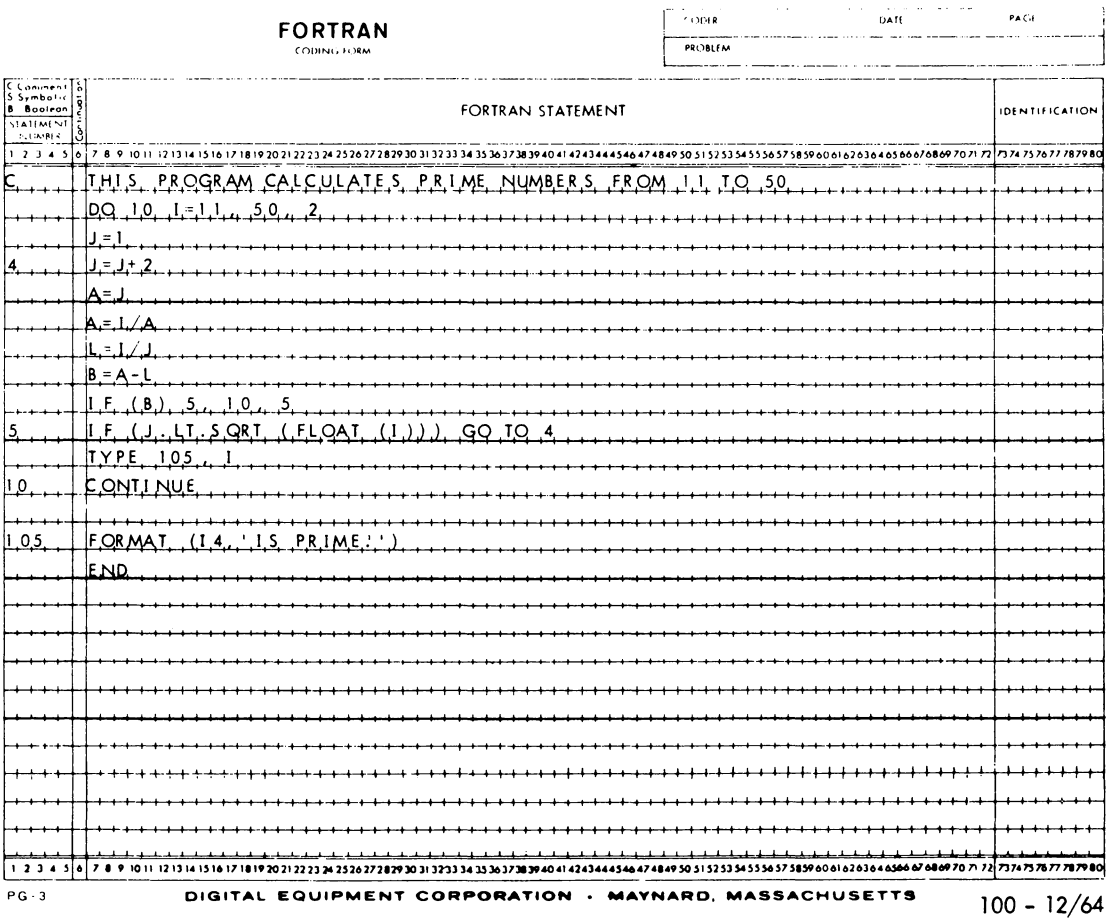


Figure 1-1 Typical FORTRAN Coding Form

have a character other than blank or zero in column 6. If a continuation line is desired when a TAB is used in the statement number field, a digit from 1 to 9 must immediately follow the TAB.

1.1.3 Statement Field

Any FORTRAN statement, as described in later sections, may appear in the statement field (columns 7-72). Except for alphanumeric data within a FORMAT statement, DATA statement, or literal constant, blanks (spaces) and TABS are ignored and may be used freely for appearance purposes. Thus the following statements are equivalent.

```
END (tab) FILE (tab) 2
END (space) FILE (space) 2
ENDFILE2
```

1.1.4 Comment Line

Any line that starts with one of the characters $ * / or the letter C in column 1 is interpreted as a line of comments. Comment lines are printed onto any listings requested but are otherwise ignored by the compiler. Columns 2-72 may be used in any format for comment purposes. A comment line must not immediately precede a continuation line.

As an aid for program debugging, the letter D in column 1 causes the line to be interpreted as a comment unless the /I switch appears in the command string. (Refer to Table 11-1 for Compile Switch options.) If the /I switch is present, the letter D in column 1 is interpreted as a space and the line is compiled as a program statement.

1.2 CHARACTER SET

The following characters are used in the FORTRAN IV language:

Blank	0	@	P
!	1	A	Q
"	2	B	R
#	3	C	S
$	4	D	T
%	5	E	U
&	6	F	V
'	7	G	W
(	8	H	X
)	9	I	Y
*	:	J	Z
+	;	K	↑
,	<	L	
-	=	M	
.	>	N	
/	?	O	

NOTE

ASCII characters greater than Z (132_8) are replaced by the error character "↑". See Chapter 12 for the internal representation of these characters.

CHAPTER 2

CONSTANTS, VARIABLES, AND EXPRESSIONS

The rules for defining constants and variables and for forming expressions are described in this chapter.

2.1 CONSTANTS

Seven types of constants are permitted in a FORTRAN IV source program: integer or fixed point, real or single-precision floating point, double-precision floating point, octal, complex, logical, and literal.

2.1.1 Integer Constants

An integer constant consists of from one to eleven decimal digits written without a decimal point. A negative constant must be preceded by a minus sign. A positive constant may be preceded by a plus sign.

Examples:
```
3
+10
-528
8085
```

An integer constant must fall within the range $-2^{35}+1$ to $2^{35}-1$. When used for the value of a subscript, the value of the integer constant is taken as modulo 2^{18}.

2.1.2 Real Constants

Real constants are written as a string of decimal digits including a decimal point. A real constant may consist of any number of digits but only the leftmost 9 digits appear in the compiled program. Real constants may be given a decimal scale factor by appending an E followed by a signed integer constant. The field following the letter E must not be blank, but may be zero.

Examples:
```
 15.
  0.0
   .579
-10.794
  5.0E3(i.e., 5000.)
  5.0E+3(i.e., 5000)
  5.0E-3(i.e., 0.005)
```

A real constant has precision to eight digits. The magnitude must lie approximately within the range 0.14×10^{-38} to 1.7×10^{38}. Real constants occupy one word of PDP-10 storage.

2.1.3 Double Precision Constants

A double precision constant is specified by a string of decimal digits, including a decimal point, which are followed by the letter D and a signed decimal scale factor. The field following the letter D must not be blank, but may be zero.

```
Examples:   24.671325982134D0
             3.6D2 (i.e., 360.)
             3.6D-2 (i.e., .036)
             3.0D0
```

Double precision constants have precision to 16 digits. The magnitude of a double precision constant must lie approximately between 0.14×10^{-38} and 1.7×10^{38}. Double-precision constants occupy two words of PDP-10 storage.

2.1.4 Octal Constants

A number preceded by a double quote represents an octal constant. An octal constant may appear in an arithmetic or logical expression or a DATA statement. Only the digits 0-7 may be used and only the last twelve digits are significant. A minus sign may precede the octal number, in which case the number is negated. A maximum of 12 octal digits are stored in each 36-bit word.

```
Examples:   "7777
            "-31563
```

2.1.5 Complex Constants

FORTRAN IV provides for direct operations on complex numbers. Complex constants are written as an ordered pair of real constants separated by a comma and enclosed in parentheses.

```
Examples:   (.70712, -.70712)
            (8.763E3,2.297)
```

The first constant of the pair represents the real part of the complex number, and the second constant represents the imaginary part. The real and imaginary parts may each be signed. The enclosing parentheses are part of the constant and always appear, regardless of context. Each part is internally represented by one single-precision floating point word. They occupy consecutive locations of PDP-10 storage.

FORTRAN IV arithmetic operations on complex numbers, unlike normal arithmetic operations, must be of the form:

$$A \pm B = a_1 \pm b_1 + i(a_2 \pm b_2)$$

$$A * B = (a_1 b_1 - a_2 b_2) + i(a_2 b_1 + a_1 b_2)$$

$$A/B = \frac{(a_1 b_1 + a_2 b_2)}{b_1^2 + b_2^2} + i\,\frac{(a_2 b_1 - a_1 b_2)}{b_1^2 + b_2^2}$$

where $A = a_1 + ia_2$, $B = b_1 + ib_2$, and $i = \sqrt{-1.}$

2.1.6 Logical Constants

The two logical constants, .TRUE. and .FALSE., have the internal values -1 and 0, respectively. The enclosing periods are part of the constant and always appear.

Logical constants may be entered in DATA or input statements as signed octal integers (-1 and 0). Logical quantities may be operated on in either arithmetic or logical statements. Only the sign is tested to determine the truth value of a logical variable.

2.1.7 Literal Constants

A literal constant may be in either of two forms:

a. A string of alphanumeric and/or special characters enclosed in single quotes; two adjacent single quotes within the constant are treated as one single quote.

b. A string of characters in the form

$$nHx_1x_2\ldots x_n$$

where $x_1x_2\ldots x_n$ is the literal constant, and n is the number of characters following the H.

Literal constants may be entered in DATA statements or input statements as a string of up to 5 7-bit ASCII characters per variable (10 characters if the variable is double-precision or complex). Literal constants may be operated on in either arithmetic or logical statements.

NOTE

Literal constants used as subprogram arguments will have a zero word as an end-of-string indicator.

Examples:
```
CALL SUB ( LITERAL CONSTANT )
'DONT''T'
5HDON'T
A = 'FIVE' + 42
B = (5HABCDE .AND. ''376)/2
```

2.2 VARIABLES

A variable is a quantity whose value may change during the execution of a program. Variables are specified by name and type. The name of a variable consists of one or more alphanumeric characters, the first one of which must be alphabetic. Only the first six characters are interpreted as defining the variable name. The type of variable (integer, real, logical, double precision, or complex) may be specified explicitly by a type declaration statement or implicitly by the IMPLICIT statement. If the variable is not specified in this manner, then a first letter of I, J, K, L, M or N indicates a fixed point (integer) variable; any other first letter indicates a floating-point (real) variable. Variables of any type may be either scalar or array variables. When used in a subscript or as an index to a DO Statement, the value of the integer variable is taken as modulo 2^{18}.

2.2.1 Scalar Variables

A scalar variable represents a single quantity.

Examples:
```
A
G2
POPULATION
```

2.2.2 Array Variables

An array variable represents a single element of an n dimensional array of quantities. The variable is denoted by the array name followed by a subscript list enclosed in parentheses. The subscript list is a sequence of integer expressions, separated by commas. The expressions may be of any form or type providing they are explicitly changed to type integer when each is completely evaluated. Each expression represents a subscript, and the values of the expressions determine the array element referred to. For example, the row vector A_i would be represented by the subscripted variable A(J), and the element, in the second column of the first row of the square matrix A, would be represented by A(1,2). Arrays may have any number of dimensions.

Examples:
```
Y(1)
STATION (K)
A (3* K+2, I, J-1)
```

The three arrays above (Y, STATION, and A) would have to be dimensioned by a DIMENSION, COMMON, or type declaration statement prior to their first appearance in an executable statement or in a DATA or NAMELIST statement. (Array dimensioning is discussed in Chapter 6).

1-Dimensional Array A(10)

A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)

CONSECUTIVE STORAGE LOCATIONS ——————————

2-Dimensional Array B(5,5)

1	B(1,1)	6	B(1,2)	11	B(1,3)	16	B(1,4)	21	B(1,5)
2	B(2,1)	7	B(2,2)	12	B(2,3)	17	B(2,4)	22	B(2,5)
3	B(3,1)	8	B(3,2)	13	B(3,3)	18	B(3,4)	23	B(3,5)
4	B(4,1)	9	B(4,2)	14	B(4,3)	19	B(4,4)	24	B(4,5)
5	B(5,1)	10	B(5,2)	15	B(5,3)	20	B(5,4)	21	B(5,5)

B(3,1) IS THE THIRD STORAGE WORD IN SEQUENCE
B(3,4) IS THE EIGHTEENTH STORAGE WORD IN SEQUENCE

3-Dimensional Array C(5,5,5)

										101	C(1,1,5)	106	C(1,2,5)	111	C(1,3,5)	116	C(1,4,5)	121	C(1,5,5)
										102	C(2,1,5)	107	C(2,2,5)	112	C(2,3,5)	117	C(2,4,5)	122	C(2,5,5)
						76	C(1,1,4)	81	C(1,2,4)	86	C(1,3,4)	91	C(1,4,4)	96	C(1,5,4)	118	C(3,4,5)	123	C(3,5,5)
						77	C(2,1,4)	82	C(2,2,4)	87	C(2,3,4)	92	C(2,4,4)	97	C(2,5,4)	119	C(4,4,5)	124	C(4,5,5)
				51	C(1,1,3)	56	C(1,2,3)	61	C(1,3,3)	66	C(1,4,3)	71	C(1,5,3)	98	C(3,5,4)	120	C(5,4,5)	125	C(5,5,5)
				52	C(2,1,3)	57	C(2,2,3)	62	C(2,3,3)	67	C(2,4,3)	72	C(2,5,3)	99	C(4,5,4)				
		26	C(1,1,2)	31	C(1,2,2)	36	C(1,3,2)	41	C(1,4,2)	46	C(1,5,2)	73	C(3,5,3)	100	C(5,5,4)				
		27	C(2,1,2)	32	C(2,2,2)	37	C(2,3,2)	42	C(2,4,2)	47	C(2,5,2)	74	C(4,5,3)						
1	C(1,1,1)	6	C(1,2,1)	11	C(1,3,1)	16	C(1,4,1)	21	C(1,5,1)	48	C(3,5,2)	75	C(5,5,3)						
2	C(2,1,1)	7	C(2,2,1)	12	C(2,3,1)	17	C(2,4,1)	22	C(2,5,1)	49	C(4,5,2)								
3	C(3,1,1)	8	C(3,2,1)	13	C(3,3,1)	18	C(3,4,1)	23	C(3,5,1)	50	C(5,5,2)								
4	C(4,1,1)	9	C(4,2,1)	14	C(4,3,1)	19	C(4,4,1)	24	C(4,5,1)										
5	C(5,1,1)	10	C(5,2,1)	15	C(5,3,1)	20	C(5,4,1)	25	C(5,5,1)										

C(1,3,2) is the 36th storage word in sequence.

C(1,1,5) is the 101st storage word in sequence.

Figure 2-1 Array Storage

Arrays are stored in increasing storage locations with the first subscript varying most rapidly and the last subscript varying least rapidly. For example, the 2-dimensional array B(I,J) is stored in the following order: B (1,1), B (2,1),..., B (I,1),B (1,2),B (2,2),...,B (I,2),...,B (I,J).

2.3 EXPRESSIONS

Expressions may be either numeric or logical. To evaluate an expression, the object program performs the calculations specified by the quantities and operators within the expression,

2.3.1 Numeric Expressions

A numeric expression is a sequence of constants, variables, and function references separated by numeric operators and parentheses in accordance with mathematical convention and the rules given below.

The numeric operators are +, -, *, /, **, denoting, respectively, addition, subtraction, multiplication, division, and exponentiation.

In addition to the basic numeric operators, function references are also provided to facilitate the evaluation of functions such as sine, cosine, and square root. A function is a subprogram which acts upon one or more quantities, called arguments, to produce a single quantity called the function value. Function references are denoted by the identifier, which names the function (such as SIN, COS, etc.), followed by an argument list enclosed in parentheses:

```
identifier(argument, argument, ..., argument)
```

At least one argument must be present. An argument may be an expression, an array identifier, a subprogram identifier, or an alphanumeric string.

Function type is given by the type of the identifier which names the function. The type of the function is independent of the types of its arguments. (See Chapter 7, Section 7.4.1.1.)

A numeric expression may consist of a single element (constant, variable, or function reference):

```
2.71828
Z(N)
TAN(THETA)
```

Compound numeric expressions may be formed by using numeric operations to combine basic elements:

```
X+3.
TOTAL/A
TAN(PI*M)
(X+3.) -(TOTAL/A) * TAN (PI*M)
```

Compound numeric expressions must be constructed according to the following rules:

a. With respect to the numeric operators +, -, *, /, any type of quantity (logical, octal, integer, real, double precision, complex or literal) may be combined with any other, with one exception: a complex quantity cannot be combined with a double precision quantity.

The resultant type of the combination of any two types may be found in Table 2-1. The conversions between data types will occur as follows:

(1) A literal constant will be combined with any integer constant as an integer and with a real or double word as a real or double word quantity. (Double word refers to both double precision and complex.)

(2) An integer quantity (constant, variable, or function reference) combined with a real or double word quantity results in an expression of the type real or double word respectively; e.g., an integer variable plus a complex variable will result in a complex subexpression. The integer is converted to floating point and then added to the real part of the complex number. The imaginary part is unchanged.

(3) A real quantity (constant, variable, or function reference) combined with a double word quantity results in an expression that is of the same type as the double word quantity.

(4) A logical or octal quantity is combined with an integer, real, or double word quantity as if it were an integer quantity in the integer case, or a real quantity in the real or double word case (i.e., no conversion takes place).

b. Any numeric expression may be enclosed in parentheses and considered to be a basic element.

```
(X+Y)/2
(ZETA)
(COS(SIN(PI*M)+X))
```

Table 2-1
Types of Resultant Subexpressions

+,-,*,/		Type of Quantity				
		Real	Integer	Complex	Double Precision	Logical, Octal, or Literal
Type of Quantity	Real	Real	Real	Complex	Double Precision	Real
	Integer	Real	Integer	Complex	Double Precision	Integer
	Complex	Complex	Complex	Complex	Not Allowed	Complex
	Double Precision	Double Precision	Double Precision	Not Allowed	Double Precision	Double Precision
	Logical, Octal, or Literal	Real	Integer	Complex	Double Precision	Logical, Octal, or Literal

c. Numeric expressions which are preceded by a + or – sign are also numeric expressions:

```
+X
-(ALPHA*BETA)
-SQRT(-GAMMA)
```

d. If the precedence of numeric operations is not given explicitly by parentheses, it is understood to be the following (in order of decreasing precedence):

Operator	Explanation
**	numeric exponentiation
*and/	numeric multiplication and division
+and-	numeric addition and subtraction

In the case of operations of equal hierarchy, the calculation is performed from left to right.

e. No two numeric operators may appear in sequence. For instance:

```
X*-Y
```

is improper. Use of parentheses yields the correct form:

```
X*(-Y)
```

By use of the foregoing rules, all permissible numeric expressions may be formed. As an example of a typical numeric expression using numeric operators and a function reference, the expression for one of the roots of the general quadratic equation:

$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$

would be coded as:

```
(-B+SQRT(B**2-4.*A*C))/(2.*A)
```

2.3.2 Logical Expressions

A logical expression consists of constants, variables, function references, and arithmetic expressions, separated by logical operators or relational operators. Logical expressions are provided in FORTRAN IV to permit the implementation of various forms of symbolic logic. Logical masks may be represented by using octal constants. The result of a logical expression has the logical value TRUE (negative) or FALSE (positive or zero) and therefore, only uses one word.

2.3.2.1 Logical Operators - The logical operators, which include the enclosing periods and their definitions, are as follows, where P and Q are expressions:

.NOT.P	Has the value .TRUE. only if P is .FALSE., and has the value .FALSE. only if P is .TRUE.
P.AND.Q	Has the value .TRUE. only if P and Q are both .TRUE., and has the value .FALSE. if either P or Q is .FALSE.
P.OR.Q	(Inclusive OR) Has the value .TRUE. if either P or Q is .TRUE., and has the value .FALSE. only if both P and Q are .FALSE.
P.XOR.Q	(Exclusive OR) Has the value .TRUE. if either P or Q but not both are .TRUE., and has the value .FALSE. otherwise.
P.EQV.Q	(Equivalence) Has the value .TRUE. if P and Q are both .TRUE. or both .FALSE., and has the value .FALSE. otherwise.

Logical expressions are evaluated by combining the full word values of P and Q (only the high-order part if P and Q are double precision, only the real part if P and Q are complex) using the appropriate logical operator. The result is TRUE if it is arithmetically negative and FALSE if it is arithmetically positive or zero.

Logical operators may be used to form new variables, for example,

```
X = Y.AND.Z
E = E.XOR."400000000000
```

2.3.2.2 Relational Operators - The relational operators are as follows:

Operator	Relation
.GT.	greater than
.GE.	greater than or equal to
.LT.	less than
.LE.	less than or equal to
.EQ.	equal to
.NE.	not equal to

The enclosing periods are part of the operator and must be present.

Mixed expressions involving integer, real, and double precision types may be combined with relationals. The value of such an expression will be .TRUE. (-1) or .FALSE. (0).

The relational operators .EQ. and .NE. may also be used with COMPLEX expressions. (Double word quantities are equal if the corresponding parts are equal.)

A logical expression may consist of a single element (constant, variable, function reference, or relation):

```
.TRUE.
X.GE.3.14159
```

Single elements may be combined through use of logical operators to form compound logical expressions, such as:

```
TVAL.AND.INDEX
BOOL(M).OR.K.EQ.LIMIT
```

Any logical expression may be enclosed in parentheses and regarded as an element:

```
(T.XOR.S).AND.(R.EQV.Q)
CALL PARITY ((2.GT.Y.OR.X.GE.Y).AND.NEVER)
```

Any logical expression may be preceded by the unary operator .NOT. as in:

```
.NOT.T
.NOT.X+7.GT.Y+Z
BOOL(K).AND..NOT.(TVAL.OR.R)
```

No two logical operators may appear in sequence, except in the case where .NOT. appears as the second of two logical operators, as in the example above. Two decimal points may appear in sequence, as in the example above, or when one belongs to an operator and the other to a constant.

When the precedence of operators is not given explicitly by parentheses, it is understood to be as follows (in order of decreasing precedence):

```
**
*,/
+,-
.GT.,.GE.,.LT.,.LE.,.EQ.,.NE.
.NOT.
.AND.
.OR.
.EQV., .XOR.
```

For example, the logical expression

```
.NOT.ZETA**2+Y*MASS.GT.K-2.OR.PARITY.AND.X.EQ.Y
```

is interpreted as

```
(.NOT.(((ZETA**2)+(Y*MASS)).GT.(K-2))).OR.(PARITY.AND.(X.EQ.Y))
```

CHAPTER 3

THE ARITHMETIC STATEMENT

3.1 GENERAL DESCRIPTION

One of the key features of FORTRAN IV is the ease with which arithmetic computations can be coded. Computations to be performed by FORTRAN IV are indicated by arithmetic statements, which have the general form:

```
A=B
```

where A is a variable, B is an expression, and = is a replacement operator. The arithmetic statement causes the FORTRAN IV object program to evaluate the expression B and assign the resultant value to the variable A. Note that the = sign signifies replacement, not equality. Thus, expressions of the form:

```
A=A+B and
A=A*B
```

are quite meaningful and indicate that the value of the variable A is to be replaced by the result of the expression to the right of the = sign.

Examples:
```
Y=1*Y
P=.TRUE.
X(N)=N*ZETA(ALPHA*M/PI)+(1.,-1.)
```

Table 3-1 indicates which type of expression may be equated to each type of variable in an arithmetic statement. D indicates that the assignment is performed directly (no conversion of any sort is done); R indicates that only the real part of the variable is set to the value of the expression (the imaginary part is set to zero); C means that the expression is converted to the type of the variable; and H means that only the high-order portion of evaluated expression is assigned to the variable.

The expression value is made to agree in type with the assignment variable before replacement occurs. For example, in the statement:

```
THETA=W*(ABETA+E)
```

if THETA is an integer and the expression is real, the expression value is truncated to an integer before assignment to THETA.

Table 3-1
Allowed Assignment Statements

Variable	Expression				
	Real	Integer	Complex	Double Precision	Logical, Octal, or Literal Constant
Real	D	C	R,D	H,D	D
Integer	C	D	R,C	H,C	D
Complex	D,R,I	C,R,I	D	H,D,R,I	D,R,I
Double Precision	D,H,L	C,H,L	R,D,H,L	D	D,H,L
Logical	D	D	R,D	H,D	D

D - Direct Replacement

C - Conversion between integer and floating point

R - Real only

I - Set imaginary part to 0

H - High order only

L - Set low order part to 0

CHAPTER 4
CONTROL STATEMENTS

FORTRAN compiled programs normally execute statements sequentially in the order in which they were presented to the compiler. However, the following control statements are available to alter the normal sequence of statement execution: GO TO, IF, DO, PAUSE, STOP, END, CALL, RETURN. CALL and RETURN are used to enter and return from subroutines.

4.1 GO TO STATEMENT

The GO TO statement has three forms: unconditional, computed, and assigned.

4.1.1 Unconditional GO TO Statements

Unconditional GO TO statements are of the form:

GO TO n

where n is the number of an executable statement. Control is transferred to the statement numbered n. An unconditional GO TO statement may appear anywhere in the source program, except as the terminal statement of a DO loop.

4.1.2 Computed GO TO Statements

Computed GO TO statements have the form:

GO TO $(n_1,n_2,\ldots,n_k),i$

where $n_1,n_2,\ldots,n_k$ are statement numbers, and i is an integer expression.

This statement transfers control to the statement numbered $n_1,n_2,\ldots,n_k$ if i has the value 1, 2, ...,k, respectively. If i exceeds the size of the list of statement numbers or is less than one, execution will proceed to the next executable statement. Any number of statement numbers may appear in the list. There is no restriction on other uses for the integer variable i in the program.

In the example

```
GO TO (20,10,5),K
```

the variable K acts as a switch, causing a transfer to statement 20 if K=1, to statement 10 if K=2, or to statement 5 if K=3.

A computed GO TO statement may appear anywhere in the source program, except as the terminal statement of a DO loop.

4.1.3 Assigned GO TO Statement

Assigned GO TO statements have two equivalent forms:

```
GO TO k
```

and

```
GO TO k, (n1,n2,n3,...)
```

where k is a variable or array element and n_1, n_2, ...n_k are statement numbers. Any number of statement numbers may appear in the list. Both forms of the assigned GO TO have the effect of transferring control to the statement whose number is currently associated with the variable k. The second form of the assigned GO TO statement passe control to the next executable statement if k is not associated with one of the statement numbers in the list. This association is established through the use of the ASSIGN statement, the general form of which is:

```
ASSIGN i TO k
```

where i is a statement number and k is a variable or array element. If more than one ASSIGN statement refers to the same integer variable name, the value assigned by the last executed statement is the current value.

