

402
(NCAR-TN/IA-70) October 1975
NCAR TECHNICAL NOTE

Revision 3 (April 1978) 501

FORTRAN Reference Manual 301

Revision 3

J. C. Adams 201
B. A. Domenico 202
P. A. Rotar 203



ATMOSPHERIC TECHNOLOGY DIVISION

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
BOULDER, COLORADO

MOI

FOREWORD

This manual describes the FORTRAN language as interpreted by the NCAR compiler. Portions of this manual have been edited from Control Data copyrighted publications with the permission of the Control Data Corporation. New sections have been added and all of the examples have been checked using computer runs. Sample programs have been run on the NCAR system.

NCAR FORTRAN uses the American National Standards Institute standard FORTRAN X3.9-1966 as a guide in order to have the current standard a subset of the NCAR FORTRAN as far as possible. However, there are a number of cases where its usage is nonstandard. Some of these have been noted. Chapter 10 contains a statement list for FORTRAN X3.9-1966, the new standard (FORTRAN 77) currently in the final release process, NCAR Control Data 7600 FORTRAN, and CRAY FORTRAN.

Astrik Deirmendjian was the programmer responsible for testing the examples. David Kitts and Sandra Fuller, who are responsible for compilers at NCAR, were very helpful in interpretation of technical material. John Snyder was the final reader for this issue. Linda Besen and Sara Ladd did the layout work and prepared the final copy. Linda generated the index based on the execution of a KWIC index program using topical subtitles. Veronica Martinez assisted the editors.

This document is the third revision of Technical Note TN/IA-70 and supersedes the FORTRAN *Reference Manual* dated October 1975.

February 1978

TABLE OF CONTENTS**CHAPTER 1**

FORTRAN CODING FORMS	1.1
Introduction	1.1
Coding Line	1.2
Statement	1.2
Statement Label	1.3
Identification Field	1.3
Comment Information	1.3
Special Comment Option	1.4
Punched Cards	1.4
Blank Cards	1.4

CHAPTER 2

FORTRAN CONCEPTS	2.1
FORTRAN Character Set	2.1
Identifiers	2.2
Alphanumeric Identifier	2.2
Constants	2.4
Integer Constants	2.4
Octal Constants	2.6
Real Constants	2.7
Double-Precision Constants	2.8
Complex Constants	2.10
Hollerith Constants	2.11
Logical Constants	2.12
Variables	2.13
Types of Variables	2.14
Integer Variables	2.14
Real Variables	2.15
Double-Precision Variables	2.15
Complex Variables	2.15
Naming Octal Constants	2.15
Alphanumeric Names (Hollerith)	2.16
Logical Variables	2.16
Classes of Variables	2.16
Simple Variable	2.16
Subscripted Variable	2.17
Arrays	2.19
Array Structure	2.20

CHAPTER 3	EXPRESSIONS: ARITHMETIC, LOGICAL, MASKING, MIXED MODE	3.1
	Introduction	3.1
	Expressions	3.2
	Arithmetic Expressions	3.2
	Standard Evaluation	3.3
	Evaluation with Optimization	3.6
	Logical Expressions	3.11
	Relational Operators	3.11
	Logical Connectives	3.13
	Masking Expressions	3.14
	Mixed-Mode Expressions	3.17
CHAPTER 4	REPLACEMENT STATEMENTS	4.1
	Introduction	4.1
	Mixed-Mode Replacement Statements	4.1
	Multiple Replacement Statements	4.4
CHAPTER 5	TYPE DECLARATIONS AND STORAGE ALLOCATION	5.1
	Introduction	5.1
	Statement Ordering	5.1
	Type Declaration	5.2
	Implicit Typing	5.4
	IMPLICIT Statement	5.4
	Symbolic Name of a Constant	5.7
	PARAMETER Statement	5.7
	Dimension Declaration	5.10
	Variable Dimensions	5.13
	Common Declaration	5.14
	Local Variables	5.14
	Common Variables	5.15
	Equivalence Declaration	5.24
	Storage Allocation	5.28
	Data Declaration	5.35
	DATA Statement	5.35
CHAPTER 6	CONTROL STATEMENTS	6.1
	Introduction	6.1
	GO TO Statements	6.2
	Unconditional GO TO Statement	6.2
	Assigned GO TO Statement	6.2
	ASSIGN Statement	6.3
	Computed GO TO Statement	6.3
	IF Statements	6.5
	Three-Branched Arithmetic IF Statement	6.5
	Two-Branched Relational IF Statement	6.7
	One-Branched Relational IF Statement	6.8
	DO Statement	6.9
	DO Loop Execution	6.13

DO Nests	6.15
Subscripting within DO Loops	6.18
Extended Range	6.18
Optimization	6.21
*FORTRAN,1	6.21
*FORTRAN,2	6.22
*FORTRAN,3	6.23
*FORTRAN,4	6.23
*FORTRAN,13	6.24
*FORTRAN,18	6.24
*FORTRAN,25	6.24
*FORTRAN,60	6.24
CONTINUE Statement	6.26
PAUSE Statement	6.26
END Statement	6.26
Normal Termination Statements	6.27
IF/THEN/ELSE	6.28
General Structure	6.28
Nesting Levels	6.29
Transfer of Control	6.29
IF THEN Statement	6.30
Form	6.30
Block	6.30
Execution	6.30
ELSE IF Statement	6.31
Form	6.31
Block	6.31
Execution	6.31
ELSE Statement	6.32
Form	6.32
Block	6.32
Execution	6.32
END IF Statement	6.33
Form	6.33
Execution	6.33
DO-Loop (FORTRAN 77)	6.37
DO-Loop Structure	6.37
DO Statement	6.37
DO-Loop Range	6.37
Terminal Statement	6.38
DO-Loop Execution	6.39
DO Statement Execution	6.40
Loop Control Processing	6.41
Execution of the Range	6.41
Incrementation Processing	6.41
Active and Inactive DO-Loops	6.42

CHAPTER 7

PROGRAM, FUNCTION, AND SUBROUTINE	7.1
Introduction	7.1
Main Program	7.2
Subprograms	7.3
Formal Parameters	7.3
Actual Parameters	7.4
Subroutine Subprogram	7.7
CALL Statement	7.8
Function	7.11
Function Definition	7.11
Function Reference	7.13
Arithmetic Statement Function	7.15
Library Functions	7.18
In-Line Algorithms	7.18
Mathematical Library Routines	7.18
Auxiliary Library Routines	7.25
Block Data	7.26
RETURN and END Statements	7.27
ENTRY Statement	7.28
EXTERNAL Statement	7.31
Variable Dimensions in Subprograms	7.33
Arrangement of FORTRAN Decks	7.35

CHAPTER 8

INPUT/OUTPUT FORMATS	8.1
Introduction	8.1
Input/Output List	8.2
Standard Subscripts	8.2
Nonstandard Subscripts	8.3
Array Transmission	8.3
FORMAT Declaration	8.7
FORMAT Specifications	8.8
Conversion Specifications	8.9
Ew.d Input	8.9
Ew.d Output	8.12
Fw.d Input	8.14
Fw.d Output	8.14
Gw.d Input	8.16
Gw.d Output	8.16
Dw.d Input	8.18
Dw.d Output	8.18
Iw Input	8.19
Iw Output	8.20
Ow Input	8.21
Ow Output	8.22
Aw Input	8.23
Aw Output	8.23
Rw Input	8.24
Rw Output	8.24
Lw Input	8.25
Lw Output	8.25

Scale Factor	8.26
Fw.d Scaling	8.27
Input	8.27
Output	8.27
Ew.d or Fw.d Scaling	8.28
Output	8.28
Editing Specifications	8.29
wX Specification	8.29
wH Input	8.30
wH Output	8.31
New Record	8.32
Repeated FORMAT Specifications	8.34
Unlimited Groups	8.35
Variable Format	8.36
CHAPTER 9	
INPUT/OUTPUT STATEMENTS	9.1
Data Transfer	9.1
Units	9.2
Source Data	9.2
The I/O List	9.3
Rules	9.3
Input/Output Terminology	9.4
Synchronous and Asynchronous	9.4
Core-to-Core Transfer	9.4
Access to Data	9.5
Keywords and Keylists	9.5
Formatted and Unformatted	9.6
Reading and Writing	9.7
Generalized Form of Synchronous Read/Write	9.8
Format	9.8
Input Parameters	9.9
Output Parameters	9.11
Syntax Error Diagnostics	9.11
Specific Forms of Synchronous I/O	9.12
Specific Output Statements Under Format Control	9.12
Unformatted Input Statements	9.15
Unformatted Output Statements	9.17
Structure of Logical Records	9.18
Generalized Form of Asynchronous READ/WRITE	9.19
Format	9.19
Input Parameters	9.20
Output Parameters	9.21
Syntax Error Diagnostics	9.22
Alternate Form for Asynchronous Statements	9.23
Parameters	9.24
Record Structure for BUFFER OUT in Binary or Character Code	9.25
Status Statement	9.25
Internal Files	9.27
ENCODE/DECODE Statements	9.27

CHAPTER 9 <i>(continued)</i>	Dataset Support Statements	9.38
	The OPEN Statement	9.38
	Form	9.38
	Ident Parameters	9.39
	Error Condition Statement Label	9.40
	Keylist Parameters	9.40
	Return Parameter	9.42
	Syntax Error Diagnostics	9.43
	The CLOSE Statement	9.44
	Form	9.44
	Input Parameters	9.44
	Error Condition Statement Label	9.45
	Return Parameter	9.45
	Syntax Error Diagnostics	9.45
	The INQUIRE Statement	9.46
	Form	9.46
	Ident Parameters	9.46
	Error Condition Statement Label	9.47
	Keylist Parameters	9.48
	Syntax Error Diagnostics	9.54
	The SETPASS Statement	9.55
	Form	9.55
	Parameters	9.55
	Syntax Error Diagnostics	9.57
	File Positioning with Sequential Access	9.58
	Positioning Statements	9.60
	Example Using Sequential Access	9.65
	Comments	9.66
	Example Using Direct Access	9.67
	Comments	9.67
	Example Using Direct and Sequential Access on the Same Unit	9.68
	Comments	9.68
	Example of User Catalog Program	9.69
	Comments	9.69
	Summary Table of FORTRAN Syntax	9.73
CHAPTER 10	FORTRAN STATEMENT LIST	10.1
	Introduction	10.1
	Program and Subprogram Statements	10.2
	Subprogram Reference and Transfer	10.2
	Arithmetic Statement Function	10.2
	Type Declaration	10.2
	Storage Allocation	10.4
	Replacement	10.4
	Branching and IF Logic	10.4
	Looping and Control Statements	10.6
	FORMAT	10.6
	Data Input	10.6

Data Output	10.6
File Positioning	10.8
Core to Core Transfer	10.8
Dataset Support	10.8
CHAPTER 11	
DIAGNOSTICS	11.1
Compilation Diagnostics	11.1
Errors Fatal to Compilation	11.5
Errors Fatal to Execution	11.7
Nonfatal Errors	11.16
Compiler Table Size Limits	11.18
CHAPTER 12	
Appendices	12.1
Appendix A: 7600 Series Character Set	12.2
Appendix B: Table of Powers of Two	12.4
Appendix C: Octal-Decimal Integer Conversion Table . . .	12.5
Appendix D: Internal Floating-Point Decimal Table . . .	12.9
Appendix E: Decimal/Binary Position Table	12.10
Appendix F: Computer Word Structure of Constants-- 7600	12.11
Appendix G: Computer Word Structure of Constants-- CRAY-1	12.12
Appendix H: FORTRAN Compilation Modifiers	12.13
CHAPTER 13	
INDEX	13.1

I

FORTRAN CODING FORMS

FORTRAN CODING FORMS

INTRODUCTION

FORTRAN is a second-level programming language commonly used with computers for solving scientific problems. It consists of statements which may be associated easily with problems set down in mathematical notation. Programs written in a second-level language usually require a systems program called a *compiler*, which scans the statements presented and reformulates these statements in machine terms. The FORTRAN language compiler described in this manual is for the Control Data 7600 computer at the National Center for Atmospheric Research (NCAR).

A *source program* written in FORTRAN is an ordered set of statements in the FORTRAN language, from which machine instructions, as well as storage areas, are defined.

After the compiler finishes the reformulation of a FORTRAN program, the resulting *machine program*, or *object program*, is available to the computer for execution.

1.2

FORTRAN Coding Forms

CODING LINE

A coding line contains 80 columns, in which characters are written one character per column. A FORTRAN line is a string of 72 characters from the NCAR FORTRAN character set.

There are four types of coding lines:

Statement	1-5 6 7-72 73-80	Statement label (optional) Blank or zero FORTRAN statement Identification field (optional)
Continuation	1-5 6 7-72 73-80	Blank FORTRAN character other than blank or zero Continued FORTRAN statement Identification field (optional)
Comment	1 2-80	Character C Comments
Data	1-80	Data

STATEMENT

The FORTRAN statement is written in columns 7-72. Statements longer than 66 columns may be carried to the next line by using a continuation line. Blanks may be used freely in FORTRAN statements to provide readability; they are ignored by the FORTRAN compiler. The character \$ may be used to write more than one statement on a coding line, but cannot be used as a separator with FORMAT or DATA statements. The \$ indicates that the next column will be interpreted as column 7.

The first line of every statement is called an initial line and must have a blank or zero in column 6. If statements occupy more than one line, all subsequent lines are continuation lines and must have a number or letter other than blank or zero in column 6. A continuation line is *not* a comment.

STATEMENT LABEL

A statement label is a string of one to five digits occupying any column position 1 through 5 of the initial coding line of the statement. This label is used to identify a particular FORTRAN statement. Any statement may have a statement label, but only statements referred to elsewhere in the program require identifying statement labels. The same label may be given to only one statement within a program unit.

IDENTIFICATION FIELD

Columns 73 through 80 may be used for identification when the FORTRAN program is to be punched on cards. The identification columns are always ignored by the compiler. Usually these columns contain sequencing information provided by the programmer. They may be left blank.

COMMENT INFORMATION

Comment information is designated by a C in column 1 of a coding line. Comment information appears in the source program and the source program listing, but it is not translated by the compiler. Comment lines may contain any character in the set for the Control Data 7600 computer. Each line must have the character C in column 1. The continuation character in column 6 does not apply to comment cards. Comments do not affect the program in any way.

**SPECIAL COMMENT
OPTION**

A special comment card has a C in column 1 and a period in column 2. Only C. comment cards will be listed when the FORTRAN list output default option is selected, (see page 12.22 in Chapter 12). This option is included to convey special information to NCAR library routine users, even when FORTRAN cards are not listed.

PUNCHED CARDS

Each line of the coding form corresponds to one 80-column card in the program deck. The terms *line* and *card* are often used interchangeably. Source programs and data can be read from cards into the computer; some output from the computer--relocatable binary programs or data--also can be punched directly onto cards.

BLANK CARDS

If a blank card appears between two statements, it is ignored. When cards are being used for data input, all 80 columns may be used.

FORTRAN CODING FORM		PROGRAM NO.			
PROGRAM	SAMPLE	PROGRAMMER J. Adams	DATE	77 80	
Statement	EOO	FORTRAN STATEMENT			CARD NO.
1 2 Mo4 5		7 8 9 10 1 2 3 4 15 6 7 8 9 20 1 2 3 4 25 6 7 8 9 30 1 2 3 4 35 6 7 8 9 40 1 2 3 4 45 6 7 8 9 50 1 2 3 4 55 6 7 8 9 60 1 2 3 4 65 6 7 8 9 70 1 2	73 75 76		
	P R, Ø G, R, A, M, S, A, M, P	L, E, , P, R, Ø, G, R, A, M, T, O, , S, H, Ø, W, C, Ø, D, I, N, G, L, I, N, E, F, Ø, R, M, A, T, S			0
C, S, A, M		D, I, M, E, N, S, I, Ø, N, M, A, T, (, 2, 5, , 2, 5,),			1
	R, E, A, L, M, A, T				2
	P, R, I, N, T, 1, 0, 5				3
, , 1, 0, 5	F, Ø, R, M, A, T, (, * 1, T, H, I, S, L, I, N, E, S, H, Ø, W, S, T, H, A, T, C, Ø, N, T, I, N, U, A, T, I, Ø, N, Ø, F, A, F, Ø, R, T, R, A, N, S, E, N, T, E, N, C, E,				4
	M, A, Y, B, E, D, Ø, N, E, W, I, T, H, A, N, Ø, N, -Z, E, R, Ø, P, U, N, C, H, I, N, C, Ø, L, 6, *,)				5
1	R, E, A, D, (, 5, , 1, 0, 0,) N, (, (, M, A, T, (, I, , J,), , I, =, 1, , N,), , J, =, 1, , N,)				6
5	F, Ø, R, M, A, T, (, I, 1, 0, , /, (, 8, E, 1, 0, , 0,),)				7
1, 0, 0	L, =, 1, , \$, D, E, T, =, 1, ., 0,				8
	D, Ø, 6, K, =, 2, , N				9
	D, Ø, 1, 0, I, =, K, , N				10
	R, A, T, =, M, A, T, (, I, , L,), /, M, A, T, (, L, , L,)				11
	D, Ø, 1, 0, J, =, K, , N				12
1, 0	M, A, T, (, I, , J,), =, M, A, T, (, I, , J,), -M, A, T, (, L, , J,), *R, A, T				13
	L, =, K,				14
6	C, Ø, N, T, I, N, U, E,				15
	D, Ø, 2, 0, I, =, 1, , N				16
	D, Ø, 5, 0, J, =, 1, , N				17
5, 0	W, R, I, T, E, (, 6, , 1, 0, 2,), (, M, A, T, (, K, , J,), , K, =, 1, , N,)				18
1, 0, 2	F, Ø, R, M, A, T, (, 1, H, 1, 2, E, 1, 0, , 2,)				19
2, 0	D, E, T, =, D, E, T, *M, A, T, (, I, , I,)				20
	W, R, I, T, E, (, 6, , 1, 0, 1,), D, E, T,				21
1, 0, 1	F, Ø, R, M, A, T, (, 1, H, 0, *D, E, T, E, R, M, I, N, A, N, T, E, Q, U, A, L, S, *E, 1, 5, , 4,)				22
	G, Ø, T, Ø, 5				23
	E, N, D,				24
					25

Example of FORTRAN Coding Form

II

FORTRAN CONCEPTS

IDENTIFIERS

CONSTANTS

VARIABLES

FORTRAN CONCEPTS

FORTRAN CHARACTER SET

The following characters may be used in writing a FORTRAN program at NCAR.

- Alphabetic

A, B, C, ..., X, Y, Z

- Numeric

0, 1, 2, ..., 7, 8, 9

- Special

blank	(space)
Equals	=
Plus	+
Minus	-
Asterisk	*
Slash	/
Left parenthesis	(
Right parenthesis)
Comma	,
Decimal point	.
Dollar sign	\$
Apostrophe	'
Colon	:

Other special characters are available but are not supported by the FORTRAN standard. The character blank is ignored by the compiler except in Hollerith fields and otherwise may be used freely to improve program readability.

IDENTIFIERS

There are two kinds of identifiers: alphanumeric and statement identifiers.

**ALPHANUMERIC
IDENTIFIER**

Alphanumeric identifiers are symbolic names used in a FORTRAN statement to identify a member of a class. An identifier is a string of one to six alphanumeric characters. The first character of an alphanumeric identifier must be a letter. Embedded blanks within an identifier are ignored. For example, the name ATEST is the same as AT EST. Identifiers are names used in the following classes of names:

<u>Type</u>	<u>FORTRAN Example</u>
Array element	<u>A(I,J) = 1.0</u>
Variables	<u>A63 = XYZZZ + S</u>
Subroutines	<u>SUBROUTINE RUNGE</u>
Entry	<u>ENTRY TEST</u>
Main programs	<u>PROGRAM MAIN6</u>
Function names	<u>FUNCTION POLY</u>
Labeled common blocks	<u>COMMON/BL14/A,B,C</u>
Block data subprograms	<u>BLOCK DATA XTEST</u>

**STATEMENT
IDENTIFIER**

Statement labels are unsigned integer constants used to identify a particular statement in the program. A statement label is a string of one to five digits placed in columns 1 through 5 of the initial line of a statement. The range is from 1 to 99999. Leading zeros are ignored, and zero is not a legal statement identifier. Within any given program or subprogram each statement identifier must be unique.

Example of Statement Labels

Statement 1 2 Mo 4 5	E O	7 8 9 10 1 2 3 4 15 6 7 8 9 20 1 2 3 4 25
1 7		E T = L A M D A
4 3		A = C O S (X) + E T
1 2 5 9		B = 4 3 . 9 1

CONSTANTS

Seven types of constants are used in FORTRAN:

Integer
Octal
Real
Double precision
Complex
Hollerith
Logical

Complex and double-precision constants are formed from real constants. The type of a constant is determined by its form and each type has a different internal representation.

**INTEGER
CONSTANTS**

An integer constant is a string of up to 18 decimal digits in the range $-(2^{59}-1) \leq N \leq (2^{59}-1)$. $2^{59}-1$ equals 576460752303423487₁₀ (37777777777777777777₈). An integer constant is an exact representation of a positive, negative, or zero integral value. Integer add and subtract may be performed on all integers in the above range; however, no trap for overflow ever occurs on integer add or subtract. In a replacement statement for a constant greater than this range, the diagnostic CONSTANT OUT OF RANGE will occur.

Due to hardware limitations, integers in the range $2^{59}-1 \leq |I| \leq 2^{48}-1$ may not be used to do integer multiply and divide. Since there is no integer divide in the hardware, it is necessary in these cases for the compiler to convert the number to real so the division can be done in floating point unit; the result of the floating point division is then truncated to an integer.

There is a hardware integer multiply used by compiled programs. The integer result will be no larger than $2^{48}-1$ in absolute value. Overflow and indefinite conditions will occur when an integer multiply exceeds $2^{48}-1$ in absolute value.

$2^{48}-1$ equals 281474976710655₁₀ (00007777777777777777₈)

For numbers greater than $2^{48}-1$ in absolute value, the print routine will output an R (meaning range error) for an I format. The number stored in the computer, however, is good for integer add and subtract.

The maximum value of the result of integer division must be less than $2^{48}-1$. A value larger than $2^{48}-1$ will cause high-order bits to be lost in the operation. No diagnostic is provided for the latter case. -0 is an integer constant as well as 0.

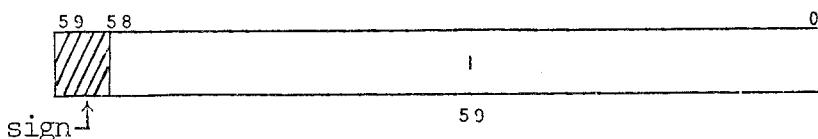
0 is 0000000000000000000000000000₈

-0 is 77777777777777777777₈

Examples

67
281474976710655
-2
-314159269
576460752303423487

The word structure for an integer constant is as follows:



**OCTAL
CONSTANTS**

Octal numbers are integer numbers in the base 8 system. Octal is used because of its easy and explicit conversion to and from binary. An octal constant consists of 1 to 20 octal digits preceded by a sign. An octal number is followed by a B suffix. The form is nB where n is the octal number. If the constant exceeds 20 digits, the compiler diagnostic is AN OCTAL NUMBER HAS MORE THAN 20 DIGITS. If a non-octal digit appears, the compiler diagnostic is ILLEGAL CHARACTER APPEARS IN A CONSTANT. In FORTRAN there is no interpretation of octal in the current standard. Note that octal is used as integer type.

Examples

Note that the B convention right-justifies the octal number in the computer word, and the word is zero-filled to the left. All octal constants should be type integer.

<u>FORTRAN Statement</u>	<u>60-Bit Word Defined</u>
I = 2374216B	00000000000000002374216
KK = 777776B	0000000000000000777776
MASK = 777000777000777B	00000777000777000777

The word structure is



REAL CONSTANTS

A real constant is represented by a string of up to 15 decimal digits. A real constant contains a decimal point and may contain an exponent representing a power of 10. The largest real number defined in the computer is 1.26501408317069E+322, base 10, (37767777777777777_8). A variable location may contain this number. If a constant is equal to or greater than 1.E+322, a compiler diagnostic is provided (CONSTANT OUT OF RANGE). However, the higher number will work in all floating point arithmetic units. When a number calculated in the computer exceeds this range, RRRRR, indicating a range error, is printed from a WRITE statement.

The smallest positive real constant is 3.13151306251401E-294, base 10, (00014000000000000000_8). If a real constant is less than this number, the constant is set to zero. No compiler diagnostic is provided.

A positive infinity in a computer word is 3777XXXXXXXXXXXXXX₈. A negative infinity is the 7's complement of positive infinity 4000XXXXXXXXXXXXXX₈. An indefinite number is 1777XXXXXXXXXXXXXX₈ in the computer word. A negative indefinite is the complement 6000XXXXXXXXXXXXXX₈.

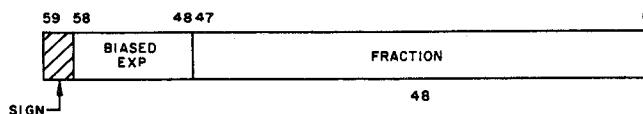
Examples

The number is n; s is the exponent to the base 10.

n.Es	3.E1 (means 3.0×10^1 ; i.e., 30.)
n.n	3.1415768
n.	314.
.n	.0749162
n.nE+s	-3.141592E+279
n.nE-s	31.41592E-01
.nEs	.31415E01
.nE+s	.31515E+01
nEs	52E6

REAL CONSTANTS
(continued)

The word structure of a floating point constant is



A constant of zero is the same as 0.0 in the computer word. However, do not mix mode in defining zeros. Use A = 0. (real) or IA = 0 (integer). The compiler will generate an extra instruction to pack or unpack the zero depending on whether or not the decimal point is present, even though its value is the same.

**DOUBLE-PRECISION
CONSTANTS**

A double-precision constant is represented by a string of up to 29 decimal digits. The forms are similar to real constants, where D is used to prefix the exponent.

.nD±s n.nD±s n.D±s

The number is n; s is the exponent to the base 10.

Double-precision constants are represented internally by two words. The D must always appear. The plus sign may be omitted for positive s.

The largest double-precision constant is
1.2650140831706913647030959170D+322 (base 10).

37767777777777777777₈ word 1
37167777777777777777₈ word 2

If a constant exceeds this range, the compiler diagnostic will be CONSTANT OUT OF RANGE. When a number calculated in the computer exceeds this range, RRRRR, indicating a range error, is printed from a WRITE statement.

The smallest double-precision constant is
3.1315130625140199656189188302D-294 (base 10).

00014000000000000000000000000000_s word 1
00000000000000000000000000000000_s word 2

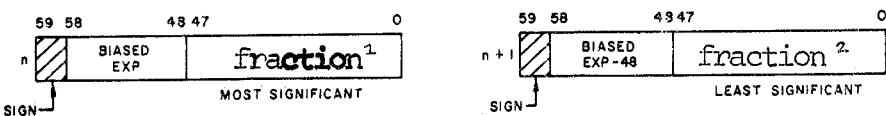
Numbers smaller than 10^{-280} have word 2 of the double-precision constant set to zero. Numbers smaller than 10^{-295} have both word 1 and word 2 set to zero. No compiler diagnostic is provided.