Examples:

```
ASSIGN 21 TO INT          ASSIGN 1000 TO INT
        .                         .
        .                         .
        .                         .
GO TO INT          GO TO INT, (2,21,1000,310)
```

An assigned GO TO statement may appear anywhere in the source program, except as the terminal statement of a DO loop.

4.2 IF STATEMENT

IF statements have two forms in FORTRAN IV: numerical and logical.

4.2.1 Numerical IF Statements

Numerical IF statements are of the form:

IF (expression) n_1,n_2,n_3

where n_1,n_2,n_3 are statement numbers. This statement transfers control to the statement numbered n_1,n_2,n_3 if the value of the numeric expression is less than, equal to, or greater than zero, respectively. All three statement numbers must be present. The expression may not be complex.

Examples:
```
IF (ETA) 4,7,12
IF (KAPPA-L (10)) 20,14,14
```

4.2.2 Logical IF Statements

Logical IF statements have the form:

IF (expression)S

where S is a complete statement. The expression must be logical. S may be any executable statement other than a DO statement or another logical IF statement (see Chapter 2, Section 2.3.2). If the value of the expression is .FALSE. (positive or zero), control passes to the next sequential statement. If value of the expression is .TRUE. (negative), statement S is executed. After execution of S, control passes to the next sequential statement unless S is a numerical IF statement or a GO TO statement; in these cases, control is transferred as indicated. If the expression is .TRUE. (negative) and S is a CALL statement, control is transferred to the next sequential statement upon return from the subroutine.

Numbers are present in the logical expression:

```
IF (B)Y=X*SIN(Z)
W=Y**2
```

If the value of B is .TRUE., the statements Y=X*SIN(Z) and W=Y**2 are executed in that order. If the value of B is .FALSE., the statement Y=X*SIN(Z) is not executed.

Examples:
```
IF (T.OR.S)X=Y+1
IF (Z.GT.X(K)) CALL SWITCH (S,Y)
IF (K.EQ.INDEX) GO TO 15
```

NOTE

Care should be taken in testing floating point numbers for equality in IF statements as rounding may cause unexpected results.

4.3 DO STATEMENT

The DO statement simplifies the coding of iterative procedures. DO statements are of the form:

DO n i=m_1,m_2,m_3

where n is a statement number, i is a nonsubscripted integer variable, and m_1,m_2,m_3 are any integer expressions. If m_3 is not specified, it is understood to be 1.

The DO statement causes the statements which follow, up to and including the statement numbered n, to be executed repeatedly. This group of statements is called the range of the DO statement. The integer variable i of the DO statement is called the index. The values of m_1,m_2, and m_3 are called, respectively, the initial, limit, and increment values of the index.

A zero increment (m_3) is not allowed. The increment m_3 may be negative if $m_1 \geq m_2$. If $m_1 \leq m_2$, the increment m_3 must be positive. The index variable can assume legal values only if $(m_2-m_i)*m_3 \geq 0$. (m_i is the current value of the index variable m_1.)

Examples:	Form	Restriction
	DO 10 I=1,5,2	
	DO 10 I=5,1,-1	
	DO 10 I=J,K,5	$J \leq K$
	DO 10 I=J,K,-5	$J \geq K$
	DO 10 L=I,J,-K	$I \leq J, K<0$ or $I \geq J, K>0$
	DO 10 L=I,J,K	$I \leq J, K>0$ or $I \geq J, K>0$

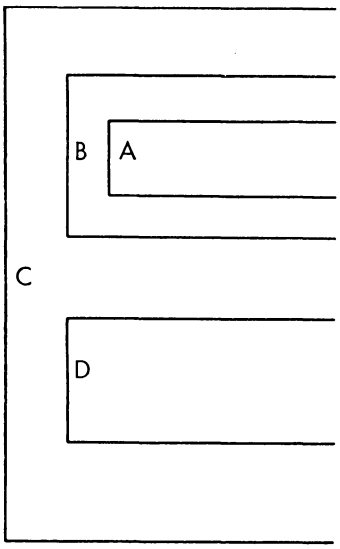
Initially, the statements of the range are executed with the initial value assigned to the index. This initial execution is always performed, regardless of the values of the limit and increment. After each execution of the range, the increment value is added to the value of the index and the result is compared with the limit value. If the value of the index is not greater than the limit value, the range is executed again using the new value of the index. When the increment value is negative, another execution will be performed if the new value of the index is not less than the limit value.

After the last execution of the range, control passes to the statement immediately following the range. This exit from the range is called the normal exit. Exit may also be accomplished by a transfer from within the range.

The range of a DO statement may include other DO statements, provided that the range of each contained DO statement is entirely within the range of the containing DO statement. When one DO loop is completely contained in another, it is said to be nested. DO loops can be nested to any depth. A transfer into the range of a DO statement from outside the range is not allowed.

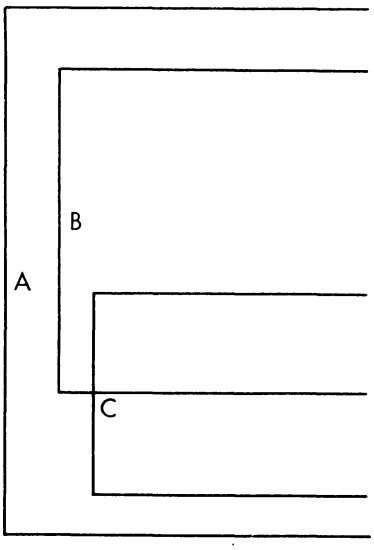
More than one DO loop within a nest of DO loops can end on the same statement. This terminal statement is considered to belong to the innermost DO loop that ends on the terminal statement. The statement label of such a terminal statement cannot be used in any GO TO or arithmetic IF statements except those that occur within the DO loop to which the terminal statement belongs.

Valid DO Loop Nest



Control must not pass from within loop A or loop B into loop D, or from loop D into loop A or loop B.

Invalid DO Loop Nest



Loop C is not fully within the range of loop B even though it is within the range of loop A.

Figure 4-1 Nested DO Loops

Within the range of a DO statement, the index is available for use as an ordinary variable. After a transfer from within the range, the index retains its current value and is available for use as a variable. The value of the index variable becomes undefined when the DO loop it controls is satisfied. The values of the initial, limit, and increment variables for the index and the index of the DO loop, may not be altered within the range of the DO statement.

The range of a DO statement must not end with a GO TO type statement or a numerical IF statement. If an assigned GO TO statement is in the range of a DO loop, all the statements to which it may transfer must be either in the range of the DO loop or all must be outside the range. A logical IF statement is allowed as the last statement of the range. In this case, control is transferred as follows. The range is considered ended when, and if, control would normally pass to the statement following the entire logical IF statement.

As an example, consider the sequences:

```
  DO 5 K = 1,4
5 IF(X(K).GT.Y(K))Y(K) = X(K)
6 ...
```

Statement 5 is executed four times whether the statement Y(K) = X(K) is executed or not. Statement 6 is not executed until statement 5 has been executed four times.

Examples:
```
DO 22 L = 1,30
DO 45 K = 2,LIMIT,-3
DO 7 X = T,MAX,L
```

4.4 CONTINUE STATEMENT

The CONTINUE statement has the form:

```
CONTINUE
```

This statement is a dummy statement, used primarily as a target for transfers, particularly as the last statement in the range of a DO statement. For example, in the sequence:

```
      DO 7 K = START,END
        :
        :
      IF (X(K))22,13,7
        :
7     CONTINUE
```

a positive value of X(K) begins another execution of the range. The CONTINUE provides a target address for the IF statement and ends the range of the DO statement.

4.5 PAUSE STATEMENT

The PAUSE statement enables the program to incorporate operator activity into the sequence of automatic events. The PAUSE statement assumes one of three forms:

```
PAUSE
PAUSE n
PAUSE 'xxxxx'
```

where n is an unsigned string of six or less octal digits, and 'xxxxx' is a literal message.

Execution of the PAUSE statement causes the message or the octal digits, if any, to be typed on the user's teletypewriter. Program execution may be resumed (at the next executable FORTRAN statement) from the console by typing "G," followed by a carriage return. Program execution may be terminated by typing "X," followed by carriage return.

Example:
```
PAUSE 167
PAUSE 'NOW IS THE TIME'
```

4.6 STOP STATEMENT

The STOP statement has the forms:

```
STOP      or
STOP n
```

where n is an unsigned string of one to five octal digits.

The STOP statement terminates the program and returns control to the monitor system. (Termination of a program may also be accomplished by a CALL to the EXIT or DUMP subroutines.) Use of the STOP statement implies a call to the EXIT subroutine.

4.7 END STATEMENT

The END statement has the form:

```
END
```

The END statement informs the compiler to terminate compilation and must be the physically last statement of the program. The END statement implies a STOP statement in a main program or a RETURN statement in a subroutine or a function. The END statement is implied by an end-of-file.

CHAPTER 5

DATA TRANSMISSION STATEMENTS

Data transmission statements are used to control the transfer of data between computer memory and either peripheral devices or other locations in computer memory. These statements are also used to specify the format of the output data. Data transmission statements are divided into the following four categories.

a. Nonexecutable statements that enable conversions between internal form data within core memory and external form data (FORMAT), or specify lists of arrays and variables for input/output transfer (NAMELIST).

b. Statements that specify transmission of data between computer memory and I/O devices: READ, WRITE, PRINT, PUNCH, TYPE, ACCEPT.

c. Statements that control magnetic tape unit mechanisms: REWIND, BACKSPACE, END FILE, UNLOAD, SKIP RECORD.

d. Statements that specify transmission of data between series of locations in memory: ENCODE, DECODE.

5.1 NONEXECUTABLE STATEMENTS

The FORMAT statement enables the user to specify the form and arrangement of data on the selected external medium. The NAMELIST statement provides for conversion and input/output transmission of data without reference to a FORMAT statement.

5.1.1 FORMAT Statement

FORMAT statements may be used with any appropriate input/output medium or ENCODE/DECODE statement. FORMAT statements are of the form:

$$n\ \text{FORMAT}\ (S_1,S_2,\ldots S_n/S_1^1,S_2^1,\ldots,S_n^1/\ldots)$$

where n is a statement number, and each S is a data field specification.

FORMAT statements may be placed anywhere in the source program. Unless the FORMAT statement contains only alphanumeric data for direct input/output transmission, it will be used in conjunction with the list of a data transmission statement.

Slashes are used to specify unit records, which must be one of the following:

a. A tape or disk record with a maximum length corresponding to a line buffer (135 ASCII characters).

b. A punched card with a maximum of 80 characters.

c. A printed line with a maximum of 72 characters for a Teletype®and either 120 or 132 characters for the line printer.

During transmission of data, the object program scans the designated FORMAT statement. If a specification for a numeric field is present (see Section 5.2.1 of this chapter) and the data transmission statement contains items remaining to be transmitted, transmission takes place according to the specifications. This process ceases and execution of the data transmission statement is terminated as soon as all specified items have been transmitted. Thus, the FORMAT statement may contain specifications for more items than are specified by the data transmission statement. Conversely, the FORMAT statement may contain specifications for fewer items than are specified by the data transmission statement.

The following types of field specifications may appear in a FORMAT statement: numeric, numeric with scale factors, logical, alphanumeric. The FORMAT statement also provides for handling multiple record formats, formats stored as data, carriage control, skipping characters, blank insertion, and repetition. If an input list requires more characters than the input device supplies for a given unit record, blanks are supplied.

5.1.1.1 Numeric Fields - Numeric field specification codes designate the type of conversion to be performed. These codes and the corresponding internal and external forms of the numbers are listed in Table 5-2.

The conversions are specified by the forms:

1.	Dw.d	
2.	Ew.d	
3.	Fw.d	
4.	Iw	
5.	Ow	
6.	Gw.d	(for real or double precision)
	Gw	(for integer or logical)
	Gw.d,Gw.d	(for complex)

respectively. The letter D, E, F, I, O, or G designates the conversion type; w is an integer specifying the field width, which may be greater than required to provide for blank columns between numbers; d is an integer specifying the number of decimal places to the right of the decimal point or, for G conversion, the number of significant digits. (For D, E, F, and G input, the position of the decimal point in the external field takes precedence over the value of d in the format.)

® Teletype is a registered trademark of Teletype Corporation.

For example,

FORMAT (I5,F10.2,D18.10)

could be used to output the line,

bbb32bbbb−17.60bbb.5962547681D+03

on the output listing.

The G format is the general format code that is used to transmit real, double precision, integer, logical, or complex data. The rules for input depend on the type specification of the corresponding variable in the data list. The form of the output conversion also depends on the individual variable except in the case of real and double-precision data. In these cases the form of the output conversion is a function of the magnitude of the data being converted. The following table shows the magnitude of the external data, M, and the resulting method of conversion.

Table 5-1
Magnitude of Internal Data

Magnitude of Data	Resulting Conversion
$0.1 \leq M < 1$	F(w-4).d, 4x
$1 \leq M < 10$	F(w-4).(d-1), 4x
.	.
.	.
$10^{d-2} \leq M < 10^{d-1}$	F(w-4). 1, 4x
$10^{d-1} \leq M < 10^{d}$	F(w-4). 0, 4x
All others	Ew.d

The field width w should always be large enough to include spaces for the decimal point, sign, and exponent. In all numeric field conversions if w is not large enough to accommodate the converted number, the excess digits on the left will be lost; if the number is less than w spaces in length, the number is right-adjusted in the field.

Table 5-2
Numeric Field Codes

Conversion Code	Internal Form	External Form
D	Binary floating point double-precision	Decimal floating point with D exponent
E	Binary floating point	Decimal floating point with E exponent
F	Binary floating point	Decimal fixed point
I	Binary integer	Decimal integer
O	Binary integer	Octal integer
G	One of the following: single precision binary floating point, binary integer, binary logical, or binary complex	Single precision decimal floating point integer, logical (T or F), or complex (two decimal floating point numbers), depending upon the internal form

5.1.1.2 Numeric Fields with Scale Factors - Scale factors may be specified for D, E, F, and G conversions. A scale factor is written nP where P is the identifying character and n is a signed or unsigned integer that specifies the scale factor.

For F type conversions (or G type, if the external field is decimal fixed point), the scale factor specifies a power of ten so that

$$\text{external number} = (\text{internal number}) * 10^{(\text{scale factor})}$$

For D, E, and G (external field not decimal fixed point) conversions, the scale factor multiplies the number by a power of ten, but the exponent is changed accordingly leaving the number unchanged except in form. For example, if the statement:

```
FORMAT (F8.3,E16.5)
```

corresponds to the line

```
bb26.451bbbb-0.41321E-01
```

then the statement

```
FORMAT (-1PF8.3,2PE16.5)
```

might correspond to the line

bbb2.645bbb−41.32157E−03

In input operations, F type (and G type, if the external field is decimal fixed point) conversions are the only types affected by scale factors.

When no scale factor is specified, it is understood to be zero. However, once a scale factor is specified, it holds for all subsequent D, E, F, and G type conversions within the same format unless another scale factor is encountered. The scale factor is reset to zero by specifying a scale factor of zero. Scale factors have no effect on I and O type conversions.

5.1.1.3 Logical Fields - Logical data can be transmitted in a manner similar to numeric data by use of the specification:

Lw

where L is the control character and w is an integer specifying the field width. The data is transmitted as the value of a logical variable in the input/output list.

If on input, the first nonblank character in the data field is T or F, the value of the logical variable will be stored as true or false, respectively. If the entire data field is blank or empty, a value of false will be stored.

On output, w minus 1 blanks followed by T or F will be output if the value of the logical variable is true or false, respectively.

5.1.1.4 Variable Field Width - The D, E, F, G, I, and O conversion types may appear in a FORMAT statement without the specification of the field width (w) or the number of places after the decimal point (d). In the case of input, omitting the w implies that the numeric field is delimited by any character which would otherwise be illegal in the field, in addition to the characters -, +, ., E, D, and blank provided they follow the numeric field. For example, input according to the format

10 FORMAT(2I,F,E,O)

might appear on the input medium as

-10,3/15.621-.0016E-10,777.

In this case, commas delimit the numeric fields, blanks may also be used as field delimiters. On output, omitting the w has the following effect:

Format	Becomes
D	D25.16
E	E15.7
F	F15.7
G	G15.7 or G25.16
I	I15
O	O15

5.1.1.5 Alphanumeric Fields - Alphanumeric data can be transmitted in a manner similar to numeric data by use of the form Aw, where A is the control character and w is the number of characters in the field. The alphanumeric characters are transmitted as the value of a variable in an input/output list. The variable may be of any type. For the sequence:

```
  READ 5,V
5 FORMAT (A4)
```

causes four characters to be read and placed in memory as the value of the variable V.

Although w may have any value, the number of characters transmitted is limited by the maximum number of characters which can be stored in the space allotted for the variable. This maximum depends upon the variable type. For a double precision variable the maximum is ten characters; for all other variables, the maximum is five characters. If w exceeds the maximum, the leftmost characters are lost on input and replaced with blanks on output. If, on input, w is less than the maximum, blanks are filled in to the right of the given characters until the maximum is reached. If, on output, w is less than the maximum, the leftmost w characters are transmitted to the external medium. Since for complex variables each word requires a separate field specification, the maximum value for w is 5. For example,

```
  COMPLEX C
  ACCEPT 1, C
1 FORMAT (2A5)
```

could be used to transmit ten alphanumeric characters into complex variable C.

5.1.1.6 Alphanumeric Data Within Format Statements - Alphanumeric data may be transmitted directly into or from the format statement by two different methods: H-conversion, or the use of single quotes.

In H-conversion, the alphanumeric string is specified by the form nH. H is the control character and n is the number of characters in the string counting blanks. For example, the format in the statement below can be used to print PROGRAM COMPLETE on the output listing.

```
FORMAT (17H PROGRAM COMPLETE)
```

The statement

```
FORMAT (16HPROGRAM COMPLETE)
```

causes ROGRAM COMPLETE to be printed.

Referring to this format in a READ statement would cause the 17 characters to be replaced with a new string of characters.

The same effect is achieved by merely enclosing the alphanumeric data in quotes. The result is the same as in H-conversion; on input, the characters between the quotes are replaced by input characters, and, on output, the characters between the quotes (including blanks) are written as part of the output data. A quote character within the data is represented by two successive quote marks. For example, referring to:

```
FORMAT (' DON''T')
```

with an output statement would cause DON'T to be printed. Referring to

```
FORMAT ('DON''T')
```

causes ON'T to be printed. The first character referenced by the FORMAT statement for output is interpreted as a carriage control character (see 5.1.1.13). TAB characters in FORMAT statements are converted to single blanks at runtime by the FORTRAN operating system.

5.1.1.7 Mixed Fields - An alphanumeric format field may be placed among other fields of the format. For example, the statement:

```
FORMAT (I5,7H FORCE=F10.5)
```

can be used to output the line:

```
bbb22bFORCE=bb17.68901
```

The separating comma may be omitted after an alphanumeric format field, as shown above.

5.1.1.8 Complex Fields - Complex quantities are transmitted as two independent real quantities. The format specification consists of two successive real specifications or one repeated real specification. For instance, the statement:

```
FORMAT (2E15.4,2(F8.3,F8.5))
```

could be used in the transmission of three complex quantities.

5.1.1.9 Repetition of Field Specifications - Repetition of a field specification may be specified by preceding the control character D, E, F, I, O, G, L, or A by an unsigned integer giving the number of repetitions desired. For example:

```
FORMAT (2E12.4,3I5)
```

is equivalent to:

```
FORMAT (E12.4,E12.4,I5,I5,I5)
```

5.1.1.10 Repetition of Groups - A group of field specifications may be repeated by enclosing the group in parentheses and preceding the whole with the repetition number. For example:

```
FORMAT (2I8,2(E15.5,2F8.3))
```

is equivalent to:

```
FORMAT (2I8,E15.5,2F8.3,E15.5,2F8.3)
```

5.1.1.11 Multiple Record Formats - To handle a group of input/output records where different records have different field specifications, a slash is used to indicate a new record. For example, the statement:

```
FORMAT (3O8/I5,2F8.4)
```

is equivalent to

```
FORMAT (3O8)
```

for the first record and

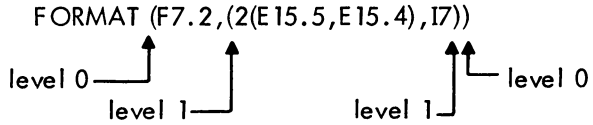
```
FORMAT (I5,2F8.4)
```

for the second record.

The separating comma may be omitted when a slash is used. When n slashes appear at the end or beginning of a format, n blank records may be written on output or records skipped on input. When n slashes appear in the middle of a format, n-1 blank records are written or n-1 records skipped.

Both the slash and the closing parenthesis at the end of the format indicate the termination of a record. If the list of an input/output statement dictates that transmission of data is to continue after the closing parenthesis of the format is reached, the format is repeated starting with that group repeat specification terminated by the last right parenthesis of level one or level zero if no level one group exists.

Thus, the statement



causes the format

F7.2,2(E15.5,E15.4),I7

to be used on the first record, and the format

2(E15.5,E15.4),I7

to be used on succeeding records.

As a further example, consider the statement

FORMAT (F7.2/(2(E15.5,E15.4),I7))

The first record has the format

F7.2

and successive records have the format

2(E15.5,E15.4),I7

5.1.1.12 Formats Stored as Data - The ASCII character string comprising a format specification may be stored as the values of an array. Input/output statements may refer to the format by giving the array name, rather than the statement number of a FORMAT statement. The stored format has the same form as a FORMAT statement excluding the word "FORMAT." The enclosing parentheses are included.

As an example, consider the sequence:

```
      DIMENSION SKELETON (2)
      READ 1, (SKELETON(I), I = 1,2)
    1 FORMAT (2A4)
      READ SKELETON,K,X
```

The first READ statement enters the ASCII string into the array SKELETON. In the second READ statement, SKELETON is referred to as the format governing conversion of K and X.

5.1.1.13 Carriage Control - The first character of each ASCII record controls the spacing of the line printer or Teletype. This character is usually set by beginning a FORMAT statement for an ASCII record with 1Ha, where a is the desired control character. The line spacing actions, listed below, occur before printing:

FORTRAN Character	Printer Character	Octal Value	Effect	Printer Channel
space	LF	012	Skip to next line with form feed after 60 lines	8
0 zero	LF,LF	012	Skip a line	8
1 one	FF	014	Form feed - go to top of next page	1
+ plus			Suppress skipping - overprint the line	
* asterisk	DC3	023	Skip to next line with no form feed	5
- minus	LF,LF,LF	012	Skip two lines	8
2 two	DLE	020	Space 1/2 of a page	2
3 three	VT	013	Space 1/3 of a page	7
/ slash	DC4	024	Space 1/6 of a page	6
. period	DC2	022	Triple space with a form feed after every 20 lines printed	4
, comma	DC1	021	Double space with a form feed after every 30 lines printed	3

NOTE: Printer control characters DLE, DC1, DC2, DC3, and DC4 affect only the line printer.

A $ (dollar sign) as a format field specification code suppresses the carriage return at the end of the Teletype or line printer line.

5.1.1.14 Spacing - Input and output can be made to begin at any position within a FORTRAN record by use of the format code

Tw

where T is the control character and w is an unsigned integer constant specifying the character position in a FORTRAN record where the transfer of data is to begin. When the output is printed, w corresponds to the (w-1)th print position. This is because the first character of the output buffer is a carriage control character and is not printed. It is recommended that the first field specification of the output format be 1x, except where a carriage control character is used.

For example,

```
2 FORMAT (T50, 'BLACK'T30, 'WHITE')
```

would cause the following line to be printed

```
Print Position 29        Print Position 49
        ↓                        ↓
      WHITE                    BLACK
```

For input, the statement

```
1 FORMAT(T35,'MONTH')
  READ (3,1)
```

cause the first 34 characters of the input data to be skipped, and the next 5 characters would replace the characters M, O, N, T, and H in storage. If an input record containing

```
ABCbbbXYZ
```

is read with the format specification

```
10 FORMAT (T7,A3,T1,A3)
```

then the characters XYZ and ABC are read, in that order.

5.1.1.15 Blank or Skip Fields - Blanks may be introduced into an output record or characters skipped on an input record by use of the specification nX. The control character is X; n is the number of blanks or characters skipped and must be greater than zero. For example, the statement

```
FORMAT (5H STEPI5, 10X2HY=F7.3)
```

may be used to output the line

```
bSTEPbbb28bbbbbbbbbbY=b-3.872
```

5.1.2 NAMELIST Statement

The NAMELIST statement, when used in conjunction with special forms of the READ and WRITE statements, provides a method for transmitting and converting data without using a FORMAT statement or an I/O list. The NAMELIST statement has the form

$$NAMELIST/X_1/A_1,A_2,\ldots,A_i/X_2/B_1,B_2,\ldots,B_i\ldots/X_m/C_1,C_2,\ldots C_n$$

where the X's are NAMELIST names, and the A's, B's, and C's are variable or array names.

Each list or variable mentioned in the NAMELIST statement is given the NAMELIST name immediately preceding the list. Thereafter, an I/O statement may refer to an entire list by mentioning its NAMELIST name. For example:

```
NAMELIST/FRED/A,B,C/MARTHA/D,E
```

states that A, B, and C belong to the NAMELIST name FRED, and D and E belong to MARTHA.

The use of NAMELIST statements must obey the following rules:

a. A NAMELIST name may not be longer than six characters; it must start with an alphabetic character; it must be enclosed in slashes; it must precede the list of entries to which it refers; and it must be unique within the program.

b. A NAMELIST name may be defined only once and must be defined by a NAMELIST statement. After a NAMELIST name has been defined, it may only appear in READ or WRITE statements. The NAMELIST name must be defined in advance of the READ or WRITE statement.

c. A variable used in a NAMELIST statement cannot be used as a dummy argument in a subroutine definition.

d. Any dimensioned variable contained in NAMELIST statement must have been defined in a DIMENSION statement preceding the NAMELIST statement.