Examples

n.nDs	3.14159254358979323D0
n.nDs	2.1843D0
n.nDs	1.D0
n.nD-s	3141.592D-03
n.nDs	3141.592D3

Two computer words define a double-precision constant: word 1 is the most significant part, word 2 the least significant part.

The structure of the two computer words defining a double-precision constant is



COMPLEX CONSTANTS

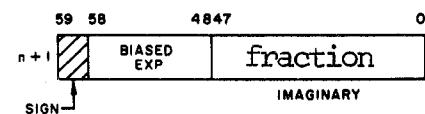
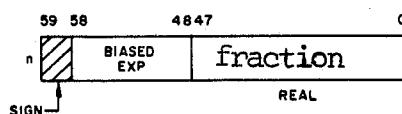
A complex constant is an ordered pair of real constants separated by a comma and enclosed in parentheses (R_1, R_2). R_1 represents the real part of the complex number, and R_2 represents the imaginary part. Either constant may be preceded by a minus sign. The two constants in the complex constant are real numbers and have the same range as real numbers. If the range of the real numbers comprising the constant is exceeded, the compiler diagnostic is CONSTANT OUT OF RANGE. Diagnostics also occur when the number pair consists of integer constants, including (0,0).

There are two words in a complex constant: word 1 is the real part of the complex number, word 2 the imaginary part.

Examples

<u>FORTRAN Representation</u>	<u>Complex Number</u>
(1., 6.55)	1. + 6.55i
(15., 16.7)	15. - 16.7i
(-14.09, 1.65E-04)	-14.09 + .000165i
(0., -1.)	-i

The structure of the two words defining a complex constant is



HOLLERITH CONSTANTS

A Hollerith constant (or alphanumeric constant) is a string of FORTRAN characters of the form nHf ; n is an unsigned decimal integer representing the length of the field f . One computer word will hold 10 Hollerith characters.

Spaces are significant in the field f . When n is not a multiple of 10, the last computer word is left-justified with spaces filling the remainder of the word. The alternate form is nRf , which is interpreted as a right-justified Hollerith constant with zero-fill for incomplete words. A Hollerith constant of the form $L = (5HABCDE)$ is also permitted.

Hollerith constants may be used in arithmetic and DATA statements. They are stored internally in console display code.

Rules

- In an arithmetic replacement statement, such as
 $L = 4HABCD$, n may not be greater than 10.
- In a DATA statement, n may not exceed 150.

```
DIMENSION L(13)
DATA(L=130H .....
1.....
2.....)
```

When the H specification is greater than 10 characters, the array defined must be appropriately dimensioned; in this case L must be dimensioned 13.

- A Hollerith constant of the form $K = (+5HABCDE)$ is also permitted.

*Rules
(continued)*

- Hollerith constants in a replacement statement are assumed by the compiler to be type integer.
- In a CALL statement, a Hollerith constant may not exceed 150 characters.

Examples in a Replacement Statement

<u>FORTRAN Statement</u>	<u>Word in Memory</u>
M = 6HABCDE	0102030405555555555555
K = 1HR	2255555555555555555555
L(1) = 10HTEMPERATUR	24051520052201242522
L(2) = 10HE	0555555555555555555555
K = 5H12345	3435363740555555555555
J = 5R12345	0000000003435363740

Examples in a DATA Statement

<u>FORTRAN Statement</u>	<u>Word in Memory</u>
DATA M/3HSUM,4HMEAN/	2325155555555555555555
DATA L/11HTEMPERATURE/	1505011655555555555555
	24051520052201242522
	0555555555555555555555

LOGICAL CONSTANTS

Logical constants are .TRUE. or .FALSE. .TRUE. is a word filled with 7's. This is -0. Any non-zero or minus-zero value is true. .FALSE. is a word all zero. This is +0.

VARIABLES

A variable in FORTRAN is a symbolic name used to identify a quantity stored in the computer, which may assume different values. The value of a variable may change during execution of a program.

The symbolic representation, that is, the variable name, is a reference to the location of the quantity. The value of that quantity is contained in the location represented by the variable name.

Example

AB = 10.5

AB is the variable name or location of the number 10.5. 10.5 is the number in AB, and is referred to as the contents of AB.

Variable names may have from one to six alphanumeric characters, and the first character must be a letter.

These names may be characterized as one of the following types:

- Integer
- Real
- Double-precision
- Complex
- Logical

VARIABLES (continued)

Unless the type of a variable name is declared in a TYPE or IMPLICIT statement (see chapter 5), the compiler assumes the name to be integer or real according to the following convention:

Assumed from name

Variables are assumed to be integer if the first character of their symbolic name is one of the characters I, J, K, L, M, or N. All other variables are assumed to be real.

Typed

The programmer may override the rule and specify any of the five types by using the FORTRAN TYPE declaration.

<u>Type</u>	<u>Statement</u>
Integer	INTEGER A, B, K
Floating point (real)	REAL I, J, T
Complex	COMPLEX A, C, C1, N
Double-precision	DOUBLE D, DD, S, L
Logical	LOGICAL L1, LT

TYPES OF VARIABLES

Integer Variables

An integer variable may be typed explicitly with a TYPE declaration, or assumed to be of integer type if the first letter in the name is I, J, K, L, M, or N. Each simple integer variable occupies one word in storage and assumes an exact integer value.

N
ITEM
M58A
ITEST
L2

Real Variables

Real variables may be typed explicitly or assumed to be real variables if the first letter is not I, J, K, L, M, or N. Each real variable occupies one word in storage and is stored in normalized floating point format. It is an approximation to a real number.

VECTOR
X
DERIV
A4S9

Double-Precision Variables

Double-precision variables must be typed as DOUBLE explicitly in a FORTRAN TYPE declaration. Each double-precision variable occupies two words of storage.

Complex Variables

Complex variables must be explicitly defined in a FORTRAN TYPE declaration. A complex variable occupies two words in storage. Each word contains a number in FORTRAN real variable format. The ordered pair of real variables represents the complex number $x_1 + ix_2$.

Naming Octal Constants

Octal constants should be defined using integer variable names to avoid having the compiler float the number if modes are mixed.

IA = 25B
IB = 0155B
MASK = 7777777B

TYPES OF VARIABLES*(continued)***Alphanumeric Names
(Hollerith)**

Names of variables containing alphanumeric messages or labels should have integer type names.

```
NAME = 4HMARY
M2S(1) = 10HTEMPERATUR
M2S(2) = 1HE
```

Logical Variables

Logical variables may be logical or integer type.

**CLASSES OF
VARIABLES**

The FORTRAN compiler recognizes two classes of variables, simple and subscripted. A *simple variable* is one that is single-valued; a *subscripted variable* is multiple-valued.

Simple Variable

A simple variable refers to a single storage location containing one quantity. The value specified by the name is always the current value stored in that location.

```
FRAN
PTEST
E9T
MAT
I3
K
```

If A has been set equal to 4.5 and is used as a variable in an equation, the value of A (which is 4.5) is used in the equation.

Subscripted Variable

A subscripted variable is an alphanumeric identifier with one, two, or three associated subscripts enclosed in parentheses. It is called an array. The value of the array element is the quantity stored in the name modified by the subscripts specified.

A(1)
B(2,3)
LTV(2,1,4)
A5(1,10,8)

If more than three subscripts appear, the compiler diagnostic given is GREATER THAN THREE DIMENSIONS IN DIMENSION STATEMENT.

If a subscripted variable appears in a replacement statement with more subscripts than were declared in the DIMENSION statement, the diagnostic TOO MANY SUBSCRIPT INDICES is given.

A standard subscript has one of the following forms: a and b are unsigned integer constants and I is a simple integer variable. Use of standard subscripts produces the most efficient code for indices.

General FormStandard Subscripted Variables

a*I+b	A(I,J,K)
I*a+b	R(I*3+2)
b+I*a	R(3+I*2)
b+a*I	R(3+4*K)
a+I	R(5+L)
I+b	B(I-1,J+1,3*K-1)
a*I	D(9)
I	XYZ(2*K+4)
a	T(2,1)

**CLASSES OF
VARIABLES**
(continued)

**Subscripted
Variable**
(continued)

A nonstandard subscript is any arithmetic expression other than the standard forms used as a subscript. If the expression is floating point, the result is truncated and used as the integer index.

Examples

Nonstandard Subscripted Variables:

A(MAXF(I,J,M))
B(J,SINF(J))
MAT(3*K*ILIM+3.5)
Q(1,-4,-2)
T(X,Y)

Limitation

- Subscripted subscripts are not allowed.

Nonstandard subscripts generate inefficient code from the compiler. Efficient code results from any of the standard forms listed above.

Arrays

An array is a named and ordered set of data stored in the computer under a symbolic name.

A single element of this array is referenced by the array name plus one, two, or three subscripts. The array name must appear in a DIMENSION statement. A TYPE or COMMON declaration may also be used, if it is appropriate.

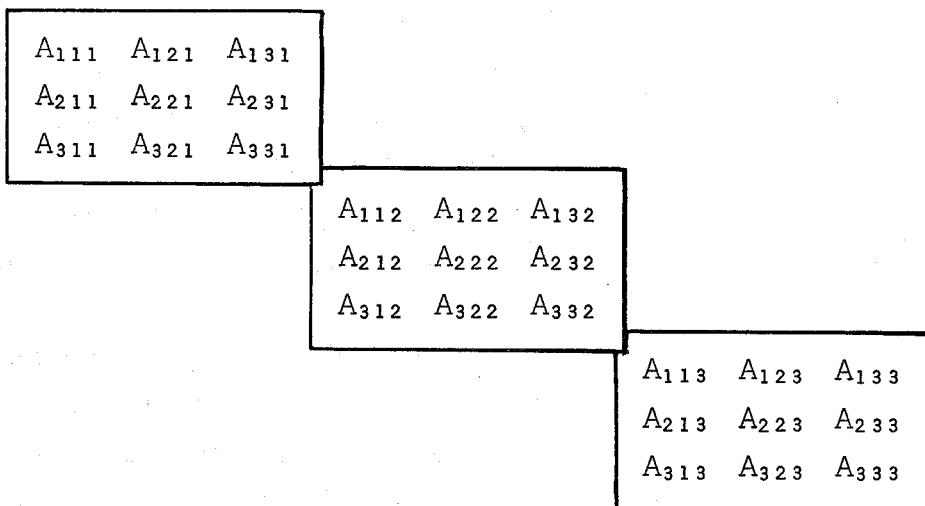
The entire array may be referenced using the array name without subscripts in I/O lists where implied DO loop notation prints the entire array as specified by the DIMENSION declaration in a DIMENSION, TYPE, or COMMON statement. In an arithmetic expression, an array name without subscripts is taken to be the first element only.

There are no diagnostics if the subscript value is greater than the dimension bound declared for that name. Program execution may be in error as a result. (Any address in the program which tries to go outside the total core used by this program will cause an execution error. The program terminates with an SCM DIRECT RANGE diagnostic on the 7600.)

**CLASSES OF
VARIABLES**
(continued)

Array Structure

Array elements are stored by columns in consecutive sequential storage locations. The array dimensioned as A(3,3,3) appears as follows:



The planes are stored in order, starting with the first, as follows:

$$\begin{array}{lll} A_{111} \rightarrow L & A_{121} \rightarrow L+3 & \dots A_{133} \rightarrow L+24 \\ A_{211} \rightarrow L+1 & A_{221} \rightarrow L+4 & \dots A_{233} \rightarrow L+25 \\ A_{311} \rightarrow L+2 & A_{321} \rightarrow L+5 & \dots A_{333} \rightarrow L+26 \end{array}$$

Array allocation is discussed under DIMENSION declaration in chapter 5. The location of an array element with respect to the first element is a function of the maximum array dimensions and the type of the array.

Given DIMENSION A(L,M,N), the location of A(i,j,k), with respect to the first element A of the array, is given by

$$A + (i - 1 + L*(j-1+M*(k-1))) * E$$

The quantity enclosed by the outer parentheses is the subscript expression. E is the element length; i.e., the number of storage words required for each element of the array. For real and integer arrays, E = 1. For complex and double-precision arrays, E = 2.

Note: A left shift rather than a multiply will be used in calculating a single element when the dimension is 64 or less and a power of 2. Use of shifts on higher powers of 2 may also occur.

Example

In an array defined by DIMENSION A(3,3,3) where A is real, the location of A(2,2,3) with respect to A(1,1,1) is

$$\begin{aligned}A(2,2,3) &= A(1,1,1) + (2-1+3(1+3(2))) \\&= A(1,1,1) + 22\end{aligned}$$

FORTRAN permits the following relaxation of the representation of subscripted variables (where N_i are integer constants):

Given $A(N_1, N_2, N_3)$,

then $A(I,J,K)$ implies $A(I,J,K)$
 $A(I,J)$ implies $A(I,J,1)$
 $A(I)$ implies $A(I,1,1)$
 A implies $A(1,1,1)$

**CLASSES OF
VARIABLES**
*(continued)***Array Structure**
(continued)

Similarly, given $A(N_1, N_2)$,

then $A(I, J)$ implies $A(I, J)$
 $A(I)$ implies $A(I, 1)$
 A implies $A(1, 1)$

and given $A(N_1)$,

then $A(I)$ implies $A(I)$
 A implies $A(1)$

The elements of a single-dimensioned array $A(N)$ may not be referred to as $A(I, J, K)$ or $A(I, J)$. Similarly, the elements of a double-dimensioned array $A(M, N)$ may not be referred to as $A(I, J, K)$. The diagnostic in such cases is TOO MANY SUBSCRIPT INDICES.

III

EXPRESSIONS ARITHMETIC, LOGICAL MASKING, MIXED MODE

EVALUATION

OPTIMIZATION

exp

EXPRESSIONS--ARITHMETIC LOGICAL, MASKING, MIXED MODE

INTRODUCTION

Three kinds of expressions are recognized by the 7600 FORTRAN compiler: arithmetic expressions; logical expressions, which may be true or false; and masking expressions which, when evaluated by the computer, have numerical values. For each type of expression there is an associated group of operators and operands.

3.2

Expressions

EXPRESSIONS

An expression may contain constants, variables (simple or subscripted), functions, and combinations of these separated by operators and/or parenthesis.

ARITHMETIC EXPRESSIONS

An arithmetic expression may contain the following operators:

<u>Operator</u>	<u>Meaning</u>	<u>Example</u>
+	Addition	A + B
-	Subtraction	C - D + 1.324
*	Multiplication	C*D
/	Division	C*D + A/D
**	Exponentiation†	A**B + 3.1415

An operator performs arithmetic using operands. An operand may be a constant, a variable (simple or subscripted), the result of the evaluation of a function or the result of the evaluation of an expression.

In the expression $(A^{**}B + 3.1415)$, A, B, and 3.1415, as well as the result of $A^{**}B$ are operands, while ** and $+$ are operators. When all operations are completed as indicated in the expression, the expression is said to be evaluated and thus has a numerical value. Operands in an arithmetic expression may be separated by only one operator ($X \text{ op op } Y$ is not valid). The only exception is unary negation, e.g. $A^{*(-)}B$.

Examples

T	$(B - \text{SQRT}(B^{**}2 - (f*A*C)))/(2.0*A)$
3.14159	GROSS - (TAX*0.05)
B + 16.427	$(\text{TEMPT} + V(M,\text{MAXF}(A,B))^{**}C)/$
$(XBAR + (B(I,J+I,K)/3.))$	$(H - \text{FACT}(K + 3))$
$-(C + \text{DELTA}^{*}\text{AERO})$	A + (-B)

STANDARD EVALUATION

The hierarchy of arithmetic evaluation is

**	Exponentiation	class 1
/	Division	}
*	Multiplication	}
+	Addition	
-	Subtraction	}
-	Unary negation	

Expressions with no embedded parentheses which contain unlike classes of operators are evaluated in the above order beginning with class 1. Expressions which contain operators of like classes are evaluated from left to right. For example, $A^{**}B^{**}C$ is evaluated $(A^{**}B)^{**}C$.

Parenthetical and function expressions (the ** operator is a function expression) are evaluated first in a left-to-right scan of the entire statement. In parenthetical expressions within parenthetical expressions, evaluation begins with the innermost expression. Parenthetical expressions are evaluated as they are encountered in the left-to-right scanning process.

When writing an integer expression, it is important to remember not only the left-to-right scanning process, but also that dividing an integer quantity by an integer quantity always yields a truncated result; thus, when the expression $1/3$ is evaluated, the result is 0.

The expression $I^{*}J/K$ will yield a result different from the expression $J/K^{*}I$ because of the order of operations and truncation in each case. For example; $4^{*}/3/2$ means 4 times 3 (or 12) divided by two resulting in 6. $3/2^{*}4$ means $3/2$ (truncated to 1) times 4 resulting in 4.

STANDARD EVALUATION*(continued)*

The NCAR compiler described in this manual will optimize code to produce more efficient machine language object code.

Arithmetic expression evaluation without optimization will apply *only* if the *FORTRAN card specifies that no optimization should be done by the compiler (see page 3.6).

Example 1

$A^{**}B/C+D^{*}E^{*}F-G$ is evaluated:

$$A^{**}B \rightarrow R_1$$

$$R_1/C \rightarrow R_2$$

$$D^{*}E \rightarrow R_3$$

$$R_3^{*}F \rightarrow R_4$$

$$R_4+R_2 \rightarrow R_5$$

$$R_5-G \rightarrow R_6 \quad \text{evaluation completed.}$$

Example 2

$A^{**}B/(C+D)^{*}(E^{*}F-G)$ is evaluated:

$$A^{**}B \rightarrow R_1$$

$$C+D \rightarrow R_2$$

$$E^{*}F-G \rightarrow R_3$$

$$R_1/R_2 \rightarrow R_4$$

$$R_4^{*}R_3 \rightarrow R_5 \quad \text{evaluation completed.}$$

Example 3

$H(I3)+C(I,J+2)^{*}(\cos(Z))^{**2}$ is evaluated:

$$\cos(Z) \rightarrow R_1$$

$$R_1^{**2} \rightarrow R_2$$

$$R_2^{*}C(I,J+2) \rightarrow R_3$$

$$R_3+H(I3) \rightarrow R_4 \quad \text{evaluation completed.}$$

The following are examples of expressions with embedded parentheses.

Example 1

$A^*(B+((C/D)-E))$ is evaluated:

$$C/D \rightarrow R_1$$

$$R_1-E \rightarrow R_2$$

$$R_2+B \rightarrow R_3$$

$R_3^*A \rightarrow R_4$ evaluation completed.

Example 2

$A^*(\sin(X)+1.)-Z/(C^*(D-E+F))$ is evaluated:

$$\sin(X) \rightarrow R_1$$

$$R_1+1. \rightarrow R_2$$

$$D-E \rightarrow R_3$$

$$R_3+F \rightarrow R_4$$

$$C^*R_4 \rightarrow R_5$$

$$A^*R_2 \rightarrow R_6$$

$$-Z \rightarrow R_7$$

$$R_7/R_5 \rightarrow R_8$$

$R_8+R_6 \rightarrow R_9$ evaluation completed.

**EVALUATION WITH
OPTIMIZATION**

Optimization of arithmetic sequences involves changing the order of operations so that the sequence is performed with speed and efficiency. The most efficient code does not always conform to conventional rules for expression evaluation. The *FORTRAN Standard* states that processors may evaluate the mathematically equivalent expression. Where the previous rules are important to the results, it is necessary for the programmer to indicate to the compiler that no reordering is to take place.

When a FORTRAN program is compiled, all optimization techniques are applied. If the order of evaluation described in the previous section must be followed precisely, the programmer must turn off the optimization on the *FORTRAN,n card, using the following codes for n. A type double or type complex turns off the optimizer for all expressions throughout the routine in which the TYPE declaration appears.

- n=14 Do not optimize divide operators (this option automatically turns off 15 as well)
- 15 Do not optimize multiply operators
- 16 Do not optimize add and subtract operators
- 0 Apply 14, 15, and 16 to prevent reordering of the arithmetic sequence

Expression: X = A/B/C/D/E

<u>With Optimization</u>	<u>Without Optimization (Code 14)</u>
B*C → R ₁	A/B → R ₁
R ₁ *D → R ₂	R ₁ /C → R ₂
R ₂ *E → R ₃	R ₂ /D → R ₃
A/R ₃ → R ₄	R ₃ /E → R ₄

To optimize, a series of divides is replaced by multiplies which are faster.

Expression: $X = A * B * C * D * E$

<u>With Optimization</u>	<u>Without Optimization (Code 15)</u>
$A * B \rightarrow R_1$	$A * B \rightarrow R_1$
$C * D \rightarrow R_2$	$R_1 * C \rightarrow R_2$
$R_1 * R_2 \rightarrow R_3$	$R_2 * D \rightarrow R_3$
$R_3 * E \rightarrow R_4$	$R_3 * E \rightarrow R_4$

NCAR's compiler will optimize because of the pipeline characteristics of the functional units in the Control Data 7600 computer. Again, optimization is done to take advantage of the pipeline characteristics.

Expression: $X = A + B - C + D$

<u>With Optimization</u>	<u>Without Optimization (Code 16)</u>
$A + B \rightarrow R_1$	$A + B \rightarrow R_1$
$D - C \rightarrow R_2$	$R_1 - C \rightarrow R_2$
$R_1 + R_2 \rightarrow R_3$	$R_2 + D \rightarrow R_3$

Expression: $X = A * B * C * D * E * F * G * H$

<u>With Optimization</u>	<u>Without Optimization (Code 15)</u>
$A * B \rightarrow R_1$	$A * B \rightarrow R_1$
$C * D \rightarrow R_2$	$R_1 * C \rightarrow R_2$
$E * F \rightarrow R_3$	$R_2 * D \rightarrow R_3$
$G * H \rightarrow R_4$	$R_3 * E \rightarrow R_4$
$R_1 * R_2 \rightarrow R_5$	$R_4 * F \rightarrow R_5$
$R_3 * R_4 \rightarrow R_6$	$R_5 * G \rightarrow R_6$
$R_6 * R_5 \rightarrow R_7$	$R_6 * H \rightarrow R_7$

Notice that resultants are also optimized under code 16.

**EVALUATION WITH
OPTIMIZATION**
(continued)
Expression: $F = A*B*X/Y$
With Optimization

$$\begin{array}{l} X/Y \rightarrow R_1 \\ A*B \rightarrow R_2 \\ R_2*R_1 \rightarrow R_3 \end{array}$$

Without Optimization
(Code 14)

$$\begin{array}{l} A*B \rightarrow R_1 \\ R_1*X \rightarrow R_2 \\ R_2/Y \rightarrow R_3 \end{array}$$

To optimize, the slower divide operation is moved up in the sequence so that R_1 and R_2 are done simultaneously with the divide covering the multiply.

Expression: $X = A+B+C+D+E$
With Optimization

$$\begin{array}{l} A+B \rightarrow R_1 \\ C+D \rightarrow R_2 \\ R_1+R_2 \rightarrow R_3 \\ R_3+E \rightarrow R_4 \end{array}$$

Without Optimization
(Code 16)

$$\begin{array}{l} A+B \rightarrow R_1 \\ R_1+C \rightarrow R_2 \\ R_2+D \rightarrow R_3 \\ R_3+E \rightarrow R_4 \end{array}$$

This is done to optimize the use of the pipelined functional units.

Expression: $B = U*R + U*R + U*R$
With Optimization

$$\begin{array}{l} U*R \rightarrow R_1 \\ R_1+R_1 \rightarrow R_2 \\ R_2+R_1 \rightarrow R_3 \\ R_3+R_1 \rightarrow R_4 \\ R_4+R_3 \rightarrow R_5 \end{array}$$

Without Optimization
(Code 15)

$$\begin{array}{l} U*R \rightarrow R_1 \\ U*R \rightarrow R_2 \\ R_1+R_2 \rightarrow R_3 \\ U*R \rightarrow R_4 \\ R_4+R_3 \rightarrow R_5 \end{array}$$

To optimize, $U*R$ is calculated only once.

Expression: $U^*T^*U^*T + A/R$

With Optimization

$$\begin{aligned} A/R &\rightarrow R_1 \\ U^*T &\rightarrow R_2 \\ R_2^*R_2 &\rightarrow R_3 \\ R_3+R_1 &\rightarrow R_4 \end{aligned}$$

In this case U^*T is calculated only once. The slow divide operation is moved forward in the string.

Without Optimization as on *FORTRAN,14,15
or *FORTRAN,14:

$$\begin{aligned} U^*T &\rightarrow R_1 \\ R_1^*U &\rightarrow R_2 \\ R_2^*T &\rightarrow R_3 \\ A/R &\rightarrow R_4 \\ R_3+R_4 &\rightarrow R_5 \end{aligned}$$

Here, no divide or multiply is optimized. The divide option turns off the multiply as well.

Without Optimization as on *FORTRAN,15:

$$\begin{aligned} A/R &\rightarrow R_1 \\ U^*T &\rightarrow R_2 \\ R_2^*U &\rightarrow R_3 \\ R_3^*T &\rightarrow R_4 \\ R_4+R_1 &\rightarrow R_5 \end{aligned}$$

In this case, the multiply option is turned off with a 15 on the *FORTRAN card, so the divide option (14) which is *not* turned off applies, and the divide is brought forward.

3.10
Expressions

**EVALUATION WITH
OPTIMIZATION**
(continued)

Expression: $C = A*B - E/R$

<u>With Full Optimization</u>	<u>Without Optimization (Codes 14, 15 or Code 14)</u>
$E/R \rightarrow R_1$	$A*B \rightarrow R_1$
$A*B \rightarrow R_2$	$-E \rightarrow R_2$
$R_2 - R_1 \rightarrow R_3$	$R_2/R \rightarrow R_3$
	$R_3 + R_1 \rightarrow R_4$

$E/R \rightarrow R_1$	$A*B \rightarrow R_1$
$A*B \rightarrow R_2$	$-E \rightarrow R_2$
$R_2 - R_1 \rightarrow R_3$	$R_2/R \rightarrow R_3$
	$R_3 + R_1 \rightarrow R_4$

The 14 option turns off both 14 and 15.

Expression: $X = A*B*C*D + G/H*P/Y - R + S$

<u>With Optimization</u>	<u>Without Optimization (Code 0)</u>
--------------------------	--

$G*P \rightarrow R_1$	$A*B \rightarrow R_1$
$H*Y \rightarrow R_2$	$R_1*C \rightarrow R_2$
$R_1/R_2 \rightarrow R_3$	$R_2*D \rightarrow R_3$
$A*B \rightarrow R_4$	$G/H \rightarrow R_4$
$C*D \rightarrow R_5$	$R_4*P \rightarrow R_5$
$S-R \rightarrow R_6$	$R_5/Y \rightarrow R_6$
$R_4*R_5 \rightarrow R_7$	$R_6+R_3 \rightarrow R_7$
$R_7+R_3 \rightarrow R_8$	$R_7-R \rightarrow R_8$
$R_8+R_6 \rightarrow R_9$	$R_8+S \rightarrow R_9$

LOGICAL EXPRESSIONS

A logical expression has the form:

$\text{exp}_1 \text{ rop } \text{exp}_2$

RELATIONAL OPERATORS

The exp_i 's are arithmetic expressions, evaluated according to rules set down in the previous section; rop is one of the following relational operators:

<u>Operator</u>	<u>Meaning</u>
.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to

The logical expression is true if the arithmetic expressions satisfy the relation specified by rop; otherwise, it is false.