5.1.2.1 Input Data For NAMELIST Statements - When a READ statement refers to a NAMELIST name, the first character of all input records is ignored. Records are searched until one is found with a $ or & as the second character immediately followed by the NAMELIST name specified. Data is then converted and placed in memory until the end of a data group is signaled by a $ or & either in the same record as the NAMELIST name, or in any succeeding record as long as the $ or & is the second character of the record. Data items must be separated by commas and be of the following form:

$$V=K_1,K_2,\ldots,K_n$$

where V may be a variable name or an array name, with or without subscripts. The K's are constants which may be integer, real, double precision, complex (written as (A, B) where A and B are real), or logical (written as T for true and F for false). A series of J identical constants may be represented by J*K where J is an unsigned integer and K is the repeated constant. Logical and complex constants must be equated to logical and complex variables, respectively. The other types of constants (real, double precision, and integers) may be equated to

any other type of variable (except logical or complex), and will be converted to the variable type. For example, assume A is a two-dimensional real array, B is a one-dimensional integer array, C is an integer variable, and that the input data is as follows:

```
$FRED A(7,2)=4, B=3,6*2.8, C=3.32$
↑
Column 2
```

A READ statement referring to the NAMELIST name FRED will result in the following: the integer 4 will be converted to floating point and placed in A(7,2). The integer 3 will be placed in B(1) and the floating point number 2.8 will be placed in B(2), B(3),..., B(7). The floating point number 3.32 will be converted to the integer 3 and placed in C.

5.1.2.2 Output Data For NAMELIST Statements - When a WRITE statement refers to a NAMELIST name, all variables and arrays and their values belonging to the NAMELIST name will be written out, each according to its type. The complete array is written out by columns. The output data will be written so that:

a. The fields for the data will be large enough to contain all the significant digits.
b. The output can be read by an input statement referencing the NAMELIST name.

For example, if JOE is a 2x3 array, the statement

```
NAMELIST/NAM1/JOE,K1,ALPHA
WRITE (u,NAM1)
```

generate the following form of output.

```
Column 2
↓
$NAM1
JOE = -6.75,          .234E-04,              68.0,
      -17.8,             0.0,          -.197E+07,
  K1 = 73.1,          ALPHA=3,$
```

5.2 DATA TRANSMISSION STATEMENTS

The data transmission statements accomplish input/output transfer of data that may be listed in a NAMELIST statement or defined in a FORMAT statement. When a FORMAT statement is used to specify formats, the data transmission statement must contain a list of the quantities to be transmitted. The data appears on the external media in the form of records.

5.2.1 Input/Output Lists

The list of an input/output statement specifies the order of transmission of the variable values. During input, the new values of listed variables may be used in subscript or control expressions for variables appearing later in the list. For example:

```
READ 13,L,A(L),B(L+1)
```

reads a new value of L and uses this value in the subscripts of A and B.

The transmission of array variables may be controlled by indexing similar to that used in the DO statement. The list of controlled variables, followed by the index control, is enclosed in parentheses. For example,

```
READ 7, (X(K),K=1,4),A
```

is equivalent to:

```
READ 7, X(1),X(2),X(3),X(4),A
```

As in the DO statement, the initial, limit, and increment values may be given as integer expressions:

```
READ 5, N, (GAIN(K),K=1,M/2,N)
```

The indexing may be compounded as in the following:

```
READ 11, ((MASS(K,L),K=1,4),L=1,5)
```

The above statement reads in the elements of array MASS in the following order:

```
MASS(1,1), MASS(2,1),...,MASS(4,1),MASS(1,2),...,MASS(4,5)
```

If an entire array is to be transmitted, the indexing may be omitted and only the array identifier written. The array is transmitted in order of increasing subscripts with the first subscript varying most rapidly. Thus, the example above could have been written:

```
READ 11, MASS
```

Entire arrays may also be designated for transmission by referring to a NAMELIST name (see description of NAMELIST statement).

5.2.2 Input/Output Records

All information appearing on external media is grouped into records. The maximum amount of information in one record and the manner of separation between records depends upon the medium. For punched cards, each card constitutes one record; on a teletypewriter a record is one line, and so forth. The amount of information contained in each ASCII record is specified by the FORMAT reference and the I/O list. For magnetic tape binary records, the amount of information is specified by the I/O list.

Each execution of an input or output statement initiates the transmission of a new data record. Thus, the statement

```
READ 2, FIRST,SECOND,THIRD
```

is not necessarily equivalent to the statements

```
READ 2, FIRST
READ 2, SECOND
READ 2, THIRD
```

since, in the second case, at least three separate records are required, whereas, the single statement

```
READ 2, FIRST,SECOND,THIRD
```

may require one, two, three, or more records depending upon FORMAT statement 2.

If an input/output statement requests less than a full record of information, the unrequested part of the record is lost and cannot be recovered by another input/output statement without repositioning the record.

If an input/output list requires more than one ASCII record of information, successive records are read.

5.2.3 PRINT Statement

The PRINT statement assumes one of two forms

```
PRINT f, list
PRINT f
```

where f is a format reference.

The data is converted from internal to external form according to the designated format. If the data to be transmitted is contained in the specified FORMAT statement, the second form of the statement is used.

Examples:
```
PRINT 16,T,(B(K),K=1,M)
PRINT F106,SPEED,MISS
```

In the second example, the format is stored in array F106.

5.2.4 PUNCH Statement

The PUNCH statement assumes one of two forms

```
PUNCH f, list
PUNCH f
```

where f is a format reference.

Conversion from internal to external data forms is specified by the format reference. If the data to be transmitted is contained in the designated FORMAT statement, the second form of the statement is used.

Examples:
```
PUNCH 12,A,B(A),C(B(A))
PUNCH 7
```

5.2.5 TYPE Statement

The TYPE statement assumes one of two forms

```
TYPE f, list
TYPE f
```

where f is a format reference.

This statement causes the values of the variables in the list to be read from memory and listed on the user's teletypewriter. The data is converted from internal to external form according to the designated format. If the data to be transmitted is contained in the designated FORMAT statement, the second form of the statement is used.

Examples:
```
TYPE 14,K,(A(L),L=1,K)
TYPE FMT
```

5.2.6 WRITE Statement

The WRITE statement assumes one of the following forms

```
WRITE (u,f) list
WRITE(u,f)
WRITE(u,N)
WRITE(u) list
WRITE(u#R,f) list
```

where u is a unit designation, f is a format reference, N is a NAMELIST name, and R is a record number where I/O is to start.

The first form of the WRITE statement causes the values of the variables in the list to be read from memory and written on the unit designated in ASCII form. The data is converted to external form as specified by the designated FORMAT statement.

The second form of the WRITE statement causes information to be read directly from the specified format and written on the unit designated in ASCII form.

The third form of the WRITE statement causes the names and values of all variables and arrays belonging to the NAMELIST name, N, to be read from memory and written on the unit designated. The data is converted to external form according to the type of each variable and array.

The fourth form of the WRITE statement causes the values of the variables in the list to be read from memory and written on the unit designated in binary form.

The fifth form of the WRITE statement causes the variables in the list to be written in the specified record of the file on the disk unit designated. Either a pound sign (#) or a single quote (') can be used to separate the unit and the record. This allows a programmer to access fixed-length records directly, and eliminates the sequential writing of data to access one or more records within the file. The file must first be defined properly by a CALL to DEFINE FILE (see Section 12.4). Output begins when the random WRITE specifying the record to which the writing is desired is given in the correct format.

5.2.7 READ Statement

The READ statement assumes one of the following forms:

```
READ f, list
READ f
READ(u,f) list
READ(u,f)
READ(u,N)
READ(u)list
READ(u#R,f) list
READ(u,f,END=C, ERR=d) list
READ(u,f,END=C) list
READ(u,f, ERR=d) list
```

where f is a format reference, u is a unit designation, N is a NAMELIST name, R is a record number where I/O is to start, C is a statement number to which control is transferred upon encountering an end-of-file, and d is the statement number to which control is transferred upon encountering an error condition on the input data.

The first form of the READ statement causes information to be read from cards and put in memory as values of the variables in the list. The data is converted from external to internal form as specified by the referenced FORMAT statement.

Example: READ 28,Z1,Z2,Z3

The second form of the READ statement is used if the data read from cards is to be transmitted directly into the specified format.

Example: READ 10

The third form of the READ statement causes ASCII information to be read from the unit designated and stored in memory as values of the variables in the list. The data is converted to internal form as specified by the referenced FORMAT statement.

Example: READ(1,15)ETA,P1

The fourth form of the READ statement causes ASCII information to be read from the unit designated and transmitted directly into the specified format.

Example: READ(N,105)

The fifth form of the READ statement causes data of the form described in the discussion of input data for NAMELIST statements to be read from the unit designated and stored in memory as values of the variables or arrays specified.

Example: READ(2,FRED)

The sixth form of the READ statement causes binary information to be read from the unit designated and stored in memory as values of the variables in the list.

Example: READ(M)GAIN,Z,AI

The seventh form of the READ statement causes information to be read from the specified record in a disk file into the variables of the list. This allows random access of fixed-length records in a disk file. The file from which records are to be read is defined by the DEFINE FILE call (see Section 12.4).

```
Example:    DOUBLE PRECISION FIL
            DIMENSION A(6)
            DATA FIL/'FILE.ONE'/
            CALL DEFINE FILE (4,30,NV,FIL,"11,"23)
            READ (4#54,5)A
```

This example reads the 54th record from FILE.ONE on the disk area belonging to programmer [11,23] into the list variables A(1) through A(6).

The eighth form of the READ statement causes control to be transferred if an end-of-file or error condition is encountered on the input data. The arguments END=c and ERR=d are optional and if both are included, either may appear first. If an end-of-file is encountered, control transfers to the statement specified by END=c. If an END parameter is not specified, I/O on that device terminates and the program halts with an error message to the user's TTY. If an error on input is encountered, control transfers to the statement specified by ERR=d. If an ERR=d parameter is not specified, the program halts with an error message to the user's TTY.

```
Example:    READ (7,7,END=888, ERR=999) A
                .
                .
                .
888         (control transfers here if an end-of-file is encountered)
                .
                .
                .
999         (control transfers here if an error on input is encountered)
```

5.2.8 REREAD Statement

The reread feature allows a FORTRAN program to reread information from the last used input file. The format used during the reread need not correspond to the original read format, and the information may be read as many times as desired.

a. To reread from an input device, the following coding would be used:

```
READ (16,100)A
      .
      .
      .
REREAD 105,A
```

The REREAD 105,A statement causes the last input device used to be reread according to format statement 105. The original read format and a subsequent reread format need not be the same.

b. The reread feature cannot be used until an input from a file has been accomplished. If the feature is used prematurely, an error message will be generated.

c. Information may be reread as many times as desired using either the same or a new format statement each time.

d. The reread feature must be used with some forethought and care since it rereads from the last input file used, i.e.:

The following example will reread from the file on Device No. 10, not Device No. 16:

```
READ (16,100)A
      .
      .
      .
READ (10,200)B
      .
      .
      .
REREAD 110,A
```

5.2.9 ACCEPT Statement

The ACCEPT statement assumes one of two forms

```
ACCEPT f, list
ACCEPT f
```

where f is a format reference.

This statement causes information to be input from the user's teletypewriter and put in memory as values of the variables in the list. The data is converted to internal form as specified by the format. If the transmission of data is directly into the designated format, the second form of the statement is used.

Examples:
```
ACCEPT 12,ALPHA,BETA
ACCEPT 27
```

5.3 DEVICE CONTROL STATEMENTS

Device control statements and their corresponding effects are listed in Table 5-3.
See Table 12-4 for more details

Table 5-3
Device Control Statements

Statement	Effect
BACKSPACE u	Backspaces designated ~~tape~~ unit one ASCII record or one logical binary record.
END FILE u	Writes an end-of-file.
REWIND u	Rewinds tape on designated unit.
SKIP RECORD u	Causes skipping of one ASCII record or one logical binary record.
UNLOAD u	Rewinds and unloads the designated ~~tape~~ unit.
ERASE u	writes 3 inches of blank tape on the designated unit.

5.4 ENCODE AND DECODE STATEMENTS

ENCODE and DECODE statements transfer data, according to format specifications, from one section of user's core to another. No peripheral equipment is involved. DECODE is used to change data in ASCII format to data in another format. ENCODE changes data of another format into data in ASCII format.

The two statements are of the form

```
ENCODE(c,f,v),L(1),...,L(N)
DECODE(c,f,v),L(1),...,L(N)
```

where

c = the number of ASCII characters
f = the format statement number
v = the starting address of the ASCII record referenced
L(1),...,L(N) = the list of variables.

A slash cannot appear in the FORMAT statement referenced by an ENCODE or DECODE statement.

Example: Assume the contents of the variables to be as follows:

A(1) contains the floating-point binary number 300.45

A(2) contains the floating-point binary number 3.0

J contains the binary integer value 1.

B is a four-word array of indeterminate contents

C contains the ASCII string 12345

```
      DO 2 J = 1,2
      ENCODE (16,10,B) J, A(J)
10    FORMAT (1X,2HA(,I1,4H) = ,F8.2)
      TYPE 11,B
11    FORMAT (4A5)
 2    CONTINUE
      DECODE (4, 12, C) B
12    FORMAT (3F1.0,1X,F1.0)
      TYPE 13,B
13    FORMAT (4F5.2)
      END
```

Array B can contain 20 ASCII characters. The result of the ENCODE statement after the first iteration of the DO loop is:

B(1)	A(1)
B(2)	=
B(3)	300.4
B(4)	5

Typed as

A(1) = 300.45

The result after the second iteration is:

B(1)	A(2)
B(2)	=
B(3)	3.0
B(4)	

Typed as

A(2) = 3.0

The result of the DECODE statement is to extract the digits 1, 2, and 3 from C and convert them to floating-point binary values and store them in B(1), B(2), and B(3). Then skip the next character (4) and extract the digit 5 from C, convert it to a floating-point binary value, and store it in B(4).

CHAPTER 6
SPECIFICATION STATEMENTS

Specification statements allocate storage and furnish information about variables and constants to the compiler. Specification statements may be divided into three categories, as follows:

a. Storage specification statements: DIMENSION, COMMON, and EQUIVALENCE.

b. Data specification statements: DATA and BLOCK DATA.

c. Type declaration statements: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, SUBSCRIPT INTEGER, and IMPLICIT.

By extending the USA Standard in regard to specification statements, PDP-10 FORTRAN IV allows the following statements to be used anywhere in the program, provided that the variables they specify appear in executable statements only after the particular specification statement. The specification statement must not appear in the range of a DO loop.

DIMENSION statement
EXTERNAL statement (described in Chapter 7)
COMMON statement
EQUIVALENCE statement
Type declaration statements
DATA statement

A sample program that incorporates these statements follows.

```
        DOUBLE PRECISION D
        DIMENSION Y(10), D(5)
        Y(1) = -1.0
        INTEGER XX(5)
        Y(2) = ABS(Y(1))
        DATA XX/1,2,3,4,5
        DO 10 I = 3,7
10      Y(I) = XX(I-2)
        COMMON Z
        Z=Y(1)*Y(2)/(Y(3) + Y(5))
        END
```

Only IMPLICIT statements and arithmetic function definition statements (described in Chapter 7) must appear in the program before any executable statement.

In addition, arrays must be dimensional before being referenced in a NAMELIST, EQUIVALENCE, or DATA statement. DOUBLE PRECISION and COMPLEX arrays must be declared before they are dimensioned.

6.1 STORAGE SPECIFICATION STATEMENTS

6.1.1 DIMENSION Statement

The DIMENSION statement is used to declare identifiers to be array identifiers and to specify the number and bounds of the array subscripts. The information supplied in a DIMENSION statement is required for the allocation of memory for arrays. Any number of arrays may be declared in a single DIMENSION statement. The DIMENSION statement has the form

DIMENSION $S_1, S_2, \ldots, S_n$

where S is an array specification.

Each array variable appearing in the program must represent an element of an array declared in a DIMENSION statement, unless the dimension information is given in a COMMON or TYPE statement. Dimension information may appear only once for a given variable.

Each array specification gives the array identifier and the minimum and maximum values which each of its subscripts may assume in the following form:

identifier(min/max,min/max,...,min/max)

The minima and maxima must be integers. The minimum must not exceed the maximum. For example, the statement

DIMENSION EDGE(-1/1,4/8)

specifies EDGE to be a two-dimensional array whose first subscript may vary from -1 to 1 inclusive, and the second from 4 to 8 inclusive.

Minimum values of 1 may be omitted. For example,

NET(5, 10)

is interpreted as:

NET(1/5,1/10)

Examples:
```
DIMENSION FORCE(-1/1,0/3,2,2,-7/3)
DIMENSION PLACE(3,3,3),JI(2,2/4),K(256)
```

Arrays may also be declared in the COMMON or type declaration statements in the same way:

```
COMMON X(10,4),Y,Z
INTEGER A(7,32),B
DOUBLE PRECISION K(-2/6,10)
```

6.1.1.1 Adjustable Dimensions - Within either a FUNCTION or SUBROUTINE subprogram, DIMENSION and TYPE statements may use integer variables in an array specification, provided that the array name and variable dimensions are dummy arguments of the subprogram. The actual array name and values for the dummy variables are given by the calling program when the subprogram is called. The variable dimensions may not be altered within the subprogram (i.e., typing the array DOUBLE PRECISION or COMPLEX after it has been dimensioned) and must be less than or equal to the explicit dimensions declared in the calling program.

Example:
```
      SUBROUTINE SBR(ARRAY,M1,M2,M3,M4)
      DIMENSION ARRAY (M1/M2,M3/M4)
                  .
                  .
                  .
      DO 27 L=M3,M4
      DO 27 K=M1,M2
                  .
                  .
                  .
27    ARRAY(K,L)=VALUE
                  .
                  .
                  .
      END
```

The calling program for SBR might be:

```
DIMENSION A1(10,20),A2(1000,4)
            .
            .
            .
CALL SBR(A1,5,10,10,20)
            .
            .
            .
CALL SBR(A2,100,250,2,4)
            .
            .
            .
END
```

6.1.2 COMMON Statement

The COMMON statement causes specified variables or arrays to be stored in an area available to other programs. By means of COMMON statements, the data of a main program and/or the data of its subprograms may share a common storage area.

The common area may be divided into separate blocks which are identified by block names. A block is specified as follows:

```
/block identifier/identifier,identifier,...,identifier
```

The identifier enclosed in slashes is the block name. The identifiers which follow are the names of the variables or arrays assigned to the block and are placed in the block in the order in which they appear in the block specification. A common block may have the same name as a variable in the same program.

The COMMON statement has the general form

```
COMMON/BLOCK1/A,B,C/BLOCK2/D,E,F/...
```

where BLOCK1,BLOCK2,... are the block names, and A,B,C,... are the variables to be assigned to each block. For example, the statement

```
COMMON/R/X,Y,T/C/U,V,W,Z
```

indicates that the elements X,Y, and T are to be placed in block R in that order, and that U,V,W, and Z are to be placed in block C.

Block entries are linked sequentially throughout the program, beginning with the first COMMON statement. For example, the statements

```
COMMON/D/ALPHA/R/A,B/C/S
COMMON/C/X,Y/R/U,V,W
```

have the same effect as the statement

```
COMMON/D/ALPHA/R/A,B,U,V,W/C/S,X,Y
```

One block of common storage, referred to as blank common, may be left unlabeled. Blank common is indicated by two consecutive slashes. For example,

```
COMMON/R/X,Y//B,C,D
```

indicates that B, C, and D are placed in blank common. The slashes may be omitted when blank common is the first block of the statement.

```
COMMON B,C,D
```

Storage allocation for blocks of the same name begins at the same location for all programs executed together. For example, if a program contains

```
COMMON A,B/R/X,Y,Z
```

as its first COMMON statement, and a subprogram has

```
COMMON/R/U,V,W//D,E,F
```

as its first COMMON statement, the quantities represented by X and U are stored in the same location. A similar correspondence holds for A and D in blank common.

Common blocks may be any length provided that no program attempts to enlarge a given common block declared by a previously loaded program.

Array names appearing in COMMON statements may have dimension information appended if the arrays are not declared in DIMENSION or type declaration statements. For example,

```
COMMON ALPHA,T(15,10,5),GAMMA
```

specifies the dimensions of the array T while entering T in blank common. Variable dimension array identifiers may not appear in a COMMON statement, nor may other dummy identifiers. Each array name appearing in a COMMON statement must be dimensioned somewhere in the program containing the COMMON statement.

6.1.3 EQUIVALENCE Statement

The EQUIVALENCE statement causes more than one variable within a given program to share the same storage location. The EQUIVALENCE statement has the form

$$\text{EQUIVALENCE}(V_1, V_2, \ldots), (V_k, V_{k+1}, \ldots), \ldots$$

where the V's are variable names.

The inclusion of two or more references in a parenthetical list indicates that the quantities in the list are to share the same memory location. For example,

```
EQUIVALENCE(RED,BLUE)
```

specifies that the variables RED and BLUE are stored in the same location.

The relation of equivalence is transitive; e.g., the two statements,

```
EQUIVALENCE(A,B),(B,C)
EQUIVALENCE(A,B,C)
```

have the same effect.

The subscripts of array variables must be integer constants.

Example: EQUIVALENCE(X,A(3),Y(2,1,4)),(BETA(2,2),ALPHA)

6.1.4 EQUIVALENCE and COMMON

Identifiers may appear in both COMMON and EQUIVALENCE statements provided the following rules are observed.

a. No two quantities in common may be set equivalent to one another.

b. Quantities placed in a common block by means of EQUIVALENCE statements may cause the end of the common block to be extended. For example, the statements

```
COMMON/R/X,Y,Z
DIMENSION A(4)
EQUIVALENCE(A,Y)
```

causes the common block R to extend from X to A(4), arranged as follows:

```
X
Y A(1)      (same location)
Z A(2)      (same location)
  A(3)
  A(4)
```

c. EQUIVALENCE statements which cause extension of the start of a common block are not allowed. For example, the sequence

```
COMMON/R/X,Y,Z
DIMENSION A(4)
EQUIVALENCE(X,A(3))
```

is not permitted, since it would require A(1) and A(2) to extend the starting location of block R.

6.2 DATA SPECIFICATION STATEMENTS

The DATA statement is used to specify initial or constant values for variables. The specified values are compiled into the object program, and become the values assumed by the variables when program execution begins.

6.2.1 DATA Statement

The data to be compiled into the object program is specified in a DATA statement. The DATA statement has the form

DATA list/d_1,d_2,.../,list/d_k,d_{k+1},.../,...

where each list is in the same form as an input/output list, and the d's are data items for each list.

Indexing may be used in a list provided the initial, limit, and increment (if any) are given as constants. Expressions used as subscripts must have the form

$$c_1*i\pm c_2$$

where c_1 and c_2 are integer constants and i is the induction variable. If an entire array is to be defined, only the array identifier need be listed. Variables in COMMON may appear on the lists only if the DATA statement occurs in a BLOCK DATA subprogram. (See Chapter 7, Section 7.6)

The data items following each list correspond one-to-one with the variables of the list. Each item of the data specifies the value given to its corresponding variable with no implied type conversion. Thus, integer variables can only be defined numerically by integer constants, real variables by real constants, double precision variables by double precision constants, and so forth. Refer to Section 2.1 for definitions of the various constants. Data items may be numeric constants, alphanumeric strings, octal constants, or logical constants. For example,

```
DATA ALPHA, BETA/.5, 16.E-2/
```

specifies the value .5 for ALPHA and the value .16 for BETA.

Alphanumeric data is packed into words according to the data word size in the manner of A conversion; however, excess characters are not permitted. The specification is written as nH followed by n characters or is imbedded in single quotes. Double precision variables must have at least six characters assigned to them in DATA statements.

Octal data is specified by the letter O or the character ", followed by a signed or unsigned octal integer of one to twelve digits.

Logical constants are written as .TRUE.,.FALSE., T, or F.

Example:
```
DATA NOTE,K/4HFOOT, O-7712/
DATA QUOTE/'QUOTE'/
```

Any item of the data may be preceded by an integer followed by an asterisk. The integer indicates the number of times the item is to be repeated. For example,

```
DATA(A(K),K=1,20)/61E2, 19*32E1/
```

specifies 20 values for the array A; the value 6100 for A(1); the value 320 for A(2) through A(20). To cause an array or part of an array to be initialized to blanks, the blank areas must be specified explicitly in the DATA statement. For example,

```
DATA(A(I),I=1,10)/'12345',9*' '/
```

causes the first word of A to contain 12345 in ASCII and the next nine words of the array to be blank.

6.2.2 BLOCK DATA Statement

The BLOCK DATA statement has the form:

```
BLOCK DATA
```

This statement declares the program which follows to be a data specification subprogram. Data may be entered into labeled or blank common.

The first statement of the subprogram must be the BLOCK DATA statement. The subprogram may contain only the declarative statements associated with the data being defined.

Example:

```
BLOCK DATA
COMMON/R/S,Y/C/Z,W,V
DIMENSION Y(3)
COMPLEX Z
DATA Y/1E-1,2*3E2/,X,Z/11.877D0,(-1.41421,1.41421)/
END
```

Data may be entered into more than one block of common in one subprogram.

6.3 TYPE DECLARATION STATEMENTS

The type declaration statements INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, IMPLICIT, and SUBSCRIPT INTEGER are used to specify the type of identifiers appearing in a program. An identifier may appear in only one type statement. Type statements may be used to give dimension specifications for arrays.

The explicit type declaration statements have the general form

```
type identifier,identifier,identifier...
```

where type is one of the following:

```
INTEGER,REAL,DOUBLE PRECISION,COMPLEX,LOGICAL,
SUBSCRIPT INTEGER
```

In addition, for the sake of compatibility the following types have been made equivalent:

```
SUBSCRIPT INTEGER is equivalent to INTEGER*2
INTEGER is equivalent to INTEGER*4
REAL is equivalent to REAL*4
DOUBLE PRECISION is equivalent to REAL*8
LOGICAL is equivalent to LOGICAL*1 and LOGICAL*4
COMPLEX is equivalent to COMPLEX*8
```

The listed identifiers are declared by the statement to be of the stated type. Fixed-point variables in a SUBSCRIPT INTEGER statement must fall between -2^{27} and 2^{27}.

6.3.1 IMPLICIT Statement

The IMPLICIT statement has the form

$$\text{IMPLICIT } type_1(a_1,a_2,\ldots),\ldots,type_2(a_3,a_4,\ldots)$$

where type represents INTEGER, REAL, LOGICAL, COMPLEX, DOUBLE PRECISION, or one of the equivalent types listed in Section 6.3, and $a_1a_2,\ldots$ represent single alphabetic characters, each separated by commas, or a range of characters (in alphabetic sequence) denoted by the first and last characters of the range separated by a minus sign (e.g., (A-D)).

This statement causes any program variable which is not mentioned in a type statement, and whose first character is one of those listed in the IMPLICIT statement, to be classified according to the type appearing before the list in which the character appears. As an example, the statement

```
IMPLICIT REAL(A-D,L,N-P)
```

causes all variables starting with the letters A through D,L, and N through P to be typed as real, unless they are explicitly declared otherwise.

The initial state of the compiler is set as if the statement

```
IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
```

were at the beginning of the program. This state is in effect unless an IMPLICIT statement changes the above interpretation; i.e., identifiers, whose types are not explicitly declared, are typed as follows.

a. Identifiers beginning with I, J, K, L, M, or N are assigned interger type.
b. Identifiers not assigned integer type are assigned real type.

If the program contains an IMPLICIT statement, this statement will override throughout the program the implicit state initially set by the compiler. No program may contain more than one IMPLICIT declaration for the same letter.

CHAPTER 7
SUBPROGRAM STATEMENTS

FORTRAN subprograms may be either internal or external. Internal subprograms are defined and may be used only within the program containing the definition. The arithmetic function definition statement is used to define internal functions.

External subprograms are defined separately from (i.e., external to) the programs that call them, and are complete programs which conform to all the rules of FORTRAN programs. They are compiled as closed subroutines; i.e., they appear only once in the object program regardless of the number of times they are used. External subprograms are defined by means of the statements FUNCTION and SUBROUTINE.

7.1 DUMMY IDENTIFIERS

Subprogram definition statements contain dummy identifiers, representing the arguments of the subprogram. They are used as ordinary identifiers within the subprogram definition and indicate the sort of arguments that may appear and how the arguments are used. The dummy identifiers are replaced by the actual arguments when the subprogram is executed.

7.2 LIBRARY SUBPROGRAMS

The standard FORTRAN IV library for the PDP-10 includes built-in functions, FUNCTION subprograms, and SUBROUTINE subprograms, listed and described in Chapter 8. Built-in functions are open subroutines; that is, they are incorporated into the object program each time they are referred to by the source program. FUNCTION and SUBROUTINE subprograms are closed subroutines; their names derive from the types of subprogram statements used to define them.

7.3 ARITHMETIC FUNCTION DEFINITION STATEMENT

The arithmetic function definition statement has the form:

```
identifier(identifier,identifier,...)=expression
```

This statement defines an internal subprogram. The entire definition is contained in the single statement. The first identifier is the name of the subprogram being defined.

Arithmetic function subprograms are single-valued functions with at least one argument. The type of the function is determined by the type of the function identifier.

The identifiers enclosed in parentheses represent the arguments of the function. These are dummy identifiers; they may appear only as scalar variables in the defining expression. Dummy identifiers have meaning and must be unique only within the defining statement. Dummy identifiers must agree in order, number, and type with the actual arguments given at execution time.

Identifiers, appearing in the defining expression, which do not represent arguments are treated as ordinary variables. The defining expression may include external functions or other previously defined arithmetic statement functions.

All arithmetic function definition statements must precede the first executable statement of the program.

Examples:
```
SSQR(K)=K*(K+1)*(2*K+1)/6
ACOSH(X)=(EXP(X/A)+EXP(-X/A))/2
```

In the last example above, X is a dummy identifier and A is an ordinary identifier. At execution time, the function is evaluated using the current value of the quantity represented by A.

7.4 FUNCTION SUBPROGRAMS

A FUNCTION subprogram is a single-valued function that may be called by using its name as a function name in an arithmetic expression, such as FUNC(N), where FUNC is the name of the subprogram that evaluates the corresponding function of the argument N. A FUNCTION subprogram begins with a FUNCTION statement and ends with an END statement. It returns control to the calling program by means of one or more RETURN statements.

7.4.1 FUNCTION Statement

The FUNCTION statement has the form:

```
FUNCTION identifier(argument,argument,...)
```

This statement declares the program which follows to be a FUNCTION subprogram. The identifier is the name of the function being defined. This identifier must not be used as a dummy argument or appear in any nonexecutable statement in the program other than as a scalar variable in a TYPE statement. It must appear as a scalar variable and be assigned a value during execution of the subprogram which is the function value.

Arguments appearing in the list enclosed in parentheses are dummy arguments representing the function argument. The arguments must agree in number, order, and type with the actual arguments used in the calling program. FUNCTION subprogram arguments may be expressions, alphanumeric strings, array names, statement labels preceded by an asterisk (*) or dollar sign ($), or subprogram names.

Dummy arguments may appear in the subprogram as scalar identifiers, array identifiers, subprogram identifiers, or an asterisk (*) or dollar sign ($), denoting statement labels in the calling program. A function must have at least one dummy argument. Dummy arguments representing array names must appear within the subprogram in a DIMENSION statement, or one of the type statements that provide dimension information. Dimensions given as constants must equal the dimensions of the corresponding arrays in the calling program. In a DIMENSION statement, dummy identifiers may be used to specify adjustable dimensions for array name arguments. For example, in the statement sequence:

```
FUNCTION TABLE(A,M,N,B,X,Y)
                :
DIMENSION A(M,N),B(10),C(50)
```

The dimensions of array A are specified by the dummies M and N, while the dimension of array B is given as a constant. The various values given for M and N by the calling program must be those of the actual arrays which the dummy A represents. The arrays may each be of different size but must have two dimensions. The arrays are dimensioned in the programs that use the function.

Dummy dimensions may be given only for dummy arrays. In the example above the array C must be given absolute dimensions, since C is not a dummy identifier. A dummy identifier may not appear in an EQUIVALENCE statement in the FUNCTION subprogram.

Dummy arguments representing statement labels can be used only in connection with the RETURN statement. When the value of the function is not required, a FUNCTION subprogram can be used as a SUBROUTINE subprogram by utilizing the optional return. When the optional return appears in a FUNCTION subprogram, the value of the function is stored on return only if RETURN or RETURN i (where i = 0) is used.

Example: FUNCTION LIST (A,$,C)

A function must not modify any arguments which appear in the FORTRAN arithmetic expression calling the function. Modification of implicit arguments from the calling program, such as variables in COMMON and DO loop indexes, is not allowed. The only FORTRAN statements not allowed in a FUNCTION subprogram are SUBROUTINE, BLOCK DATA, and another FUNCTION statement.

7.4.1.1 Function Type - The type of the function is the type of identifier used to name the function. This identifier may be typed, implicitly or explicitly, in the same way as any other identifier. Alternatively, the function

may be explicitly typed in the FUNCTION statement itself by preceding the word FUNCTION with one of the types or equivalent types described in Section 6.3. For example:

```
INTEGER FUNCTION
REAL FUNCTION
COMPLEX FUNCTION
LOGICAL FUNCTION
DOUBLE PRECISION FUNCTION
REAL*8 FUNCTION
```

Thus, the statement

```
COMPLEX FUNCTION HPRIME(S,N)
```

is equivalent to the statements

```
FUNCTION HPRIME(S,N)
COMPLEX HPRIME
```

Examples:

```
FUNCTION MAY(RANGE,EP,YP,ZP)
COMPLEX FUNCTION COT(ARG)
DOUBLE PRECISION FUNCTION LIMIT(X,Y)
FUNCTION WORK (A,$,C)
```

7.5 SUBROUTINE SUBPROGRAMS

A SUBROUTINE subprogram may be multivalued and can be referred to only by a CALL statement. A SUBROUTINE subprogram begins with a SUBROUTINE statement and returns control to the calling program by means of one or more RETURN statements.

7.5.1 SUBROUTINE Statement

The SUBROUTINE statement has the form:

```
SUBROUTINE identifier(argument,argument,...)
```

This statement declares the program which follows to be a SUBROUTINE subprogram. The first identifier is the subroutine name. This identifier cannot be used as a dummy argument or appear in any nonexecutable statement in the program other than as a scalar variable in a TYPE statement. The subroutine name can, however, be used as a scalar variable in any executable statement in the program. The arguments in the list enclosed in parentheses are dummy arguments representing the arguments of the subprogram. The dummy arguments must agree in number, order, and type with the actual arguments used by the calling program.

SUBROUTINE subprograms may have expressions, alphanumeric strings, array names, statement labels, and subprogram names as arguments. The dummy arguments may appear as scalar, array, subprogram identifiers, or an

asterisk (*) or dollar sign ($) denoting a statement label in the calling program. Dummy arguments representing statement labels can be used only in connection with the RETURN statement.

Dummy identifiers which represent array names must be dimensioned within the subprogram by a DIMENSION or type declaration statement. As in the case of a FUNCTION subprogram, either constants or dummy identifiers

may be used to specify dimensions in a DIMENSION statement. The dummy arguments must not appear in an EQUIVALENCE or COMMON statement in the SUBROUTINE subprogram.

A SUBROUTINE subprogram may use one or more of its dummy identifiers to represent results. The subprogram name is not used for the return of results. A SUBROUTINE subprogram need not have any argument at all.

Examples:
```
SUBROUTINE FACTOR(COEFF,N,ROOTS)
SUBROUTINE RESIDU(NUM,N,DEN,M,RES)
SUBROUTINE SERIES
SUBROUTINE TYPE(A,$,B,*)
```

The only FORTRAN statements not allowed in a function subprogram are FUNCTION, BLOCK DATA, and another SUBROUTINE statement.

7.5.2 CALL Statement

The CALL statement assumes one of two forms:

```
CALL identifier
CALL identifier (argument,argument,...,argument)
```

The CALL statement is used to transfer control to SUBROUTINE subprogram. The identifier is the subprogram name.

The arguments may be expressions, array identifiers, alphanumeric strings, subprogram identifiers, or statement labels of the calling program preceded by an asterisk (*), dollar sign ($), or ampersand (&). Arguments may be of any type, but must agree in number, order, type, and array size (except for adjustable arrays, as discussed under the DIMENSION statement) with the corresponding arguments in the SUBROUTINE statement of the called subroutine. Unlike a function, a subroutine may produce more than one value and cannot be referred to as a basic element in an expression.

A subroutine may use one or more of its arguments to return results to the calling program. If no arguments at all are required, the first form is used.

Examples:
```
CALL EXIT
CALL SWITCH(SIN,2.LE.BETA,X**4,Y)
CALL TEST(VALUE,123,275)
CALL TYPE(A,$10,B,*20,&30)
```

The identifier used to name the subroutine is not assigned a type and has no relation to the types of the arguments. Arguments which are constants or formed as expressions must not be modified by the subroutine.

7.5.3 RETURN Statement

The RETURN statement has one of two forms:

```
RETURN
RETURN i
```

where i is an integer constant or an interger variable. The value of i must be positive, and specifies that the return is to the i-th argument of the referencing statement (where the i-th argument is a statement number preceded by a $ or *). If i=0, the return is the same as with the first form of the RETURN statement.

This statement returns control from a subprogram to the calling program. Normally, the last statement executed in a subprogram is a RETURN statement. Any number of RETURN statements may appear in a subprogram. For purposes of debugging functions and subroutines originally written as main programs, the RETURN statement has been made equivalent to the STOP statement in a main program.

7.6 BLOCK DATA SUBPROGRAMS

A BLOCK DATA subprogram is a data specification subprogram and is used to enter initial values into variables in COMMON for use by FORTRAN subprograms and MACRO-10 main programs (see Chapter 9). No executable statements may appear in a BLOCK DATA subprogram.

7.6.1 BLOCK DATA Statement

The BLOCK DATA statement has the form:

```
BLOCK DATA
```

This statement declares the program which follows to be a data specification subprogram and it must be the first statement of the subprogram (see Chapter 6, Section 6.2.2).

7.7 EXTERNAL STATEMENT

FUNCTION and SUBROUTINE subprogram names may be used as the actual arguments of subprograms. Such subprogram names must be distinguished from ordinary variables by their appearance in an EXTERNAL statement. The EXTERNAL statement has the form:

```
EXTERNAL identifier, identifier,...,identifier
```

This statement declares the listed identifiers to be subprogram names. Any subprogram name given as an argument to another subprogram must have previously appeared in an external declaration in the calling program (i.e., as an identifier in an EXTERNAL or CALL statement or as a function name in an expression).

Example:

```
EXTERNAL SIN,COS
          :
          :
CALL TRIGF(SIN,1.5,ANSWER)
          :
          :
CALL TRIGF(COS,.87,ANSWER)
          :
          :
END
```

```
SUBROUTINE TRIGF(FUNC,ARG,ANSWER)
          :
ANSWER = FUNC(ARG)
          :
RETURN
END
```

To reference external variables from a MACRO-10 program by name, place the variables in named COMMON. Use the name of the variable as the name of the COMMON block:

```
COMMON /A/A/B/B(13)/C/C(6,7)
```

7.8 SUMMARY OF PDP-10 FORTRAN IV STATEMENTS

CONTROL STATEMENTS

General Form	Section References
ASSIGN i to m	4.1.3
CALL name (a_1,a_2,...)	7.5.2
CONTINUE	4.4
DO i m=m_1,m_2,m_3	4.3
GO TO i	4.1.1
GO TO m	4.1.3
GO TO m, (i_1,i_2,...)	4.1.3
GO TO (i_1,i_2,...),m	4.1.2
IF (e_1)i_1,i_2,i_3	4.2.1
IF (e_2)s	4.2.2
PAUSE	4.5
PAUSE j	4.5
PAUSE 'h'	4.5
RETURN	7.5.3
RETURN i	7.5.3
STOP	4.6
END	4.7

DATA TRANSMISSION STATEMENTS

General Form	Section References
ACCEPT f	5.2.9
ACCEPT f,list	5.2.9
BACKSPACE unit	5.3
DECODE (n,f,v)list	5.4
END FILE unit	5.3

General Form	Section References
ENCODE (n,f,v)list	5.4
FORMAT (g)	5.1.1
PRINT f	5.2.3
PRINT f, list	5.2.3
PUNCH f	5.2.4
READ f	5.2.7
READ f, list	5.2.7
READ (unit, f)	5.2.7
READ (unit,f)list	5.2.7
READ (unit)list	5.2.7
READ (unit,$name_1$)	5.2.7
READ (unit #R,f)list	5.2.7
READ (unit,f,END=c,ERR=d)list	5.2.7
READ (unit,f,END=c)list	5.2.7
READ (unit,f,ERR=d)list	5.2.7
REREAD f,list	5.2.8
REWIND unit	5.3
SKIP RECORD unit	5.3
TYPE f	5.2.5
TYPE f,list	5.2.5
WRITE (unit,f)	5.2.6
WRITE (unit,f)list	5.2.6
WRITE (unit)list	5.2.6
WRITE (unit,$name_1$)	5.2.6
WRITE (unit #R,f)list	5.2.6
UNLOAD unit	5.3

SPECIFICATION STATEMENTS

General Form	Section References
BLOCK DATA	6.2.2
COMMON a(n_1,n_2,...),b(n_3,n_4,...),...	6.1.2
COMMON /blk1/a,b/blk2/c,d/...	6.1.2
COMPLEX a(n_1,n_2,...),b(n_3,n_4,...),...	6.3
DATA t,u,.../k_1,k_2,k_3,.../ v,w,.../k_4,k_5,k_6,.../...	6.2.1

DIMENSION $a(n_1,n_2,\ldots),b(n_1,n_2,\ldots),\ldots$	6.1.1
DOUBLE PRECISION $a(n_1,n_2,\ldots),b(n_3,n_4,\ldots),\ldots$	6.3
EQUIVALENCE $(a(n_1,\ldots),b(n_2,\ldots),\ldots),\ldots$ $(c(n_3,\ldots),d(n_4,\ldots),\ldots),\ldots$	6.1.3
EXTERNAL y,z,...	7.7
IMPLICIT $type_1(l_1\text{-}l_2),type_2(l_3\text{-}l_4),\ldots$	6.3.1
INTEGER $a(n_1,n_2,\ldots),b(n_3,n_4,\ldots),\ldots$	6.3
LOGICAL $a(n_1,n_2,\ldots),b(n_3,n_4,\ldots),\ldots$	6.3
NAMELIST $/name_1/a,b,\ldots/name_2/c,d,\ldots$	5.1.2
REAL $a(n_1,n_2,\ldots)b(n_3,n_4,\ldots),\ldots$	6.3
SUBSCRIPT INTEGER $a(n_1,n_2,\ldots),b(n_3,\ldots),\ldots$	6.3

ARITHMETIC STATEMENT FUNCTION DEFINITION

General Form	Section Reference
name(a,b,...)=e	7.3

NOTE:

$a_1,a_2,\ldots$	are expressions
a,b,c,d	are variable names
blk1,blk2	are block names
c	is the statement number to which control is transferred upon encountering an end-of-file
d	is the statement number to which control is transferred upon encountering an error condition on the input data.
e	is an expression
e_1	is a noncomplex expression
e_2	is a logical expression
f	is a format number
g	is a format specification
'h'	is an alphanumeric
$i,i_1,i_2,\ldots$	are statement numbers
j	is an integer constant
$k_1,k_2,\ldots$	are constants of the general form j*k where k is any constant
$l_1,l_2,\ldots$	are letters

General Form	Section Reference
list	is an input/output list
m	is an integer variable name
m_1, m_2, m_3	are integer expressions
$n_1, n_2, \ldots$	are dimension specifications
n	are the number of ASCII characters
name	is a subroutine or function name
$name_1, name_2$	are NAMELIST names
#R	is a record number where I/O begins
s	is a statement (not DO or logical IF)
t,u,v,w	are variable names or input/output lists
$type_1, type_2, \ldots$	are type specifications
unit	is an integer variable or constant specifying a logical device number
v	is the starting address of the ASCII record referenced
y,z	are external subprogram names

SECTION II
THE RUN TIME SYSTEM

The five chapters of this section contain information on LIB40, SUBPROGRAM calling sequences, accumulator usage, compiler switches and diagnostic messages, and FORTRAN user programming.

CHAPTER 8
LIB40

LIB40 is a single file which contains all of the programs in the FORTRAN library. It is composed of three groups of programs:

(1) The FORTRAN Operating System.
(2) Science Library.
(3) FORTRAN Utility Subprograms.

There are two forms of LIB40, one for the KA-10 and the other for the KI-10. The KA-10 library will run on the KI-10, but will not take advantage of the speed of the KI-10. The KI-10 library will not run on the KA-10 because of the hardware differences. Also, the library used must match the compiler used, i.e., KA-10 compiled code must use the KA-10 LIB40 and the KI-10 compiled code must use the KI-10 LIB40.

8.1 THE FORTRAN OPERATING SYSTEM

The system programs in the FORTRAN Operating System act as the interface between the user's program and the PDP-10. All of these programs are invisible to the user's program. The FORTRAN Operating System is loaded automatically from LIB40 and resides in the user's core area along with the user's main programs and any library functions and subroutines that his programs reference.

8.1.1 FORSE.

FORSE. is the main program of the FORTRAN Operating System and is loaded whenever a FORTRAN main program is in core.. The primary functions of FORSE. are

a. FORMAT statement processing,
b. Dispatching of all UUOs, and
c. Control of I/O devices at runtime.

8.1.1.1 FORMAT Processing - FORSE. assumes that all FORMAT statements are syntactically correct since the syntax of each statement is checked by the compiler. FORSE. scans the FORMAT statements and performs the indicated I/O operations. FORSE. invokes the required conversion routine to actually do data conversion. The conversion routine that is used is a function of the conversion indicated in the FORMAT statement and of the data type of the element in the I/O list.

8.1.1.2 UUO Dispatching - Some UUOs are handled minimally by FORSE. (NLIN, NLOUT, MTOP), but the others are handled almost entirely within FORSE.

8.1.1.3 I/O Device Control - FORSE. executes the required carriage control of output devices that are physical listing devices (LPT, TTY) and stores the carriage control character at the beginning of each line if the output is going to a retrievable medium for deferred listing. When listings are deferred, the appropriate switch in PIP can be used to list the file and execute the required carriage control.

8.1.1.4 Additional Functions of FORSE. - FORSE. is responsible for the following:

a. Control of REREAD and ENCODE/DECODE features.

b. Interaction with EOFTST and READ (unit,f,END=C)list to handle end-of-file testing.

c. Control of the assignment of devices to software channels.

d. Control of the handling of filenames for I/O associated with directory devices.

e. Control of the opening and closing of data files.

f. Control the handling of the functions associated with the MAGDEN, BUFFER, IBUFF, OBUFF, DEFINE FILE, TRAPS, and RELEASE subroutines.

8.1.2 I/O Conversion Routines

The I/O conversion routines convert data from internal PDP-10 format to external format or vice versa. The calls to these routines are implied by FORMAT and data transfer statements in the FORTRAN source program. The routines reside as relocatable binary files in LIB40. REL.

Table 8-1
I/O Conversion Routines

Routine	Description
ALPHI.	Alphanumeric ASCII input conversion
ALPHO.	Alphanumeric ASCII output conversion
FLIRT.*	Floating point and double precision input conversion
FLOUT.*	Floating point and double precision output conversion
INTI.	Integer input conversion
INTO.	Integer output conversion
LINT.	Logical input conversion
LOUT.	Logical output conversion

*FLIRT. contains two entry points, FLIRT and DIRT.
FLOUT. contains two entry points, FLOUT and DOUBT.

Table 8-1 (Cont)
I/O Conversion Routines

Routine	Description
BINWR.	Binary I/O
OCTI.	Octal input conversion
OCTO.	Octal output conversion
NMLST.	Namelist

8.1.3 FORTRAN UUOs

Operation codes 000 through 077 in the PDP-10 are programmed operators, sometimes referred to as UUO's (Unimplemented User Operators) since from a hardware point of view their function is not prespecified. Some of these op-codes trap to the Monitor and the rest trap to the user program. FORTRAN UUO's trap to the FORTRAN Operating System UUO Handler and are then processed.

Table 8-2
FORTRAN UUOs

UUO	Op Code	Meaning
RESET.	015	Resets all devices, clears tables and flags.
IN.	016	Initializes device for formatted input, does a LOOKUP.
OUT.	017	Initializes device for formatted output, does an ENTER.
DATA.	020	Converts one data element from external to internal format or vice versa depending upon whether input or output is being done. Actual data transfer takes place.
FIN.	021	Terminates data transfer statements.
RTB.	022	Initializes device for unformatted input, similar to IN.
WTB.	023	Initializes device for unformatted output, similar to OUT.
MTOP.	024	Performs Magtape operations, rewind, rewind and unload, backspace, end file, skip, write blank record.
SLIST.	025	Converts entire arrays from external to internal format or vice versa depending upon whether input or output is being done. Actual data transfer takes place.
INF.	026	IFILE. Sets up input filename, similar to IN. but with specified filename.
OUTF.	027	OFILE. Sets up output filename, similar to OUT. but with specified filename.
RERED.	030	REREAD. Reread last record.
NLI.	031	Namelist input.

Table 8-2 (Cont)
FORTRAN UUOs

UUO	Op Code	Meaning
NLO.	032	Namelist output.
DEC.	033	DECODE.
ENC.	034	ENCODE.

8.2 SCIENCE LIBRARY AND FORTRAN UTILITY SUBPROGRAMS

The Science Library and FORTRAN Utility Subprograms extend the capabilities of the FORTRAN language. These subprograms are called explicitly by the user. The subprograms include the built-in FORTRAN math functions and the user-called utility subroutines which provide optional I/O capabilities and control of and information about the program's environment. The optional I/O capabilities and environmental control are achieved by the subroutines from interactions with the FORTRAN Operating System.

8.2.1 FORTRAN IV Library Functions

This section contains descriptions of all standard function subprograms provided with the FORTRAN IV library for the PDP-10. These functions are called by using the function mnemonic as a function name in an arithmetic expression. The function mnemonics in Table 8-3 have the types specified unless their types are explicitly or implicitly changed. (Refer to Section 6.3, "Type Declaration Statements" and Section 6.3.1, "IMPLICIT Statement.")

Table 8-3
FORTRAN IV Library Functions

Function	Mnemonic	Definition	Number of Arguments	Type of Argument	Type of Function	External Calls
Absolute value:						
Real	ABS	|arg|	1	Real	Real	
Integer	IABS	|arg|	1	Integer	Integer	
Double precision	DABS	|arg|	1	Double	Double	
Complex to real	CABS	$c=(x^2+y^2)^{1/2}$	1	Complex	Real	SQRT
Conversion:						
Integer to real	FLOAT*		1	Integer	Real	
Real to integer	IFIX*	Result is largest integer $\leq a$	1	Real	Integer	
Double to real	SNGL		1	Double	Real	
Real to double	DBLE		1	Real	Double	
Integer to double	DFLOAT		1	Integer	Double	
Complex to real (obtain real part)	REAL		1	Complex	Real	
Complex to real (obtain imaginary part)	AIMAG		1	Complex	Real	
Real to complex	CMPLX	$c=Arg_1+i*Arg_2$	2	Real	Complex	
Truncation:						
Real to real	AINT	Sign of arg * largest integer $\leq$ |arg|	1	Real	Real	
Real to integer	INT*		1	Real	Integer	
Double to integer	IDINT		1	Double	Integer	
Remaindering:						
Real	AMOD	The remainder when Arg 1 is divided by Arg 2	2	Real	Real	ERROR., TRAPS
Integer	MOD		2	Integer	Integer	
Double precision	DMOD		2	Double	Double	
Maximum Value:						
	AMAX0	$Max(Arg_1, Arg_2, \ldots)$	≥ 2	Integer	Real	FLOAT
	AMAX1			Real	Real	
	MAX0			Integer	Integer	
	MAX1			Real	Integer	IFIX
	DMAX1			Double	Double	
Minimum Value:						
	AMIN0	$Min(Arg_1, Arg_2, \ldots)$	≥ 2	Integer	Real	FLOAT
	AMIN1			Real	Real	
	MIN0			Integer	Integer	
	MIN1			Real	Integer	IFIX
	DMIN1			Double	Double	

*These functions are not used on the KI-10 because they are unnecessary.