$C = B.\text{LT}.(I^*\text{COS}(X-1.0))$

Logical expressions are evaluated as illustrated in the example, A.EQ.B. This is equivalent to the question: does $A - B = 0$?

The following table shows how the 7600 compiler treats the logical expressions. In the following expressions, $A_0 = 0$, $A_1 > 0$.

<u>Computer Calculation</u>	<u>Test</u>	<u>Evaluated Expression (in octal)</u>
$A_0.\text{EQ}.A_1$	$A_0 - A_1$	Not zero 000000000000000000000000 (false)
$A_0.\text{NE}.A_1$	$A_0 - A_1$	Zero 777777777777777777777777 (true)
$A_0.\text{GT}.A_1$	$A_1 - A_0$	Zero or plus 000000000000000000000000 (false)
$A_0.\text{GT}.A_1$	$A_0 - A_1$	Minus (other than -0) 000000000000000000000000 (false)
$A_0.\text{LT}.A_1$	$A_0 - A_1$	Zero or plus 777777777777777777777777 (true)
$A_0.\text{LE}.A_1$	$A_1 - A_0$	Minus (other than -0) 777777777777777777777777 (true)

RELATIONAL OPERATORS
(continued)

The difference is computed and tested. If the test is satisfied, the relation is true and the expression is assigned the value -0 or .TRUE. (777777777777777777). If the relation is false, the expression is set to 0 or .FALSE.

If A is true, A is set to 777777777777777777B

If A is false, A is set to 000000000000000000000000B

Logical expressions of the following forms are allowed:

I.LT.R , I.LT.D , I.LT.C

where I=integer, R=real, D=double, and C=complex. An expression of the form I.GE.0 is treated as being true if I assumes the value -0.

Logical expressions are converted internally to arithmetic expressions according to the rules of mixed-mode arithmetic (see section below on mixed mode expressions). When complex expressions are tested, only the real part is used in the comparison.

- $A_1 \text{ rop } A_2 \text{ rop } A_3 \dots$ is not a valid logical expression (see logical connectives, p. 3.13).
- A logical expression of the form $A_1 \text{ rop } A_2$ is evaluated from left to right. The logical expressions $A_1 \text{ rop } A_2$, $A_1 \text{ rop } (A_2)$, and $(A_1) \text{ rop } (A_2)$ are equivalent.
- A false logical expression is assigned the value +0; a true logical expression is assigned the value -0.
- Relational operators may not be used in multiple replacement statements.
- In a logical replacement statement, the mode of the resultant is ignored; the result is true or false (i.e., -0 or +0).

Examples

A.LT.16.

T = Q(I)*Z.LE.3.141592

B = C.NE.D + E

R(I).GE.R(I-1)

K.LT.16

I.EQ.J(K)

(I).EQ.(J(K))

LOGICAL CONNECTIVES

Logical connectives may be used with relational operators in the parenthetical expressions used in an IF statement. A logical expression may be of the form

IF(lexp₁ con lexp₂...)n₁,n₂

The terms lexp_i are logical expressions as shown in the examples in the previous section, where lexp_i = A₁ rop A₂ and con is a connective belonging to the following set:

<u>Connective</u>	<u>Meaning</u>
.NOT.	Negation (this does <i>not</i> mean complement)
.AND.	Conjunction
.OR.	Disjunction

Connectives are defined as follows:

.NOT.REL ₁	Is false if and only if REL ₁ is true
REL ₁ .AND.REL ₂	Is true if and only if REL ₁ and REL ₂ are both true
REL ₁ .OR.REL ₂	Is false if and only if REL ₁ and REL ₂ are both false (this is not an exclusion or)

LOGICAL CONNECTIVES
(continued)

- Results of logical expressions in an IF statement are either true or false:

True	777777777777777777
False	00000000000000000000
- The scan is based on the connectives as they appear from left to right. Evaluation terminates as soon as the truth values can be established for an expression.
- Where .NOT. is used alone, as in (.NOT.A.EQ.B), no negation is involved. The compiler reverses the test (i.e., (A.NE.B)) in order to set truth values.
- Logical variables may be set true or false in a replacement statement by declaring:
$$\begin{aligned} L1 &= .TRUE. \\ L2 &= .FALSE. \end{aligned}$$
- REL $con_1 con_2 REL$ is never legal where con_i is either .AND. or .OR.
- In a combined logical expression with connectives, the resultant is set to true or false regardless of mode

$B - C \leq A < B + C$

is written as

$B - C .LE. A .AND. A .LT. B + C$

MASKING EXPRESSIONS

In a FORTRAN masking expression, 60-bit logical arithmetic is performed bit by bit on the operands within the expression. The operands may be real- or integer-type variables or constants. No mode conversion is performed during evaluation. Although the form of masking operators is the same as logical connectives, the meanings are very different. They are defined below, listed according to hierarchy of operation.

- .NOT. Complement the operand bit by bit
- .AND. Form the bit-by-bit logical product of two operands:

$$\begin{array}{r}
 0101 \text{ p} \\
 \cdot \text{AND.} 1101 \text{ v} \\
 \hline
 0101
 \end{array}$$

- .OR. Form the bit-by-bit logical sum of two operands:

$$\begin{array}{r}
 0101 \text{ p} \\
 \cdot \text{OR.} 1101 \text{ v} \\
 \hline
 1101 \text{ (this is not an exclusive or)}
 \end{array}$$

The operations are described below:

P	V	<u>p.AND.v</u>	<u>p.OR.v</u>	<u>.NOT.p</u>
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Example 1

Let MASK1 = 770077000000000000000B
 MASK2 = 7777B
 MASK3 = 777700000000000000000B
 IB = 10HAAAAAAMARY (01010101010115012231B in octal)
 CI = 1.0 (17204000000000000000B in octal)

Then .NOT.MASK1 is 007700777777777777B
 MASK3.AND.CI is 172000000000000000000B
 MASK2.OR.MASK3 is 777700000000000007777B
 MASK2.AND.IB is 00000000000000002231B

Example 2

Let MASK1 = 77770000777700007777B
 MASK2 = 10HABCDABCDAB (01020304010203040102B in octal)
 A = 1.0 (1720400000000000000B in octal)
 I = 000000000111111111B

Then the resultant for

R = MASK1.AND.MASK2 is 01020000010200000102B
 IJ = MASK2.OR.I is 01020304011313151113B
 KI = A.OR..NOT.I is 777777777666666666B

MASKING EXPRESSIONS
(continued)

The user should note that these masking operations, plus the exclusive OR, can also be performed using the in-line functions AND, OR, COMPL, and X R described in the Library Routines Manual (NCAR Technical Note TN-IA-67). In general programs, using the masking functions rather than masking operators will be more portable.

Rules

- Let B_i be masking expressions, variables, or constants of any type. The following are masking expressions:

.NOT. B_1 B_1 .AND. B_2 B_1 .OR. B_2

- .NOT. may appear with .AND. or .OR. only as follows:

.AND. .NOT.
.OR. .NOT.
.AND. (.NOT. ...)
.OR. (.NOT. ...)

- Masking expressions of the following forms are evaluated from right to left:

A .AND. B .AND. C is evaluated as follows:

B .AND. C $\rightarrow R_1$
A .AND. R $\rightarrow R_2$

- Masking expressions must not contain parenthetical arithmetic expressions, but any function subprogram call is allowed. If L5 = MASK.AND.X*I is written, the computer diagnostic is ILLEGAL USE OF A REPLACEMENT STATEMENT.
- Only real- and integer-type variables may be used. Other variable types result in the diagnostic MASKING OPNDS MUST BE REAL OR INTEGER.
- The masking forms of .AND., .OR., and .NOT. may not be used in an IF statement.
- Masking operations do *not* produce truth values.
- The mode of the variables in a masking statement is ignored.

- Masking expressions are not permissible as actual parameters.
The diagnostic is AN OPERATOR IS MISSING OR USED IMPROPERLY.

MIXED-MODE EXPRESSIONS

Mixed-mode expressions are permitted using the NCAR FORTRAN compiler. In discussing mixed mode it is helpful to distinguish between simple and combined expressions. A simple expression does not contain parenthetical delimiters within the expression. $A^*R/\cos(X)$ and $C+D*I$ are simple expressions. A combined expression is an expression consisting of more than one simple expression including nests. $A^*(\sin(X)+1) - Z/(C^*(D-E+F))$ is an example of a combined expression.

A precedence of expressions and operators can be established for mixed expressions. This order is shown below.

<u>Expression Type</u>	<u>Operators</u>
Arithmetic	** / * + -
Relational	.GT. , .GE. , .LT. , .LE. , .EQ. , .NE.
Logical connectives	.NOT. .AND. .OR.
Masking	.NOT. .AND. .OR.

In the following expression

$A+B.LT.C-D$

the arithmetic expressions are evaluated first, then the relational operators.

$A+B \rightarrow R_1$

$C-D \rightarrow R_2$

$R_1.LT.R_2 \rightarrow R_3$

R_3 will have a truth value.

3.18 Expressions

A table of permissible combinations of expression types with the mode of result may be helpful in interpreting mixed expressions.

Rules

- The order of dominance of the operand types within a simple expression from the highest to the lowest is

Complex
Double
Real
Integer
Relational

The dominant type is used for all arithmetic operations within a simple expression.

- All simple expressions in a combined expression are evaluated first, starting with the innermost nest of parentheses. The final scan uses the mode of the dominant resultant of all simple expressions or variables remaining after the initial scan.
- Double-precision and complex arithmetic are evaluated by in-line arithmetic functions.
- A simple arithmetic expression is evaluated in the mode of the dominant operand type.
- Expressions which contain Hollerith or octal constants are ordered as type integer in the order of dominance list above. If these are in a mixed expression of higher dominance, they will be converted to the appropriate type. (Using Hollerith constants in a mixed-mode expression is usually a mistake.)
- Expression optimization is never done by the compiler when there is mixed mode.

- The operator `**` combines constants, variables, expressions, and subscripted variables. The following list contains the combinations of mode that are allowed. Any combination other than these will result in the execution diagnostic **ILLEGAL EXPONENT**.

<u>Base</u>	<u>Exponent</u>	<u>Result</u>
Integer	Integer	Integer
Real	Integer	Real
	Real	Real
Double	Integer	Double
	Real	Double
	Double	Double
Complex	Integer	Complex

Some illustrations of simple and combined expressions that may be used with the `**` operator are given below:

T**I	Real to integer
I**(J - 1)	Integer to integer
RE(K,J + 1)**(2.1*COS(X))	Real to real
A**(I + K-1)	Real to integer
(4.23D0 + 1)**(XTEMP)	Double to real
C**(I*J + 3)	Complex to integer
(4.23D0 + 1)**(1.D0 + 3)	Double to double

Example 1

Given real A,B; integer I,J. The simple expression $A*B - I + J$ is real because the dominant operand type is real. The expression is evaluated:

$A*B \rightarrow R_1$ real
 Convert I to real
 Convert J to real
 $R_1 - I \rightarrow R_2$ real
 $R_2 + J \rightarrow R_3$ real evaluation completed.

3.20
Expressions

Rules
(continued)

Example 2

The use of parentheses can change the evaluation. Given $A*B-(I+J)$, the simple expression $(I+J)$ is evaluated first in integer. The evaluated expression $(I+J)$ is then converted to real. $A*B-(I+J)$ is evaluated:

$I+J \rightarrow R_1$ integer
Convert R_1 to real
 $A*B \rightarrow R_2$ real
 $R_2-R_1 \rightarrow R_3$ real evaluation completed.

Example 3

Given $A*B-J*K/J$. This simple expression is evaluated as follows where the dominant mode is real:

$A*B \rightarrow R_1$
Convert J to real $\rightarrow R_2$
Convert K to real
 R_2*K
Bring in J again and convert to real
 $R_2*K/J \rightarrow R_3$
 $R_3+R_1 \rightarrow R_4$ evaluation completed.

Example 4

Given $A*B-(J*K/J)$. This combined expression is evaluated as follows where the simple expression $(J*K/J)$ is evaluated as type integer:

$J*K \rightarrow R_1$
 $R_1/J \rightarrow R_2$ (where result is type integer)
Convert R_2 to real $\rightarrow R_3$
 $A*B \rightarrow R_4$
 $R_4-R_3 \rightarrow R_5$ evaluation completed.

Example 5

Given complex C1, C2; real A,B,F. The combined expression A*(C1/C2) + B*F is complex. The expression is evaluated:

C1/C2 → R₁ complex

Convert A to complex

A*R₁ → R₂ complex

Convert B to complex

Convert F to complex

B*F → R₃ complex

R₂ + R₃ → R₄ complex evaluation completed.

Example 6

Consider the expression C1/C2 + (A-B) where the operands are defined as the example above. The expression is evaluated:

A-B → R₁ real

Convert R₁ to complex

C1/C2 → R₂ complex

R₁ + R₂ → R₃ complex evaluation completed.

*Rules
(continued)*

Example 7

Given $C1*D + R / I$ where $C1$ is complex; D is double precision; R is real; I is integer. In this simple expression all variables are raised to the dominant operand type, and complex arithmetic is used in the entire expression.

Convert D to complex†

$C1*D \rightarrow R_1$ complex

Convert R to complex

Convert I to complex

$R / I \rightarrow R$ complex

$R_1 + R_2 \rightarrow R_3$ complex evaluation completed.

Example 8

Given $C1*D + (R/I)$ where the variables are the same as in the example above. In this combined expression the simple expression (R/I) is evaluated first in its dominant mode which is real. The order and mode of the evaluation is as follows:

Convert I to real

$R/I \rightarrow R_1$ real

Convert R_1 to complex

Convert D to complex

$C1*D \rightarrow R_2$ complex

$R_1 + R_2 \rightarrow R_3$ complex evaluation complete.

† Truncate D to the most significant portion, which is used as the real part of a complex number; the imaginary part is set to zero.

IV

REPLACEMENT STATEMENTS

EVALUATION

MIXED MODE

MULTIPLE

v = exp

REPLACEMENT STATEMENTS

INTRODUCTION

The general form of a replacement statement is $A = \text{exp}$, where exp is an expression and A is a variable name, simple or subscripted. The operator, $=$, means that the value of A is replaced by the value of the evaluated expression, exp , with conversion for mode, if necessary. If A is a logical variable, exp must be a logical expression.

Examples

```
A = -A
B(3,4) = CALC(I+1)*BETA + 2.3478
XITHETA = 7.4*DELTA + A(I,J,K)**BETA
RESPNS = SIN(ABAR(INV + 2,JBAR)) / ALPHA(J)
JMAX = 19
AREA = SIDE1*SIDE2
PERIM = 2.*(SIDE1 + SIDE2)
```

MIXED-MODE REPLACEMENT STATEMENTS

The type of an evaluated expression is determined by the type of the dominant operand (see Chapter 3). This, however, does not restrict the types that identifier A may assume in the replacement statement where $A = \text{expression}$. The converted result of a complex expression may replace A when A is real. The following table shows the $A = \text{exp}$ relationship for all the standard modes. The mode of A determines the mode of the result; the dominant operand in the expression determines the mode of exp .

**MIXED-MODE
REPLACEMENT
STATEMENTS**
(continued)

Given: C_i, A_C complex
 D_i, A_D double
 R_i, A_R real
 I_i, A_I integer

Example 1

$$A_C = C_1 * C_2 - C_3 / C_4$$

$$(6.90525, 15.39287) = (4.4, 2.1) * (3.0, 2.0) - (3.3, 6.8) / (1.1, 3.4)$$

The expression is complex. Therefore, the result of the expression evaluation is a two-word, floating point quantity which represents a complex number. This complex quantity is stored in A_C as a two-word complex result, since A_C is type complex.

Example 2

$$A_R = C_1$$

$$4.4000 = (4.4, 2.1)$$

The expression is complex. A_R is real; therefore the real part of C_1 replaces A_R .

Example 3

$$A_R = C_1 * (0., -1.)$$

$$2.100 = (4.4, 2.1) * (0., -1.)$$

The expression is complex. A_R is real; the real part of the result of the complex multiplication replaces A_R .

Example 4

$$A_I = R_1 / R_2 * (R_3 - R_4) + I_1 - (I_2 * R_5)$$

$$13 = 8.4 / 4.2 * (3.1 - 2.1) + 14 - (1 * 2.3)$$

The expression is real. A_I is integer; the result of the expression evaluation, a real, is converted to an integer replacing A_I .

Table 4-1. Replacement Statement A = exp

A is an Identifier
 exp is an Expression
 $\phi(f)$ is the Evaluated Expression

Type of A \ Mode of $\phi(f)$	Complex	Double	Real	Integer†	Logical	Masking
Complex	Store real and imaginary parts of $\phi(f)$ in real and imaginary parts of A.	Round $\phi(f)$ to real. Store in real part of A. Store zero in imaginary part of A.	Store $\phi(f)$ in real part of A. Store zero in imaginary part of A.	Convert $\phi(f)$ to real and store in real part of A. Store zero in imaginary part of A.	Store $\phi(f)$ in real part of A. Store zero in imaginary part of A.	Illegal
Double	Store real part of $\phi(f)$ in A and replace the least significant part with plus zero.	Store $\phi(f)$ (most and least significant parts) in A (most and least significant parts).	If $\phi(f)$ is ±, affix #0 as least significant part. Store in A (most and least significant parts).	Convert $\phi(f)$ to real. Fill least significant part with plus zeros. Store in A (most and least significant parts).	Store $\phi(f)$ in most significant part of A. Store zero in least significant part.	Illegal
Real	Store real part of $\phi(f)$ in A. Imaginary part is lost.	Round $\phi(f)$ to real and store in A. Least significant part of $\phi(f)$ is lost.	Store $\phi(f)$ in A.	Convert $\phi(f)$ to real. Store in A.	Store $\phi(f)$ in A.	Store $\phi(f)$ in A.
Integer	Truncate real part of $\phi(f)$ to integer. Store in A. Imaginary part is lost.	Truncate $\phi(f)$ to integer and store in A. The least significant part of $\phi(f)$ is lost.	Truncate $\phi(f)$ to integer. Store in A.	Store $\phi(f)$ in A.	Store $\phi(f)$ in A.	Store $\phi(f)$ in A.
Logical	Store real part of $\phi(f)$ in A. Imaginary part is lost.	Store most significant part of $\phi(f)$ in A. Least significant part is lost.	Store $\phi(f)$ in A.	Store $\phi(f)$ in A.	Store $\phi(f)$ in A.	Store $\phi(f)$ in A.

† Hollerith constants and octal constants are assumed to be type integer in an expression.

Example 5

$$\begin{aligned} A_D &= D_1^{**}2*(D_2+(D_3*D_4)) + (D_2*D_1*D_2) \\ 4.96800000000000D+01 &= 2.0D^{**}2*(3.2D+(4.1D*1.0D)) \\ &\quad + (3.2D*2.0D*3.2D) \end{aligned}$$

The expression is double precision. A_D is double precision; the result of the expression evaluation, a double-precision floating quantity, replaces A_D .

Example 6

$$\begin{aligned} A_I &= C_1*R_1-R_2+I_1 \\ 33 &= (4.4,2.1)*8.4 - 4.2 + 0.4 \end{aligned}$$

The expression is complex. Since A_I is integer, the truncated real part of the evaluated expression replaces A_I .

MULTIPLE REPLACEMENT STATEMENTS

The multiple replacement statement is a generalization of the replacement statements discussed earlier in this chapter, and its form is

$$\psi_n = \psi_{n-1} = \dots = \psi_2 = \psi_1 = \text{expression}$$

For example, $A = B = C = 4.*\cos(x) + 32$. The ψ_i are variables of any type, and the multiple replacement statement replaces each of the variables ψ_1, \dots, ψ_n with the value of the expression following the last equals. This is done with the standard conventions used in mixed-mode arithmetic statements, as shown in the following examples, where each successive replacement $\psi_i = \psi_{i-1}$ is done based only on the modes of ψ_i and ψ_{i-1} (see example 5 below).

Given that A is real; C1,C2 is complex; D is double; and I is integer:

Example 1

A = D = 3.1415926535897932384626D	3.1415926535897932384626D → D
	3.141592654 → A

Note: In octal representation, A is not rounded; in this case, conversion of A from octal to decimal causes A to appear as if it had been rounded.

Example 2

I = A = 4.6	4.6 → A
	4. → I

Example 3

A = I = 4.6	4 → I
	4.0 → A

Example 4

I = A = C1 = (10.2,3.0)	10.2 → C1 real
	3.0 → C1 imaginary
	10.2 → A
	10 → I

Example 5

C2 = A = I = C1 = (13.4,16.2)	13.4 C1 real
	16.2 C1 imaginary
	13 I
	13.0 A
	13.0 C2 real
	0.0 C2 imaginary

Only one expression is permissible and it must be the last replacement in the set of replacements. If an expression is introduced in the middle of a string, the diagnostic ILLEGAL USE OF A REPLACEMENT STATEMENT will be printed.

V

TYPE DECLARATIONS AND STORAGE ALLOCATION

```
COMPLEX nlist
DOUBLE PRECISION nlist
REAL nlist
INTEGER nlist
LOGICAL nlist
IMPLICIT typ(let[,let]...) [,typ(let[,let]...)]...
PARAMETER (param=exp[,param=exp]...)
DIMENSION v(n1[,n2][,n3]) [v(n1) [,n2] [,n3]]...
COMMON/blk1/nlist[/blk2/nlist...]
EQUIVALENCE(nlist)[,(nlist)]...
DATA nlist1/clist1/[[,]nlist2/clist2/[,]...nlistn/clistn/]
```


TYPE DECLARATIONS AND STORAGE ALLOCATION

INTRODUCTION

The FORTRAN code discussed in this chapter determines the types of variables that are used and where they will be stored in the program. Since this information must be established before the program is executed, the TYPE and storage allocation instructions are nonexecutable and are handled at compile time.

STATEMENT ORDERING

TYPE, PARAMETER, IMPLICIT, DIMENSION, COMMON and EQUIVALENCE are specification statements. They must appear ahead of the first executable statement in the program; otherwise, the diagnostic is DECLARATIVE STATEMENTS MUST PRECEDE ALL ARITHMETIC STATEMENTS. The IMPLICIT statement must appear before any other specification statement, except PARAMETER. The PARAMETER statement must precede any specification statement that contains a parameter reference. DATA declarations may appear anywhere in a program unit following the specification statements.

TYPE DECLARATION

The TYPE declaration statement provides the compiler with information on the internal representation and names of variables and functions. Both the name and the datum identified by the name have a type. There are five variable types declared by one of the following statements:

<u>Statement</u>	<u>Characteristics</u>
COMPLEX nlist	2 words/element; floating point
DOUBLE PRECISION nlist	2 words/element; floating point
REAL nlist	1 word/element; floating point
INTEGER nlist	1 word/element; integer
LOGICAL nlist	1 word/element; integer

nlist is a string of identifiers separated by commas; positive integer constant subscripts are permitted. An example of nlist is:

A, B1, CAT, D36F, GAR(1,2,3)

Rules

- The TYPE declaration is nonexecutable and must precede the first executable statement.
- A TYPE declaration for type double or complex turns off the optimizer for expressions in the entire routine in which the type declaration appears.
- If an identifier is declared in two or more TYPE declarations, the diagnostic ATTEMPT TO DOUBLY TYPE A VARIABLE will appear.

- An identifier not declared in a TYPE declaration or in IMPLICIT typing has a default type of real or integer. Type integer is assumed if the first letter of the name is I, J, K, L, M or N; for any other letter, it is type real.
- The TYPE statement may also provide dimension information. When subscripts appear in nlist, the associated identifier is the name of an array, and the product of the subscripts determines the amount of storage to be reserved for that array. Where the amount of storage is specified in more than one declarative statement, a diagnostic appears.
- Type logical is essentially type integer. However, any expression containing a logical variable may ignore the mixed-mode conventions. Results from the expression are unpredictable; for example, L + X, where L is logical and X is real, generates a simple unnormalized add of the two variables, resulting in meaningless code.

Examples

```
COMPLEX ALPHA,B21,INTERIM
DOUBLE PRECISION POST(10,10),NO,BILL
REAL K(2,3),RE,IN2F
INTEGER X,Y,GAR(1,2,3)
DOUBLE PRECISION RL,MASS(10,10)
```

For a discussion of typing a function name, see Function Definition in Chapter 7.

IMPLICIT TYPING

Implicit typing is used to identify a type associated with single letters that appear as the first letter of names of variables, constants, arrays or functions. It is used primarily to modify the typing conventions that are used in FORTRAN as default types which apply when no explicit typing appears among the specification statements of a program unit. Explicit typing overrides both the default type and the implicit type of a name.

IMPLICIT STATEMENT

The form of the IMPLICIT statement is:

```
IMPLICIT typ(let1[,let2]...)[,typ(let3[,let4]...)]...
```

where typ may be COMPLEX, DOUBLE PRECISION, REAL, INTEGER or LOGICAL; and let may be a list of single letters, a₁[,a₂]..., or a range of letters, a₁-a₂, or a combination of single letters and ranges of letters.

Examples

```
IMPLICIT REAL (I,N)
IMPLICIT INTEGER (A,C,X)
IMPLICIT COMPLEX (C,V-Z)
IMPLICIT REAL (I),INTEGER (A)
IMPLICIT LOGICAL (A-C),INTEGER (D-F,R-T,Z)
IMPLICIT INTEGER (A-J),REAL (K-Z)
```

Rules

- Any letter appearing as a single letter or within a range of letters in an IMPLICIT statement determines the type of those symbolic names beginning with that letter.
- The symbolic names affected may be names of constants, variables, arrays or external functions in the program unit.
- The type applies to all variables that are used within a main program, or within a particular subprogram. The implicit typing does not apply outside the range of a program unit.
- Implicit typing may not be used to modify the type of in-line functions.
- Implicit typing may not be used to specify a type for processor-supplied external routines such as SIN, TAN, etc.
- Explicit type statements take precedence over any implicit typing specified in a program unit.
- IMPLICIT statements must appear before any other specification statements except the PARAMETER statement.
- A particular letter may be used only once in any IMPLICIT statement or statements appearing in a given program unit.
- IMPLICIT statements refer to letters and may not contain explicit symbolic names of more than one letter. Therefore no dimension information may be supplied.

5.6

Type Declarations and Storage Allocation

Example 1

```
*FORTRAN,FL
CARD APPROXIMATE PROGRAM LOCATION
NUMBER
1      0      PROGRAM TEST
2      0      IMPLICIT INTEGER (A,C),REAL (I-K),DOUBLE (X-Z)
3      0      IMPLICIT COMPLEX (E,Q)
4      0      ABC = 2.8
5      0      IVAR = 1.5
6      0      JVAR = 3.6
7      0      CB = JVAR
8      0      XE = 10.5252525205
9      0      ZA = -XE
10     0      EBAR = (3.2E2,-6.666)
11     0      WRITE (6,102) ABC
12     23     102 FORMAT(1H05X,"ABC="I3/)
13     23     WRITE (6,104) IVAR,JVAR
14     34     104 FORMAT(1H05X,"IVAR=F4.1,5X,"JVAR=F4.1/)
15     34     WRITE (6,105) CB
16     43     105 FORMAT(1H05X,"CB="I3/)
17     43     WRITE (6,106) ZA
18     52     106 FORMAT(1H05X,"ZA="D16.8/)
19     52     END

LENGTH OF ROUTINE TEST      111
VARIABLE ASSIGNMENTS
ABC = 64      IVAR = 63      JVAR = 62      CB = 61      XE = 57      ZA = 55
EBAR = 53

SUBROUTINES CALLED
OUTPTC EXIT
COMPILE TIME = 21 MILLISECS

PROGRAM SPACE IS 1642

ORIGIN ENTRY POINTS AND LOCATIONS
4      TEST      4
115    Q8QERR    115
130    EXIT      143      END      143      STOP      133
150    KODER     150
1203   OUTPTC    1203      OTPTER    1636

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS      54077
THE PROGRAM CURRENTLY USES 946
THE LOADING PROCESS USED 8835
THE MAXIMUM LCM IS       449602
THE PROGRAM CURRENTLY USES 4785
LCM AREA MAP BY BUFFER TYPE
  MISC      185  SYSIOU      6  PR      2048  RANFO      64  SYSSAT      512
  SYSOM1    1024  SYSSCM    946

ABC= 2
IVAR= 1.5      JVAR= 3.6
CB= 3
ZA= -1.05252525D+06

TERMINATION DATE = 06/27/77
TERMINATION TIME = 13/15/19

TOTAL CPU TIME IN MILLISECONDS =      89
PPU TIME IN MILLISECONDS =      400
PAGES PRINTED =            3
TOTAL RESOURCES USED =      14

DISK BLOCK USAGE SUMMARY
ODD UNITS          EVEN UNITS
DISK 0             DISK 1
LIMIT ON BLOCKS ALLOWED 4095           4095
MAXIMUM BLOCKS USED   3               0
PLIB BLOCKS FOR THIS PROJECT 1
```

**SYMBOLIC NAME
OF A CONSTANT**

The PARAMETER statement is used to give a symbolic name to a constant. The name is used subsequently in specifications or in executable statements within a program unit. Thus, the symbolic name of the constant replaces the constant itself in FORTRAN statements. A useful application might be in a program unit containing many references to variables in a DIMENSION statement having the same dimension specification. Changing the value of the constant is simplified using the symbolic name of a constant and fewer programming errors occur at the time of the program change.

PARAMETER STATEMENT

The form of the PARAMETER statement is

PARAMETER (param₁=exp₁[,param₂=exp₂]...)

where param is the symbolic name of a constant; and exp is an expression containing constants and/or symbolic names of constants previously defined.

Examples

```
PARAMETER (PI=3.1412)
PARAMETER (X=4.265E-9)
PARAMETER (Y=PI**2.,I=300)
PARAMETER (K=I+100,Z=X**Y)
```

Rules

- exp must be a constant expression.
- Type of constants is determined by the name according to the same rules that govern the type of symbolic names of variables, arrays and functions.
- The value is defined by the constant expression, exp.
Operands within exp may be constants or symbolic names of constants.
- The value of the constant replaces the symbolic name.
(String substitution in a statement is not used.)
- The symbolic name of a constant may not be used subsequently as a variable, an array or a function.
- A parameter must be defined before it is used. The definition may occur in the same or a preceding parameter statement within that program unit.
- A parameter may be defined only once in a PARAMETER statement within a program unit.
- Parameters, that is, symbolic names of constants, may not appear in a FORMAT statement.
- Parameters may not have their type changed in type statements following the PARAMETER statement.

Example 2

```

*FORTRAN>FL
CARD APPROXIMATE
NUMBER PROGRAM LOCATION
1 0 PROGRAM TEST
2 0 PARAMETER (A=5*2,JK=2)
3 0 PARAMETER (IP=A+JK+3,JP=5)
4 0 DIMENSION L(JP,JP)
5 0 II = IP
6 31 JJ = JP
7 31 WRITE (6,102) II,JJ
8 44 102 FORMAT(1HO,5X,"II="I3,5X,"JJ="I3/)
9 44 END

LENGTH OF ROUTINE TEST 53
VARIABLE ASSIGNMENTS
L - 0 II - 46 JJ - 45
SUBROUTINES CALLED
OUTPTC EXIT
COMPILE TIME = 7 MILLISECS

PROGRAM SPACE IS 1604

ORIGIN ENTRY POINTS AND LOCATIONS
4 TEST 35
57 Q80ERR 57
72 EXIT 105 END 105 STOP 75
112 KODER 112
1145 OUTPTC 1145 OPTTER 1600
OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS 54077
THE PROGRAM CURRENTLY USES 916
THE LOADING PROCESS USED 8805
THE MAXIMUM LCM IS 449602
THE PROGRAM CURRENTLY USES 4755
LCM AREA MAP BY BUFFER TYPE
MISC 185 SYSIOU 6 PR 2048 RANFO 64 SYSSAT 512
SYSDM1 1024 SYSSCM 916

II= 10 JJ= 5

TERMINATION DATE = 06/27/77
TERMINATION TIME = 13/15/29

TOTAL CPU TIME IN MILLISECONDS = 204
PPU TIME IN MILLISECONDS = 349
PAGES PRINTED = 2
TOTAL RESOURCES USED = .15

DISK BLOCK USAGE SUMMARY
ODD UNITS EVEN UNITS
DISK 0 DISK 1
LIMIT ON BLOCKS ALLOWED 4095 4095
MAXIMUM BLOCKS USED 3 0
PLIB BLOCKS FOR THIS PROJECT 1

```

**DIMENSION
DECLARATION**

A subscripted variable represents an element of an array. For example, A(8) in an expression refers to the eighth element of the array A. Storage is reserved for arrays in the nonexecutable statements DIMENSION, COMMON, or TYPE. DIMENSION A(8) means save eight elements to be referenced by the array name A.

DIMENSION v₁(n₁[,n₂][,n₃])[,v₂(n₁[,n₂][,n₃])]...]

The variable names v_i, may have one, two, or three integer constant subscripts separated by commas, as in X(2,50,10). If more than three subscripts appear after an array name in a DIMENSION statement, the compiler diagnostic is GREATER THAN THREE DIMENSIONS IN DIMENSION STATEMENT. The DIMENSION declaration is nonexecutable and must precede the first executable statement.

The number of computer words reserved for an array is determined by the product of the subscripts in the subscript string and the type of the variable.

ARRA(2,4,3) would have $2 \times 4 \times 3$ elements, or 24.

A maximum of 65,535 elements may be reserved in any one array. If the maximum is exceeded, the diagnostic is DIMENSIONED ARRAY TOO LARGE FOR CORE. Practically speaking, room for instructions and system requirements is needed for any program. In the declarations

COMPLEX ATOM
DIMENSION ATOM(10,20),

the number of elements in the array ATOM is 200. Two words are used to store a complex element; therefore, the number of computer words reserved is 400. This is also true for double-precision arrays. The maximum array size for arrays with

two-word elements is 32,767. For real and integer arrays, the number of words in an array equals the number of elements in the array.

If an array is dimensioned in more than one declaration statement, the diagnostic is AN ARRAY NAME PREVIOUSLY USED.

Examples

```
DIMENSION A(20,2,5)
DIMENSION MATRIX(10,10,10),VECTOR(100),ARRAY(16,27)
```

No diagnostic is provided during execution for index values that exceed the DIMENSION specification but stay within the program field length , for example:

```
DIMENSION A(10),B(15)
A(14) = 42.
B(02) = 91.
```

In the above example, no index check is made. A(14) is actually B(4). The replacement statement is executed and B(4) is destroyed. In order to add a check on dimensions, many more instructions would need to be added to a compiled program; the check instructions would be required every time an index is calculated. This is too costly in terms of run time and overall efficiency of the program. (When needed for program debugging, an index-checking facility can be activated through the use of a program preprocessor such as NCAR's FRED).

**DIMENSION
DECLARATION**
(continued)

Negative subscripts may be used to access dimensioned variables during program execution. For example, in the statement DIMENSION A(10),B(15), the following correspondences apply. Element B(-2) is the same as element A(8). The 0th element precedes B(1), B(-1) precedes B(0), etc.

Extended Indices of A in Positive Direction	A(1)	B(-9)	Extended Indices of B in Negative Direction
	A(2)	B(-8)	
	A(3)	B(-7)	
	A(4)	B(-6)	
	:	:	
	A(8)	B(-2)	
	A(9)	B(-1)	
	A(10)	B(0)	
	A(11)	B(1)	
	A(12)	B(2)	
Extended Indices of A in Positive Direction	A(13)	B(3)	
	A(14)	B(4)	
	:	:	
	A(23)	B(13)	
	A(24)	B(14)	
	A(25)	B(15)	

If a subscript exceeds the program field in either direction, the execution diagnostic SCM DIRECT RANGE will be printed on the Control Data 7600.

A general formula for calculating a single element of an array with more than one subscript is included under "Array Structure" in chapter 2.

**VARIABLE
DIMENSIONS**

When an array identifier and its dimensions appear as formal parameters in a function or subroutine, the dimensions may be assigned through the actual parameter list accompanying the function reference or subroutine call. Notice that the array name, as well as the DIMENSION statement identifiers M and N, must appear in the parameter list. The dimensions must not exceed the maximum array size specified by the DIMENSION declaration in the calling program. However, no compiler diagnostic is provided and storage will be overwritten in the array area during execution. (See Variable Dimensions in Subprograms in Chapter 7.) For example:

```
SUBROUTINE VAR(A,M,N)
DIMENSION A(M,N)
```

If either A, M, or N does not appear in the formal parameter list or if they appear in COMMON, the compiler diagnostic is VARIABLE DIMENSIONED IDENTIFIER NOT IN FORMAL PARAMETER LIST.

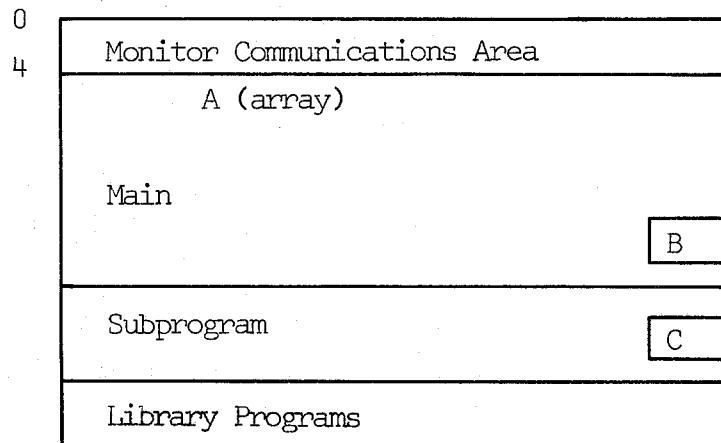
Variable dimensions may not be used in a main program; the diagnostic will be VARIABLE DIMENSIONED IDENTIFIER NOT IN FORMAL PARAMETER LIST.

COMMON DECLARATION**LOCAL VARIABLES**

When a programmer uses variable names in a program or declares arrays in a DIMENSION statement, the compiler provides storage within the program or subprogram for these variables and arrays. A place is saved for them within the program and the name of the variable or array is sometimes called a "place holder." Since these variables are located *only* within the program or subprogram using them, they are often called "local variables."

Example

System Reference to Start of User's Program:



In this example, the array A and the variables B and C are local variables. They are stored only within the program or subprogram referring to them.

COMMON VARIABLES

The COMMON declaration may establish blocks of storage for variables and arrays that are available generally to the main program, to various subprograms, or to both. A particular common block area is available to any program unit which declares this block in a declaration statement and *only* to those program units with this declaration.

**COMMON
VARIABLES**
(continued)

The program shown in Example 3 is loaded in the following locations:

0	Monitor Communications Area
4 ₈	Program TEST
12 ₈	T1 Common
14 ₈	Blank Common
17 ₈	Subprogram TT
26 ₈	T2 Common
30 ₈	.41 Common
32 ₈	BLKA Common
FL	

The Variable Assignments table specifies a relative address for each variable. If the variable is in common, Cxx appears after the address. (xx is merely a count of the blocks in any one routine. This count does not necessarily agree from program to subprogram.) D and E are in common T1; A, B, and C are in blank common; X and Y are local variables where no common block is specified. Common blocks are loaded as they appear in the program; T1 is first, then blank common. Thus T1 is C 0, blank common is C 1. Notice that both of these common blocks are loaded following the program, which is loaded into 4₈. The subroutine has common blocks not declared in the main program, T2, .41, BLKA. These are loaded following the subprogram TT which first declares them to be common blocks. The order of common blocks in the subroutine is

C 0	blank common
C 1	T2
C 2	.41
C 3	BLKA

Example 3

```

*FORTRAN,FL

CARD APPROXIMATE
NUMBER PROGRAM LOCATION
1 0 PROGRAM TEST
2 0 COMMON/T1/D,E
3 0 COMMON A,B
4 0 COMMON C
5 0 X=4.0
6 0 Y=5.0
7 0 END

LENGTH OF ROUTINE TEST 6
VARIABLE ASSIGNMENTS
D - OC 0 E - 1C 0 A - 0C 1 B - 1C 1 C - 2C 1 X - 5
Y - 4
SUBROUTINES CALLED
EXIT
COMMON BLOCKS AND LENGTHS
T1 - 2 - 3
COMPILE TIME = 5 MILLISECS

CARD APPROXIMATE
NUMBER PROGRAM LOCATION
1 0 SUBROUTINE TT
2 0 COMMON A,B,C
3 0 COMMON /T2/ F,G
4 0 COMMON/41/ H,I
5 0 COMMON /BLKA/U,V
6 0 END

LENGTH OF ROUTINE TT 7
VARIABLE ASSIGNMENTS
A - DC 0 B - 1C 0 C - 2C 0 F - 0C 1 G - 1C 1 H - 0C 2
I - 1C 2 U - 0C 3 V - 1C 3
COMMON BLOCKS AND LENGTHS
- 3 T2 - 2 .41 - 2 BLKA - 2
COMPILE TIME = 3 MILLISECS

PROGRAM SPACE IS 54

ORIGIN ENTRY POINTS AND LOCATIONS
4 TEST 4
17 TT 17
34 EXIT 47 END 47 STOP 37

COMMON BLOCKS LOCATION
T1 12 14 T2 26 .41 30 BLKA 32

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS 54101
THE PROGRAM CURRENTLY USES 69
THE LOADING PROCESS USED 7949
THE MAXIMUM LCM IS 449602
THE PROGRAM CURRENTLY USES 3899
LCM AREA MAP BY BUFFER TYPE
MISC 185 SYSIOU 6 PR 2048 RANFO 64 SYSSAT 512
SYSDM1 1024 SYSSCH 60

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11:36:24

TOTAL CPU TIME IN MILLISECONDS = 82
PPU TIME IN MILLISECONDS = 209
PAGES PRINTED = 3
TOTAL RESOURCES USED = .11

DISK BLOCK USAGE SUMMARY
ODD UNITS EVEN UNITS
DISK 0 DISK 1
LIMIT ON BLOCKS ALLOWED 4095 4095
MAXIMUM BLOCKS USED 0 3
PLIB BLOCKS FOR THIS PROJECT 1

```

**COMMON
VARIABLES**
(continued)

The program shown in Example 4 is loaded as follows:

0	Monitor Communications Area
4 ₈	Blank Common
7 ₈	Program TEST
15 ₈	T1 Common
17 ₈	Subprogram TT
26 ₈	T2 Common
30 ₈	.41 Common
32 ₈	BLKA Common
FL	

When the reference to blank common appears in the first program loaded, and it is the first common block specified, then blank common is loaded into 4₈. (For programs that are approaching the maximum core size, some saving of core is made at load time by having blank common at 4₈. The loader, which is about 10,000₈, will then use the same space as the blank common storage area; after this, it is set to zero.) Blank common must appear as the first common block or it will follow the program, as in Example 3.

Areas of common information may be specified by the declaration:

```
COMMON nlist
COMMON /blk1/nlist[/blk2/nlist ...]
```

Example 4

#FORTRAN,FL

CARD NUMBER	APPROXIMATE PROGRAM LOCATION
1	0 PROGRAM TEST
2	0 COMMON A,B
3	0 COMMON C
4	0 COMMON/T1/D,E
5	0 X=4.0
6	0 Y=5.0
7	0 END

LENGTH OF ROUTINE TEST 6

VARIABLE ASSIGNMENTS

A	- DC 0 B -	1C 0 C -	2C 0 D -	0C 1 E -	1C 1 X -	5
Y	- 4					

SUBROUTINES CALLED

EXIT

COMMON BLOCKS AND LENGTHS

-	3 T1 -	2
---	--------	---

COMPILE TIME = 5 MILLISECS

CARD NUMBER	APPROXIMATE PROGRAM LOCATION
1	0 SUBROUTINE TT
2	0 COMMON A,B,C
3	0 COMMON /T2/ F,G
4	0 COMMON/H1/ H,I
5	0 COMMON /BLKA/ U,V
6	0 END

LENGTH OF ROUTINE TT 7

VARIABLE ASSIGNMENTS

A	- DC 0 B -	1C 0 C -	2C 0 F -	0C 1 G -	1C 1 H -	0C 2
I	- 1C 2 U -	DC 3 V -	1C 3			

COMMON BLOCKS AND LENGTHS

-	3 T2 -	2 .41 -	2 BLKA -	2
---	--------	---------	----------	---

COMPILE TIME = 3 MILLISECS

PROGRAM SPACE IS 54

ORIGIN ENTRY POINTS AND LOCATIONS

7 TEST	7
17 TT	17
34 EXIT	47 END 47 STOP 37

COMMON BLOCKS LOCATION

4 T1	15 T2	26 .41	30 BLKA	32
------	-------	--------	---------	----

OVERALL CORE USE STATISTICS (DECIMAL)

THE MAXIMUM SCM IS 54101

THE PROGRAM CURRENTLY USES 60

THE LOADING PROCESS USED 7946

THE MAXIMUM LCM IS 449602

THE PROGRAM CURRENTLY USES 3899

LCM AREA MAP BY BUFFER TYPE

MISC 185	SYSIOU 6 PR	2048 RANFO 64 SYSSAT 512
SYSDM1 1024	SYSSCH 60	

TERMINATION DATE = 06/21/77

TERMINATION TIME = 11:37:02

TOTAL CPU TIME IN MILLISECONDS = 49

PPU TIME IN MILLISECONDS = 248

PAGES PRINTED = 3

TOTAL RESOURCES USED = .10

DISK BLOCK USAGE SUMMARY

ODD UNITS	EVEN UNITS
DISK 0	DISK 1
LIMIT ON BLOCKS ALLOWED 4095	4095
MAXIMUM BLOCKS USED 0	3
PLIB BLOCKS FOR THIS PROJECT 1	

**COMMON
VARIABLES**
(continued)

blk_i is a common block identifier up to six characters long; it is the common block name. An alphanumeric identifier must start with a letter. A numeric identifier starting with a number must contain only numbers. Leading zeros in identifiers are ignored. Zero by itself is an acceptable common block identifier. The following are COMMON identifiers:

/AZ13/	/1/
/MAXMUS/	/146/
/Z/	/6600/
/XRAY/	/0/

nlist is a string of identifiers representing simple and subscripted variables. If a nonsubscripted array name appears in nlist, the dimensions must be defined by a TYPE or DIMENSION declaration in that program. If an array is dimensioned in more than one declaration, a compiler diagnostic is given.

The common block identifier is omitted for blank common.
Common || nlist and common nlist both denote blank common.

Examples

```
COMMON A,B,C
COMMON /BLOCKA/A(15),B,C/123/DEL(5,2),ECHO,X
COMMON B,A /VECTOR/VECTOR(15),VECTORA,VECTORB
```

Rules

- COMMON is nonexecutable and must precede the first executable statement. Any number of COMMON declarations may appear in a program.
- A COMMON declaration with a given name specified, /NAME/, may appear only once in any given program or subprogram. If one card is not long enough to hold all variable names in the list, continuation cards may be used.

```
COMMON/A1/A,B,C,...,
1E,Q,R(10,100)
```

- If a repeated name such as COMMON/A1/E,Q,R(10,100) is used instead of a continuation card, the diagnostic will be DUPLICATE BLOCK NAME.
- Blank common is the only declaration that may be used more than once in the same program and it must appear consecutively, such as

```
COMMON A,B  
COMMON C,D,E
```
- If DIMENSION or COMMON or TYPE declarations appear together, the order is immaterial.
- Common block identifiers are used only for block identification within the compiler; they may be used elsewhere in the program as other kinds of identifiers except as subroutine and program names in the same job.
- An identifier in one common block may not appear as an identifier in another common block. (If it does, the name is doubly defined.)
- The order of array storage within a common block is determined by the list declaration.
- At the beginning of program execution the contents of all common areas are zero except those areas in named COMMON which were specified in a DATA declaration. (This can be changed by using *RUN,I; then at the beginning of program execution the contents of all common areas are negative indefinite (see chapter 2) except those areas in named COMMON which were specified in a DATA declaration.)
- Blank common may not be entered with data using a DATA statement. A diagnostic will be given that blank common may not be preset.

Rules
(continued)

- The length of a common block in computer words is determined from the number and type of the list variables. In the following statements, the length of common block A is 12 computer words. The origin of the common block is Q(1).

```
COMMON/A/Q(4), R(4),S(2)
REAL Q,R
COMPLEX S
```

Block A

Origin	Q(1)
	Q(2)
	Q(3)
	Q(4)
	R(1)
	R(2)
	R(3)
	R(4)
S(1)	Real part
S(1)	Imaginary part
S(2)	Real part
S(2)	Imaginary part

- If a subprogram does not use all of the locations reserved in a common block, unused variables may be necessary in the COMMON declaration to insure proper correspondence of common areas.

Main program	COMMON/SUM/A,B,C,D
Subprogram	COMMON/SUM/DUMMY(2),C

In the above example, only the variable C is used in the subprogram. The unused variable DUMMY is necessary to space over the area reserved by A and B.

- The first appearance of a common block establishes its length. If a longer block with the same name appears in a subprogram later, the diagnostic ATTEMPT TO CHANGE LENGTH OF COMMON BLOCK FROM XXXXXX TO XXXXXX is given.

- The length of a common block must not be extended by subprograms using the block. However, the length may be shortened by subprograms. The symbolic names used within the block may differ as shown below.
- Each subprogram using a common block assigns the allocation of words in the block. The identifiers used within the block may differ as to name, type, and number of elements, although the block identifier itself must remain the same.
- The loader will overlay any blank common when it appears *first* in a program, regardless of the size of the common length declared (larger or smaller). Therefore, when core is short, as in a large program, and every location is needed, blank common must appear first so that the loader can overlay it. If blank common does not appear first, the program may not load.

Example 5

```
PROGRAM MAIN
COMPLEX C
COMMON/TEST/C(20)/36/A,B,Z
.
.
END
```

The length of TEST is 40 computer words. The subprogram may rearrange the allocation of words as in:

```
SUBROUTINE ONE
COMMON/TEST/A(10),G(10),K(10)
COMPLEX A
.
.
END
```

The length of this TEST is also 40 words. The first 10 elements (20 words) of the block represented by A are complex elements. Array G is the next 10 words and array K is the last 10 words. Within the subprogram, elements of G are treated as real floating point quantities and elements of K are treated as integer quantities.

**EQUIVALENCE
DECLARATION**

The EQUIVALENCE declaration permits variables to share the same storage locations. It is not a replacement statement and thus does not equate variables mathematically. The equivalence variables share the memory location(s) of the leftmost variable identifier. The variable on the left is called a referenced identifier in this program. Any name in the equivalence list calls out this memory location; thus, more than one identifier for a cell is established. The general form is EQUIVALENCE (nlist[,nlist])...

EQUIVALENCE (A,B,C,...),(X(1),Y(4),Z(2)...)

(A,B,C,...) is an EQUIVALENCE group of two or more simple or single-subscripted variable names. The equivalence group is defined as C shares the location of B, B shares locations of A; thus, C shares locations of A. A is the referenced identifier and appears on the left. The equivalence group is order-dependent. If one or more variables in the group are in COMMON, one of the COMMON variables must be the referenced identifier.

A multiply-subscripted variable can be represented only by a singly-subscripted variable. The correspondence is:

$A(i,j,k)$ is the same as $A((i+(j-1)*I + (k-1)*I*j))$

where i,j,k are integer constants; I and J are the integer constants appearing in DIMENSION A(I,J,K). For example, in DIMENSION A(2,3,4), the element A(1,1,2) is represented by A(7).

EQUIVALENCE is most commonly used when two or more arrays can share the same storage locations. The lengths may be different or equal.

Example 6

```
DIMENSION A(10,10),I(100)
EQUIVALENCE (A,I)
8 READ 10,A
      .
      .
9 READ 20,I
      .
      .
END
```

The EQUIVALENCE declaration assigns the first element of array A and array I to the same storage location. The READ statement 8 stores the A array in consecutive locations. All operations using A should be completed before statement 9 is executed, since the values of array I are read into the storage locations previously occupied by A.

**EQUIVALENCE
DECLARATION**

(continued)

Example 7

*FORTRAN,FL

CARD NUMBER	APPROXIMATE PROGRAM LOCATION	TEST	47
1	0	PROGRAM TEST	
2	0	DIMENSION A(10),B(2,5)	
3	0	EQUIVALENCE (A(5),B(3))	
4	0	EQUIVALENCE (X,1)	
5	0	I=17204000000000000000000000000000	
6	15	WRITE (6,100) X	
7	26	100 FORMAT (*0X=F5.1)	
8	26	B(1,2)=99.	
9	26	WRITE (6,101) A(5)	
10	37	101 FORMAT (*0A(5)=F5.2)	
11	37	END	
LENGTH OF ROUTINE		TEST	47
VARIABLE ASSIGNMENTS			
A	- 0 B -	2 X - 14 I - 14	
SUBROUTINES CALLED			
OUTPTC	EXIT		
COMPILE TIME = 10 MILLISECS			
PROGRAM SPACE IS 1577			
ORIGIN ENTRY POINTS AND LOCATIONS			
4	TEST	21	
53	080ERR	53	
65	EXIT	100	END 100 STOP 70
105	KODER	105	
1140	OUTPTC	1140	OPTER 1573
OVERALL CORE USE STATISTICS (DECIMAL)			
THE MAXIMUM SCM IS		54101	
THE PROGRAM CURRENTLY USES		911	
THE LOADING PROCESS USED		8800	
THE MAXIMUM LCM IS		449602	
THE PROGRAM CURRENTLY USES		4750	
LCM AREA MAP BY BUFFER TYPE			
MISC	185	SYSIOU	6 PR 2048 RANFO 64 SYSSAT 512
SYSMD1	1024	SYSSCM	911
X= 1.0			
A(5)=99.00			
TERMINATION DATE = 06/21/77			
TERMINATION TIME = 11:37:39			
TOTAL CPU TIME IN MILLISECONDS = 61			
PPU TIME IN MILLISECONDS = 191			
PAGES PRINTED = 2			
TOTAL RESOURCES USED = 8			
DISK BLOCK USAGE SUMMARY			
ODD UNITS		EVEN UNITS	
DISK 0		DISK 1	
LIMIT ON BLOCKS ALLOWED 4095		4095 0	
MAXIMUM BLOCKS USED 3		0	
PLIB BLOCKS FOR THIS PROJECT 1			

In example 7, the EQUIVALENCE declaration places A(5) in the same memory cell as B(3). Apply the formula above to attain a linear subscript for B. With dimensions at (2,5), B(1,2) is the same array element as B(3). Storage is arranged as in the following table

Origin	<u>Subscript for B</u>	
A(1)	<u>Double</u>	<u>Single</u>
A(2)		
A(3)	B(1,1) or	B(1)
A(4)	B(2,1)	B(2)
A(5)	B(1,2)	B(3)
A(6)	B(2,2)	B(4)
A(7)	B(1,3)	B(5)
A(8)	B(2,3)	B(6)
A(9)	B(1,4)	B(7)
A(10)	B(2,4) B(1,5)	B(8) B(9)
		B(10)

A and B share eight memory cells. In order to provide the correct equivalence specified, A requires two cells ahead of B for A(1) and A(2); B must be extended by two cells to include B(1,5) and B(2,5). The total block reserved for A and B together is 12 memory locations.