Table 8-3 (Cont)
FORTRAN IV Library Functions

Function	Mnemonic	Definition	Number of Arguments	Type of Argument	Type of Function	External Calls
Transfer of Sign:						
Real	SIGN	$\{ Sgn(Arg_2)*\|Arg_1\| \}$	2	Real	Real	
Integer	ISIGN		2	Integer	Integer	
Double precision	DSIGN		2	Double	Double	
Positive Difference:						
Real	DIM	$\{ Arg_1-Min(Arg_1, Arg_2) \}$	2	Real	Real	
Integer	IDIM		2	Integer	Integer	
Exponential:						
Real	EXP	$\{ e^{Arg} \}$	1	Real	Real	ERROR.
Double	DEXP		1	Double	Double	
Complex	CEXP		1	Complex	Complex	EXP,SIN,COS, ALOG,ERROR.
Logarithm:						
Real	ALOG	$\log_e$ (Arg)	1	Real	Real	ERROR.
	ALOG10	$\log_{10}$ (Arg)	1	Real	Real	ERROR.
Double	DLOG	$\log_e$ (Arg)	1	Double	Double	
	DLOG10	$\log_{10}$ (Arg)	1	Double	Double	
Complex	CLOG	$\log_e$ (Arg)	1	Complex	Complex	ALOG,ATAN2, SQRT,ERROR.
Square Root:						
Real	SQRT	$(Arg)^{1/2}$	1	Real	Real	ERROR.
Double	DSQRT	$(Arg)^{1/2}$	1	Double	Double	
Complex	CSQRT	$c=(x+iy)^{1/2}$	1	Complex	Complex	SQRT
Sine:						
Real (radians)	SIN	$\{ \sin (Arg) \}$	1	Real	Real	
Real (degrees)	SIND		1	Real	Real	
Double (radians)	DSIN		1	Double	Double	
Complex	CSIN		1	Complex	Complex	SIN,SINH,COSH, ALOG,EXP
Cosine:						
Real (radians)	COS	$\{ \cos (Arg) \}$	1	Real	Real	
Real (degrees)	COSD		1	Real	Real	
Double (radians)	DCOS		1	Double	Double	
Complex	CCOS		1	Complex	Complex	SIN,SINH,COSH, ALOG,EXP

Table 8-3 (Cont)
FORTRAN IV Library Functions

Function	Mnemonic	Definition	Number of Arguments	Type of		External Calls
				Argument	Function	
Hyperbolic:						
Sine	SINH	sinh (Arg)	1	Real	Real	EXP, ERROR.
Cosine	COSH	cosh (Arg)	1	Real	Real	EXP, ERROR.
Tangent	TANH	tanh (Arg)	1	Real	Real	EXP
Arc - sine	ASIN	asin (Arg)	1	Real	Real	ATAN, SQRT, ERROR.
Arc - cosine	ACOS	acos (Arg)	1	Real	Real	ATAN, SQRT, ERROR.
Arc tangent						
Real	ATAN	atan (Arg)	1	Real	Real	
Double	DATAN	atan (Arg)	1	Double	Double	
quotient of two arguments	ATAN2	atan (Arg_1/Arg_2)	2	Real	Real	ATAN, ERROR., TRAPS
	DATAN2	atan (Arg_1/Arg_2)	2	Double	Double	DATAN, ERROR.
Complex Conjugate	CONJG	Arg=X + iY, C=X - iY	1	Complex	Complex	
Random Number	RAN	result is a random number in the range of 0 to 1.0.	1	Integer, Real, Double, or Complex	Real	

8.2.2 FORTRAN IV Library Subroutines

This section contains descriptions of all standard subroutine subprograms provided within the FORTRAN IV library for the PDP-10. These subprograms are closed subroutines and are called with a CALL statement.

Table 8-4
FORTRAN IV Library Subroutines

Subroutine Name	Effect
BUFFER	Allows the programmer to specify buffering for a device at one of fifteen levels. CALL BUFFER (unit*, in/out, number) where in/out is 1 for input buffering only, 2 for output buffering only, or 3 for both, and number is the level of buffering ($1 \leq$ number ≤ 15). If number is not specified, 2 is assumed. In calls to two entries in BUFFER, IBUFF and OBUFF, the programmer can specify a non-standard buffer size if the records in his data files exceed standard buffer sizes set by the Monitor. (See Table 12-1.) The programmer cannot change buffer sizes for the disk; IBUFF and OBUFF are designed primarily for Magtape. CALL IBUFF (d,n,s) where d is the device number, n is the number of buffers, and s is the size of buffer.
CHAIN	Reads a segment of coding (Chain file) into core and links it to a program already residing in core. CALL CHAIN (type,device,file) where type is 0 (the next Chain file is read into core immediately above the permanent resident area) or type is 1 (the next Chain file is read into core immediately above the FORTRAN IV program which marks the end of the removable resident). Device is 0,1,2,... FORTRAN IV logical device number (Chain files can be stored on DSK, MTA, or DTA only) corresponding to the device where the Chain file can be found. File is 0 for reading the next file from the selected magnetic tape or 1,2,... for the number of the magnetic tape unit where the Chain file is located.
DATE	Places today's date as left-justified ASCII characters into a dimensioned 2-word array. CALL DATE (array) where array is the 2-word array. The date is in the form dd-mmm-yy

*For explanation, see page 7-10.

Table 8-4 (Cont)
FORTRAN IV Library Subroutines

Subroutine Name	Effect
DATE (cont)	where dd is a 2-digit day (if the first digit is 0, it is converted to a blank), mmm is a 3-digit month (e.g., MAR), and yy is a 2-digit year. The date is stored in ASCII code, left-justified in the two words.
DUMP	Causes particular portions of core to be dumped and is referred to in the following form: CALL DUMP (L_1,U_1,F_1,...,L_n,U_n,F_n) where L_i and U_i are the variable names which give the limits of core memory to be dumped. Either L_i or U_i may be upper or lower limits. F_i is a number indicating the format in which the dump is to be performed: 0=octal, 1=real, 2=integer, and 3=ASCII. If F is not 0,1,2,3, the dump is in octal. If F_n is missing, the last section is dumped in octal. If U_n and F_n are missing, an octal dump is made from L to the end of the job area. If L_n, U_n, and F_n are missing, the entire job area is dumped in octal. The dump is terminated by a call to EXIT.
EOF1(unit*)	Skips one end-of-file terminator when found and returns the value TRUE if an end-of-file was found and FALSE if it was not found. Subsequent terminators produce an error message.
EOFC(unit*)	Skips more than one end-of-file terminators when found and returns the value TRUE if an end-of-file was found or FALSE if it was not found.
ERRSET	Allows the user to control the typeout of execution-time arithmetic error messages, ERRSET is called with one argument in integer mode. CALL ERRSET(N) Typeout of each type of error message is suppressed after N occurances of that error message. IF ERRSET is not called, the default value of N is 2.
EXIT	Returns control to the Monitor and, therefore, terminates the execution of the program.
IFILE	Performs LOOKUPs for files to be read from DECtape and disk. CALL IFILE(unit*,filnam) where filnam is a filename consisting of five or fewer ASCII characters enclosed in single quotes ('). e.g., CALL IFILE (12, 'FILE1')

*For explanation, see page 7-10.

Table 8-4 (Cont)
FORTRAN IV Library Subroutines

Subroutine Name	Effect
ILL	Sets the ILLEG flag. If the flag is set and an illegal character is encountered in floating-point/double-precision input, the corresponding word is set to zero. CALL ILL
LEGAL	Clears the ILLEG flag. If the flag is set and an illegal character is encountered in floating-point/double-precision input, the corresponding word is set to zero. CALL LEGAL
MAGDEN	Allows specification of magnetic tape density and parity. CALL MAGDEN(unit*,density,parity) where density is the tape density desired (200=200 bpi, 556=556 bpi, or 800=800 bpi) and parity is the tape parity desired (0=odd, 1=even). Even parity is intended for use with BCD-coded tapes only.
OFILE	Performs ENTERs for files to be written on DECtape and disk. CALL OFILE (unit*,filnam) where filnam is a filename consisting of five ASCII characters.
PDUMP	Is referred to in the following form: CALL PDUMP(L_1,U_1,F_1,...,L_n,U_n,F_n) where the arguments are the same as those for DUMP. PDUMP is the same as DUMP except that control returns to the calling program after the dump has been executed.
RELEAS	Closes out I/O on a device initialized by the FORTRAN Operating System and returns it to the uninitialized state. CALL RELEAS (unit*)
SAVRAN	SAVRAN is called with one argument in integer mode. SAVRAN sets its argument to the last random number (interpreted as an integer) that has been generated by the function RAN.
SETRAN	SETRAN has one argument which must be a non-negative integer $< 2^{31}$. The starting value of the function RAN is set to the value of this argument, unless the argument is zero. In this case, RAN uses its normal starting value.

*For explanation, see page 7-10.

Table 8-4 (Cont)
FORTRAN IV Library Subroutines

Subroutine Name	Effect
SLITE(i)	Turns sense lights on or off. i is an integer expression. For $1 \leq i \leq 36$ sense light i will be turned on. If i = 0, all sense lights will be turned off.
SLITE(i,j)	Checks the status of sense light i and sets the variable j accordingly and turns off sense light i. If i is on, j is set to 1; and if i is off, j is set to 2.
SSWTCH(i,j)	Checks the status of data switch i($0 \leq i \leq 35$) and sets the variable j accordingly. If i is set down, j is set to 1; and, if i is up, j is set to 2.
TIME	Returns the current time in its argument(s) in left-justified ASCII characters. If TIME is called with one argument, CALL TIME(X) the time is in the form hh : mm where hh is the hours (24-hour time) and mm is the minutes. If a second argument is requested, CALL TIME(X,Y) the first argument is returned as before and the second has the form ss.t where ss is the seconds and t is the tenths of a second.

CHAPTER 9

SUBPROGRAM CALLING SEQUENCES

This chapter describes the conventions used in writing MACRO subprograms which can be called by FORTRAN IV programs, and FORTRAN subprograms which can be linked to MACRO main programs. The reader is assumed to be familiar with the following texts:

MACRO-10 Assembler (DEC-10-AMZB-D)
Section 2.5.8 "Linking Subroutines"
Figure 7-1, "Sample Program, CLOG"

Timesharing Monitors: (DEC-T9-MTZC-D)
Section 3.2.2 "Loading Relocatable Binary Files"

Science Library and FORTRAN Utility Subprograms (DEC-10-SFLE-D)
How to Use This Manual - FORTRAN calling sequences

9.1 MACRO SUBPROGRAMS CALLED BY FORTRAN MAIN PROGRAMS

9.1.1 Calling Sequences

The FORTRAN calling sequence, in the main program, for a subroutine is

FORTRAN Code	MACRO Code (Generated by Compiler)
CALL subprog (adr_1, adr_2,...)	JSA 16, subprog ARG $code_1$, adr_1 ARG $code_2$, adr_2 .

where

subprog	is the name of the subprogram
adr_1, adr_2,...	are the addresses of the arguments
$code_1$, $code_2$	are the accumulator fields of the ARG instructions which indicate the type of argument being passed to the subprogram. These codes are as follows:

0	Integer argument	4	Octal argument
1	Unused	5	Hollerith argument
2	Real argument	6	Double-precision argument
3	Logical argument	7	Complex argument

An example of a FORTRAN calling sequence for a subroutine and the MACRO-10 coding generated by the compiler is given below.

FORTRAN Code	MACRO Code
CALL PROG1 (REAL,INT)	JSA 16, PROG1
	ARG 02, REAL
	ARG 00, INT

The MACRO code generated by the compiler is the same for subroutines and functions; however, the FORTRAN code is different.

9.1.2 Returning of Answers

A subroutine returns to its answers in specified locations in the main program. These locations are often given as argument names or as variable names.

A function returns its answer in accumulator 0 (if a single word result) or in accumulators 0 and 1 (if a double-precision or complex result). A function may also return its answer in specified locations (given by argument names in the CALL) or variable names; in any event, however, it must return an answer in accumulator 0 (or accumulators 0 and 1).

A MACRO subprogram access COMMON by declaring as external common block names for labelled COMMON and by declaring .COMM. as external for blank common. A common block name always refers to the same core location as the first element following the block name in a COMMON statement. MACRO subprograms may refer to the remainder of the variables in the common block through additive globals.

9.1.3 Use of Accumulators

For accumulator usage, see Chapter 10, Accumulator Conventions for PDP-10 Main Programs and Subprograms.

9.1.4 Examples of Subprogram Linkage

Three examples of subprogram linkage, one of a subroutine, one of a function subprogram, and one of a FORTRAN main program and MACRO subprogram both referencing COMMON, are given below.

9.1.4.1 Example of a Subroutine Linkage - The coding of the subroutine in this example is followed by the calling sequence.

```
ENTRY     SUBA
SUBA:     0
          MOVE      1,@0(16)        ;GET FIRST ARGUMENT
          IMULI     1, 12           ;MULTIPLY BY 10
          MOVEM     1,@0(16)        ;RETURN RESULT IN ARGUMENT
          JRA       16, 1(16)       ;RETURN TO MAIN PROGRAM
```

FORTRAN Calling Sequence		MACRO Code (Generated by Compiler)
CALL	SUBA(INT)	JSA 16, SUBA ARG 00, INT

9.1.4.2 Example of a Function Subprogram Linkage - The coding of the function subprogram in this example is followed by the calling sequence.

```
ENTRY     FNC
FNC:      0
          MOVE      00,@0(16)       ;PICK UP FIRST ARGUMENT
          MOVE      01,@1(16)       ;PICK UP SECOND ARGUMENT
          IMUL      00, 01          ;MULTIPLY BOTH ARGUMENTS
                                    ;RESULT IN AC0
          JRA       16, 2(16)       ;RETURN WITH ANSWER IN AC0
```

FORTRAN Calling Sequence		MACRO Code (Generated by Compiler)
X = FNC	(I, 10)	JSA 16, FNC ARG 00, I ARG 00, CONST.

9.1.4.3 Example of a FORTRAN Main Program and a MACRO Subprogram Both Referencing COMMON.

```
T               F40             V013        28-NOV-69   12:24

1M              BLOCK           0

                MOVE            02,D
                FADR            02,B+7
                FADR            02,C+2
                MOVEM           02,A+1

                JSA             16,SUB2

                JSA             16,EXIT
MAIN.%          RESET.          00,0
                JRST            1M

COMMON
C               /.COMM./                    0
A               /A/             0
B               /B/             0
D               /D/             0

SUBPROGRAMS

FORSE.
JOBFF
SUB2
EXIT

SCALARS

D               0

ARRAYS
```

```
DIMENSION A(5),B(3,4),C(3)

COMMON C

COMMON/A/A/B/B/D/D

A(2)=B(2,3)+C(3)+D


CALL SUB2

END
```

```
A           0
B           0
C           0

MAIN.       ERRORS DETECTED: 0

  2K CORE USED



.MAIN       MACRO.V36    12:23   28-NOV-69

                                                        EXTERNAL .COMM.,A,B,D
                                                        ENTRY       SUB2
            000000      000000      000000      SUB2:   0
            000001      200000      000002              MOVE        0,A+2           ;GET A(3)
            000002      202000      000003              MOVEM       0,B+3           ;STORE IN B(1,2)
            000003      200000      000000              MOVE        0,.COMM.        ;GET C
            000004      202000      000000              MOVEM       0,D             ;STORE IN D
            000005      267716      000000              JRA         16,(16)         ;RETURN TO FORTRAN PROGRAM
                                                        END                         ;END

NO ERRORS DETECTED

PROGRAM BREAK IS 000006

            SYMBOL TABLE

A                       000000 EXT          B           000000 EXT      D           000004' EXT
SUB2                    000000' INT         .COMM.      000003' EXT
```

003466 IS THE PROGRAM BREAK

STORAGE MAP

MAIN.	000140	000035
MAIN.	000146	
.COMM.	000150	
A	000153	
B	000160	
D	000174	
.MAIN	000175	000006
SUB2	000175	
JOBDAT	000203	000000
FORSE.	000203	002374
BUFCA.	001624	
BUFHD.	002337	
CHINN.	001121	
CLOS.	002002	
CLOSI.	002000	
CLROU.	001763	
CLRSY.	001770	
DADDR.	002276	
DATA.	000000	
DEPOT.	001004	
DEVIC.	002477	
DEVNO.	002172	
DYNDV.	002212	
DYNND.	002356	
ENDLN.	001047	
EOFFL.	002205	
EOFTS.	001214	
EOL.	002275	
FI.	001112	
FIN.	000000	
FMTBG.	002274	
FMTEN.	002273	
FNCTN.	001751	
IORTR.	000334	
LOOK.	002034	
MTOP.	000000	
MTPZ.	002030	
NLI.	000000	
NLO.	000000	
FORSE.	000203	
IIB.	001141	
IN.	000000	
INF.	000000	
INP.	002007	
INPDV.	002203	
NXTCR.	001162	
NXTLN.	001172	
ONLY1.	002204	
OUT.	000000	
OUTF.	000000	
OUTT.	002013	
OVFLS.	002202	
PAKFL.	002176	
RERDV.	002501	
RERED.	000000	
RESET.	000000	
RIN.	000245	
RTB.	000000	
SESTA.	002020	
SETOU.	001755	
SLIST.	000000	
STAT.	001774	
TCNT1.	002506	
TCNT2.	002507	
TEMP.	002232	
TNAM1.	002133	
TANM2.	002132	
TPNTR.	002505	
TYPE.	002504	
UUOH.	001234	
WAIT.	002024	
WTB.	000000	
XIO.	000424	
ERROR.	002577	000431

BPHSE.	002777	
DEVER.	002667	
DPRER.	002767	
DUMER.	003041	
ENDTP.	002772	
ERROR.	002577	
ILLCH.	002634	
ILLMG.	003007	
ILRED.	003025	
ILUUO.	003051	
INIER.	002654	
LISTB.	002737	
LOGEN.	002627	
MSNG.	002707	
NMLER.	003020	
NOROM.	002720	
PARER.	003034	
QTY1	003170	
REDER.	002746	
TBLER.	002700	
UUOM	003067	
WLKER.	002731	
EXIT	003230	000002
EXIT	003230	
EXIT.	003231	
IOADR.	003232	000014
IOADR.	003232	
DALPHI	003246	000002
ALPHI.	003246	
DALPHO	003250	000002
ALPHO.	003250	
DDIRT	003252	000002
DIRT.	003252	
DDOUBT	003254	000002
DOUBT.	003254	
DFLIRT	003256	000002
FLIRT.	003256	
DFLOUT	003260	000002
FLOUT.	003260	
DINTI	003262	000002
INTI.	003262	
DOCTI	003264	000002
OCTI.	003264	
DINTO	003266	000002
INTO.	003266	
DOCTO	003270	000002
OCTO.	003270	
DLINT	003272	000002
LINT.	003272	
DLOUT	003274	000002
LOUT.	003274	
DNMLST	003276	000003

DELIM.	003300	
NMLST.	003276	
DTFMT	003301	000002
TFMT.	003301	
DBINWR	003303	000002
BINDT.	003303	
BINEN.	003303	
BINWR.	003303	
INPT.	003303	
DTPFCN	003305	000002
TPFCN.	003305	
DEVTB.	003307	000123
DATTB.	003363	
DEVLS.	003344	
DEVND.	003352	
DEVTB.	003307	
DVTOT.	000035	
MBFBG.	003352	
MTABF.	003353	
MTACL.	003421	
NEG1.	000005	
NEG2.	000007	
NEG3.	000003	
NEG5.	000002	
TABP1.	003363	
TABPT.	003362	
PDLST.	003432	000025
PDLST.	003432	
ILL	003457	000007
ILL	003457	

ILLEG.	003465
LEGAL	003462

LOADER 3K CORE
3+3K MAX 1225 WORDS FREE

9.2 MACRO MAIN PROGRAMS WHICH REFERENCE FORTRAN SUBPROGRAMS

9.2.1 Calling Sequences

The MACRO code which calls the FORTRAN subprogram should be the same as that produced by the FORTRAN IV compiler when it calls a subroutine. That is:

MACRO Code

```
JSA 16, subprog
ARG code1, adr1
ARG code2, adr2
      .
      .
```

where

subprog	is the name of the subprogram
adr_1, adr_2,...	are the addresses of the arguments
$code_1$, $code_2$	are the accumulator fields of the ARG instruction which indicate the type of argument being passed to the subprogram. These codes are as follows:

0	Integer argument
1	Unused
2	Real argument
3	Logical argument
4	Octal argument
5	Hollerith argument
6	Double-precision argument
7	Complex argument

Both subroutines and functions are called in this manner.

9.2.2 Returning of Answers

A FORTRAN subroutine returns its answers in specified locations in the main program. These locations may be given as variable names in COMMON or as argument names.

A FORTRAN function returns its answer in accumulator 0, if a single word result, or in accumulators 0 and 1, if a double-precision or complex result. A function may also return its answer in specified locations given by argument names in the CALL, or variable names in COMMON; in any event, however, it must return an answer in accumulator 0 (or accumulators 0 and 1).

If it is desired to reference a common block of data in both the MACRO main program and the FORTRAN subprogram, it is necessary to set up the common area first by loading a FORTRAN BLOCK DATA program before the MACRO main program and the FORTRAN subprogram.

9.2.3 Example of Subprogram Linkage

The following is an example of a FORTRAN subroutine being called by a MACRO main program. Both programs reference common data. Read and write statements have been omitted for simplification. Because the FORTRAN operating system, FORSE., sets up I/O channels at run time, the MACRO programmer must be sure not to initialize a device on a channel that FORSE. will then try to use, unless he releases the device before FORSE. is called. FORSE. initializes the first device encountered in the user program on software channel 1, the second on channel 2, etc.

It is possible to release a device from its associated channel in a FORTRAN program by a call to the subroutine RELEAS. Channels one through seventeen are available for I/O. If a FORTRAN user wishes to write MACRO programs which do I/O, he may use either FORTRAN UUO's or the channel numbers less than or equal to seventeen but greater than the largest number used by FORSE.