In the statement EQUIVALENCE (X,I) a floating point name is declared equivalent to an integer name. To avoid mixed mode when it is not applicable, choose the appropriate name. In Example 7, the octal constant is stored in I, but printed as a floating point number using X. The same address is used for both X and I.

Rules

- EQUIVALENCE is nonexecutable and must precede the first executable statement.
- DIMENSION, COMMON, EQUIVALENCE, DATA, or TYPE declarations may appear together in any order.

Rules
(continued)

- Any full or multiword variable may be made equivalent to any other full or multiword variable. The variables may be with or without subscripts.
- No diagnostic is provided if a variable with a subscript appears in an EQUIVALENCE statement if the variable has not been dimensioned. The equivalences in the storage block will be correct and include enough space for all variables that are implied in equivalence lists. EQUIVALENCE (A(3),B) where A is not dimensioned leaves two cells for A(1),A(2) ahead of B.
- The EQUIVALENCE declaration does not rearrange COMMON, but arrays may be defined as equivalent so that the length of the common block is changed. The origin of the common block must not be changed by the EQUIVALENCE declaration. The elements of COMMON arrays may be equivalenced to other COMMON variables in the same COMMON blocks or to the elements of arrays not in COMMON.
- No element of a formal parameter list may appear in an EQUIVALENCE statement within a subroutine. If it does, the compiler diagnostic is FORMAL PARAMETER ERROR IN EQUIVALENCE.

Storage Allocation

Storage is allocated differently depending on whether the storage array is in COMMON. The following simple examples illustrate changes in block lengths caused by the EQUIVALENCE declaration.

Example 8

No variables are in COMMON.

```
*FORTRAN L
CARD APPROXIMATE
NUMBER PROGRAM LOCATION
1 0 PROGRAM TEST
2 0 DIMENSION A(3),B(2),C(4)
3 0 EQUIVALENCE (A(3),C(2))
4 0 END

          ASCENT   TEST
          A      BSS    5B
          B      BSS    2B
          C      EQU    A+1B
          TEST   BSS    0B
          ENDING. BSS    0B
          RJ     EXIT

LENGTH OF ROUTINE TEST 10
COMPILE TIME = 4 MILLISECS

PROGRAM SPACE IS 34

ORIGIN ENTRY POINTS AND LOCATIONS
4 TEST 13
14 EXIT 27 END 27 STOP 17

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS 54101
THE PROGRAM CURRENTLY USES 44
THE LOADING PROCESS USED 7933
THE MAXIMUM LCM IS 449602
THE PROGRAM CURRENTLY USES 3883
LCM AREA MAP BY BUFFER TYPE
  MISC 185 SYSIOU 6 PR 2048 RANFO 64 SYSSAT 512
  SYSMD1 1024 SYSSCM 44

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11/38/45

TOTAL CPU TIME IN MILLISECONDS = 57
PPU TIME IN MILLISECONDS = 245
PAGES PRINTED = 2
TOTAL RESOURCES USED = * 8

DISK BLOCK USAGE SUMMARY
  ODD UNITS EVEN UNITS
  DISK 0           DISK 1
LIMIT ON BLOCKS ALLOWED 4095 4095
MAXIMUM BLOCKS USED 3 0
PLIB BLOCKS FOR THIS PROJECT 1
```

The origin of the storage block is A with five locations followed by B with two locations. C is the same as A+1. There are seven cells in the block. The arrangement is as follows in core:

LOC origin	A(1)
LOC+1	A(2) C(1)
LOC+2	A(3) ↔ C(2)
LOC+3	C(3)
LOC+4	C(4)
LOC+5	B(1)
LOC+6	B(2)

5.30

Type Declarations and
Storage Allocation

STORAGE ALLOCATION
(continued)

Example 9

Two arrays in COMMON are equivalenced.

```
*FORTRAN,L
CARD APPROXIMATE PROGRAM LOCATION
NUMBER
1 0 PROGRAM TEST
2 0 COMMON A(3),B(2),C(4)
3 0 EQUIVALENCE (A(3),C(2))
4 0 END

ASCENT TEST
BLOCK 5B
COMMON A$3B
COMMON B$2B
C EQU A+1B
TEST BSS OB
ENDING. BSS OB
RJ EXIT

LENGTH OF ROUTINE TEST 1
COMPILE TIME = 3 MILLISECS

PROGRAM SPACE IS 32

ORIGIN ENTRY POINTS AND LOCATIONS
11 TEST 11
12 EXIT 25 END 25 STOP 15

COMMON BLOCKS LOCATION
4
OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS 54101
THE PROGRAM CURRENTLY USES 42
THE LOADING PROCESS USED 7926
THE MAXIMUM LCM IS 449602
THE PROGRAM CURRENTLY USES 3881
LCM AREA MAP BY BUFFER TYPE
MISC 185 SYSICU 6 PR 2048 RANFO 64 SYSSAT 512
SYSDM1 1024 SYSSCM 42

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11:38:11

TOTAL CPU TIME IN MILLISECONDS = 60
PPU TIME IN MILLISECONDS = 212
PAGES PRINTED = 2
TOTAL RESOURCES USED = 8

DISK BLOCK USAGE SUMMARY
ODD UNITS EVEN UNITS
DISK 0 DISK 1
LIMIT ON BLOCKS ALLOWED 4095 4095
MAXIMUM BLOCKS USED 0 3
PLIB BLOCKS FOR THIS PROJECT 1
```

The origin of the common block is A with three locations.
B follows after A with two locations. C is defined as A+1.
The total block length is five locations. The arrangement in
core is:

LOC origin	A(1)
LOC+1	A(2) ↔ C(1)
LOC+2	A(3) ↔ C(2)
LOC+3	B(1) ↔ C(3)
LOC+4	B(2) ↔ C(4)

Example 10

A is in COMMON, B is not in COMMON. Notice that A must appear as the referenced identifier. Sa and Sb are the subscript of A and B:

Sb \leq Sa is a permissible subscript arrangement
 Sb $>$ Sa is not permissible

```
*FORTRANL
CARD APPROXIMATE PROGRAM LOCATION
NUMBER
1      0      PROGRAM TEST
2      0      COMMON /2/A(4)
3      0      DIMENSION B(5)
4      0      EQUIVALENCE (A(3),B(2))
5      0      END

                                ASCENT    TEST
                                •2      BLOCK    6B
                                COMMON   A+4B
                                B       EQU     A+1B
                                TEST    BSS     0B
                                ENDING  BSS     0B
                                RJ      EXIT

LENGTH OF ROUTINE TEST      1
COMPILE TIME = 4 MILLISECS

PROGRAM SPACE IS 33

ORIGIN ENTRY POINTS AND LOCATIONS
4      TEST      4
13     EXIT      26      END      26      STOP      16

COMMON BLOCKS LOCATION
•2      5

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS      54101
THE PROGRAM CURRENTLY USES    43
THE LOADING PROCESS USED    7932
THE MAXIMUM LCM IS        449602
THE PROGRAM CURRENTLY USES    3882
LCM AREA MAP BY BUFFER TYPE
MISC      185  SYSIOU      6  PR      2048  RANFO      64  SYSSAT      512
SYSDM1    1024  SYSSCM      43

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11:39:58

TOTAL CPU TIME IN MILLISECONDS =      36
PPU TIME IN MILLISECONDS =      219
PAGES PRINTED =      2
TOTAL RESOURCES USED =      • 7

DISK BLOCK USAGE SUMMARY
ODD UNITS          EVEN UNITS
DISK 0             DISK 1
LIMIT ON BLOCKS ALLOWED 4095      4095
MAXIMUM BLOCKS USED      3          0
PLIB BLOCKS FOR THIS PROJECT 1
```

Block 2 in COMMON is six cells long. A is the origin; B is now in COMMON; B(1) is equivalent to A(2). Locations are arranged as follows:

LOC origin	A(1)	
LOC+1	A(2)	B(1)
LOC+2	A(3) \leftrightarrow	B(2)
LOC+3	A(4)	B(3)
LOC+4		B(4)
LOC+5		B(5)

STORAGE ALLOCATION
(continued)Example 11

B is in COMMON, A is not in COMMON. Notice that B must appear as the referenced identifier. Sa and Sb are the subscripts of A and B:

Sa ≤ Sb is a permissible subscript
Sa > Sb is not permissible

```

CARD      APPROXIMATE #FORTRANSL
NUMBER   PROGRAM LOCATION
1        0      PROGRAM TEST
2        0      COMMON /3/ B(4)
3        0      DIMENSION A(5)
4        0      EQUIVALENCE (B(2),A(1))
5        0      END

                                ASCENT    TEST
                                •3       BLOCK    6B
                                0       COMMON   B•4B
                                1       EQU      B+1B
                                0       TEST     BSS     DB
                                0       ENDING. BSS     DB
                                0       RJ      EXIT
0100400000          LENGTH OF ROUTINE TEST      1
COMPILE TIME = 5      MILLISECS

PROGRAM SPACE IS 33

ORIGIN  ENTRY POINTS AND LOCATIONS
4      TEST      4
13     EXIT      26      END      26      STOP      16

COMMON BLOCKS LOCATION
•3      5

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS 54101
THE PROGRAM CURRENTLY USES 43
THE LOADING PROCESS USED 7932
THE MAXIMUM LCM IS 449602
THE PROGRAM CURRENTLY USES 3882
LCM AREA MAP BY BUFFER TYPE
MISC      185  SYSIOU      6  PR      2048  RANFO      64  SYSSAT      512
SYSDM1    1024  SYSSCM      43

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11/39/19

TOTAL CPU TIME IN MILLISECONDS = 94
PPU TIME IN MILLISECONDS = 227
PAGES PRINTED = 2
TOTAL RESOURCES USED = .10

DISK BLOCK USAGE SUMMARY
ODD UNITS      EVEN UNITS
DISK 0          DISK 1
LIMIT ON BLOCKS ALLOWED 4095      4095
MAXIMUM BLOCKS USED 0          3
PLIB BLOCKS FOR THIS PROJECT 1

```

The array in COMMON should appear first in the EQUIVALENCE statement. Common block 3 is six cells long with B as the origin; A(1) is the same cell as B(2). The common block was extended two cells because of the dimension of A.

LOC origin	B(1)
LOC+1	B(2) ↔ A(1)
LOC+2	B(3) A(2)
LOC+3	B(4) A(3)
LOC+4	A(4)
LOC+5	A(5)

Example 12

A,B are both in Common. If A appears before B in the COMMON statement and Sa and Sb are the subscripts of A and B,

Sa \geq Sb is permissible
Sa < Sb is not permissible

```

CARD          APPROXIMATE      *FORTRAN*
NUMBER       PROGRAM LOCATION
1            0      PROGRAM TEST
2            0      COMMON /1/ A(5),B(7)
3            0      EQUIVALENCE (A(4),B(3))
4            0      END

                                ASCENT    TEST
                                •1      BLOCK   108
                                0      COMMON   A,58
                                1      EQU     A+1B
                                0      TEST    BSS    OB
                                0      ENDING. BSS    OB
                                0      RJ      EXIT

LENGTH OF ROUTINE TEST      1
COMPILE TIME = 3 MILLISECS

PROGRAM SPACE IS 35

ORIGIN      ENTRY POINTS AND LOCATIONS
6      TEST      4
15     EXIT      30      END      30      STOP      20

COMMON BLOCKS      LOCATION
•1      5

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS      54101
THE PROGRAM CURRENTLY USES      45
THE LOADING PROCESS USED      7934
THE MAXIMUM LCM IS      449602
THE PROGRAM CURRENTLY USES      3884
LCM AREA MAP BY BUFFER TYPE
  MISC      185  SYSIOU      6  PR      2048  RANFO      64  SYSSAT      512
  SYSDM1    1024  SYSSCM      45

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11/40/31

TOTAL CPU TIME IN MILLISECONOS =      23
PPU TIME IN MILLISECONOS =      299
  PAGES PRINTED =      2
TOTAL RESOURCES USED =      • 8

DISK BLOCK USAGE SUMMARY
  ODD UNITS      EVEN UNITS
  DISK 0          DISK 1
LIMIT ON BLOCKS ALLOWED      4095      4095
MAXIMUM BLOCKS USED          3          0
PLIB BLOCKS FOR THIS PROJECT      1

```

STORAGE ALLOCATION*(continued)*

The block is 8 (10_8) words long; the origin is at A; B(1) is the same as A(2), extending the block by three words.

LOC origin	A(1)	
LOC+1	A(2)	B(1)
LOC+2	A(3)	B(2)
LOC+3	A(4)	\leftrightarrow B(3)
LOC+4	A(5)	B(4)
LOC+5		B(5)
LOC+6		B(6)
LOC+7		B(7)

If B appears before A in the Common statement,

Sa \leq Sb is permissible

Sa > Sb is not permissible

EQUIVALENCE (A(3),B(4)) would change the origin of the common block; thus, it is not allowed.

DATA DECLARATION

The DATA statement provides initial values for variables, arrays and array elements. The DATA statement is non-executable and may appear in a routine anywhere after the specification statements (DIMENSION, COMMON, EQUIVALENCE, TYPE, IMPLICIT, PARAMETER and EXTERNAL).

DATA STATEMENT

The form of the DATA statement is:

```
DATA klisti/clisti/[[[,]klistj/clistj/]...]
```

where klist_i is a list of names of variables, arrays and array elements. clist_i is a list of constants of the form c or r*c, where c is a constant or the symbolic name of a constant and r, the repetition count, is a non-zero positive integer constant or the symbolic name of such a constant. The form r*c is equivalent to r successive appearances of the constant c.

The commas between successive klist/clist/ structures are optional.

When the program begins execution, each element of klist_i will have the value of the constant in the corresponding position of clist_i.

Rules

- Dummy arguments (formal parameters), elements of blank COMMON and function names cannot be assigned initial values in a DATA statement. An entity may have an initial value assigned only once in a program.
- There must be a one-to-one correspondence between the elements of $klist_i$ and $clist_i$. If an array name without subscripts appears in $klist_i$, there must be one constant in $clist_i$ for each element of the array. The ordering of array elements is determined by the array element subscript value (see chapter 2).
- The type of each entry in $klist_i$ must be the same as the corresponding entry in $clist_i$ if either is type LOGICAL. Otherwise the $clist_i$ constant will be converted to the type of the corresponding element in $klist_i$ according to the rules for mixed mode replacement statements (see chapter 3).
- Each subscript in $klist$ must be an integer constant expression except for implied DO variables.

- The implied DO loop in a DATA statement has the general form:

(dlist, j=m₁, m₂[, m₃])

where dlist is a list of array element names and implied DO lists; j is an integer variable, the implied DO variable; and m₁, m₂ and m₃ are each an integer constant expression except that the expression may contain the implied DO variables of other implied DO loops that have this loop in their ranges.

The iteration count and values of the implied DO variables are established by m₁, m₂ and m₃ exactly as they are for the explicit DO loop (see chapter 6) except that the iteration count must be positive. If the third parameter, m₃, is omitted, a default value of 1 is used.

*Rules
(continued)*Example

```

PARAMETER (KK=4,L=3,M=5)
DOUBLE PRECISION D
DIMENSION A(6),B(5,5),C(10),ID(M,M,M)
DATA (A(I),I=1,6)/1.,2.,5./,(C(I),I=1,10)/5*1.,5*2./
DATA ((B(J,I),I=1,J),J=1,5)/15*1/
DATA (((ID(I,J,K),I=1,M),J=1,M),K=1,M)/125*2./
DATA D/4.5/
DATA ICH/'ABCD'

```

The above DATA statements cause initial values to be assigned so that when program execution begins, the variables and array elements will be defined as follows:

- Array elements A(1) and A(2) will contain the floating point values 1. and 2., respectively. Elements A(3) through A(6) will contain the floating point value 5..
- Array elements C(1) through C(5) will contain the floating point value 1.. Elements C(6) through C(10) will contain the floating point value 2..
- The diagonal and lower triangular elements of the array B will contain the floating point value 1..
- Each element of the three-dimensional array ID will contain an integer value of 2..
- The variable D will contain the double precision value 4.5D0.
- The variable ICH will contain the character string 'ABCD'.

VI

CONTROL STATEMENTS

```
GO TO s
GO TO lab[(s1[,s2]...)]
ASSIGN s TO lab
GO TO (s1[,s2]...)[,]i
IF (exp) s1,s2,s3
IF (exp) s1,s2
IF (exp) st
IF (exp) THEN
ELSE IF (exp) THEN
ELSE
END IF
DO s[,]i = exp1,exp2[,exp3]
CONTINUE
PAUSE [string]
END
CALL EXIT
STOP [string]
```


CONTROL STATEMENTS

INTRODUCTION

Program execution normally proceeds from each statement to the statement immediately following it in the program. The normal execution sequence begins with the first executable statement of the main program. Control statements may be used to alter this sequence or to cause a number of iterations of the same program section.

Control may be transferred to an executable statement only; a transfer to a nonexecutable statement results in a program error.

Users of the 7600 may currently choose one of two DO-loop interpretations on the NCAR compiler. The default version of the compiler accepts a syntax which is closer to the FORTRAN 66 standard. However, the ",25" modifier on the *FORTRAN control card invokes a version of the compiler which accepts a DO-loop syntax which conforms to the FORTRAN 77 standard. A separate section of this chapter is devoted to a description of each syntax.

GO TO STATEMENTS**UNCONDITIONAL
GO TO STATEMENT**

GO TO s

This statement causes an unconditional transfer to the statement labeled s.

- If there is no statement labeled s, the diagnostic THESE STATEMENT LABELS ARE MISSING will be printed followed by a list of the missing statement labels.
- If s is a variable instead of an integer constant, no diagnostic is provided. The compiler assumes it is an assigned GO TO.

**ASSIGNED GO TO
STATEMENT**GO TO lab[, (s₁[,s₂]...)]

This statement acts as a many-branch GO TO; lab is a simple integer variable assigned a label value s_i in a preceding ASSIGN statement. The s_i are statement labels. As shown, the parenthetical statement label list need not be present; however, if it is not, DO loop optimization will not take place where the GO TO is in a loop.

When the list is omitted, the comma after lab must be omitted. A simple integer variable, lab cannot be defined as the result of a computation. (For such a capability, see the computed GO TO statement). No compiler diagnostic is given if lab is computed or not defined, but the object code generated by the compiler is incorrect. If the address lab is not in the program, a termination with no comment may occur.

ASSIGN STATEMENT

ASSIGN s to lab

This statement is used to assign statement labels that will appear with the assigned GO TO statement; s is a statement label and lab is a simple integer variable. If s is a variable name instead of a statement number, the diagnostic is UNRECOGNIZED STATEMENT.

ASSIGN 10 TO LSWTCH

:

GO TO LSWTCH,(5,10,14,20)

Control is transferred to statement 10.

COMPUTED GO TO STATEMENT

GO TO (s₁[,s₂]...)[,]i

This statement acts as a many-branch GO TO where i is preset or computed prior to its use in the GO TO. The branch will go to s_i, depending on the current value of i. That is, the branch will go to the ith statement in the list.

The s_i are statement labels and i is an integer variable or expression. If i < 1 or if i > m, a transfer is not executed, and the next statement following the GO TO is executed.

Limitations

- For proper operations, i must *not* be specified by an ASSIGN statement. No compilation diagnostic is given for this error, but the object code generated by the compiler is incorrect and the program may stop without comment.

COMPUTED GO TO**STATEMENT**

(continued)

Limitations

(continued)

- If i is a floating point expression, a nonfatal diagnostic will be printed, NON-INTEGER EXPRESSION NOT ALLOWED IN FORTRAN STANDARD EXECUTION MAY CONTINUE. The value of the floating expression is truncated to an integer and execution continues.

Example

```
ISWTCH = 1
GO TO (10,20,30),ISWTCH
:
10 JSWTCH = ISWTCH + 1
      GO TO (11,21,31),JSWTCH
```

Control is transferred first to statement 10 and from there to statement 21.

IF STATEMENTS

IF statements may be three-, two-, or one-branch statements.

**THREE-BRANCH
ARITHMETIC
IF STATEMENT**

IF(exp) s₁,s₂,s₃

exp is an arithmetic expression, and the s_i are statement labels. This statement tests the evaluated expression exp and jumps as follows:

If exp = 0, jump to statement s₂

If exp > 0, jump to statement s₃

Otherwise, exp < 0, go to s₁

Example 1

C JUMP - ,0, + depending on value of A
 IF(A) 1,2,3

Whether A is 0 or -0, the jump to 2 is taken; if A is greater than 0, the jump is to 3, if A is less than 0, the jump is to 1.

Example 2

COMPLEX C
 IF(C-SIN(X)) 1,4,5

The mode of the evaluated expression is complex. Only the real part is tested.

Example 3

LOGICAL L
 IF(L) 1,2,3

Both true (-0) and false (+0) jump to 2. Branches to 1 and 3 will be taken only on a nonzero value of L. This is *not* a standard way of using a logical variable, but no diagnostic is provided. L is treated here as type integer.

**THREE-BRANCH
ARITHMETIC
IF STATEMENT
(continued)**

Example 4

IF(A/B**2) 3,6,6

The expression is evaluated and if it is zero or plus, a jump to 6 is taken.

Example 5

```
F = .1
E = 0.0
A = 2.
DO 40 J=1,20
40 E = E+F
41 IF(E-A) 1,2,3
```

At statement 41, A = 2.0 (17214000000000000000B). After 20 iterations of statement 40, E = 1.999 (172077777777777767B); since .1 cannot be expressed exactly as a 48 bit binary number, E does not become exactly 2.0. If the floating-point expression E-A, no zero will ever be reached; therefore, the branch to 2 will never be taken. It is recommended that the zero branch for an IF test with floating point variables not be relied on.

Example 6

```
F = 0.1
A = 2.0
E = 0.0
DO 40 J=1,20
40 E = E+F
IF((A-E)+1.E-10) 1,3,3
```

In this example, no exact zero is expected and a very small increment is added to the expression to insure that truncation or round-off does not send the branch to 1 for values of E very close to A.

**TWO-BRANCH
RELATIONAL
IF STATEMENT**

IF (exp) s_1, s_2

exp is an expression. The s_i are statement labels. The evaluated expression is tested for true (nonzero, including -0) or false (+0) condition. If exp is true, the jump is to statement s_1 . If exp is false, the jump is to statement s_2 .

Example 1

C TWO BRANCH JUMP T,F
IF(A.GT.B.AND.A.LT.C) 5,6

(A.GT.B) is evaluated. If it is false, the branch to 6 is immediately taken. If (A.GT.B) is true, then (A.LT.C) is evaluated, and a branch to 5 is taken if it is true; otherwise a branch to 6 is taken.

Example 2

IF(A-B) 1,2

Since (A-B) is arithmetic, no -0 could result from this expression since a subtract is generated. However, the test is (false) for any nonzero value (as true) with a branch to 1; a zero will branch to 2.

Example 3

A = 0.0
B = -A
20 IF(B) 19,29

In this case B is equal to -0. A negative test is done first where -0 goes to the 19 (true) branch. (-0 in this test is true.) A zero test then branches to 29 (false), all non-zero numbers branching to 19. (The compiler might also generate a zero test first, where both +0 and -0 would be true.)

**ONE-BRANCH
RELATIONAL
IF STATEMENT**

IF(exp) st

exp is an expression and st is a statement. If exp is true (nonzero including -0), execute statement st. If exp is false (+0), do not execute statement st.

In an IF(exp) st statement, st may *not* be an END statement, nor may it be another conditional or a DO. If a simple branch is required, use the full GO TO s statement.

Examples

```
C      IF EXP TRUE, A=2.0
          IF(A.LE.2.5) A=2.0
          IF(VALUE*4.73.GT.ATEM.OR.VALUE.LT.150.0)STEP=TRUE
          IF(P.AND.Q)GO TO 427
          IF(PARAM)PARAM=A(I)+SINF(B(3))
```

If the form of an IF is incorrect, the diagnostic IF STATEMENT FORMAT ERROR is provided. (See Chapter 3 for information about evaluation of expressions.)

DO STATEMENTDO s[,]i = exp₁,exp₂[,exp₃]

This statement makes it possible to repeat groups of statements and to change the value of an integer variable during the repetition. s is the statement label ending the DO loop; i is the index variable (simple integer). exp_i are the indexing parameters; they may be unsigned integer constants or simple integer variables. The initial value assigned to i is exp₁; exp₂ is the largest possible value assigned to i (it must be less than 100,000); and exp₃ is the increment added to i after each time through the loop. If exp₃ does not appear, it is assigned the value 1.

A statement label which terminates a DO loop and has not been previously referenced except in a DO statement is ignored. A later reference to such a statement will cause a missing-statement-label indication. The following example produces a diagnostic THESE STATEMENT LABELS ARE MISSING, where 5 has not been used within the loop for a branch.

```
DO 5 I=1,N
5 CONTINUE
GO TO 5
```

The DO statement, the statement labeled s, and all intermediate statements are contained in the DO loop. Statement s may not be an IF or GO TO statement, FORMAT declaration, or another DO loop.

Rules

- The indexing parameters exp₁, exp₂, and exp₃ are either constants or simple integer variables. Subscripted variables cause a diagnostic.

DO STATEMENT
(continued)

Rules
(continued)

- The indexing parameters exp_1 and exp_2 must be nonzero positive integer constants or simple integer variables. If exp_1 or exp_2 is a variable, it may have a sign.
- The incrementing parameter exp_3 must be a simple positive integer constant or variable. If exp_3 is omitted, it is assigned the value 1.
- The values of exp_1 , exp_2 , and exp_3 may be changed during the execution of the DO loop.
- A DO requires a terminal statement. The diagnostic A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO will be generated when the terminal statement is missing.

Incorrect Code[†]

```
DO 50 K=1,2
50 DO 51 J=1,2
51 CONTINUE
```

```
DO 2 I=1,10
DO 2 I=1,10
:
:
```

```
2 CONTINUE
```

```
DO 5 K=0,M
DO 6 J=-4,10
DO 15 L1 = A,B
DO 15 L3 = 1,15.4
DO 1 J=1,N(J),2
DO 15 L=1,3,-1
DIMENSION J(2)
DO 1 K=1,J
```

```
DO 2 I = 5,1
```

```
DO 8 I=1,2
8 IF(A.EQ.B)C(I)=4.0
```

```
DO 101 I=1,2
WRITE (6,101)
101 FORMAT (*.....*)
```

```
DO 5 N(I)=1,5
```

```
DO 6 I=1,3
DO 8 K=1,4
6 A = B(K)
8 CONTINUE
```

```
DO 15 L8=1,100000
:
:
```

```
15 CONTINUE
```

Diagnostic

A PREVIOUS DO TERMINATES
ON THIS DO STATEMENT

A DO VARIABLE IS USED
IN AN OUTER DO LOOP

THE PARAMETERS OF A DO LOOP
MUST BE AN UNSIGNED INTEGER
CONSTANT OR SIMPLE INTEGER
VARIABLE

A DO LOOP OUT OF RANGE

A DO LOOP TERMINATES AT
THIS STATEMENT

THIS STATEMENT
DOES NOT FOLLOW A DO
WHICH IT TERMINATES

ILLEGAL USE OF A REPLACEMENT
STATEMENT

A DO LOOP WHICH TERMINATES
AT THIS STATEMENT INCLUDES
AN UNTERMINATED DO

A DO LOOP LIMIT EXCEEDS
MACHINE CAPACITY
(Note: The limit is 99999)

[†] Incorrect code errors are given in italics. Note that many of these statements are acceptable according to the FORTRAN 77 standard version of the DO-loop described later in this chapter.

6.12

Control Statements

DO STATEMENT

(continued)

The following programming sample shows standard DO loop notation.

```

*FORTRAN,FL
CARD APPROXIMATE
NUMBER PROGRAM LOCATION
1      0      PROGRAM SAMP
2      9      DIMENSION A(6,5,3)
3     132      N=3 $M=5 $L=6
4     132      WRITE (6,102)
5     149      102 FORMAT (1H1)
6     140      DO 2 K=1,N
7     140      DO 3 J=1,M,1
8     141      WRITE (6,101) J,K
9     151      101 FORMAT (* A(I,*I1*,*I1*)*)
10    151      DO 4 I=1,6,N
11    151      4 A(I,J,K)=I+J+K
12    165      3 WRITE (6,100) (A(I,J,K),I=1,L,N)
13    205      100 FORMAT (1H+15X8F10.0)
14    205      2 CONTINUE
15    210      END

LENGTH OF ROUTINE   SAMP      230
VARIABLE ASSIGNMENTS
A      -      0      N      -      217      M      -      216      L      -      215      K      -      214      J      -      213
I      -      212

SUBROUTINES CALLED
OUTPTC  EXIT
COMPILE TIME = 23 MILLISECS
```

PROGRAM SPACE IS 1727

ORIGIN	ENTRY POINTS AND LOCATIONS
4	SAMP 135
234	Q80ERR 234
246	EXIT 261
266	K0ER 266
1316	OUTPTC 1315

OVERALL CORE USE STATISTICS (DECIMAL)

THE MAXIMUM SCM IS	53861
THE PROGRAM CURRENTLY USES	999
THE LOADING PROCESS USED	7059
THE MAXIMUM LCM IS	468706
THE PROGRAM CURRENTLY USES	2240

LCM AREA MAP BY BUFFER TYPE

MISC	147	SYSIOU	6 PR	512 RANFO	64 SYSSAT	512
SYSSCM	909					

A(I,1,1)	3	6
A(I,2,1)	4	7
A(I,3,1)	5	8
A(I,4,1)	6	9
A(I,5,1)	7	10
A(I,1,2)	4	7
A(I,2,2)	5	8
A(I,3,2)	6	9
A(I,4,2)	7	10
A(I,5,2)	8	11
A(I,1,3)	5	8
A(I,2,3)	6	9
A(I,3,3)	7	10
A(I,4,3)	8	11
A(I,5,3)	9	12

TOTAL CPU TIME IN MILLSECONDS =	149
PPU TIME IN MILLSECONDS =	515
PAGES PRINTED =	3
CARDS PUNCHED =	1
TOTAL RESOURCES USED =	.17

**DO LOOP
EXECUTION**

DO s[,]i = exp₁,exp₂[,exp₃]

The initial value of the index of a DO, exp₁, is increased by exp₃ and compared with exp₂ after executing the DO loop once, and if i does not exceed exp₂, the loop is executed a second time. After this step, i is again increased by exp₃ and again compared with exp₂; this process continues until i exceeds exp₂. Control then passes to the statement immediately following statement s, and the DO loop is satisfied. This form of a loop is executed at least once. (See end of chapter for zero-trip DO loop.) Should exp₁ exceed exp₂ on the initial entry to the loop, the loop is executed just once and control then passes to the statement following statement s. As shown in the examples below, after the DO loop is satisfied, i is not equal to exp₂ in many cases.

Example 1

```
N = 1
M = 5
DO 5 K=M,N
C(K) = 42.*A(K)
      :
5 CONTINUE
```

After the DO is satisfied, K = 5, and the DO has been executed once. The test is done for completion of the loop at the end. The index K was not stored since it was never used as a variable, only as an index. Therefore, K is 5, not 6, after completion.

Example 2

```
M3 = 1
M2 = 1
M1 = 5
DO 50 I=M1,M2,M3
I = I
      :
50 CONTINUE
```

**DO LOOP
EXECUTION**
(continued)

Example 2 (continued)

After completion, $I = 4$. It has been decremented by $M3 = -1$ and stored because of the statement $I = I$. When the DO loop is satisfied, the index variable i is no longer well defined. Thus, this loop with $M3$ negative executes only once.

Example 3

```
DO 2 I=1,4
A = B(I)
2 CONTINUE
```

After the DO is satisfied $I = 1$. The index is left in a B-register and never stored, so after the DO, the index value is one (exp_1).

Example 4

```
DO 2 I=1,4
K = I
A = B(I)
2 CONTINUE
```

After the DO is satisfied $I = 5$. The index is forced to store by the statement $K = I$. When the loop is finished $I = 4 + 1$ since the increment (exp_3) is added at the end of the loop.

Example 5

```
DO 20 I=1,4
DO 20 J=2,4
DO 20 K=1,4
A = B(I,J,K)
20 CONTINUE
```

After the triple nest is satisfied, $I = 5$, $J = 5$, and $K = 1$. No store is forced in the inner loop; outer loop indices are always stored.

Example 6

```
DO 20 I=1,4
DO 20 J=2,4
DO 20 K=1,4
20 KN = K
```

After execution of the above triple nested DO loop, a store has been forced for K since K is used as a variable as well as an index, and I = 5, J = 5, and K = 5 even though KN = 4.

Example 7

```
N = 4
DO 5 K=1,N
IF (K.EQ.N) GO TO 6
:
:
5 CONTINUE
6 .....
```

If a transfer out of the DO loop occurs before the DO is satisfied, the value of K is preserved and may be used in subsequent statements. At the end of the loop, K is equal to N and a branch to 6 is taken. The store is forced by the expression K.EQ.N. K = 4 since the incrementing at the end of the loop is bypassed.

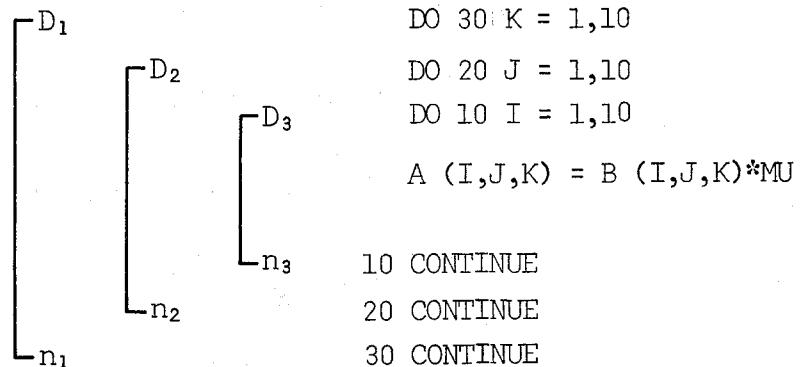
DO NESTS

When a DO loop contains another DO loop, the grouping is called a DO nest. Nesting may not exceed 19 levels. The last statement of a nested DO loop either must be the same as the last statement of the outer DO loop or must occur before it. If D_1, D_2, \dots, D_m represent DO statements where the subscripts indicate that D_1 appears before D_2 , D_2 appears before D_3 , and n_1, n_2, \dots, n_m represent the corresponding terminuses of the D_i , then n_m must appear at or before n_{m-1} .

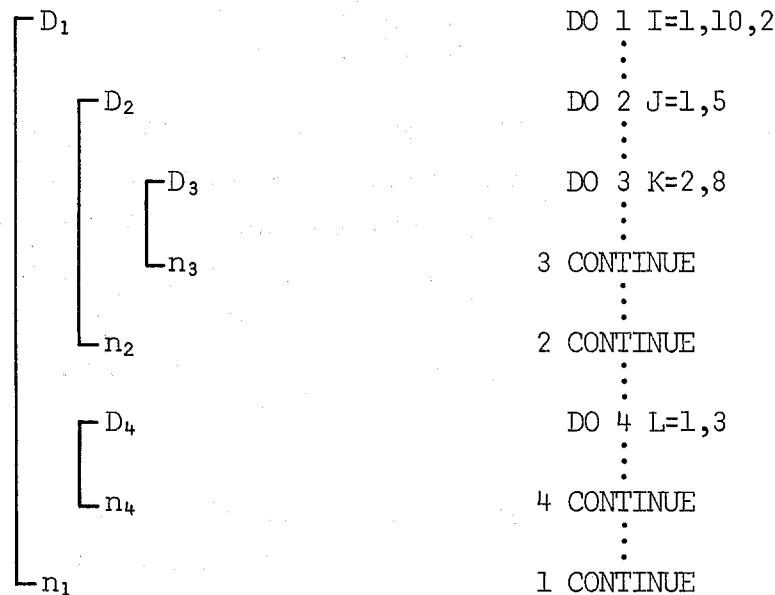
DO NESTS

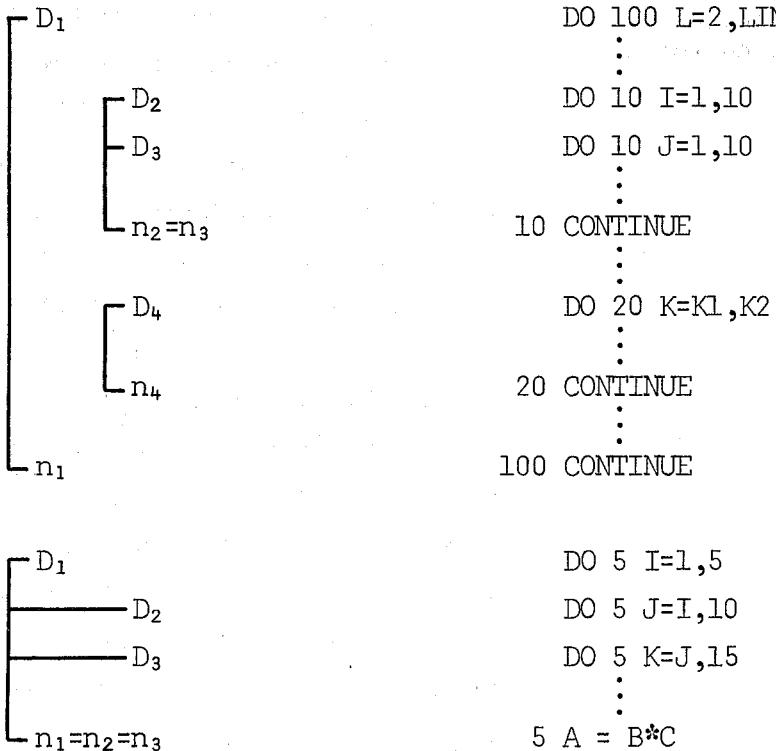
(continued)

The index variable of the inner nest should be the first subscript of three-dimensional arrays, where optimization is a consideration.



DO loops may be nested in common with other DO loops.



*Limitations*

- Nineteen DO loops may be nested. The twentieth DO loop will cause the compiler diagnostic THE NESTING CAPACITY OF THE COMPILER HAS BEEN EXCEEDED.
- If a FORTRAN statement is the terminal statement of more than one DO statement, the statement label of that FORTRAN statement may not be used in an IF or a GO TO statement outside the most deeply nested DO.[†] Even though this construct is sometimes allowed on other compilers, no diagnostic is issued.

[†] This limitation follows *USA Standard FORTRAN*, American National Standards Institute Inc., Publ. No. USAS X3.9-1966, 36 pp., 1966.

**SUBSCRIPTING
WITHIN DO LOOPS**

Where variables appear with subscripts inside a DO loop, standard subscripts should be used whenever possible. (see the section of Subscripted Variables in Chapter 2 for a definition of standard subscripts.) Standard index functions should not be separated out and given a new name (which would force an unnecessary store of the index). They should be left in the subscript even when used repeatedly. Nonstandard subscripts force additional code to be generated with a DO loop. Standard subscripts result in concise object code where index functions are calculated in B-registers. However, when there are more than seven different standard index functions in a DO loop, the 8th, 9th, ... index functions will force a store for the value of these index functions. In all cases, standard index functions are more efficient than nonstandard index functions, but the first seven standard index functions are compiled most efficiently.

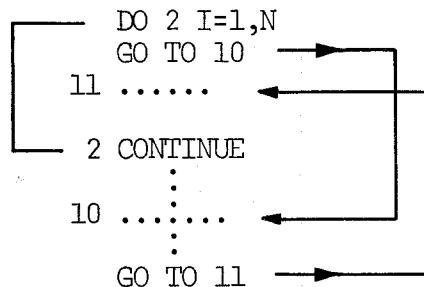
Example

In the following example $I + 1$ and $I - 1$ are not separated out and are not given new variable names for use in the index function.

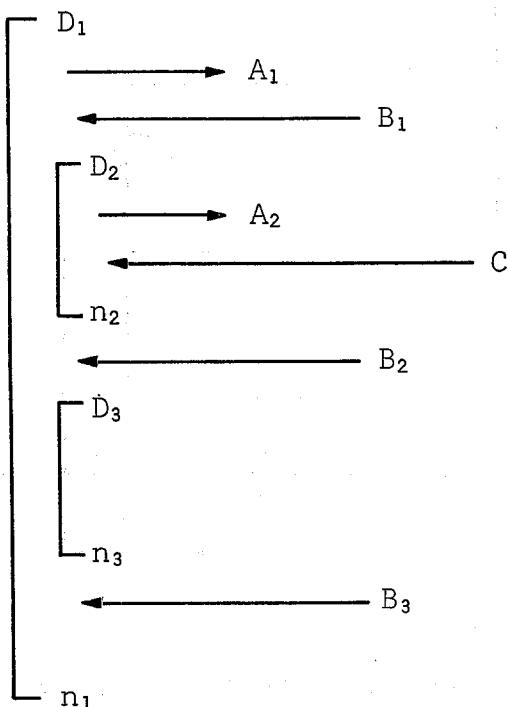
```
DO 2 I=1,N
2 A(I+1)=B(I+1)-X(I-1)*Y(I-1)
```

EXTENDED RANGE

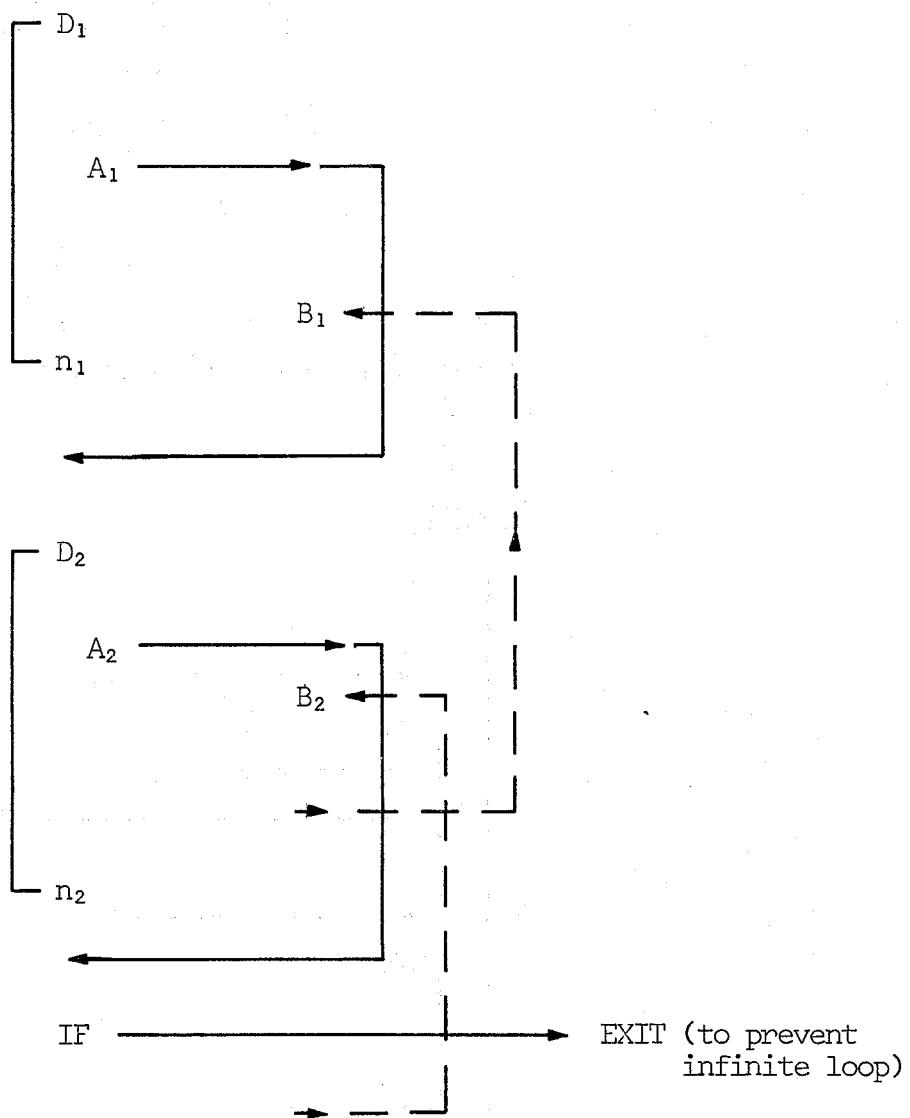
A DO loop has an "extended range" if a GO TO or an IF branches out of the range of the DO where code is executed and a reentry back into the DO is performed. (*Note:* An initial entry into the end of a loop is illegal without first entering at the start of that loop. This illegal entry is not flagged, however, because of implementation of extended range. Thus no diagnostic message is issued.)

Example 1

Example 1 shows an exit to statement 10, a reentry to statement 11. A reentry occurs anytime after the GO TO 10 statement.

Example 2

Example 2 shows three nested loops with D₂ and D₃ at the same nesting level. If an exit is taken at A₁, reentry may be effected at points B₁, B₂, and B₃. Reentry may not be taken into loop D₂ or D₃, except at the DO statement, since the control variables will not have been set. If an exit is taken at A₂, possible reentry may be made at C, B₁, B₂, or B₃. No error messages are generated for improperly positioned reentries.

EXTENDED RANGE
(continued)
Example 3

In example 3 an extended range DO loop is within the extended range of another loop. The control variables and the \exp_1 , \exp_2 , \exp_3 parameters may not be redefined during execution of the immediate or extended range of that DO. Care should be taken in defining the terminal statement of nested DO loops whenever extended range and reentry are used. Indices should be used as variables to force a store when using extended range. No error messages are generated for improperly positioned reentries.

OPTIMIZATION

The compiler will optimize the innermost DO of a nest so that the fastest possible time is achieved. The optimizing will be done provided none of the following statements appear inside the range of the innermost DO.

- A CALL to a subroutine or programmer-defined function
- Any input/output statement
- An assigned GO TO (without a list)
- Any branch out of the range of the DO
- A function call by name where an index variable is either in COMMON or in the parameter list

The optimizer code may be turned off by placing the number corresponding to a particular optimizing technique on the *FORTRAN card. The following list of codes pertains to DO loop optimization.

*FORTRAN,1

Do not remove constant statements from a DO loop.

Example 1

```
DO 2 I=2,N
A = B
.
.
2 CONTINUE
```

A *FORTRAN card which allows full optimization will remove the statement A = B from inside the DO loop.

A *FORTRAN,1 card will turn off the optimization, and A = B will remain inside the DO loop.

*FORTRAN,1
(continued)

Example 1 (continued)

When, under optimization, the compiler removes this constant from the loop, a nonfatal compiler error occurs, and a message is printed locating the statement displaced.

Example 2

```
DO 2 I=2,N
C = B
CALL T
2 CONTINUE
```

The statement C = B will not be removed because the CALL statement inside the loop turns off the optimization.

Example 3

```
COMMON B
DO 11 I=1,N
A = B
R = COS(X(I))
:
11 CONTINUE
```

A function type subprogram provided by the system, however, *does* allow optimization if no codes appear on the *FORTRAN card to turn off the optimization. The statement A = B will be removed from the DO loop. If a subscript value appears in COMMON or in the parameter list of a function reference, the code will not be optimized.

*FORTRAN,2

Do not remove constant subexpressions from a DO.

```
DO 9 I=1,N
A = (B*Z)+(B*Z)+S(I)
9 CONTINUE
```

If the code is optimized (*FORTRAN), the phrase (B^Z) will be calculated ahead of the loop and stored in a temporary cell. Inside the loop, the save cell is recovered and used in the expression. No notice is given in the diagnostic list in this case.

If the code is *not* optimized (*FORTRAN,2), (B^Z) is left inside the loop.

***FORTRAN,3**

Do not precalculate addresses in index registers.

```
DO 9 I=1,N
S(I) = R(J)
9 CONTINUE
```

If the code is optimized (*FORTRAN), the address of S_i is calculated in an index register outside the loop, as well as R_j. Incremented addresses in index registers are used inside the loop for bringing in addresses and storing addresses.

Without optimization (*FORTRAN,3), all addressing for S_i and R_j is done inside the loop.

***FORTRAN,4**

Do not optimize variant global functions. A global index function depends on a factor which is common throughout the range of a DO loop. A variant changes as a function along with the DO variable. A variant global function is a combination of these two factors.

```
KK = 2
DO 21 L=2,30
J = L+1
21 R(J) = R(J+1)+V(J,KK)
```

***FORTRAN,4**

(continued)

Using *FORTRAN, the address calculation of $V(J, KK)$ is optimized. KK is a "global" index, whereas the complete address is "variant". KK is removed from the loop, and the global factor is saved in a B-register.

Without optimization using a *FORTRAN,4 card, a memory reference to KK is done each time inside the loop.

***FORTRAN,13**

Do not set B-registers to constants.

```
DO 20 K=1,18,2
  S(K) = S(K) + ...
  .
  .
  20 CONTINUE
```

Without optimization, the 2 (m_3 in the DO loop notation DO s I = m_1 , m_2 , m_3) is saved in a B-register outside the loop. The index is incremented using this B-register.

Without optimization as on a *FORTRAN, 13, the constant 2 would be used throughout the loop.

***FORTRAN,18**

With a *FORTRAN,18 card, all of the previous DO loop optimization is turned off.

***FORTRAN,25**

This card causes the compiler to interpret DO loops according to the FORTRAN 77 standard. (See the next section of this chapter.)

***FORTRAN,60**

Do not optimize any of the program whatsoever. This option removes FORTRAN and object code optimization.

Wherever possible, the following techniques are *always* applied.

- Use A_0 as a constant 1 in the logic. (Any external function other than the ** operator will turn this off.)

- Place no statement label on a CONTINUE statement unless it is specifically referred to in a branch statement.
- Fetches are placed at the end of the D0 after incrementing. At the last execution through the D0 loop, an extra fetch is done. This extra fetch may cause a bounds error beyond the last array element. Since this technique speeds loop execution time, it is worth taking the chance of getting a bounds error.

Note: The use of variable dimensions in a subprogram causes more indexing logic to be generated for D0 loops. If speed, rather than generality, is important, do not use variable dimensions.

**CONTINUE
STATEMENT**

CONTINUE

The CONTINUE statement is most frequently used as the last statement of a DO loop to provide loop termination when a GO TO or IF would normally be the last statement of the loop. If CONTINUE is used elsewhere in the source program, it acts as a do-nothing instruction and control passes to the next sequential program statement. The CONTINUE statement should contain a statement label in columns 1 through 5. If it is not labeled, it is ignored by the compiler. If a label contains a character in column 6, the diagnostic ILLEGAL MARK IN COLUMN SIX is printed. If the label contains a letter, ILLEGAL CHARACTER APPEARS IN A CONSTANT is the diagnostic.

**PAUSE
STATEMENT**

PAUSE [string] string ≤ 5 characters

PAUSE stops program execution with the message string displayed on the console. An operator entry from the console can continue (xxx GO) or terminate (xxx DROP) the program. (xxx is the job control point on the CRT.) Program continuation goes to the statement immediately following PAUSE. If string is omitted, it is understood to be blank. If string appears as more than five characters, only the first five characters appear with the PAUSE at the console.

END STATEMENT

END

END must be the very last statement in a program or subprogram, and there should be only one END card for each unit. It is executable in the sense that it affects termination of the program in the absence of a CALL EXIT or a return from a subprogram in the absence of a RETURN.

**NORMAL
TERMINATION
STATEMENTS**

CALL EXIT

EXIT is a subprogram on the system library. (See the *Library Routines Manual* for a discussion of the EXIT subprogram.) This routine will terminate a program normally. An alternate form is the FORTRAN statement STOP. It may be omitted immediately preceding the END card.

STOP [string] string \leq 6 characters

STOP is an alternate entry point to the subroutine EXIT. This routine will perform a normal termination and print the value string, if it is present in the STOP statement. If string is omitted, it is understood to be blank.

IF/THEN/ELSE**GENERAL STRUCTURE**

The IF/THEN/ELSE structure is used to control the execution sequence in a program. The structure begins with an IF-THEN (block IF) statement and ends with an END IF statement. Optionally, one or more ELSE IF statements and one ELSE statement can also be used in the structure. The IF-THEN, ELSE IF and ELSE statements have blocks of statements associated with them, so the general IF/THEN/ELSE structure looks like the following:

```
IF(exp1) THEN
    [statement block]
ELSE IF(exp2) THEN
    [statement block]
    :
ELSE IF(expn) THEN
    [statement block]
ELSE
    [statement block]
END IF
```

where exp_i is a logical expression. Each of these statements and the associated blocks are described in detail below.