The FORTRAN RESET. UUO should be the first instruction executed in any program which accesses FORTRAN subroutines. For this reason the FORTRAN operating system, which contains the FORTRAN UUO handler routine, must be declared external in the MACRO main program. This causes FORSE. to be loaded. In general, any program in the FORTRAN library referenced in a MACRO program must be declared external. This results in the searching of LIB40 by the Linking Loader and loading the referenced program.

```
BLKDTA.F4        F40      V016     22-JAN-70        15:46

1M       BLOCK   0                                  BLOCK DATA

                                                    COMMON/A/A/B/B/C/C

                                                    COMMON D

                                                    DIMENSION A(5),B(2,3)

                                                    END

DAT.     BLOCK   0

COMMON
A        /A/     0
B        /B/     0
C        /C/     0
D        /.COMM./        0

SUBPROGRAMS

JOBFF

SCALARS

C        0
D        0

ARRAYS

A        0
B        0

DAT.     ERRORS DETECTED:  0

2K CORE USED
```

```
.MAIN    MACRO.V40 16:05 22-JAN-70
START .MAC

                                        ENTRY    START
                                        EXTERNAL              .COMM.,A,B,C,ARGS,FORSE.,EXIT.
        000000   015000   000000   START:  RESET.   00,0                  ;DO FORTRAN UUO RESET, FOUND IN FORSE.
        000001   200000   000000           MOVE     0,A                   ;GET A(1)
        000002   202000   000000           MOVEM    0,B                   ;STORE IN B(1,1)
        000003   200000   000000           MOVE     0,C                   ;GET C
        000004   202000   000000           MOVEM    0,.COMM.              ;STORE IN D
        000005   200040   000002           MOVE     1,A+2                 ;GET A(3)
        000006   202040   000005           MOVEM    1,B+5                 ;STORE IN B(2,3)
        000007   266700   000000           JSA      16,ARGS               ;GO TO FORTRAN SUBROUTINE ARGS
                                           JSA      16,EXIT.              ;EXIT. FORTRAN EXIT ROUTINE WHICH PRINTS
        000010   266700   000000                                          ;OUT SUMMARIES AND ALSO CALLS MONITOR
                                                                          ;LEVEL EXIT UUO. USER HAS OPTION TO USE
                                                                          ;EITHER
                                           END      START                 ;END

NO ERRORS DETECTED

PROGRAM BREAK IS 000011

START    .MAC    SYMBOL TABLE

A                000001' EXT          ARGS                 000007' EXT      B          000002' EXT
C                000003' EXT          EXIT.                000010' EXT      FORSE.     000000  EXT
START            000000' ENT          .COMM.               000004' EXT
```

```
ARGS.F4 F40        V016      22-JAN-70        15:46

1M        BLOCK    0                                    SUBROUTINE ARGS

                                                        COMMON /A/A/B/B/C/C

                                                        COMMON D

                                                        DIMENSION A(5),B(2,3)

                                                        A(1)=B(1,1)+C+D

          MOVE     02,C
          FADR     02,D
          FADR     02,B
          MOVEM    02,A

                                                        RETURN

          JRST     2M                                   END

          JRST     2M
ARGS%     ARG      00,0
          MOVEM    15,TEMP.
          MOVEM    16,TEMP.+1
          JRST     1M
2M        MOVE     15,TEMP.
          MOVE     16,TEMP.+1
          JRA      16,0(16)

COMMON
A         /A/      0
B         /B/      0
C         /C/      0
D         /.COMM./        0

SCALARS

ARGS      17
C         0
D         0
```

```
ARRAYS

A          0
B          0

ARGS     ERRORS DETECTED:  0
2K CORE USED



003471 IS THE LOW SEGMENT BREAK
.MAIN   STORAGE MAP      16:06      22-JAN-70
STARTING ADDRESS 000155 PROG .MAIN FILE START
DAT.    000140   000015

        DAT.     000140   A        000140   B        000145   C        000153
        .COMM.   000154

.MAIN   000155   000011

        START    000155

ARGS    000166   000020

        ARGS     000174

JOBDAT  000206   000000

FORSE.  000206   002374

        BUFCA.   001627   BUFHD.   002342   CHINN.   001124   CLOS.    002005
        CLOSI.   002003   CLROU.   001766   CLRSY.   001773   DADDR.   002301
        DATA.    000000   DEPOT.   001007   DEVIC.   002502   DEVNO.   002175
        DYNDV.   002215   DYNND.   002361   ENDLN.   001052   EOFFL.   002210
        EOFTS.   001217   EOL.     002300   FI.      001115   FIN.     000000
        FMTBG.   002277   FMTEN.   002276   FNCTN.   001754   FORSE.   000206
        IIB.     001144   IN.      000000   INF.     000000   INP.     002012
        INPDV.   002206   IORTR.   000337   LOOK.    002037   MTOP.    000000
        MTPZ.    002033   NLI.     000000   NLO.     000000   NXTCR.   001165
        NXTLN.   001175   ONLY1.   002207   OUT.     000000   OUTF.    000000
        OUTT.    002016   OVFLS.   002205   PAKFL.   002201   RERDV.   002504
```

	RERED.	000000	RESET.	000000	RIN.	000250	RTB.	000000
	SESTA.	002023	SETOU.	001760	SLIST.	000000	STAT.	001777
	TCNT1.	002511	TCNT2.	002512	TEMP.	002235	TNAM1.	002136
	TNAM2.	002135	TPNTR.	002510	TYPE.	002507	UUOH.	001237
	WAIT.	002027	WTB.	000000	XIO.	000427		
ERROR.	002602	000431						
	BPHSE.	003002	DEVER.	002672	DPRER.	002772	DUMER.	003044
	ENDTP.	002775	ERROR.	002602	ILLCH.	002637	ILLMG.	003012
	ILRED.	003030	ILUUO.	003054	INIER.	002657	LISTB.	002742
	LOGEN.	002532	MSNG.	002712	NMLER.	003023	NOROM.	002723
	PARER.	003037	QTY1	003173	REDER.	002751	TBLER.	002703
	UUOM	003072	WLKER.	002734				
EXIT	003233	000002						
	EXIT	003233	EXIT.	003234				
IOADR.	003235	000014						
	IOADR.	003235						
DALPHI	003251	000002						
	ALPHI.	003251						
DALPHO	003253	000002						
	ALPHO.	003253						
DDIRT	003255	000002						
	DIRT.	003255						
DDOUBT	003257	000002						
	DOUBT.	003257						
DFLIRT	003261	000002						
	FLIRT.	003261						
DFLOUT	003263	000002						
	FLOUT.	003263						
DINTI	003265	000002						
	INTI.	003265						

DOCTI	003267	000002						
	OCTI.	003267						
DINTO	003271	000002						
	INTO.	003271						
DOCTO	003273	000002						
	OCTO.	003273						
DLINT	003275	000002						
	LINT.	003275						
DLOUT	003277	000002						
	LOUT.	003277						
DNMLST	003301	000003						
	DELIM.	003303	NMLST.	003301				
DTFMT	003304	000002						
	TFMT.	003304						
DBINWR	003306	000002						
	BINDT.	003306	BINEN.	003306	BINWR.	003306	INPT.	003306
DTPFCN	003310	000002						
	TPFCN.	003310						
DEVTB.	003312	000123						
	DATTB.	003366	DEVLS.	003347	DEVND.	003355	DEVTB.	003312
	DVTOT.	000035	MBFBG.	003355	MTABF.	003356	MTACL.	003424
	NEG1.	000005	NEG2.	000007	NEG3.	000003	NEG5.	000002
	TABP1.	003366	TABPT.	003365				
PDLST.	003435	000025						
	PDLST.	003435						
ILL	003462	000007						
	ILL	003462	ILLEG.	003470	LEGAL	003465		

LOADER 3K CORE
3+3K MAX 1222 WORDS FREE

CHAPTER 10

ACCUMULATOR CONVENTIONS FOR MAIN PROGRAMS AND SUBPROGRAMS

10.1 LOCATIONS

Locations specified in the calling sequence for a FORTRAN subprogram may be either required locations or defined locations. A required location is a memory location whose address is specified in the calling sequence for a subprogram. For example, X is a required location in the calling sequence

```
JSA 16, SQRT
ARG     X
```

A defined location is a memory location whose address is specified in the definition of a calling sequence. The location does not appear in the calling sequence. For example in the calling sequence

```
MOVEI 16, MEMORY
PUSHJ 17, DFAS.0
```

MEMORY is required, and AC0, AC1, and AC2 are defined by DFAS.0.

10.2 ACCUMULATORS

10.2.1 Accumulators 0 and 1

When used for subprograms called by JSA, accumulators 0 and 1 may be used at any time without restoring their original contents. These accumulators cannot be required locations. A FORTRAN function returns its answer in accumulator 0 (if a single word result) or in accumulators 0 and 1 (if a double-precision or complex result). A function may also return its answer in specified locations (given by argument names in the CALL) or variable names; in any event, an answer must be returned either in accumulator 0 or in accumulators 0 and 1.

When used for subprograms called by PUSHJ 17, adr, accumulators 0 and 1 may have their contents destroyed. Some subprograms by their definition return an argument in accumulator 0 or 1.

10.2.2 Accumulators 2 Through 15

Accumulators 2 through 15 must not be destroyed by FORTRAN functions, but may be destroyed by FORTRAN subroutines. (Presently subroutines must preserve the contents of accumulator 15.) The contents of these accumulators must not be destroyed by subprograms called by PUSHJ unless the definition of the subroutines requires it.

10.2.3 Accumulators 16 and 17

Accumulator 16 should be used only for JSA-JRA subprogram calls unless the definition of the subprogram sequence requires otherwise. The contents of accumulator 16 may be destroyed by subprograms called by PUSHJ 17, adr.

Accumulator 17 must be used only for pushdown list operations.

10.3 UUOS

User UUO's are not considered subprograms and may not change any locations except those required for input and the contents of accumulators 0 or 1.

10.4 SUBPROGRAMS CALLED BY JSA 16, ADDRESS

The calling sequence is

```
JSA 16, address
ARG     adr1
ARG     adr2
        :
        :
ARG     adrN
```

where each ARG adrN corresponds to one argument of the subprogram.

There may or may not be arguments. If there are arguments, they must be in accumulators 2 through 15. Subroutines called with the FORTRAN CALL statement may, by definition, return an argument in accumulator 0 or 1. Subprograms that are FORTRAN functions (such as SIN or SQRT) may destroy the contents of accumulators 0 and 1. Results are returned in accumulator 0 for single word results and accumulators 0 and 1 for double word results.

10.5 SUBPROGRAMS CALLED BY PUSHJ 17, ADDRESS

See section 10.2. In addition, three consecutive accumulators are required for double-precision addition, subtraction, multiplication, and division operations. The contents of the third accumulator may be destroyed. The

"to memory" modes also leave the answer in the defined accumulators. The two arguments of the double-precision operation cannot be in the same accumulators. Complex addition, subtraction, multiplication, and division operations do not destroy locations except those required for the answer and accumulator 16. The two arguments of the complex operation must not be in the same accumulator.

10.6 SUBPROGRAMS CALLED BY UUOS

Subprograms called by UUO's may change the contents of accumulators 0 and 1 only.

Table 10-1
Accumulator Conventions for
PDP-10 FORTRAN IV Compiler and Subprograms

Subprogram Called By: Accumulators	JSA		PUSHJ	UUO
	Functions	Subroutines		
0, 1	1) May be destroyed. 2) May not be used to pass arguments. 3) A result must be returned in 0 or 0 and 1.	1) May be destroyed. 2) May not be used to pass arguments. 3) Results must not be returned.	1) May be destroyed. 2) May be used to pass arguments if the subprogram is defined with an argument in 0 or 0 and 1. 3) Results may be returned if the subprogram is so defined.	1) May be destroyed. 2) May be used to pass arguments except as defined. 3) Results must not be returned.
2-15	1) Must be preserved. 2) Arguments may be passed. 3) Results may be returned if required by calling sequence.	1) May be destroyed. 2) Arguments may be passed. 3) Results must not be returned.	1) Must be preserved unless the definition of subprogram forces results to be returned. 2) Arguments may be passed. 3) Results may be returned if the subprogram is so defined.	1) Must be preserved. 2) Arguments may be passed. 3) Results must not be returned.
16 Reserved for JSA-JRA Operations (except as noted for PUSHJ)	1) Must be preserved. 2) May not be used to pass arguments. 3) Results must not be returned.	1) Must be preserved. 2) May not be used to pass arguments. 3) Results must not be returned.	1) Is destroyed. 2) Used for argument address. 3) Results must not be returned.	1) Must be preserved. 2) May not be used to pass arguments. 3) Results must not be returned.
17 Reserved for Pushdown List Operations	1) Must be preserved. 2) May not be used to pass arguments. 3) Results must not be returned.	1) Must be preserved. 2) May not be used to pass arguments. 3) Results must not be returned.	1) Must be preserved. 2) May not be used to pass arguments. 3) Results must not be returned.	1) Must be preserved. 2) May not be used to pass arguments. 3) Results must not be returned.

CHAPTER 11

SWITCHES AND DIAGNOSTICS

11.1 FORTRAN SWITCHES AND DIAGNOSTICS

Table 11-1
FORTRAN Compiler Switch Options

Switch	Meaning
A†	Advance magnetic tape reel by one file.
B†	Backspace magnetic tape reel by one file.
C†	Generate a CREF-type cross-reference listing. (DSK:CREF.TMP assumed if no list-dev specified)
	Complement: Do not produce cross-reference information (standard procedure).
E	Print an octal listing of the binary program produced by the compiler in addition to the symbolic listing output.
	Complement: Do not produce octal listing (standard procedure).
I	Translate the letter D in column 1 as a space and treat the line as a normal FORTRAN statement.
	Complement: Translate the letter D in column 1 as a comment character and treat the line as a comment (standard procedure).
M	Include MACRO coding in the output listing.
	Complement: Eliminate the MACRO coding from the output listing (standard procedure).
N	Suppress output of error messages on the Teletype.
	Complement: Output error messages on TTY (standard procedure).
S	If the compiler is running on the KA-10, produce code for execution on the KI-10 and vice-versa.
T†	Skip to the logical end of the magnetic tape reel.
W†	Rewind the magnetic tape reel.
Z†	Zero the DECtape directory.

†Switches A through C and T, W, and Z must immediately follow the device name or filename.ext to which the individual switch applies.

Table 11-2
FORTRAN Compiler Diagnostics
(Command Errors)

Message	Meaning
?BINARY OUTPUT ERROR dev:filename.ext	An output error has occurred on the device specified for the binary program output.
?CANNOT FIND dev:filename.ext	Filename.ext cannot be found on this device.
?DEVICE INPUT ERROR for command string	Device error occurred while attempting to read Monitor command file.
IMPROPER IO FOR DEVICE dev:	An input device is specified for output (or vice versa) or an illegal data mode was specified (e.g., binary output to TTY).
ILLEGAL MEMORY REFERENCE AT loc COMPILATION TERMINATED	An illegal memory reference has occurred and compilation has stopped. The current output files will be closed and the next source files read.
?INPUT DATA ERROR dev:filename.ext	A read error has occurred on the source device.
?x IS A BAD SWITCH	This specified switch is not recognizable.
?x IS AN ILLEGAL CHARACTER	A character in a command string typein is not recognizable (e.g., FORM-FEED).
?dev: IS NOT AVAILABLE	Either the device does not exist or it has been assigned to another job.
LINKAGE ERROR	Input device error while doing Dump Mode I/O, or not enough core was available to execute the newly loaded program.
?LINKAGE ERROR FOR dev:filename	Specified dev:filename appears in a ! Monitor command string, but cannot be run for some reason.
?LISTING OUTPUT ERROR	An output error has occurred on the device specified for the listing output.
?NO ROOM FOR filename.ext	The directory on dev: DTAn is full and cannot accept filename.ext as a new file, or a protection failure occurred for a DSK output file.
?NO FILE NAMED filename.ext	An illegal filename has been used.
?NOT ENOUGH CORE FOR LINKAGE	Not enough core available to load (with dump mode I/O) the program specified in a ! Monitor command string.
?SYNTAX ERROR IN COMMAND STRING	A syntax error has been detected in a command string typein (e.g., the ← has been omitted).
?X SWITCH ILLEGAL AFTER LEFT ARROW	Cannot change machine type with a file or clear source directory.
?X SWITCH ILLEGAL AFTER FIRST STANDARD FILE	Cannot clear directory after start of compilation (Batch Mode).
?X SWITCH, NO LISTING FILE	A CREF listing requires a listing file.
?INSUFFICIENT CORE - COMPILATION TERMINATED	The compiler has insufficient table space to compile the program.

Table 11-2 (Cont)
FORTRAN Compiler Diagnostics
(Command Errors)

Message	Meaning
WORK STACK OVERFLOW AT loc COMPILATION TERMINATED	The pushdown list used by the compiler for machine language subroutine calls has overflowed. Compilation has stopped. The current output files will be closed and the next source file read.

Table 11-3
FORTRAN Compiler Diagnostics
(Compilation Errors)

	Message	Meaning
I-1	DUPLICATED DUMMY VARIABLE IN ARGUMENT STRING	A dummy variable (identifier) may appear only once in any one argument set representing the arguments of a subprogram. (See Section 7.3)
I-2	ARRAY NAME ALREADY IN USE	Any attempt to re-dimension a variable or redefine a scalar as an array is illegal. (See Section 6.1.1)
I-3	ATTEMPT TO REDEFINE VARIABLE TYPE	Once a variable has been defined as either complex, double precision, integer, logical, or real it may not be defined again. (See Section 2.2, 6.3)
I-4	NOT A VARIABLE FORMAT ARRAY	The variable which contains the FORMAT specification read-in at object time must be a dimensioned variable, i.e., an array (see Section 5.1.1) or a subprogram argument was used as a NAMELIST name with the subprogram (see Section 5.1.2).
I-5	NAME ALREADY USED AS NAMELIST NAME	After a NAMELIST name has been defined, it may appear only in READ or WRITE statements and may not be defined again. (See Section 5.1.2)
I-6	DUPLICATED NAMELIST NAME	A NAMELIST name has already been used as a scalar array or global dummy argument. (See Section 5.1.2)
I-7	A NAME APPEARS TWICE IN AN EXTERNAL STATEMENT	A subprogram name has been declared EXTERNAL more than once. (See Section 7.7)
I-8	ARGUMENT TYPE DOESN'T AGREE WITH FUNCTION SPEC	The actual arguments for a function do not agree in type with the dummy arguments in the specification of the function.
I-9	THIS FUNCTION REQUIRES MORE ARGUMENTS	Not enough arguments were supplied for a function.
I-10	SUBPROGRAM NAME ALREADY IN USE	A subprogram name has appeared in another statement as a scalar or array variable, arithmetic function statement name, or COMMON block name. (See Section 7.5)
I-11	DUMMY ARGUMENT IN DATA STATEMENT	Dummy arguments may not appear in DATA statements. (See Section 6.2.1)

Table 11-3 (Cont)
FORTRAN Compiler Diagnostics
(Compilation Errors)

Message		Meaning
I-12	NOT A SCALAR OR ARRAY	The variable defining the starting address for an ENCODE/DECODE statement must be a scalar or an array. (See Section 5.4) The I/O unit name of a READ/WRITE statement is not a scalar or array. (See Sections 5.2.6, 5.2.7) An attempt to ASSIGN a label number to a variable that is not a scalar or array. (See Section 2.2) An attempt to GO TO through a variable that is not a scalar or array. (See Section 4.1)
I-13	ILLEGAL USE OF DUMMY ARGUMENT	Dummy arguments may be used with functions or subprograms only. (See Sections 7.4.1, 7.5.1)
I-14	ILLEGAL DO LOOP PARAMETER	The DO index must be a non-subscripted integer variable while the initial, limit, and increment values of the index must be an integer expression - the index may not be zero. (See Section 4.3)
I-15	I/O VARIABLES MUST BE SCALARS OR ARRAYS	Referencing data in an I/O statement other than scalars or arrays is illegal. (See Section 5.2)
I-16	A CONFLICT EXISTS WITH A COMMON DECLARATION	The function name used was previously declared a scalar variable in a COMMON statement.
S-1	ILLEGAL NAME OR DELIMITER OR KEY CHARACTER	A variable name doesn't start with an alphabetic character, or a delimiter such as the left parenthesis that begins a format is missing, or a key character such as the letter D in BLOCK DATA is missing.
S-2	STATEMENT KEYWORD NOT RECOGNIZED	A statement keyword such as ERASE was not recognized, possibly due to misspelling (e.g., ERASC 16).
S-3	ILLEGAL FIELD SPECIFICATION	The field width or decimal specification in a FORMAT statement must be integer. The number of Hollerith characters in an H specification must be equal to the number specified. (See Sections 5.1.1.1, 5.1.1.6)
S-4	SCALAR VARIABLE - MAY NOT BE SUBSCRIPTED	An undimensioned variable (a scalar variable) is being illegally subscripted (see Section 2.2.1) or a scalar variable is subscripted in an ENCODE/DECODE statement (see Section 5.4).
S-5	ILLEGAL TYPE SPECIFICATION	The type of constant specified is illegal or misspelled. (See Section 2.1)
S-6	ARGUMENT IS NOT SINGLE LETTER	Arguments in parentheses must be single letters in IMPLICIT statement. (See Section 6.3.1)
S-7	'NAMELIST' NOT FOLLOWED BY "/"	The first character following NAMELIST must be /. (See Section 5.1.2)
S-8	ILLEGAL CHARACTER IN LABEL	A non-numeric character was detected in the label field of the statement, possibly because tabs or spaces are missing.

Table 11-3 (Cont)
FORTRAN Compiler Diagnostics
(Compilation Errors)

Message		Meaning
S-9	MISSING COMMA OR SLASH IN SPECIFICATION STATEMENT	A specification statement (see Section 7.8) requires a comma or slash and it is missing.
S-10	ILLEGAL ARITHMETIC "IF" - TOO MANY LABELS	An arithmetic "IF" statement must have no more or less than three statement labels to transfer to. Special optimization will occur if two of the labels are the same, or one or more labels refer to the next statement.
S-11	A NUMBER WAS EXPECTED	Only arrays which are subprogram arguments can have adjustable dimensions. (See Section 6.1.1.1)
S-12	IMPLICIT TYPE RANGE OVERLAPS PREVIOUS SPECIFICATION	An implicit type range encompasses a character that has already been given an implicit type.
S-13	ATTEMPT TO USE AN ARRAY OR FUNCTION NAME AS A SCALAR	Variables may be either scalar or array but not both. Variables appearing in a DIMENSION statement must be subscripted when used. (See Section 2.2) Function names must be followed by at least one argument enclosed in parentheses (See Section 7.4).
S-14	ARRAY NOT SUBSCRIPTED	See S-13
S-15	ILLEGAL USE OF AN ARITHMETIC FUNCTION NAME	Arithmetic function definition statement name is being used without arguments (i.e., as a scalar) in an arithmetic expression. (See Section 7.3)
S-16	MULTIPLE RETURN ILLEGAL WITHOUT STATEMENT LABEL ARG	A dollar sign ($) or an asterisk (*) must have appeared in the argument list of this subprogram to represent the position of a statement label argument in the call.
S-17	INCORRECT PAREN COUNT OR MISSING IMPLIED DO INDEX	The number of left and right parentheses does not match, or an undefined index variable was used in defining a DO loop (see Section 5.2.1), or the number of implied DO loops and the number of matching parentheses differ in a DATA statement. (See Section 6.2.1)
S-18	INVALID INDEX IN DO-LOOP OR IMPLIED DO-LOOP	The index of a DO statement must be a non-subscripted integer variable and must not be zero. (See Section 4.3) The index is not used as a subscript in a DATA list. (See Section 6.2.1)
S-19	EQUIVALENCE REQUIRES TWO OR MORE ELEMENTS	The EQUIVALENCE statement must have more than one argument because it causes variables to share the same location. (See Section 6.1.3)
S-20	ILLEGAL DEFINITION OF AN ARITHMETIC STATEMENT FUNCTION	The statement function continues past its recognized end point.
S-21	MISSING COMMA IN INPUT/OUTPUT LIST	An input/output list continues past its recognized end point.
S-22	STATEMENT CONTINUES PAST RECOGNIZED END POINT	A statement other than those mentioned above continued past its recognized end point.
S-23	ILLEGAL COMPLEX CONSTANT	The parentheses of the complex constant enclose a logical, Hollerith, or complex constant.

Table 11-3 (Cont)
FORTRAN Compiler Diagnostics
(Compilation Errors)

Message		Meaning
O-1	BLOCK DATA NOT SEPARATE PROGRAM	Block Data must exist as a separate program. (See Sections 6.2.2, 7.6)
O-2	SUBROUTINE IS NOT A SEPARATE PROGRAM	A subroutine following a main program or another subroutine subprogram may have no statement between it and the preceding programs END statement and must begin with a SUBROUTINE statement. The previous program must have been terminated properly. (See Section 7.5)
O-3	STATEMENT OUT OF PLACE	The IMPLICIT specification statement and any arithmetic function definition statement must appear before any executable statement. (See Chapter 6)
O-4	EXECUTABLE STATEMENTS ILLEGAL IN BLOCK DATA	Block DATA statements cannot contain executable statements.
A-1	MINIMUM VALUE EXCEEDS MAXIMUM VALUE	Minimum value of an array exceeds the maximum value specified. (See Section 6.1.1)
A-2	ATTEMPT TO ENTER A VARIABLE INTO COMMON TWICE	A variable name may appear in COMMON statement only once. (See Section 6.1.2)
A-3	ATTEMPT TO EQUIVALENCE A SUBPROGRAM NAME OR DUMMY ARGUMENT	An identifier defined as a subprogram name cannot appear in EQUIVALENCE statements in the defining program. Dummy argument identifiers of a subprogram may not appear in EQUIVALENCE statements in that subprogram. (See Sections 6.1.3, 7.1)
A-4	NOT A CONSTANT OR DUMMY ARGUMENT	Only constant and dummy arguments may be used as arguments in dimension statements. (See Section 7.4.1)
A-5	CAUTION ** COMMON VARIABLE PASSED AS ARGUMENT	The variable may be multiply defined in the called subprogram. (See Sections 7.4.1, 7.5.1)
M-1	TOO MANY SUBSCRIPTS	An array variable appears with more subscripts than specified. (See Sections 2.2.2, 6.1.1)
M-2	WRONG NUMBER OF SUBSCRIPTS	An array variable appears with too few subscripts. (See Sections 2.2.2, 6.1.1)
M-3	CONSTANT OVERFLOW	Too many significant digits in the formation of a constant or the exponent is too large. (See Section 2.1)
M-4	ILLEGAL 'IF' ARGUMENT	Logical IF or DO statement adjacent to a logical IF statement, or illegal expression within a logical IF statement. (See Sections 4.2.2, 4.3)
M-5	ILLEGAL CONVERSION IMPLIED	Attempt to mix double precision and complex data in the same expression. (See Section 2.3.1)
M-6	LABEL OUT OF RANGE OR ARRAY TOO LARGE	Illegal statement label (See Section 1.1.1) or array size is greater than $2^{18}-1$.
M-7	UNTERMINATED HOLLERITH STRING	A missing single quote or fewer than n characters following an "nH" specification. (See Section 5.1.1.6)

Table 11-3 (Cont)
FORTRAN Compiler Diagnostics
(Compilation Errors)

Message	Meaning
M-8 SYSTEM ERROR - NO MORE SPACE FOR RECURSIVE STORAGE	The compiler's work roll is too small to hold the parts of all the subexpressions this statement implies. Break this statement or reassemble the compiler with a larger work-roll parameter (WORLEN=150_8 at present).
M-9 TOO MUCH DATA - WRONG ARRAY SIZE OR LITERAL TOO LONG	The list of DATA constants defines more words than the list of DATA variables specifies. This may be due to an array of the wrong size in the list of DATA variables, or definition of an integer, real, or logical DATA variable with a Hollerith constant of more than five characters.
M-10 ILLEGAL DO LOOP CLOSE	Illegal statement terminating a DO loop. (See Section 4.3)
M-11 MORE DATA NEEDED - LITERAL TOO SHORT OR TYPE CONVERSION EXPECTED	The list of DATA constants defines fewer words than the list of DATA variables specifies. This may be due to a double precision or complex DATA variable defined with a Hollerith constant of less than six characters, or a double precision DATA variable defined with a real constant.
M-12 NON-INTEGER PARAMETER IN 'DO' STATEMENT	DO statement parameters must be integers. (See Section 4.3)
M-13 NON-INTEGER SUBSCRIPT	Array subscripts must be integer constants, variables, or expressions. (See Section 4.3)
M-14 ILLEGAL COMPARISON OF COMPLEX VARIABLES	The only comparison allowed of complex variables is .NE. or .EQ. (See Sections 2.2, 2.3)
M-15 TOO MANY CONTINUATION CARDS	More than 19 continuation cards. (See Section 1.1.2)
M-16 NON-INTEGER I/O UNIT OR CHARACTER COUNT	The I/O unit variable of a READ/WRITE statement, or the character count variable of an ENCODE/DECODE statement, is not an integer variable. (See Sections 5.2.6, 5.2.7, 5.4)
M-17 SYSTEM ERROR-ROLL OUT OF RANGE	Compiler error. Report this message and its circumstances via a Software Trouble Report.
M-18 SYSTEM ERROR - NO MORE SPACE FOR RECURSIVE CALLS	The compiler's exit roll is too small to hold the return addresses for all the recursive subroutine calls this statement requires to be compiled. Break up the statement or reassemble the compiler with a larger exit roll parameter (EXLEN1=201_8 at present).
M-19 ILLEGAL USE OF STATEMENT LABEL	A GO TO or IF statement transfers to itself.
M-20 ILLEGAL RECURSIVE CALL	The statement function called itself. Recursive calls are illegal in the FORTRAN language.
EXCESSIVE COUNT	The number specified is greater than the maximum possible number of characters in a statement.
OPEN DO LOOPS	The list of statements are specified in DO statements but not defined.
UNDEFINED LABELS	The list of labels that do not appear in the label field.