Nesting Levels

Each of the statement blocks within an IF/THEN/ELSE structure may itself contain another IF/THEN/ELSE structure. That is, these structures may be nested. However, the inner IF/THEN/ELSE structure must be entirely contained within one statement block of the outer structure. There may be as many as 21 nesting levels.

Transfer of Control

It is important to note that control may not be transferred into a statement block from outside that block. The NCAR compiler does *not* check for statements which result in such a transfer, but the effect of the transfer is undefined.

IF THEN STATEMENT*Form*

The IF-THEN (block IF) statement has the form

IF(exp) THEN

where exp is a logical expression.

Block

This statement has associated with it an IF block which consists of all the executable statements, if any, following the IF-THEN statement up to the next ELSE IF, ELSE or END IF statement on the same nesting level.

Execution

The execution of an IF-THEN statement causes evaluation of the logical expression exp.

- If the value of exp is true, execution continues with the first statement of the IF block. After execution of the last statement of the block, control is transferred to the corresponding END IF statement.

- If the value of exp is false, control is transferred to the next ELSE IF, ELSE or END IF statement on the same nesting level.

ELSE IF STATEMENT

Form

The ELSE IF statement has the form

ELSE IF(exp) THEN

where exp is a logical expression.

Block

This statement has associated with it an ELSE IF block which consists of all the executable statements, if any, following the ELSE IF statement up to the next ELSE IF, ELSE or END IF statement on the same nesting level.

Execution

The execution of an ELSE IF statement causes evaluation of the logical expression exp.

- If the value of exp is true, execution continues with the first statement of the ELSE IF block. After execution of the last statement of the block, control is transferred to the corresponding END IF statement.

- If the value of exp is false, control is transferred to the next ELSE IF, ELSE or END IF statement on the same nesting level.

ELSE STATEMENT*Form*

The ELSE statement has the form

ELSE

Block

This statement has associated with it an ELSE block which consists of all the executable statements, if any, following the ELSE statement up to the next END IF statement on the same nesting level.

Execution

The execution of an ELSE statement itself has no effect. Once control is transferred to an ELSE statement, normal execution sequence continues with the first statement of the ELSE block.

END IF STATEMENT

Form

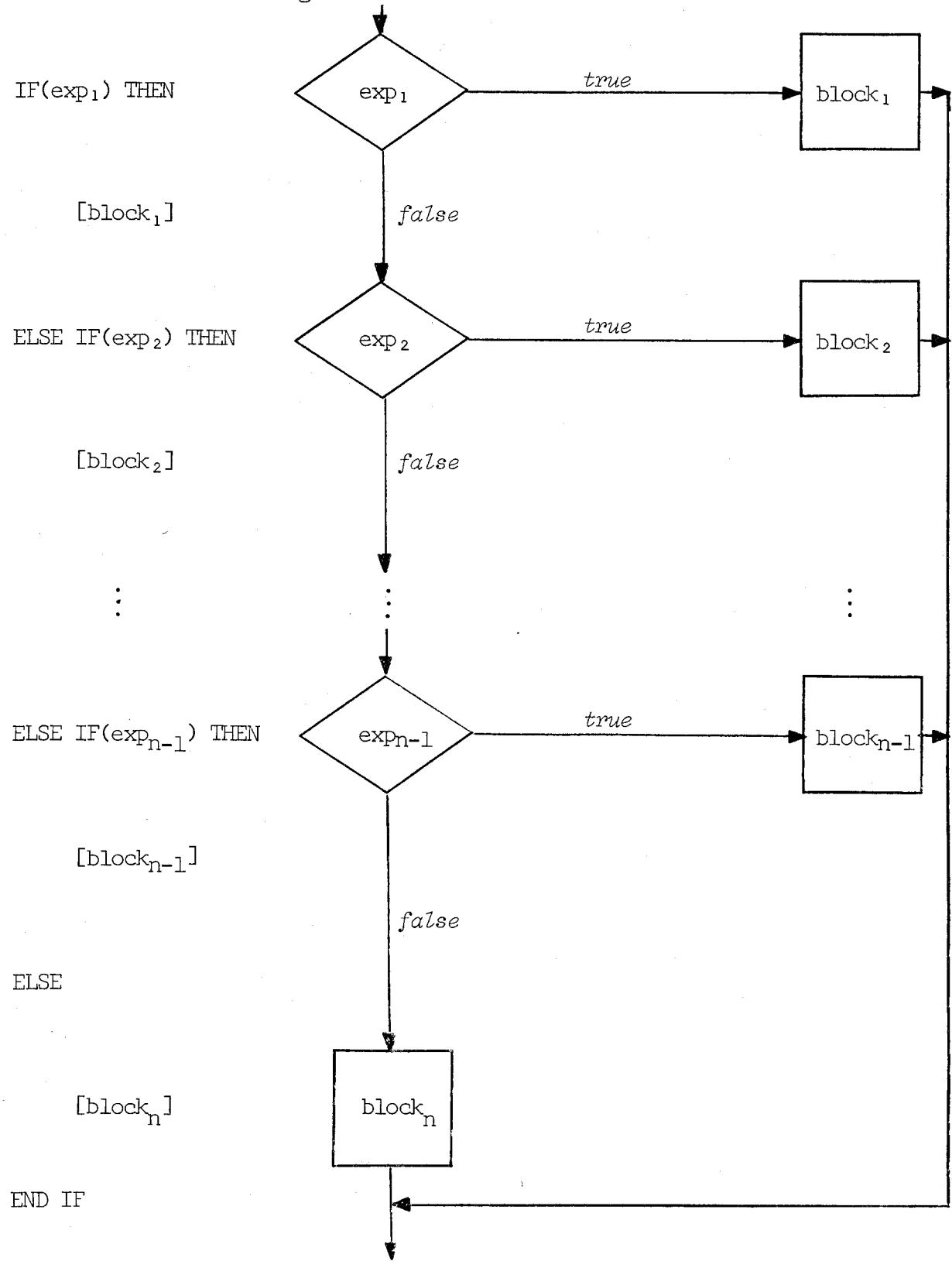
The END IF statement has the form

END IF

Execution

Execution of the END IF statement has no effect. The normal execution sequence continues with the first executable statement following the END IF statement.

Flow diagram for IF/THEN/ELSE structure:



Sample Program Illustrating IF/THEN/ELSE

CARD NUMBER	APPROXIMATE PROGRAM LOCATION	CODE
1		*FORTRAN,FL
2		
3		0 C TEST BLOCK IF STATEMENT, SECTION 11.6 IN FORTAN 77
4		0 C
5		0 IMPLICIT INTEGER (A,B,C,D)
6		0 PARAMETER (L=5)
7		0 DIMENSION A(L),B(L),C(L),D(L)
8		0 PRINT 100
9		31 100 FORMAT (#1#)
10		31 K=0
11		31 N=0
12		31 1 CONTINUE
13		74 IF (K .EQ. 0) THEN
14		75 DO 2 I=1,L
15		35 A(I)=1
16		35 2 CONTINUE
17		42 PRINT 101,K
18		52 101 FORMAT (#0IF BLOCK 1 K=# I2/)
19		52 ELSE IF (K .LE. L) THEN
20		56 PRINT 102,K
21		65 102 FORMAT (#0ELSE IF 1 K=# I2/)
22		66 IF (N .EQ. 0) THEN
23		67 PRINT 103,N
24		76 103 FORMAT (#0IF BLOCK 2 N=# I2/)
25		76 DO 4 I=1,L
26		76 C(I)=A(I)
27		76 4 CONTINUE
28		103 ELSE IF (N .LE. L) THEN
29		107 PRINT 104,N
30		116 104 FORMAT (#0ELSE IF 2 N=# I2/)
31		116 DO 5 I=1,L
32		116 D(I)=D(I)+10
33		116 5 CONTINUE
34		124 ELSE
35		125 PRINT 105,N
36		134 105 FORMAT (#0ELSE 2 N=# I2/)
37		134 END IF
38		134 N=K
39		134 DO 6 I=1,L
40		134 B(I)=B(I)+1
41		134 6 CONTINUE
42		142 ELSE
43		143 PRINT 106,K
44		152 106 FORMAT (#0ELSE 1 K=# I2/)
45		152 PRINT 108,(B(I),I=1,L)
46		166 108 FORMAT (#0B(I)=# 5I8//)
47		166 PRINT 110,(D(I),I=1,L)
48		202 110 FORMAT (#0D(I)=# 5I8//)
49		202 STOP
50		203 END IF
51		203 K=K+1
52		203 GO TO 1
53		205 END

Output from Sample Program BLKIF

IF BLOCK 1 K= 0

ELSE IF 1 K= 1

IF BLOCK 2 N= 0

ELSE IF 1 K= 2

ELSE IF 2 N= 1

ELSE IF 1 K= 3

ELSE IF 2 N= 2

ELSE IF 1 K= 4

ELSE IF 2 N= 3

ELSE IF 1 K= 5

ELSE IF 2 N= 4

ELSE 1 K= 6

B(I)= 5 5 5 5 5

D(I)= .40 .40 .40 .40 .40

STOP

DO-LOOP (FORTRAN 77)

The FORTRAN 77 version of the DO loop will be accepted when the *FORTRAN, 25 control card is used.

DO-LOOP STRUCTURE

The DO-loop structure makes it possible to repeat groups of statements while changing the value of a variable called the DO variable. As described below, the DO-loop consists of a DO statement followed by a group of statements called the DO-loop range. The last statement of the range is referred to as the terminal statement.

DO STATEMENT

The DO statement, used to specify the beginning of a DO-loop, has the form:

DO s[,] i = exp₁,exp₂[,exp₃]

Where: s is the label of the terminal statement of the DO-loop

i is the name of an integer, real or double precision variable, called the DO variable.

exp₁, exp₂ and exp₃ are each an integer, real or double precision expression.

DO-LOOP RANGE

The range of a DO-loop consists of the executable statements between the DO statement and the associated terminal statement. The terminal statement is the last statement within the range of the DO-loop.

TERMINAL STATEMENT

The terminal statement of a DO-loop must follow the associated DO statement and must be an executable statement. The terminal statement must not be a GO TO, RETURN, STOP, END, DO, ELSE IF, ELSE, END IF, or any IF statement other than a logical IF. If the terminal statement is a logical IF, it may contain any executable statement except a DO, block IF, ELSE IF, ELSE, END IF, END or another logical IF statement.

Rules

- More than one DO-loop may have the same terminal statement.
- If a DO statement appears within the range of another DO-loop, the entire range of the DO-loop must be contained within the range of the outer loop.
- If a DO statement appears within an IF block, ELSE IF block or ELSE block, the entire range of the DO-loop must be contained within that block.
- If a block IF statement appears within the range of a DO-loop, the corresponding END IF statement must also appear within the range of that DO-loop.

DO-LOOP EXECUTION

The execution of a DO-loop consists of the following sequence of operations, each of which is described in more detail below:

- Execution of the DO statement
- Loop control processing
- Execution of statements in the range of the DO-loop
- Incrementation processing

The last three steps will be repeated until either loop control processing determines that no iterations remain to be performed or until the execution of a statement in the loop range causes a jump out of the loop.

**DO STATEMENT
EXECUTION**

The execution of a DO statement consists of the following sequence of operations:

- The initial parameter m_1 , the terminal parameter m_2 , and the incrementation parameter m_3 are established by evaluating e_1 , e_2 and e_3 respectively. If necessary, the result is converted to the type of the DO variable i according to the rules of arithmetic conversion (see chapter 3). If e_3 does not appear, m_3 has the value one. m_3 must not be zero.
- The DO variable i becomes defined with the value of the initial parameter m_1 .
- The iteration count is established according to the formula

$$\text{MAX}(\text{INT}((m_2-m_1+m_3)/m_3), 0)$$

Note that the iteration count is zero whenever:

$$\begin{aligned} & m_1 > m_2 \text{ and } m_3 > 0, \\ \text{or} \quad & m_1 < m_2 \text{ and } m_3 < 0. \end{aligned}$$

At the completion of execution of the DO statement, loop control processing begins.

**LOOP CONTROL
PROCESSING**

To determine whether further execution of the range of the DO-loop is required, the iteration count is tested.

- If the iteration count is zero, the DO-loop range is *not* executed. The execution sequence continues as though the DO-loop were not present.
- If it is not zero, the first statement in the range is executed next.

**EXECUTION
OF THE RANGE**

Statements in the range of a DO-loop are executed until the terminal statement is reached. Unless execution of one of the statements in the range results in the transfer of control out of the loop, execution of the terminal statement is followed by incrementation processing described below. Note that the DO variable may neither be redefined nor become undefined during the execution of the range of the DO-loop.

**INCREMENTATION
PROCESSING**

Incrementation processing consists of the following sequence of operations:

- The DO variable is incremented by the value of the incrementation parameter m_3 .
- The iteration count is decremented by one.
- Execution continues with loop control processing described above.

**ACTIVE AND INACTIVE
DO-LOOPS**

Initially inactive, a DO-loop becomes active only when its DO statement is executed.

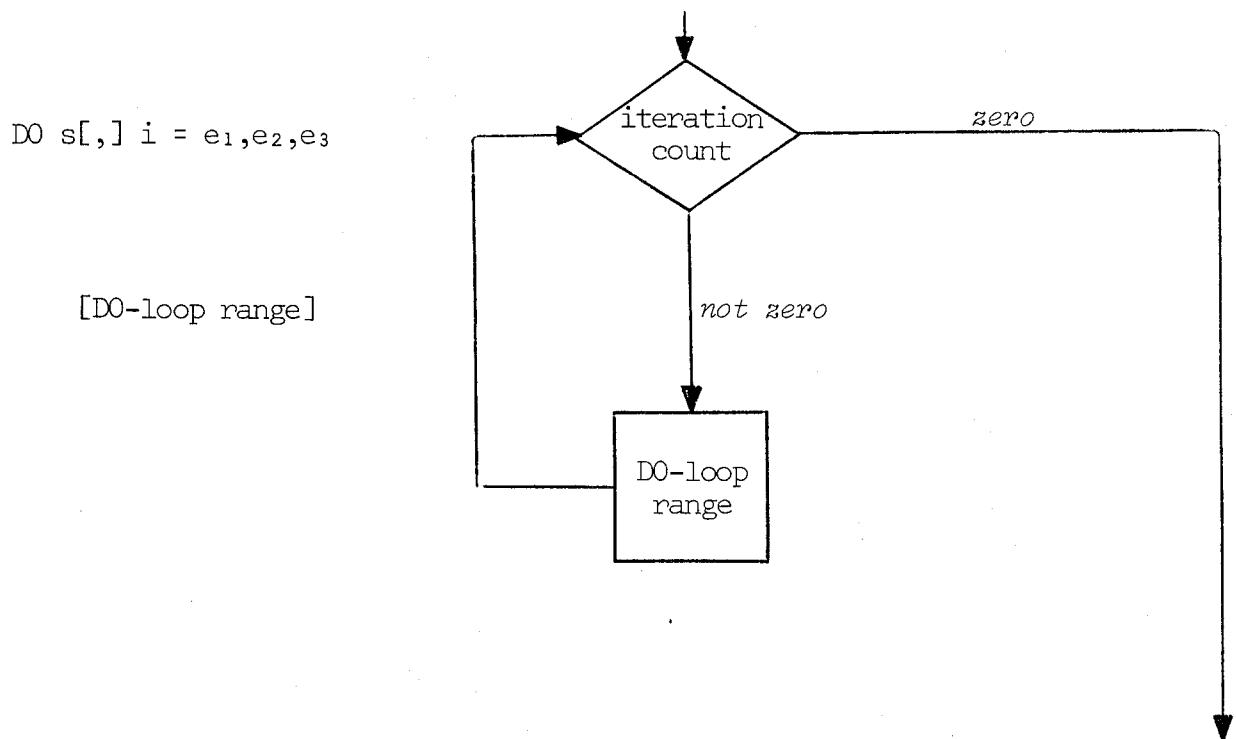
An active DO-loop becomes inactive only when:

- loop control processing determines the iteration count to be zero
- a RETURN statement is executed within its range
- control is transferred outside the range of the DO-loop
- program execution is terminated for any reason

When a DO-loop becomes inactive, the DO variable retains its last defined value unless it has become undefined.

Transfer of control into the range of a DO-loop from outside the range is not permitted. The NCAR compiler does *not* check for statements which result in such transfers, but the effect of the transfer is undefined.

Flow diagram for DO-loop (FORTRAN 77):



VII

PROGRAM, FUNCTION, AND SUBROUTINE

ENTRY POINTS

TRANSFER STATEMENTS

ARITHMETIC STATEMENT FUNCTION

```
PROGRAM pgm
SUBROUTINE sub [(d1[,d2]...)]
CALL en [(a1[,a2]...)]
[typ] FUNCTION fun [d1[,d2]...]
fun (a1[,a2]...)
fun (d1[,d2]...) = exp
BLOCK DATA [sub]
RETURN
END
ENTRY en
EXTERNAL en1[,en2]...
```


PROGRAM, FUNCTION, AND SUBROUTINE

INTRODUCTION

A job is a complete set of control cards, programs, and data which may be submitted to the computer to be compiled and/or assembled, loaded, and run. FORTRAN programs may be main programs or subprograms (functions and subroutines). A job will contain one main program with or without subprograms.

A program is an ordered set of computer instructions which performs a mathematical algorithm or some data processing task or tasks. If the job has only one main program with few, if any, subprograms, it is a series of tasks done in order and is said to be written "in-line." In contrast, in a "modular" design, separate tasks may be programmed in subprograms. The main program then calls the subprograms as the overall program design requires. The main program then may be referred to as a "driver," and the collection of programs is said to be designed using a modular approach.

In the following pages, main program, subroutine, function, and arithmetic statement are defined, and the use of parameters is discussed.

MAIN PROGRAM

The first statement of a main program must be the PROGRAM card. The form is

PROGRAM pgm

where pgm is a symbolic identifier consisting of one to seven alphanumeric characters and starting with a letter.

The program card is optional; however, the main program must have an END card. A main program may not be called as a subprogram or called by itself recursively.

SUBPROGRAMS

A subprogram is a separate computational procedure which may be referenced for execution by a main program or another subprogram. Subprograms may be called by or may call another subprogram, but a subprogram may not call itself. Communication of parameters may be done by means of an argument list or through a COMMON declaration (see Chapter 5). When an argument list is used, a distinction is made between formal and actual parameters. Parameters are transmitted by name, not by value. If a constant is a parameter, the compiler provides a temporary name for it.

FORMAL PARAMETERS

A formal parameter is the name used in the argument list of the subprogram being called. These names are local to the subprogram and are dummy names used to reference the actual parameters specified by the calling program. Examples of formal parameters are the variables in the following argument lists:

Rules

```
SUBROUTINE SUB(X,Y,YMIN,XMAX,ITITL)
FUNCTION FUN(V,W,I)
```

- Formal parameters may be array names, simple variables, or names of library functions and function and subroutine subprograms.
- Since formal parameters are local to the subprogram containing them, they may be the same as names appearing outside the procedure with no conflict.
- No element of a formal parameter list may appear in an EQUIVALENCE or DATA statement within the subroutine. If it does, the compiler diagnostics resulting are
 FORMAL PARAMETER ERROR IN EQUIVALENCE
 ILLEGAL USE OF FORMAL PARAMETER IN DATA STATEMENT

*Rules
(continued)*

- When a formal parameter represents an array, it should be dimensioned within the subprogram. If it is not dimensioned, the array name must appear without subscripts in the subprogram, and only the first element of the array is then available to the subprogram being called. Otherwise, the subprogram will look for a missing subroutine.

ACTUAL PARAMETERS

An actual parameter is specified in the argument list of the calling program, and is often a variable which is stored in the calling program.

The following are permissible forms of an actual parameter:

Arithmetic expression
Logical expression
Constant
Simple or subscripted variable
Array name
Function or subroutine name (see EXTERNAL statement)

```
CALL SUBA(A,B,C(1,2),1.2E-3,4HTEST)
CALL DER(T,I,J,COS(X))
CALL UUB(A,B,COS(X),B,4.13E-2,(2.1,4.3),.TRUE.)
```

Note: Masking expressions are *not* legal.

Example 1

When an actual parameter is the name of a function or subroutine, that name must also appear in an EXTERNAL statement in the calling program. Note that functions from the FORTRAN library may not be used as externals. Functions supplied by the program are acceptable.

```

PROGRAM TEST
EXTERNAL ST,SU
.
.
.
CALL S2(ST,X,ANS)
.
.
.
CALL S2(SU,A,AN)
.
.
.
END

SUBROUTINE S2(F,X,R)
.
.
.
R = F(X)
.
.
.
RETURN
END

FUNCTION ST(X)
ST=X*X*X
RETURN
END

FUNCTION SU(X)
DATA U/.3333333333333 /
SU=X**U
RETURN
END

```

Example 2

Actual and formal parameters must agree in order, type, and number. No diagnostic is provided, but the code will be incorrect.

```

PROGRAM MAIN
.
.
.
CALL UB(A,42,4.13E-2,I,(2.1,4.3))
.
.
.
END

SUBROUTINE UB(A,K,T,L,C)
COMPLEX C
.
.
.
END

```

ACTUAL PARAMETERS
(continued)
Example 3

Parameters are treated as names by a subprogram. Any name in the parameter list may be used to find a value and to store a value. Therefore, changes to any parameter may be made by the subprogram, whether the actual parameter is a constant or a variable. Care should be used with constants in a parameter list because the subprogram can change the parameter value, even though the actual parameter is the same number in the CALL statement. Note in the following example that the statement $X = 2.0$, where X is a formal parameter, changes the actual parameter, which is a constant 1.0.

```

PROGRAM MAIN
  .
  .
CALL S(1.0,A)
END

SUBROUTINE S(X,Y)
  .
  .
T=X+Y
X=2.0
  .
RETURN
END

```

After the call to subroutine S, any further reference to the constant 1.0 in the main program will generate incorrect results since the storage cell allocated to hold the constant 1.0 was reset to 2.0 during execution of the subroutine. To avoid this hazard, subroutine S can be rewritten as:

```

SUBROUTINE S(XX,YY)
X=XX
Y=YY
  .
  .
T=X+Y
X=2.0
  .
RETURN
END

```

**SUBROUTINE
SUBPROGRAM**

A subroutine subprogram may or may not return values, depending on the parameter list or COMMON. No value is associated with the subroutine name.

The first statement of a subroutine subprogram must be in the following form:

SUBROUTINE sub [(d₁,[,d₂]...)]

where sub is the alphanumeric identifier and d_i are formal parameters. The parameter list is optional.

Rules

- The subroutine name may not appear anywhere within the subroutine except in the SUBROUTINE statement itself.
- Any of the formal parameters may be used to return output to the calling program if those parameters are defined or redefined in the subprogram procedure.

CALL STATEMENT

To call a subprogram, use the following statement:

```
CALL en [(a1[,a2]...)]
```

where en is the name of the entry into the subroutine being called, and a_i are actual parameters. More than 64 arguments will produce the diagnostic MORE THAN 64 ARGUMENTS IN CALL OR FUNCTION. The subprogram name may not appear in any declarative statement in the calling program, with the exception of the EXTERNAL statement.

The CALL statement transfers control to the subroutine. A RETURN statement in the subroutine returns control to the next executable statement following the CALL statement in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until the DO loop is satisfied; each trip through the loop executes a CALL to the subroutine.

Example 1

A, B, C are formal parameters.

```
SUBROUTINE TEST(A,B,C)
C = A + B*EXP(A+.3)
RETURN
END
```

Some calls have different actual parameters. For example:

```
CALL TEST(A,B,C)
CALL TEST(A(I+1)-T/U,R(I)+C(J),T)
CALL TEST(SIN(Q5),VEC(I+J),OVEC(L))
```

Example 2

COMMON is used for matrix variables.

```
SUBROUTINE MATMUL
COMMON/MAT/X(20,20),Y(20,20),Z(20,20)
DO 10 I=1,20
DO 10 J=1,20
Z(I,J) = 0
DO 10 K=1,20
10 Z(I,J) = Z(I,J) + X(I,K)*Y(K,J)
RETURN
END
```

Operations in MATMUL are performed on variables contained in the common block MAT. This block must be defined in all programs calling the subroutine MATMUL.

```
PROGRAM TEST
COMMON/MAT/AB(20,20),CD(20,20),EF(20,20)
.
.
CALL MATMUL
.
.
END
```

CALL STATEMENT
(continued)
Example 3

```

SUBROUTINE AGMT(SUB,ARG)
COMMON/ABL/XP(100)
ARG=0.
DO 5 I=1,100
5 ARG=ARG+XP(I)
CALL SUB(ARG)
RETURN
END

```

This example uses both COMMON and an argument list. The subprogram used as an actual parameter must have its name declared in an EXTERNAL statement, as in the following calling program.

```

PROGRAM MAIN
.
.
.
COMMON/ABL/ALST(100)
EXTERNAL SQDEV
CALL AGMT(SQDEV,V1)
.
.
.
END

```

The subroutine SQDEV, which is passed to subroutine AGMT as an EXTERNAL reference, also needs to be defined:

```

SUBROUTINE SQDEV(ZSUM)
COMMON/ABL/Z(100)
ZMEAN=ZSUM/100.
DO 5 I=1,100
5 Z(I)=(Z(I)-ZMEAN)*(Z(I)-ZMEAN)
RETURN
END

```

FUNCTION

A function subprogram is an independent set of instructions which returns a result to the program for the function name. This result is placed in the equation at the point where the function is referenced. Parameters are used in the same way as in subroutines.

```

PROGRAM MAIN
.
.
T = 1. - F(X)
.
.
END
FUNCTION F(A)
.
.
F = A*A + R
RETURN
END

```

The result F, calculated in the function subprogram, is placed directly in the equation for T in the main program where F(X) is referenced. (There are library functions such as COS(X), EXP(X), etc., that are used in this way.)

**FUNCTION
DEFINITION**

The general syntax for a function definition statement is:

[type] FUNCTION fun [(d₁[,d₂]...)]

The first statement of a function subprogram must be one of the following forms:

```

FUNCTION fun [d1[,d2]...]
INTEGER FUNCTION fun [(d1[,d2]...)]
DOUBLE FUNCTION fun [(d1[,d2]...)]
DOUBLE PRECISION FUNCTION fun [(d1[,d2]...)]
COMPLEX FUNCTION fun [(d1[,d2]...)]
REAL FUNCTION fun [(d1],d2]...)]
LOGICAL FUNCTION fun [(d1],d2]...)]

```

**FUNCTION
DEFINITION**
(continued)

where fun is an alphanumeric identifier and d_1 are formal parameters. A FUNCTION statement may appear without parameters. The mode of the results stored in the name is determined in the function name statement or with the conventional type indication used for variables. When the type indicator is omitted, the mode is determined by the first character of the function name.

INTEGER FUNCTION F2(I)

is the same as the following two statements:

FUNCTION F2(I)
INTEGER F2

The name of the function must not appear in a DIMENSION declaration in the calling program. No diagnostic is provided, but the program will not call the function. The name must appear within the function subprogram at least once as the left-hand identifier in a replacement statement; otherwise the diagnostic will be A FUNCTION NAME WAS NOT USED AS A REPLACEMENT STATEMENT.