Table 11-3 (Cont)
FORTRAN Compiler Diagnostics
(Compilation Errors)

Message	Meaning
MULTIPLY DEFINED LABELS	The list of labels that appeared more than once in the label field.
ALLOCATION ERRORS	The list of EQUIVALENCEd COMMON variables which have attempted to extend the beginning of a COMMON block.

Table 11-4
FORTRAN Operating System Diagnostics
(Execution Errors)

Message	Meaning
?BLOCK TOO LARGE OR QUOTA EXCEEDED ON dev	The user's program attempted to add blocks to a random access file, which caused the block to be too large or caused him to exceed his disk quota.
?CANNOT ACCESS FORTR.SHR- GETSEG ERROR CODE xx	An error occurred when a GETSEG UUO was issued to access FORTR.SHR. The codes are listed in Appendix E of the TOPS-10 Monitor Calls manual.
?dev: NOT AVAILABLE	FORSE. tried to initialize a device which either does not exist or has been assigned to another job.
?DEVICE NUMBER n IS ILLEGAL	A nonexistent device number was selected.
?DEVICE NUMBER n MUST BE DSK FOR RANDOM ACCESS	The device for random access operations must be disk.
?DIRECT ACCESS DEVICE NUMBER n IS ILLEGAL	Only devices 1 through 17 can be used for random access.
ENCODE - DECODE ERROR	The character count in an ENCODE or DECODE statement was incorrect.
END OF FILE ON dev:	A premature end-of-file has occurred on an input device.
?END OF TAPE ON dev:	The end of tape marker has been sensed during input or output
?FILE NAME filename.ext NOT ON DEVICE dev:	Filename.ext cannot be found in the directory of the specified device.
?ILLEGAL CHARACTER, x, IN FORMAT	The illegal character x is not valid for a FORMAT statement.
?ILLEGAL CHARACTER, x, IN INPUT STRING	The illegal character x is not valid for this type of input.
?ILLEGAL MAGNETIC TAPE OPERATION, TAPE dev:	An attempt was made to skip a record after performing output on a magnetic tape.
?ILLEGAL PHYSICAL RECORD COUNT, TAPE dev:	FORSE. has encountered an inconsistency in the physical record count on a magnetic tape.
?ILLEGAL USER UUO uuu AT USER loc	An illegal user UUO to FORSE. was encountered at location loc.
?INPUT DEVICE ERROR ON dev:	A data transmission error has been detected in the input from a device.

FORTRAN Operating System Diagnostics
(Execution Errors)

Message	Meaning
?LIBRARY (FORTR.SHR) AND USER PROGRAM VERSION NUMBERS ARE DIFFERENT	The user's executable program is using an obsolete version of the library. The program should be recompiled so that the correct version of the library is used.
?MORE THAN 15 DEVICES REQUESTED	Too many devices have been requested.
?NAMELIST SYNTAX ERROR	Improper mode of I/O (octal or Hollerith), incorrect variable name.
?NO ROOM FOR FILE filename.ext ON DEVICE dev:	There is no room for the file in the directory of the named device.
?NOT ENOUGH CORE FOR BUFFERS	Either a call to BUFFER or a random access operation tried to set up a buffer ring when not enough core was available.
program name NOT LOADED	A dummy routine was loaded instead of the real one. Generally, this error occurs when a loaded program is patched to include a call to a library program which was not called by the original program at load time.
?OUTPUT DEVICE ERROR ON dev:	A data transmission error has been detected during output to a device.
?OUTPUT FIELD WIDTH OVERFLOW	A field overflowed on output and was filled with asterisks.
?PARITY ERROR ON dev:	A parity error has been detected.
?REREAD EXECUTED BEFORE FIRST READ	A reread was attempted before initializing the first input device.
?TAPE RECORD TOO SHORT ON UNIT n	The data list is too long on a binary tape READ operation.
?dev: WRITE PROTECTED	The device is WRITE locked.
WARNING! / IS ILLEGAL IN ENCODE-DECODE, END OF FORMAT ASSUMED	A slash was used in the FORMAT statement referenced by an ENCODE or DECODE statement. Since slashes are illegal in these statements, the operating system assumes that the slash is the end of the format.

The following messages are typed twice, when the error occurs, and in a final summary. When the error occurs, the PC value is appended to the message. When the message appears in the final summary, the number of times that the error occurred in the program is appended to the message.

?ACOS OF ARG > 1.0 IN MAGNITUDE
?ASIN OF ARG > 1.0 IN MAGNITUDE
?ATTEMPT TO TAKE LOG OF NEGATIVE ARG
?ATTEMPT TO TAKE SQRT OF NEGATIVE ARG
?FLOATING DIVIDE CHECK
?FLOATING OVERFLOW
?FLOATING UNDERFLOW
?INTEGER DIVIDE CHECK
?INTEGER OVERFLOW

Table 11-4 (Cont)
FORTRAN Operating System Diagnostics
(Execution Errors)

Message	Meaning
The following messages are issued when a LOOKUP, ENTER, or RENAME UUO error occurs. The number in parentheses indicates the error code. Refer to Appendix E in the TOPS-10 Monitor Calls manual.	
?(0) ILLEGAL FILENAME WAS NOT FOUND FILE xx ON DEVICE yy	
?(1) NO DIRECTORY FOR PROJECT -PROGRAMMER NUMBER FILE xx ON DEVICE yy.	
?(2) PROTECTION FAILURE FILE xx ON DEVICE yy	
?(3) FILE WAS BEING MODIFIED FILE xx ON DEVICE yy	
?(4) RENAME FILE NAME ALREADY EXISTS FILE xx ON DEVICE yy	
?(5) ILLEGAL SEQUENCE OF UUOS FILE xx ON DEVICE yy	
?(6) BAD UFD OR BAD RIB FILE xx ON DEVICE yy	
?(7) NOT A SAV FILE FILE xx ON DEVICE yy	
?(10) NOT ENOUGH CORE FILE xx ON DEVICE yy	
?(11) DEVICE NOT AVAILABLE FILE xx ON DEVICE yy	
?(12) NO SUCH DEVICE FILE xx ON DEVICE yy	
?(13) NOT TWO RELOC REG. CAPABILITY FILE xx ON DEVICE yy	
?(14) NO ROOM OR QUOTA EXCEEDED FILE xx ON DEVICE yy	
?(15) WRITE LOCK ERROR FILE xx ON DEVICE yy	
?(16) NOT ENOUGH MONITOR SPACE FILE xx ON DEVICE yy	
?(17) PARTIAL ALLOCATION ONLY FILE xx ON DEVICE yy	
?(20) BLOCK NOT FREE ON ALLOCATION FILE xx ON DEVICE yy	
NOTE With the exception of the messages ILLEGAL USER UUO uuu AT USER loc and ENCODE/DECODE ERROR, all messages are followed by a second message LAST FORTRAN I/O AT USER LOC adr	

Several arithmetic error conditions can occur during execution time.

a. Overflow - An attempt was made to create either a positive number greater than the largest representable positive number or a negative number greater in magnitude than the most negative representable number (in the appropriate mode).

Example: For I an integer,

$377777777777 < I < 400000000000$ (octal)

b. Underflow - An attempt was made to create either a positive non-zero number smaller than the smallest representable positive non-zero number or a negative number smaller in magnitude than the negative number whose magnitude is the smallest representable.

Example: For X a real non-zero number,

$$777400000000 < X < 000400000000$$

c. Divide Check - An attempt was made to divide by zero.

d. Improper Arguments for LIB40 math routines - For example, an attempt was made to find the arc sine of an argument greater than 1.0.

When overflow, underflow, or divide check errors occur in the user's FORTRAN program, the Monitor calls the LIB40 routine OVTRAP. This routine replaces the resulting numbers, if the numbers are floating point, with either zero in the case of underflow or ± the largest representable number in the cases of overflow and divide check. OVTRAP does not affect numbers in integer mode.

Overflow, underflow, and divide check errors occurring in LIB40 math routines are handled differently from when they occur in the user's program: only if the final answer from a routine is in error is an error condition considered to exist. If the answer is floating point, it is set to the appropriate value as for user program errors. Integer answers are handled in various ways. (See the Science Library and FORTRAN Utility Subprograms, DEC-10-SFLE-D.)

When an error condition occurs in a user program or in a final answer from a LIB40 math routine, an error message is typed. Presently there are eight distinct error messages.

Error Message No.	Error Message
1	INTEGER OVERFLOW PC=nnnnnn
2	INTEGER DIVIDE CHECK PC=nnnnnn
3	FLOATING OVERFLOW PC=nnnnnn
4	FLOATING UNDERFLOW PC=nnnnnn
5	FLOATING DIVIDE CHECK PC=nnnnnn
6	ATTEMPT TO TAKE SQRT OF NEGATIVE ARG
7	ACOS OF ARG > 1.0 IN MAGNITUDE
8	ASIN OF ARG > 1.0 IN MAGNITUDE
final	1 ? FATAL I/O ERROR

NOTE

nnnnnn = location at which the error occurred.

After two typeouts of a particular error message, further typeout of that error message is suppressed. At the end of execution, a summary listing the actual number of times each error message occurred is typed out. If the user wishes to permit more than two typeouts for each error message, he may do so by calling the routine ERRSET at the beginning of the executable part of his main program. ERRSET accepts one argument in integer mode. This argument is the number of typeouts that are permitted for each error message before suppression occurs. This routine is used to obtain the PC information which would otherwise be lost. Alternatively, because of the slow-

ness of the Teletype output, the user may wish to suppress typeout of the messages entirely. This can be done by calling ERRSET with an argument of zero. Suppression of typeout can also be accomplished during execution by typing ↑O on the Teletype.

Error messages and the summary are output to the Teletype (or the output device when running BATCH), regardless of the device assignments that have been made.

The treatment of overflow, underflow, and divide check errors in MACRO programs (those that are loaded with OVTRAP) can, to a certain extent, be manipulated by the user. (See OVTRAP in the Science Library and FORTRAN Utility Subprogram manual.)

12.1 ASCII CHARACTER SET

Table 12-1
ASCII Character Set

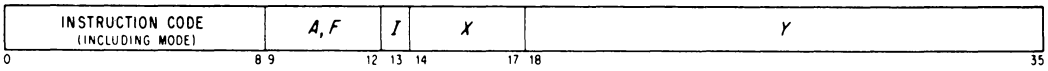
SIXBIT	Character	ASCII 7-Bit†	SIXBIT	Character	ASCII 7-Bit†	Character	ASCII 7-Bit†
00	Space	040	40	@	100	`	140
01	!	041	41	A	101	a	141
02	"	042	42	B	102	b	142
03	#	043	43	C	103	c	143
04	$	044	44	D	104	d	144
05	%	045	45	E	105	e	145
06	&	046	46	F	106	f	146
07	'	047	47	G	107	g	147
10	(	050	50	H	110	h	150
11	)	051	51	I	111	i	151
12	*	052	52	J	112	j	152
13	+	053	53	K	113	k	153
14	,	054	54	L	114	l	154
15	-	055	55	M	115	m	155
16	.	056	56	N	116	n	156
17	/	057	57	O	117	o	157
20	0	060	60	P	120	p	160
21	1	061	61	Q	121	q	161
22	2	062	62	R	122	r	162
23	3	063	63	S	123	s	163
24	4	064	64	T	124	t	164
25	5	065	65	U	125	u	165
26	6	066	66	V	126	v	166
27	7	067	67	W	127	w	167
30	8	070	70	X	130	x	170
31	9	071	71	Y	131	y	171
32	:	072	72	Z	132	z	172
33	;	073	73	[	133	{	173
34	<	074	74	\	134	|	174
35	=	075	75	]	135	}	175
36	>	076	76	↑	136	~	176
37	?	077	77	←	137	Delete	177

†FORTRAN IV also accepts the following control codes in 7-bit ASCII:

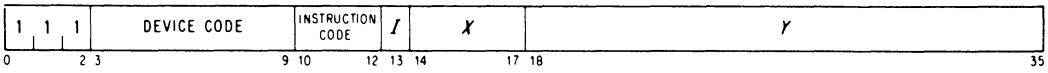
Horizontal Tab	011	Carriage Return	015
Line Feed	012	Form Feed	014

12.2 PDP-10 WORD FORMATS

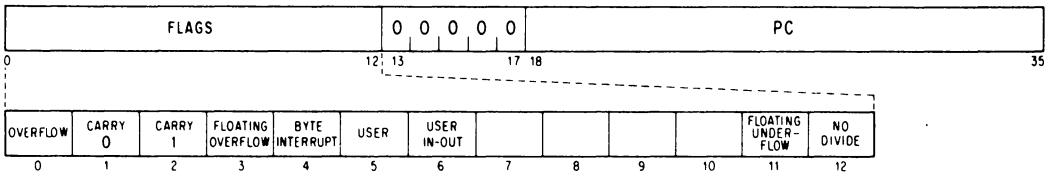
BASIC INSTRUCTIONS



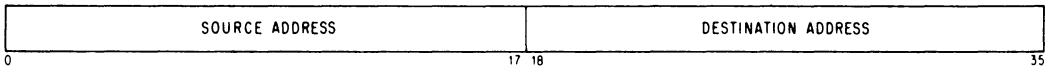
IN-OUT INSTRUCTIONS



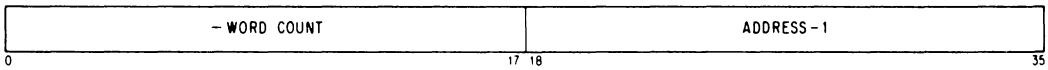
PC WORD



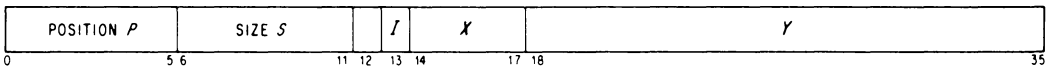
BLT POINTER [XWD]



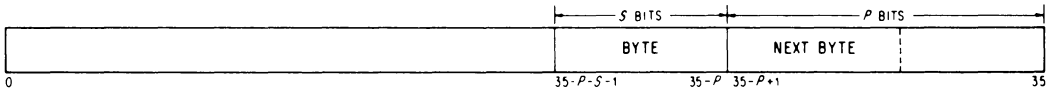
BLKI / BLKO POINTER, PUSHDOWN POINTER, DATA CHANNEL CONTROL WORD [IOWD]



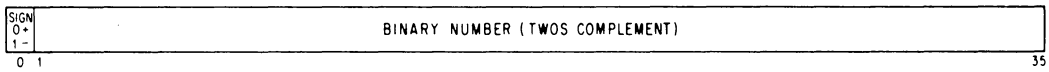
BYTE POINTER



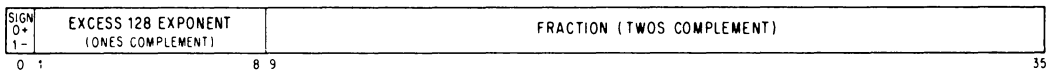
BYTE STORAGE



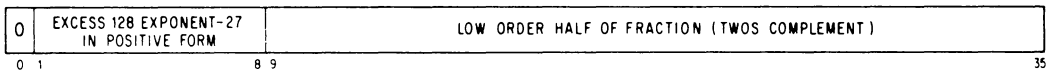
FIXED POINT OPERANDS



FLOATING POINT OPERANDS



LOW ORDER WORD IN DOUBLE LENGTH FLOATING POINT OPERANDS



In addition to the arithmetic functions, the PDP-10 FORTRAN IV library (LIB40) contains several subprograms which control FORTRAN IV I/O operations at runtime. The I/O subprograms are compatible with the PDP-10 Monitors.

In general FORTRAN IV I/O is done with double buffering unless the user has either specified otherwise through calls to IBUFF and OBUFF or is doing random access I/O to the disk. In these cases, single buffers are used. The standard buffer sizes for the devices normally available to the user are given in Table 12-2. Note that the devices and buffer sizes are determined by the Monitor and may be changed by a particular installation. Also a user may specify buffer sizes for magtape operations through the use of IBUFF and OBUFF.

The logically first device in a FORTRAN program is initialized on software I/O channel one, the second on software I/O channel two, and so forth. Software I/O channel 0 is reserved for error message and summary output. The SIXBIT name of the device that is initialized on channel N can be found in a dynamic device table at location DYNDV. + N. A device may be initialized for input and output on the same I/O channel. Devices are initialized only once and are released through either the CALL [SIXBIT/EXIT/] executed at the end of every FORTRAN program or the LIB40 subroutine RELEAS.

Table 12-2
PDP-10 FORTRAN IV Standard Peripheral Devices

Name	Mnemonic	Input/Output		Buffer Size In Words	Operation
		Formatted	Unformatted		
Card Punch	CDP	Yes	Yes	26	WRITE
Card Reader	CDR	Yes	Yes	28	READ
Disk (includes disk packs and drums)	DSK	Yes	Yes	128	READ/WRITE
DECtapes	DTA	Yes	Yes	127	READ/WRITE
Line Printer	LPT	Yes	No	26	WRITE
Magtape	MTA	Yes	Yes	128	READ/WRITE
Plotter	PLT	Yes	Yes	36	WRITE
Paper Tape Punch	PTP	Yes	Yes	33	WRITE
Paper Tape Reader	PTR	Yes	Yes	33	READ
Pseudo Teletype	PTY	Yes	No	17	READ/WRITE
Teletype - User	TTY	Yes	No	17	READ/WRITE
Teletype - Console	CTY	Yes	No	17	READ/WRITE

12.3.1 Logical and Physical Peripheral Device Assignments

Logical and physical device assignments are controlled by either the user at runtime or a table called DEVTB. The first entry in DEVTB. is the length of the table. Each entry after the first is a sixbit ASCII device name. The position in the table of the device name corresponds to the FORTRAN logical number for that device. For example, in Table 12-3, magnetic tape 0 is the 16th entry in DEVTB. Therefore, the statement

```
WRITE (16, 13)A
```

refers to magnetic tape 0. The last five entries in DEVTB. correspond to the special FORTRAN statements READ, ACCEPT, PRINT, PUNCH, and TYPE. Any device assignments may be changed by reassembling DEVTB.

If the user gives the Monitor command

```
ASSIGN DSK 16
```

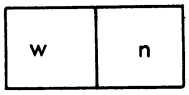
prior to the running of his program, a file named FOR16.DAT would be written on the disk. Similarly, the Monitor command

```
ASSIGN LPT 16
```

causes output to go to the line printer.

12.3.2 DECtape and Disk Usage

12.3.2.1 Binary Mode - In binary mode, each block contains 127 data words, the first of which is a record control word of the form:



where w is the word count specifying the number of FORTRAN data words in the block (126 for a full block) and n is 0 in all but the last block of a logical record, in which case n is the number of blocks in the logical record. A logical record contains all the data corresponding to one READ or WRITE statement, that is, the maximum number of logical records per disk/DECtape block is one.

12.3.2.2 ASCII Mode - In ASCII mode, blocks are packed with as many full lines (a line is a unit record as specified by a format statement) as possible. Lines always begin with a new word. If a line terminates in the middle of a word, the word is filled out with null characters and the next line begins with the next word. Lines are not split across blocks. Such a file is created by FORTRAN during output or by PIP with the A switch. FORTRAN input files must be in this format.

Table 12-3
Device Table for FORTRAN IV

```
TITLE     DEVTB  V.017
SUBTTL    1-APR-69

ENTRY     DEVTB.,DEVND.,DEVLS.,DVTOT.
ENTRY     MTABF.,MBFBG.,TABPT.,TABP1.
ENTRY     MTACL.,DATTB.,NEG1.,NEG2.,NEG3.,NEG5.
P=17

DEVTB.:   EXP      DEVND.-.        ;NO. OF ENTRIES
                                   ;LOGICAL#/FILENAME/DEVICE
          SIXBIT   .DSK.           ; 1 FOR01.DAT     DISC
CDRPOS:   SIXBIT   .CDR.           ; 2 FOR02.DAT     CARD READER
LPTPOS:   SIXBIT   .LPT.           ; 3 FOR03.DAT     LINE PRINTER
          SIXBIT   .CTY.           ; 4 FOR04.DAT     CONSOLE TELETYPE
TTYPOS:   SIXBIT   .TTY.           ; 5 FOR05.DAT     USER TELETYPE
          SIXBIT   .PTR.           ; 6 FOR06.DAT     PAPER TAPE READER
PTPPOS:   SIXBIT   .PTP.           ; 7 FOR07.DAT     PAPER TAPE PUNCH
          SIXBIT   .DIS.           ; 8 FOR08.DAT     DISPLAY
          SIXBIT   .DTA1.          ; 9 FOR09.DAT     DECTAPE
          SIXBIT   .DTA2.          ; 10 FOR10.DAT
          SIXBIT   .DTA3.          ; 11 FOR11.DAT
          SIXBIT   .DTA4.          ; 12 FOR12.DAT
          SIXBIT   .DTA5.          ; 13 FOR13.DAT
          SIXBIT   .DTA6.          ; 14 FOR14.DAT
          SIXBIT   .DTA7.          ; 15 FOR15.DAT
          SIXBIT   .MTA0.          ; 16 FOR16.DAT    MAGNETIC TAPE
          SIXBIT   .MTA1.          ; 17 FOR17.DAT
          SIXBIT   .MTA2.          ; 18 FOR18.DAT
          SIXBIT   .FORTR.         ; 19 FORTR.DAT    ASSIGNABLE DEVICE, FORTR

          SIXBIT   .DSK0.          ; 20 FOR20.DAT    DISK
          SIXBIT   .DSK1.          ; 21 FOR21.DAT
          SIXBIT   .DSK2.          ; 22 FOR22.DAT
          SIXBIT   .DSK3.          ; 23 FOR23.DAT
          SIXBIT   .DSK4.          ; 24 FOR24.DAT
          SIXBIT   .DEV1.          ; 25 FOR25.DAT    ASSIGNABLE DEVICES
          SIXBIT   .DEV2.          ; 26 FOR26.DAT
          SIXBIT   .DEV3.          ; 27 FOR27.DAT
          SIXBIT   .DEV4.          ; 28 FOR28.DAT
DEVLS.:   SIXBIT   .DEV5.          ; 29 FOR29.DAT      V.006
          SIXBIT   .REREAD.        ;       -6        REREAD
          SIXBIT   .CDR.           ;       -5        READ
          SIXBIT   .TTY.           ;       -4        ACCEPT
          SIXBIT   .LPT.           ;       -3        PRINT
          SIXBIT   .PTP.           ;       -2        PUNCH
DEVND.:   SIXBIT   .TTY.           ;       -1        TYPE
```

12.3.2.3 File Names - File names may be declared for DECtapes or the disk through the use of the library subprograms IFILE and OFILE. In order to make an entry of the file name FILE1 on unit u, the following statement could be used:

```
CALL OFILE (u,'FILE1')
```

Similarly, the following statements might be used to open the file, RALPH, for reading:

```
RALPH=5HRALPH
CALL IFILE(u,RALPH)
```

After writing a file, the END FILE u statement must be given in order to close the current file and allow for reading or writing another file or for reading or rewriting the same file. If no call to IFILE or OFILE has been given before the execution of a READ or WRITE referencing DECtape or the disk the file name FORnn.DAT is assumed where nn is the FORTRAN logical number used in the I/O statement that references device nn.

The FORTRAN programmer can make logical assignments such that each device has its own unique file as intended, but each can be on the DSK. In order to use the devices available, the programmer can make assignments at run time and assign the DSK to those not available.

For example, the FORTRAN logical device numbers, e.g., 1 = DSK, 2 = CDR, 3 = LPT, are used in the file name. The written file names are FOR01.DAT, FOR02.DAT, etc. The same is true for READ. For example, a WRITE (3, 1) A, B, C, in the FORTRAN program generates the file name FOR03.DAT on the DSK if the DSK has been assigned LPT or 3 prior to running the program. (Note: REREAD rereads from the file belonging to the device last referenced in a READ statement, not FOR-6.DAT, as usual.) The programmer must, of course, realize his own mistake in assigning the DSK as the TTY in the case that FORSE tries to type out error messages or PAUSE messages.

More than one DSK File may be accessed, without making logical assignments at runtime, by using logical device numbers 1, and 20 through 24 in the FORTRAN program. Logical device number 19 refers to logical device FORTR which must be assigned at runtime and accesses file name FORTR.DAT to maintain compatibility with the past system of default file name FORTR.DAT. In all cases when the operating system fails to find a file specified, an attempt will be made to read from file FORTR.DAT as before.

The magnetic tape operation REWIND is simulated on DECtape or the disk; a REWIND closes the file and clears the filename. A call to IFILE or OFILE should be made after a REWIND to open the file and preserve the filename, if desired. A program which uses READ, WRITE, END FILE, and REWIND for magnetic tape need only have the logical device number changed or assigned to a DECtape or disk at runtime in order to perform the proper input/output sequences on DECtape or the disk.

12.3.3 Magnetic Tape Usage

Magnetic tape and disk/DECtape I/O are different in the following ways. When a READ is issued, a record is read in for both magnetic tape and disk. If a WRITE is then issued, the next sequential record is written on

magnetic tape but not on disk. When one or more READs have been executed on a disk file and a WRITE is issued, the next record is written. Unless records are written past the existing end-of-file, that end-of-file is not changed, i.e., the file is not truncated.

12.3.3.1 Binary Mode - The format of binary data on magnetic tape is similar to that for DECtape except that the physical record size depends on the magnetic tape buffer size assigned in the Time-Sharing Monitor or by IBUFF/OBUFF (see Section 8.2.2). Normally, the buffer size is set at either 129 or 257 words so that either 128 or 256 word records are written (containing a control word and 127 or 255 FORTRAN data words).

The first word, control word, of each block in a binary record contains information used by the operating system. The left half of the first word contains the word count for that block. The right half of the first word contains a null character except for the last block in a logical record. In this case, the right half of the first word contains the number of blocks in the logical record.

12.3.3.2 ASCII Mode - The format for ASCII data is the same as that used on DECtape.

12.3.3.3 Backspacing and Skipping Records - Both the BACKSPACE u and SKIP RECORD u statements are executed on a logical basis for binary records and on a line basis for ASCII records.

a. Binary Mode - Both BACKSPACE and SKIP RECORD space magnetic tape physically over one (1) logical record; i.e., the result of one WRITE (u) statement.

b. ASCII Mode - ASCII records are packed, that is WRITE (u, f) statements do not cause physical writing on the tape until the output buffers are full or a BACKSPACE, END FILE, or REWIND command is executed by the program. BACKSPACE and SKIP RECORD on ASCII record space over one (1) line.

c. BACKSPACE and SKIP RECORD following WRITE ASCII commands.

(1) BACKSPACE closes the tape, writes 2 EOF's (tapemark) and backspaces over the last line.

(2) SKIP RECORD cannot be used during a WRITE operation. This is an input function only.

12.4 RANDOM ACCESS PROGRAMMING

In random access programming, data is obtained from (or placed into) storage, where the time required for this access is independent of the location of the data most recently obtained from (or placed into) storage. Random access programming allows a programmer to access any record within a file with a single READ or WRITE statement independent of the location of the previously accessed record within that file. For example, a programmer may read or write only the 10th record in a file if he wishes. Random I/O is desirable when only a few records in a large file are to be accessed, or when a file is to be read or wirtten in a non-sequential manner, as in a sort.

Random access applies only to data files on the disk with fixed-length record sizes. Any fixed-length record file (formatted or unformatted) which has been written on the disk with FORTRAN or with PIP using the A switch may be read or rewritten non-sequentially.

12.4.1 How to Use Random Access

A programmer may directly access fixed-length records in a disk file by defining the structure of the file with a CALL DEFINE FILE and then specifying the record he wishes to access with a READ or WRITE statement. The file from which records are to be accessed is defined as follows:

```
CALL DEFINE FILE (U,S,V,F,PJ,PG)
```

where

U = the unit number expressed as an integer. The number must refer to the disk. The numbers from 1 to 10 are available unless a particular installation decides to change this range.

S = the size of the records within the file expressed as an integer. The size is specified by the number of characters per record for formatted records, and the number of words per record for unformatted records. The size of the records must be constant within the file and may be from 1 to 132 characters in formatted records, or one word to any size limited by core in unformatted records.

V = the associated integer variable. Contains any integer value. The record number which would be accessed next if I/O were to continue sequentially is returned as an integer in the associated variable after each random read or write. The associated variable may be used in the I/O statements as part of the integer expression which defines the record number.

F = the filename and extension. This may be zero, in which case standard default names are used.

PJ = the project number in octal of the disk area being accessed.

PG = the programmer number in octal of the disk area being accessed. The project-programmer numbers may be zero, in which case the user's disk area is accessed. Note that the writing on another user's disk area is restricted by the monitor.

I/O begins when the random WRITE or READ is specified in the correct format. (See Sections 5.2.6 and 5.2.7.)

12.4.2 Restrictions

A number of restrictions are imposed in random access programming:

a. A logical unit may not be used for sequential and then random I/O in the same program unless an intervening CALL to RELEASE is issued. For example, if sequential I/O is done to unit 3 then random I/O to unit 3 is illegal and will fail.

b. If the name of a file to be accessed randomly is specified in a DATA statement or is read in at run-time, the user must use a full 6-character filename and a 3-character extension.

c. Mixed formatted and unformatted files are not accessible randomly.

d. Before random I/O is performed through a READ or WRITE statement, the file must be properly defined through a CALL to DEFINE FILE.

e. All FORTRAN data files must be created by FORTRAN or PIP with the A switch.

f. The records within the file must be of a fixed length.

g. Random access is used for disk files only.

h. Access to files is controlled by the file protection scheme in effect at each installation. (Refer to the Timesharing Monitors Manual for a discussion of file access privileges.)

i. CALLs to IFILE or OFILE open files for sequential I/O. They must not be issued for units to be used for random I/O. If it is desired to open a file for sequential I/O on a unit that has been used for random I/O, a CALL to RELEASE must be issued before a CALL to IFILE or OFILE.

12.4.3 Examples

Example 1:

Assume a standard FORTRAN program, the purpose of which is to read the Kth record in a file and ignore all other records. A section of the program might be as follows:

```
          .
          .
          DO 10 I=1, K
10        READ (1,1) A,B,C
1         FORMAT (3A5)
```

If K is a large number, time is wasted in obtaining the Kth record using sequential I/O. Now consider a program written to perform the same function using random access:

```
          CALL DEFINE FILE (1,15,N,0,0,0)
            .
            .
            .
          READ (1#K,1) A,B,C

1         FORMAT (3A5)
```

Note that the default filename FOR01.DAT and the user's project-programmer number are used in both examples.

Example 2:

Consider a program the purpose of which is to change the contents of the Kth record within the file FOR01.DAT on the user's disk area. Using sequential I/O, the code might be as follows:

```
                    .
                    .
                    .
          DO 10 I=1, K-1
          READ (1,1) A,B,C
10        WRITE (2,1) A,B,C
          READ (1,1) A,B,C
          WRITE (2,1) D,E,F
          DO 20 I=K+1,NEND
          READ (1,1) A,B,C
20        WRITE (2,1) A,B,C
1         FORMAT (3A5)
            .
            .
            .
```

There would be two files on the disk, FOR01.DAT and FOR02.DAT, which are identical except for the Kth record. The code that accomplishes the same result using random access is:

```
      CALL DEFINE FILE (1,15,N,0,0,0)

      WRITE (1#K,1) D,E,F
1     FORMAT (3A5)
        .
        .
        .
```

A new file is not created; the old file remains with the Kth record changed.

Example 3:

The following code creates a new file for random output by first writing K blank records and then updating the file in non-sequential output:

```
C     40 SPACES PER RECORD USERS
C     NEED NO WORRY ABOUT CARRIAGE
C     RETURNS AND LINE FEEDS.

      DIMENSION A(8), B(8)
      DO 10 I=1,K
10    WRITE (1,1) A
        .
        .
        .
      CALL DEFINE FILE (2,40,N,'FOR01.DAT',0,0)
      N=3
      DO 20 I=1,5
20    WRITE (2#N*8,2) B
1     FORMAT (8A5)
2     FORMAT (4I5,2A5,F10.3)
        .
        .
        .
```

Example 4:

Read a 1000 record file, the records of which are 27 characters long, backwards. The file is named FOR01.DAT and resides on the user's disk area. The following program creates a disk file and then reads it backwards. (Note that the same unit number may not be used for both sequential and random I/O in the same program):

```
          DIMENSION A(6)
          CALL DEFINE FILE (2,27,NV,'FOR01.DAT',0,0)
          DO 10 I=1, 1000
10        WRITE (1,1) I
          REWIND (1)
1         FORMAT ('THIS IS RECORD NUMBER', I5)
          NV=1002
          DO 20 I=1,1000
20        READ (2#NV-2,2) A
2         FORMAT(5A5,A2)
          END
```

Example 5:

Use random WRITES to change every 7th record, beginning with record 10, in the file named DATA on the user's disk area. The file contains 100 records, each of which is 35 characters long.

```
          DIMENSION LIST(7)
          CALL DEFINE FILE(5,35,NV,'DATA',0,0)
          DO 10 I=10,100,7
10        WRITE (5#I,5) LIST
5         FORMAT (2A5,5I5)
          END
```

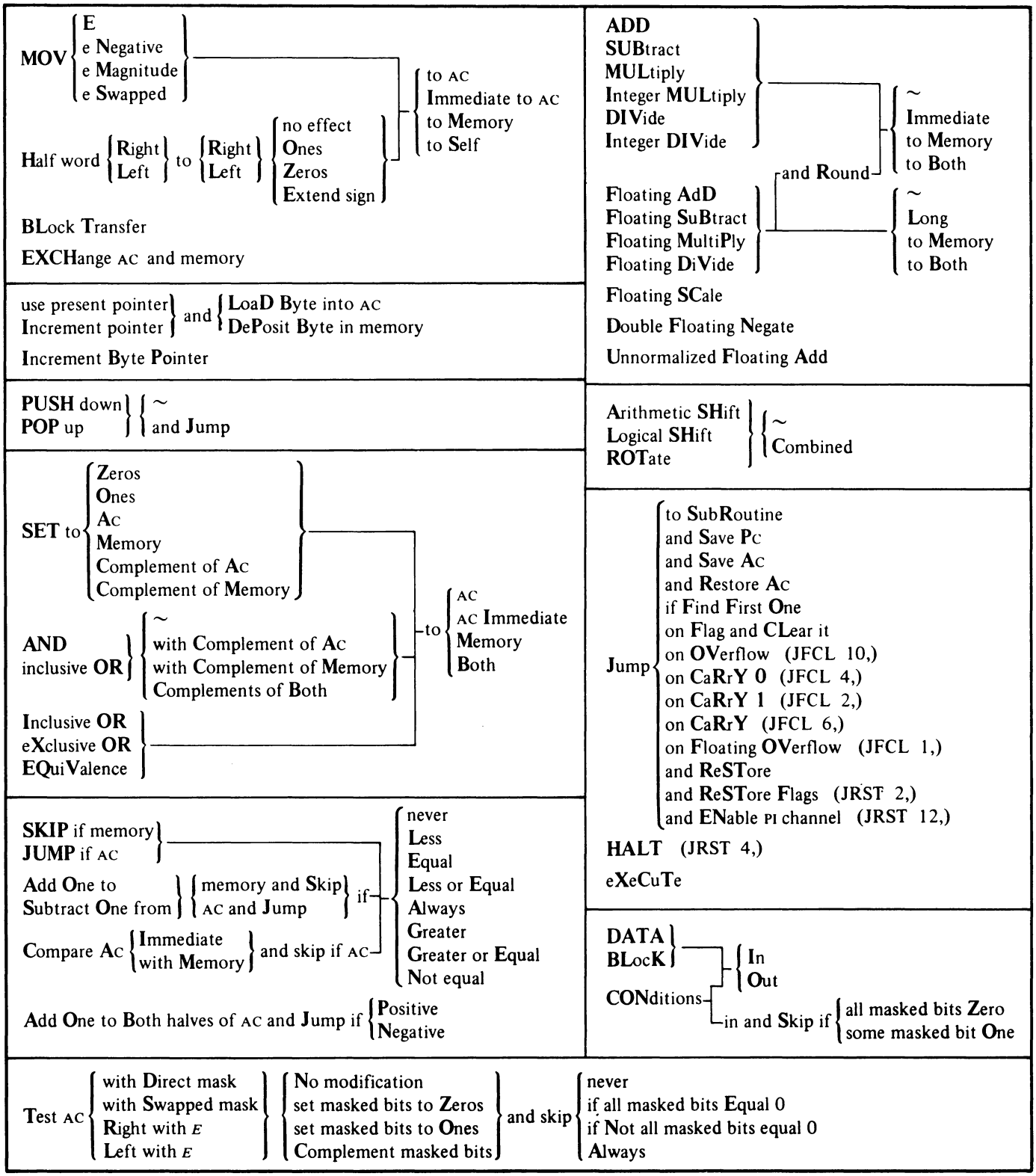
Example 6:

Read one-word binary records, starting with record 26 and ending with record 7, from file FOR07.DAT. The following program creates a 50-record file of the numbers from 1 to 50, reads the file backwards, and types the contents of the record it read, NP, along with the contents of the associated variable, NV. Note that FORTRAN binary output creates files with a maximum of one record per disk block.

```
.TY RINTST
C         BINARY RANDOM ACCESS TEST
C
          DOUBLE PRECISION FIL
          DATA FIL /'FOR07.DAT'/
          CALL DEFINE FILE (2,1,NV,FIL,0,0)
          DO 7 I=1,50
          WRITE(7)I
7         CONTINUE
          END FILE (7)
          NV=28
          DO 2 I=1,20
          READ(2#NV-2)NP
          WRITE(5,5)NP,NV
2         CONTINUE
5         FORMAT(' NP= ',I3,' NV= ',I3)
          END
```

```
RUN DSK BINTST

NP=  26 NV=  27
NP=  25 NV=  26
NP=  24 NV=  25
NP=  23 NV=  24
NP=  22 NV=  23
NP=  21 NV=  22
NP=  20 NV=  21
NP=  19 NV=  20
NP=  18 NV=  19
NP=  17 NV=  18
NP=  16 NV=  17
NP=  15 NV=  16
NP=  14 NV=  15
NP=  13 NV=  14
NP=  12 NV=  13
NP=  11 NV=  12
NP=  10 NV=  11
NP=   9 NV=  10
NP=   8 NV=   9
NP=   7 NV=   8
```


MOV
E
e Negative
e Magnitude
e Swapped
to AC
Immediate to AC
to Memory
to Self
Half word
Right
Left
to
Right
Left
no effect
Ones
Zeros
Extend sign
BLock Transfer
EXCHange AC and memory
ADD
SUBtract
MULtiply
Integer MULtiply
DIVide
Integer DIVide
and Round
~
Immediate
to Memory
to Both
Floating AdD
Floating SuBtract
Floating MultiPly
Floating DiVide
~
Long
to Memory
to Both
use present pointer
Increment pointer
and
LoaD Byte into AC
DePosit Byte in memory
Increment Byte Pointer
Floating SCale
Double Floating Negate
Unnormalized Floating Add
PUSH down
POP up
~
and Jump
Arithmetic SHift
Logical SHift
ROTate
~
Combined
SET to
Zeros
Ones
Ac
Memory
Complement of Ac
Complement of Memory
AND
inclusive OR
~
with Complement of Ac
with Complement of Memory
Complements of Both
to
AC
AC Immediate
Memory
Both
Inclusive OR
eXclusive OR
EQuiValence
Jump
to SubRoutine
and Save Pc
and Save Ac
and Restore Ac
if Find First One
on Flag and CLear it
on OVerflow (JFCL 10,)
on CaRrY 0 (JFCL 4,)
on CaRrY 1 (JFCL 2,)
on CaRrY (JFCL 6,)
on Floating OVerflow (JFCL 1,)
and ReSTore
and ReSTore Flags (JRST 2,)
and ENable PI channel (JRST 12,)
HALT (JRST 4,)
eXeCuTe
SKIP if memory
JUMP if AC
Add One to
Subtract One from
memory and Skip
AC and Jump
if
Compare Ac
Immediate
with Memory
and skip if AC
never
Less
Equal
Less or Equal
Always
Greater
Greater or Equal
Not equal
Add One to Both halves of AC and Jump if
Positive
Negative
DATA
BLocK
In
Out
CONditions
in and Skip if
all masked bits Zero
some masked bit One
Test AC
with Direct mask
with Swapped mask
Right with E
Left with E
No modification
set masked bits to Zeros
set masked bits to Ones
Complement masked bits
and skip
never
if all masked bits Equal 0
if Not all masked bits equal 0
Always

APPENDIX A
THE SMALL FORTRAN IV COMPILER

This compiler runs in 5.5K of core, and to the user, is identical to the large compiler, with the exception of the following language differences. Operating procedures are given in the Systems User's Guide (DEC-10-NGCC-D).

Language Differences

The IMPLICIT, DATA, and NAMELIST statements are not recognized; constant strings are not collapsed (for example, A=5*3 will not be treated as A=15).

INDEX

ACCEPT Statement, 5-1, 5-20, 7-7

Accumulator, 10-1

Accumulator conventions, 10-1

Adjustable dimensions, 6-3

A format, 5-6

Alphanumeric fields, 5-6

ALPHI., 8-2

ALPHO., 8-2

Argument, def., 2-6

Arithmetic error conditions, 11-10

Arithmetic function definition statement, 7-1, 7-9

Arithmetic operations on complex numbers, 2-3

Arithmetic statement, 3-1

Array dimensioning, 2-4, 6-2

Array variables, 2-4

ASCII character set, 12-1

ASCII mode

- DECtape, 12-4
- disk, 12-4
- magnetic tape, 12-7

Assigned GO TO statement, 4-2, 7-7

ASSIGN Statement, 4-2, 7-7

BACKSPACE statement, 5-1, 5-20, 7-7

Binary mode

- DECtape, 12-4
- disk, 12-4
- magnetic tape, 12-7

BINWR., 8-3

Blank common, 6-4

Blank fields, 5-11

Blank records, 5-9

BLOCK DATA statement, 6-8, 7-6, 7-8

BLOCK DATA subprogram, 7-6

Block identifier, 6-4

Block name, 6-4

Buffer, 12-3

- sizes, 12-3

BUFFER subroutine, 8-8

CALL statement, 7-5, 7-7

Carriage control, 5-7, 5-10

Chain files, 8-8

CHAIN subroutine, 8-8

Character set, 1-3, 12-1

Closed subroutines, 7-1

Coding form, 1-2

Comment line, 1-3

Common block, 6-4

COMMON statement, 6-4, 6-6, 7-8

Common storage, 6-4

Compiler diagnostics

- command errors, 11-2
- compilation errors, 11-3

Compiler switches, 11-1

COMPLEX (type declaration statement), 6-8, 7-8

Complex constants, 2-2

Complex fields, 5-8

Complex subexpression, 2-7

Compound expressions

- logical, 2-10
- numeric, 2-7

Computed GO TO statement, 4-1, 7-7

Constants

- integer, 2-1
- real, 2-1
- double precision, 2-2
- octal, 2-2
- complex, 2-2
- logical, 2-3
- literal, 2-3

CONTINUE statement, 4-6, 7-7

Control statements, 4-1, 7-7

CALL, 7-5
GO TO, 4-1
IF, 4-2
DO, 4-4
RETURN, 7-5
CONTINUE, 4-6
PAUSE, 4-6
STOP, 4-7
END, 4-7

Data record, 5-15

DATA statement, 6-6, 7-8

Data specification statements, 6-1

DATA, 6-6, 7-8
BLOCK DATA, 6-8, 7-6, 7-8

Data specification subprogram, 6-8

Data transmission statements, 5-1, 7-7

ACCEPT, 5-20
DECODE, 5-21
ENCODE, 5-21
PRINT, 5-15
PUNCH, 5-16
READ, 5-18
REREAD, 5-19
TYPE, 5-16
WRITE, 5-16

DATA. UUO, 8-3

DATE subroutine, 8-8

DECODE statement, 5-21, 7-7

DECtape usage, 12-4

DEC. UUO, 8-4

Defined locations, 10-1

DEFINE FILE, 5-17, 12-8

Device assignments, 12-4

Device control statements, 5-20, 7-7, 7-8

BACKSPACE, 5-20
END FILE, 5-20
REWIND, 5-20
SKIP RECORD, 5-20
UNLOAD, 5-20

Device table, 12-5

DEVTB., 12-4

D format, 5-2 - 5-6

Diagnostic messages

command, 11-2
compilation, 11-3
execution, 11-8

DIMENSION statement, 6-2, 7-8

adjustable dimension, 6-3

DIRT., 8-2

Disk usage, 12-4

DO loops, 4-4

DO statement, 4-4, 7-7

DOUBLE PRECISION (type declaration statement), 6-8, 7-9

Double precision constants, 2-2

Double word, 2-7, 2-9

DOUBT., 8-2

Dummy arguments, 7-2, 7-3

Dummy identifiers, 7-1, 7-2

DUMP, 8-9

E format, 5-2 - 5-6

ENCODE statement, 5-21, 7-8

ENC. UUO, 8-4

END FILE statement, 5-20

END statement, 4-7

EOF1 subroutine, 8-9

EOFC subroutine, 8-9

EQUIVALENCE statement, 6-5

ERRSET subroutine, 8-9

EXIT subroutine, 8-9

Expressions, 2-6

numeric, 2-6
logical, 2-8

EXTERNAL statement, 7-6, 7-9

External subprograms, 7-1

F format, 5-2 - 5-6

Field delimiters, 5-6

Field specifications, 5-2

Field width, 5-2 - 5-6

FIN. UUO, 8-3

FLIRT., 8-2

FLOUT., 8-2

Formats stored as data,5-9

FORMAT statement, 5-1

- numeric fields, 5-2
- logical fields, 5-5
- variable field width, 5-5
- alphanumeric fields, 5-6
- mixed fields, 5-7
- complex fields, 5-8
- multiple records, 5-8
- blank fields, 5-11

FORSE., 8-1

- format processing, 8-1
- I/O device control, 8-2
- UUO dispatching, 8-2

FORTRAN operating system, 8-1

- FORSE., 8-1
- I/O conversion routines, 8-2
- FORTRAN UUO's, 8-3

FORTRAN program and MACRO subprogram linkage, example of, 9-3

FORTRAN UUOs, 8-3

Function, def., 2-6

Function identifier, 2-6, 7-2

FUNCTION statement, 7-2

FUNCTION subprograms, 7-2

- FUNCTION statement, 7-2

Function subprogram linkage, example of, 9-3

Function type, 2-6, 7-3

Function value, 2-6

G Format, 5-2 - 5-6

GO TO statement

- assigned, 4-2, 7-7
- computed, 4-1, 7-7
- unconditional, 4-1, 7-7

H-conversion, 5-6, 5-7

Hierarchy

- of numeric operators, 2-8, 2-10
- of logical operators, 2-9
- of relational operators, 2-9

IBUFF, 8-8

IFILE subroutine, 8-9

I format, 5-2 - 5-6

IF statement

- logical, 4-3, 7-7
- numerical, 4-3, 7-7

ILL subroutine, 8-10

IMPLICIT statement, 6-9, 7-9

INF. UUO, 8-3

Instruction set, 12-13

INTEGER (type declaration statement), 6-8, 7-9

Integer constants, 2-1, 6-6

Internal subprograms, 7-1

INTI., 8-2

INTO., 8-2

IN. UUO, 8-3

I/O conversion routines, 8-2

I/O list, 5-14

I/O records, 5-15

LEGAL subroutine, 8-10

L format, 5-5

LIB40, 8-1

Library functions, 8-4

Library subprograms, 7-1, 8-4

Library subroutines, 8-8

Line continuation field, 1-1

Line format, 1-1

Line spacing, 5-10

LINT., 8-2

Literal constants, 2-3

Locations

defined, 10-1
required, 10-1

LOGICAL (type declaration statement), 6-8, 7-9

Logical constants, 2-3

Logical expressions, 2-8

Logical fields, 5-5

Logical IF statement, 4-3, 7-7

Logical operators, 2-9, 2-10

LOOPS, DO, 4-4

LOUT., 8-2

MACRO main programs, 9-10

MACRO subprograms, 9-1

MAGDEN subroutine, 8-10

Magnetic tape usage, 12-6

Magnitude

of integer constants, 2-1
of real constants, 2-1
of double-precision constants, 2-2

Mixed fields, 5-7

MTOP. UUO, 8-3

Multiple record formats, 5-8

termination of, 5-9

NAMELIST statement, 5-1, 5-11, 7-9

input data, 5-12
output data, 5-13

Nested DO Loops, 4-4, 4-5

NLI. UUO, 8-3

NLO. UUO, 8-4

NMLST., 8-3

Non-executable statements

FORMAT statement, 5-1
NAMELIST statement, 5-11

Normal exit of a DO statement, 4-4

Numeric expressions, 2-6

Numeric fields, 5-2

repetition of, 5-8
repetition of groups, 5-8

Numeric IF statement, 4-3, 7-7

Numeric operations, 2-8

Numeric operators, 2-6

OBUFF, 8-8

Octal constants, 2-2

OCTI., 8-3

OCT., 8-3

OFILE subroutine, 8-10

O format, 5-2 - 5-6

Open subroutines, 7-1

Operating system diagnostics, 11-8

Operators

logical, 2-9
numeric, 2-6
relational, 2-9
priorities of, 2-10

OUT. UUO, 8-3

OUTF. UUO, 8-3

PAUSE statement, 4-6, 7-7

PDUMP subroutine, 8-10

Precision

of double-precision constants, 2-2
of real constants, 2-1

PRINT statement, 5-15, 7-8

Priorities of operators, 2-8, 2-10

PUNCH statement, 5-16, 7-8

Range of a DO statement, 4-5

Random access of records, 12-7

WRITE, 5-17, 7-8
READ, 5-18, 7-8

READ statement, 5-17, 7-8

REAL (type declaration statement), 6-8, 7-9

Real constants, 2-1

Relational operators, 2-9

RELEAS subroutine, 8-10

Repetition

of field specifications, 5-8
of groups, 5-8

Replacement operator, 3-1

Required locations, 10-1

REREAD statement, 5-19, 7-8

RERED. UUO, 8-3

RESET. UUO, 8-3

RETURN statement, 7-5, 7-7

REWIND statement, 5-20, 7-8

RTB. UUO, 8-3

SAVRAN subroutine, 8-10

Scalar variables, 2-4

Scale factor, 2-1, 2-2, 5-4

SETRAN subroutine, 8-10

SKIP RECORD statement, 5-20, 7-8

SLIST. UUO, 8-3

SLITE subroutine, 8-11

SLITET subroutine, 8-11

Spacing, 5-10a

Specification statements, 6-1, 7-8

data specification, 6-6
storage specification, 6-2
type declaration, 6-8

SSWTCH subroutine, 8-11

Statement number field, 1-1

Statement numbers, 1-1

Statement field, 1-2

STOP statement, 4-7, 7-7

Storage specification statements, 6-2

COMMON, 6-4
DIMENSION, 6-2
EQUIVALENCE, 6-5

Stored formats, 5-9

SUBSCRIPT INTEGER (type declaration statement), 6-8, 7-9

Subprogram calling sequences, 9-1

Subprogram linkage, example of, 9-2

Subroutine linkage, example of, 9-2

SUBROUTINE statement, 7-4

Subroutine subprograms, 7-4

SUBROUTINE statement, 7-4
CALL statement, 7-5
RETURN statement, 7-5

Symbolic logic, 2-8

Tab, horizontal, 1-1

Termination of a program, 4-7

T format, 5-10a

TIME subroutine, 8-11

Type declaration statements, 6-8, 7-8

TYPE statement, 5-16, 7-8

Unconditional GO TO statement, 4-1, 7-7

Unit records, 5-2

UNLOAD statement, 5-20, 7-8

Variable field width, 5-5

Variables

Scalar, 2-4
Array, 2-4

Word format, 12-2

WRITE statement, 5-16, 7-8

WTB. UUO, 8-3

X format, 5-11