The function name may also appear as an element of an input list within the subprogram or as an actual parameter of a subprogram reference.

**FUNCTION
REFERENCE**

fun (a₁[,a₂]...)

where fun identifies the function being called. It is an alphanumeric identifier, and its type is determined in the same way as a variable identifier. DOUBLE and COMPLEX function references must have a TYPE statement in the calling program. If the default option for typing variables has not been assumed, then specific type statements for the function name are required in the calling program. The a_i are actual parameters. A function reference *must* have at least one parameter, even if it is a dummy. A function reference may appear any place in an expression that an operand may be used. The evaluated function has a value associated with the function name. This value is returned and used in the expression as an operand.

When a function reference is encountered in an expression, control is transferred to that function. When a RETURN statement or an END in the function subprogram is encountered, control is returned to the statement containing the function reference, where expression evaluation is continued.

Example 1

```
FUNCTION GRATER(A,B)
IF(A.GT.B)GO TO 2
GRATER = A-B
RETURN
2 GRATER = A+B
RETURN
END
```

Reference to the function GRATER might be:

W(I,J) = FA + FB - GRATER(C-D,3.*AX/BX)

**FUNCTION
REFERENCE
(continued)****Example 2**

```
FUNCTION PHI(ALPHA,PHI2)
PHI=PHI2(ALPHA)
RETURN
END
```

This function can be referenced:

```
EXTERNAL SNE
C = D-PHI(Q(K),SNE)
```

where SNE is later defined as:

```
FUNCTION SNE(S)
SNE=S
IF(S.LE.0) RETURN
SNE=-SNE
RETURN
END
```

The replacement statement with the function PHI will be executed as if it had been written PHI = SNE(Q(K)).

The name of the function may not be used in an EXTERNAL statement within itself. The diagnostic is VARIABLE IDENTIFIER IN EXTERNAL STATEMENT.

**ARITHMETIC
STATEMENT
FUNCTION**

The statement function is a single expression defined exclusively in the program or subprogram containing it. The definition is placed before the first executable statement of the program and is of the form

$$\text{fun } (d_1[, d_2] \dots) = \text{exp}$$

The name fun of the statement function is an alphanumeric identifier; a value is always associated with the name. The name of the statement function applies only to the program containing the definition, and it must not appear in an EXTERNAL statement.

The d_i are formal parameters and must be simple variables or array names. The d_i are thus dummy arguments indicating type, number, and order of arguments. They may have the same name as variables appearing in the rest of the program, since they are formal parameters only.

The expression exp may be any arithmetic expression which contains references to library functions, other statement functions, or function subprograms.

The identifiers appearing in the expression which are not parameters have their current program values.

Examples

Definition:

```
DIMENSION B(10,10)
Q2(X,Y,Z) = 2*X+Y**2 - Z*COS(A)
C(A,I,J) = A(I,J)
```

Reference:

```
AND = Q2(A+B,COS(Y),T) + A*COS(Y)
Z = C(B,1,2)
```

**ARITHMETIC
STATEMENT
FUNCTION**
(continued)

Q2 is the name of the statement function; the actual parameters a_i may be any arithmetic expression. In the example, A+B is the actual parameter for the formal parameter X, COS(Y) for Y, and T for Z. Actual parameters may be subscripted variables.

During compilation, the arithmetic statement function definition is inserted in the code where the actual reference is made. This reference appears as an operand in an expression.

Rules

- The statement function name must not appear in a DATA, DIMENSION, EQUIVALENCE, COMMON, or EXTERNAL statement; the name may appear in a TYPE declaration, but cannot be dimensioned.
- Statement function names must not appear as actual or formal parameters in subprogram references, since they are defined only locally.
- Actual and formal parameters must agree in number and order. However, the type of the statement function name or the type of formal parameters is ignored; the mode of the evaluated statement function is determined by the mode of the expression given the actual parameters.[†] When the number of entries in the call is not the same as the number of entries in the statement function, the diagnostic ARITHMETIC STATEMENT FUNCTION PARAMETERS DO NOT AGREE IN NUMBER is provided. If the order does not agree, no diagnostic is provided and care must be taken in defining actual parameters.

[†] This is not standard in the American National Standard Institute interpretation of the FORTRAN language.

- All statement functions must precede the first executable statement of the program or subprogram, but they must follow all declarative statements (DIMENSION, TYPE, etc.), or a diagnostic is provided that DECLARATIVE STATEMENTS MUST PRECEDE ALL ARITHMETIC STATEMENTS.
- A statement function may not call itself.
- A statement function must have at least one argument
- Statement functions may generate more instructions in the code than the same program written without the use of statement functions. When the argument is an index function, the compiled code generates a nonstandard index function. This may slow down the program execution.

Examples

Definition:

```
COMPLEX Z
Z(X,Y) = (1.,0.)*EXP(X)*COS(Y) + (0.,1.)*EXP(X)*SIN(Y)
```

Reference:

```
COMPLEX ZZ
ZZ = Z(A,B)
```

**LIBRARY
FUNCTIONS**

Mathematical algorithms, such as sine and tangent, that are used frequently have been written and stored in a reference library and are available to the programmer through the compiler. These may be functions or subroutines or in-line algorithms.

**IN-LINE
ALGORITHMS**

A number of these routines are available as in-line algorithms. Code to accomplish these functions is inserted directly in the program requesting them by the compiler. Table 1 gives the in-line functions available on the NCAR compiler.

**MATHEMATICAL
LIBRARY
ROUTINES**

Certain mathematical routines and functions are supplied by the system library and need not be submitted with the program. These routines are automatically loaded if the user program references them. The names of the FORTRAN library routines, listed alphabetically, are given in Table 2.

Table 7-1. In-Line Functions Available on NCAR Compiler

<u>Description</u>	<u>Forms</u>	<u>Parameter Type</u>	<u>Result Mode</u>
Absolute Value			
To obtain absolute value	ABS(X) [ABSF(X)]	Real	Real
To obtain absolute value	IABS(I) [XABSF(I)]	Integer	Integer
To obtain absolute value	DABS(D)	Double	Double
Conversion			
To truncate a real argument	AINT(X) [INTF(X)]	Real	Real
To truncate a real number	INT(X) [XINTF(X)] IFIX(X) [XFIXF(X)]	Real	Integer
To float an integer number to floating point	FLOAT(I) [FLOATF(I)]	Integer	Real
To obtain the imaginary part of a complex argument	AIMAG(C)	Complex	Real
To obtain a double-precision number from a real number	DBLE(X)	Real	Double
To obtain the real part of a complex argument	REAL(C)	Complex	Real
To convert real to complex	Cmplx(X ₁ , X ₂)	Real	Complex
To obtain the complex conjugate of an argument	CONJG(C)	Complex	Complex
To convert double-precision argument to single-precision	SNGL(D)	Double	Real

Table 7-1. In-Line Functions Available on NCAR Compiler (continued)

<u>Description</u>	<u>Forms</u>	<u>Parameter Type</u>	<u>Result Mode</u>
Location†			
To determine the address of the variable name stated in the argument	ALOC(VAR _i) [LOCF(VAR _i)]	Not applicable	Real
	LOC(VAR _i) [XLOCF(VAR _i)]	Not applicable	Integer
Maximum (or minimum) of n Arguments			
	AMAX0(I ₁ , I ₂ , ..., I _n) [MAX0F(I ₁ , I ₂ , ..., I _n)]	Integer	Real
	AMAX1(X ₁ , X ₂ , ..., X _n) [MAX1F(X ₁ , X ₂ , ..., X _n)]	Real	Real
	MAX0(I ₁ , I ₂ , ..., I _n) [XMAX0F(I ₁ , I ₂ , ..., I _n)]	Integer	Integer
	MAX1(X ₁ , X ₂ , ..., X _n) [XMAX1F(X ₁ , X ₂ , ..., X _n)]	Real	Integer
	AMIN0(I ₁ , I ₂ , ..., I _n) [MIN0F(X ₁ , X ₂ , ..., X _n)]	Integer	Real
	AMIN1(X ₁ , X ₂ , ..., X _n) [MIN1F(X ₁ , X ₂ , ..., X _n)]	Real	Real
	MIN0(I ₁ , I ₂ , ..., I _n) [XMIN0F(I ₁ , I ₂ , ..., I _n)]	Integer	Integer
	MIN1(X ₁ , X ₂ , ..., X _n) [XMIN1F(X ₁ , X ₂ , ..., X _n)]	Real	Integer

† The "real" mode of result may not be practical when dealing with addresses. Integer is the mode of result used with most applications.

Table 7-1. In-Line Functions Available on NCAR Compiler (continued)

<u>Description</u>	<u>Forms</u>	<u>Parameter Type</u>	<u>Result Mode</u>
Modulus			
$X_1 = (X_1/X_2)X_2$	AMOD(X ₁ ,X ₂) [MODF(X ₁ ,X ₂)]	Real	Real
where			
(X_1/X_2) = integer part	MOD(I ₁ ,I ₂) [XMODF(I ₁ ,I ₂)]	Integer	Integer
Positive Difference			
$a_1 = \text{MIN}(a_1, a_2)$	DIM(X ₁ ,X ₂) [DIMF(X ₁ ,X ₂)]	Real	Real
	IDIM(I ₁ ,I ₂) [XDIMF(I ₁ ,I ₂)]	Integer	Integer
Sign			
Sign of X_2 times absolute value of X_1	SIGN(X ₁ ,X ₂) [SIGNF(X ₁ ,X ₂)]	Real	Real
Sign of I_2 times absolute value of I_1	ISIGN(I ₁ ,I ₂) [XSIGNF(I ₁ ,I ₂)]	Integer	Integer

Table 7-1. In-Line Functions Available on NCAR Compiler (continued)

<u>Description</u>	<u>Forms</u>	<u>Parameter Type</u>	<u>Result Mode</u>
Logical Functions			
Logical product: Bit-by-bit logical AND. Variable number of arguments. n=2 or 3 or...or 63.	A=AND(X,...) B=AND(A ₁ ,A ₂ ,...,A _n)	any type	no mode
Logical sum: Bit-by-bit logical OR. Variable number of arguments. n=2 or 3 or...or 63.	A=OR(A ₁ ,A ₂ ,...,A _n)	any type	no mode
Exclusive OR: Bit-by-bit exclusive OR. Variable number of arguments. n=2 or 3 or...or 63.	A=XOR(A ₁ ,A ₂ ,...,A _n)	any type	no mode
Complement: Bit-by-bit Boolean complement of A (i.e., not A)	B=COMPL(A)	any type	no mode
Shifting			
Shift: Shift the number in A,I bit positions: left circular if I > 0; right with sign extension if I < 0 , 0≤ I ≤60	B=SHIFT(A,I)	A - any type I - integer	no mode

Table 7-2. FORTRAN Library Functions

<u>Name</u>	<u>Definition</u>	<u>Parameter Mode</u>	<u>Result Mode</u>
ACOS(X)	Arcosine	Real	Real
ALOG(X)	Natural log of X	Real	Real
ALOG10(X)	Log to the base 10 of X	Real	Real
ASIN(X)	Arcsine	Real	Real
ATAN(X)	Arctangent X radians	Real	Real
ATAN2(X ₁ ,X ₂)	Arctangent X ₁ /X ₂	Real	Real
CABS(C)	Absolute value	Complex	Real
CBAIEX(C,I)†	C**I	Complex (to integer)	Complex
CCOS(C)	Complex cosine	Complex	Complex
CEXP(C)	Complex exponential	Complex	Complex
CLOG(C)	Complex log function	Complex	Complex
COS(X)	Cosine X radians	Real	Real
CSIN(C)	Complex sine	Complex	Complex
CSQRT(C)	Complex square root	Complex	Complex
CUBRT(Y)	Cube root	Real	Real
DATAN(D)	Double arctangent	Double	Double
DATAN2(D ₁ ,D ₂)	Double arctangent D ₁ /D ₂	Double	Double
DBADEX(D ₁ ,D ₂)†	D ₁ **D ₂	Double (to double)	Double
DBAIEX(D,I)†	D**I	Double (to integer)	Double
DBAREX(D,X)†	D**X	Double (to real)	Double
DCOS(D)	Double cosine	Double	Double
DEXP(D)	Double exponential function (e ^D)	Double	Double
DLOG(D)	Natural log of D (base e)	Double	Double

† These routines must be called with a ** operator in FORTRAN. They may not be used as functions. Use of the function name is only allowed in assembly language. These routines must save and restore A0 as well as B-registers used. Other library functions need only save the B-registers.

Table 7-2. FORTRAN Library Functions (continued)

<u>Name</u>	<u>Definition</u>	<u>Parameter Mode</u>	<u>Result Mode</u>
DLOG10(D)	Log to base 10 of D	Double	Double
DMAX1(D ₁ ,D ₂ ,...,D _n)	Maximum of n arguments	Double	Double
DMIN1(D ₁ ,D ₂ ,...,D _n)	Minimum of n arguments	Double	Double
DMOD(D ₁ ,D ₂)	D ₁ modulo D ₂ D ₁ -(AINT(D ₁ /D ₂)D ₂)	Double Double	Double Double
DSIGN(D ₁ ,D ₂)	Sign of D ₂ times D ₁	Double	Double
DSIN(D)	Sine of double-precision argument	Double	Double
DSQRT(D)	Square root of double	Double	Double
EXP(X)	e to x th power (e ^x)	Real	Real
IBAIEX(I ₁ ,I ₂)†	I ₁ **I ₂	Integer (to integer)	Integer
IDINT(D)	Convert double-precision argument to integral floating-point number (rounded)	Double	Real
RANF(X)	Random number generator; uniform distribution 0. < RANF(X) < 1.	Real	Real
RBAIEX(X,I)†	X**I	Real (to integer)	Real
RBAREX(X ₁ ,X ₂)†	X ₁ **X ₂	Real (to real)	Real
SIN(X)	Sine X radians	Real	Real
SQRT(X)	Square root of X	Real	Real
TAN(X)	Tangent X radians	Real	Real
TANH(X)	Hyperbolic tangent X radians	Real	Real

† These routines must be called with a ** operator in FORTRAN. They may not be used as functions. Use of the function name is only allowed in assembly language. These routines must save and restore A0 as well as B-registers used. Other library functions need only save the B-registers.

**AUXILIARY
LIBRARY
ROUTINES**

A number of miscellaneous subroutines are also included in the library and need not be submitted with the user deck. These are not the standard routines associated with STANDARD FORTRAN programs, and may not be available at other installations. These are essentially NCAR routines. Included in Table 7-3 is a list of the routines.

Table 7-3. Auxiliary Library Routines

<i>Subroutine</i>	<i>Entry Points</i>	<i>Subroutine</i>	<i>Entry Points</i>	<i>Subroutine</i>	<i>Entry Points</i>
ACGOER	ACGOER	INQUIRE	INQUIRE	PWRY	PWRY
BACKSPACE	BACKSP, IFENDF, REWINM	INPUTC	INPUTC	Q8QRSD	Q8QRSD, NUMP
BRANRD	BRANRD, BRANCK, BRANST, BRANRL	INPUTS	INPUTS	RDBK	RLBK, WTBK, RDBK
BUFFEI	BUFFEI, BUFFEO, IOCHEK, LENGTHF	IOB	INPUTB, OUTPTB	RPTIN/ RPTOUT	RPTIN, RPTOUT
BUFRD	BUFINT, BUFWT, BUFRD, BUFCL	IOPROC	BSTAPE, RDTAPE, WRTAPE, IOWAIT	SAVEF	SAVEF
BUFILE		JOBID	JOBID	SETPASS	SETPASS
CLOSE	CLOSE	KODER	KODER	SKIPFILE	BACKFILE, SKIPFILE
CLD	CLOCKF, DATEF	KRAKER	KRAKER	SORT	SORT
CURVED	CURVED	LCMREQ	LCMRD, LCMWT	SSW	SSW
DASHD	DASHD	MACHINE	MACHINE	SYSRCL	SYSRCL
DEBUG	DEBUG	NEWVOL	NEWVOL	TIMEF	TIMEF
DUMP	DUMP	NTIG	NTIG	ULIBER	ULIBER
ENCD	ENCD	OUTPTC	OUTPTC	UNLCRD	UNLCRD, UNLCWT, UNLCK
ENDFIL	ENDFIL	OUTPTS	OUTPTS	UNLOAD	UNLOAD
EXIT	END, EXIT, STOP	OPEN	OPEN	VARGCK	VARGCK
FLRGCK	FLRGCK	OVERLAY	OVERLAY	VOLCPY	VOLCPY
GBYTE	GBYTE, GBYTES, SBYTE, SBYTES	PAUSE	PAUSE		
		PDUMP	PDUMP		
		PSYMD	PSYMD		

BLOCK DATA

A BLOCK DATA subprogram is a separate, nonexecutable subprogram used for initialization of data. The first statement of a BLOCK DATA subprogram must have the following form:

BLOCK DATA [sub]

sub is a symbolic identifier consisting of from one to seven alphanumeric characters and starting with a letter.

Rules

- Only data statements and specification statements may be included. No executable statements are allowed.
- There may be more than one BLOCK DATA subprogram in a deck, each having a unique alphanumeric name.
- There may be only one unnamed or blank BLOCK DATA subprogram.
- If the deck contains OVERLAYS and SEGMENTS, use a named BLOCK DATA subprogram, e.g., BLOCK DATA BLKDATA.
- DATA may not be entered in blank COMMON with a DATA statement.
- All specification statements for a COMMON block must be included, such as TYPE, DIMENSION, and EQUIVALENCE statements, even though DATA statements do not appear for all variables in that block.
- DATA may be initialized in more than one COMMON block in any one BLOCK DATA subprogram.

Example

```
BLOCK DATA IRT
COMMON/XYZ/A(4)
DATA A/1.,2.,3.,4./
```

RETURN AND END STATEMENTS

A subprogram may contain one or more RETURN statements to indicate the end of logic flow within the subprogram and return control to the calling program. The form is:

RETURN

In function subprograms, control returns to the statement containing the function reference. In subroutine subprograms, control returns to the next executable statement following the CALL. A RETURN statement in the main program causes the diagnostic RETURN NOT ALLOWED IN MAIN PROGRAM.

END

The main program and every subprogram must contain one END statement, and it must be the last statement of the program or subprogram. The END card in a main program generates a CALL EXIT. The END statement in a subprogram generates a RETURN.

ENTRY STATEMENT

The ENTRY statement provides alternate entry points to a function or subroutine subprogram. Its form is

```
ENTRY en
```

where en is a symbolic identifier consisting of one to seven alphanumeric characters and starting with a letter.

The ENTRY statement should appear directly ahead of the first statement to be executed when the subprogram is called by that entry name. Only one name may be given a particular entry point.

Example 1

```
SUBROUTINE T1(A,B)
  .
  .
  ENTRY T2
  .
  .
  ENTRY T3
  .
  .
END
```

In the example, arguments appear on the SUBROUTINE card, not on the ENTRY statement card. The ENTRY card just establishes the place to enter the subprogram.

In the calling program, the reference to the entry name is made just as if reference were being made to the FUNCTION or SUBROUTINE in which the ENTRY is embedded:

```
CALL T1(X,Y)
CALL T2(XX,YY)
CALL T3(W,Z)
```

Example 2

```
FUNCTION ARG(A,B)
ARG = A+B
RETURN
ENTRY SOLX
ARG = B-A
RETURN
ENTRY SOLY
ARG = A*B - A*B
RETURN
END
```

A reference to the function ARG in a calling program must include parameters. For the following example, the call would be

```
...
R = ARG(COS(X)*3.,T)
...
Z = SOLX(X,Y) + SOLY(X1,Y1)
...
```

In this example notice that the subprogram may be entered at SOLX and SOLY as alternate entry points to ARG, which is the function name entry. The code following the entry points must define the same variable ARG to be returned to the calling program. The entry points SOLX and SOLY do *not* specify arguments in the subprogram definition. The calling programs use the entry point name *and* the arguments to branch to the function subroutine or subprogram. The arguments appear in the call (note SOLX(X,Y) above).

Rules

- In the subprogram, the name of the entry point appears only in the ENTRY statement.
- Formal parameters do *not* appear with the ENTRY statement. The diagnostic is INCORRECT FORM FOR THE ENTRY STATEMENT.
- In a function subprogram, the value is returned in the same variable name used in the function definition, regardless of entry point.
- ENTRY may not appear within a DO loop. The diagnostic is AN ENTRY STATEMENT MAY NOT OCCUR INSIDE A DO LOOP.
- All declarative statements must appear ahead of all executable statements including the ENTRY statement.
- An ENTRY statement may not be labeled. The diagnostic is DO NOT LABEL AN ENTRY STATEMENT.
- Entry points must agree in type with the function name. The mode of the results will be incorrect, but no diagnostic is provided.
- Calling programs use the entry point name *and* the actual argument list, if any, to branch to the function subroutine or subprogram.
- The implementation described does not conform to the FORTRAN 77 specification for entry.

**EXTERNAL
STATEMENT**

When the actual parameter list which calls a function or subroutine subprogram contains a function or subroutine name as an argument, that name must be declared in an EXTERNAL statement in the calling program. Its form is

EXTERNAL en₁ [,en₂]...

where en_i is the name of a function or subroutine. The EXTERNAL statement must precede the first executable statement of any program which calls a function or subroutine subprogram using the EXTERNAL name. When it is used, EXTERNAL always appears in the calling program. It may not be used with statement functions; if it is, a compiler diagnostic is given.

Example

```
PROGRAM TEST
EXTERNAL TNG,ST
.
.
.
R = F1(TNG,X)
.
.
.
U = F1(ST,X)
.
.
.
END

FUNCTION F1(F,X)
.
.
.
F1 = F(X) - 49.3641*X*X
.
.
.
END

FUNCTION TNG(Y)
.
.
.
TNG = SIN(Y)/COS(Y)
.
.
.
END

FUNCTION ST(S)
.
.
.
ST = S**2.4+COS(S)
.
.
.
END
```

**EXTERNAL
STATEMENT**
(continued)

All programmer generated subprograms may be used as externals in an actual parameter list. FORTRAN library functions may *not* be used as externals. If the names of FORTRAN library functions appear in an EXTERNAL statement, the diagnostic INTRINSIC OR LIBRARY FUNCTION CANNOT BE DECLARED EXTERNAL will be printed.

**VARIABLE
DIMENSIONS IN
SUBPROGRAMS**

In many subprograms, especially those performing matrix manipulation, the programmer may wish to vary array dimensions each time a subprogram is called.

This is accomplished by specifying the array name and its dimensions as formal parameters in the FUNCTION or SUBROUTINE statement. The corresponding actual parameters specified in the calling program are values used by the calling program as dimensions. The maximum dimensions that any given array may assume are determined by dimensions in a DIMENSION, COMMON, or TYPE statement in the calling program at compile time.

Rules

The following rules must be adhered to or the program will fail in ways that are difficult to debug. Compiler diagnostics are *not* provided in all cases, so care must be taken to write correct code.

- The formal parameters representing the array dimensions must be simple integer variables. The array name must also be a formal parameter.
- The actual parameters representing the array dimensions may be unsigned integer constants or integer variables. A floating point or a zero argument may cause an infinite loop.
- If the total number of elements of a given array in the calling program is N, then the total number of elements of the corresponding array in the subprogram may not exceed N. The code may be overwritten in the calling program.

**VARIABLE
DIMENSIONS IN
SUBPROGRAMS**
(continued)

Example

CARD NUMBER	APPROXIMATE PROGRAM LOCATION	*FORTRAN,FL
1	0	PROGRAM TEST
2	0	DIMENSION A(8,2),E(5,6)
3	0	C=1.
4	56	READ (5,100) M,N,(A(I,J),I=1,M),J=1,N
5	103	100 FORMAT (210/(8E10.0))
6	103	CALL MAT(A,M,N,C)
7	111	DC 2 I=1,5
8	111	DC 2 J=1,6
9	112	2 E(I,J)=1.C+.
10	125	CALL MAT (B,5,6,2.0)
11	134	WRITE (6,101) A
12	141	101 FCRMAT (*0A MATRIX*/(1H BF10.1))
13	141	WRITE (6,102) E
14	146	102 FCRMAT (*0B MATRIX*/(1H 5F10.1))
15	146	END

CARD NUMBER	APPROXIMATE PROGRAM LOCATION	
1	0	SUBROUTINE MAT (X,K,L,C)
2	0	DIMENSION X(K,L)
3	0	DC 2 I=1,K
4	2	DC 2 J=1,L
5	3	2 X(I,J)=X(I,J)+C
6	20	RETURN
7	21	END

A MATRIX								
2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	
2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	

B MATRIX								
4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	
5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	
6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	
7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	
8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0	
9.0	9.0	9.0	9.0	9.0	9.0	9.0	9.0	

In the example, the array X and the variable dimensions K,L must appear as formal parameters in the SUBROUTINE statement. The actual parameters in the main program vary from CALL to CALL. The second call to MAT sends the array B and its dimensions 5 and 6 as actual parameters. All three must appear in the CALL.

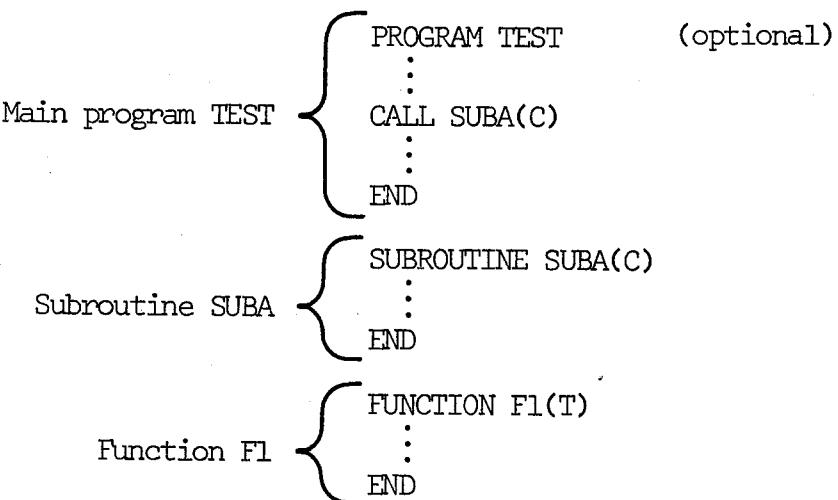
The compiler does *not* check to see if the limits of the array established by the DIMENSION statement in the main program are exceeded.

**ARRANGEMENT OF
FORTRAN DECKS**

All FORTRAN programs and subprograms are compiled as independent units. A SUBROUTINE or FUNCTION card must appear as the first card of a subprogram and the FORTRAN END card must be the last card of a program or subprogram. A PROGRAM card is optional in a main program. All statements between these two cards are assumed by the compiler to belong to one program or subprogram with a unique name

Example

Sample Deck Arrangement:



VIII

INPUT/OUTPUT FORMATS

STANDARD SUBSCRIPTS
NONSTANDARD SUBSCRIPTS
ARRAY TRANSMISSION
CONVERSION SPECIFICATIONS
EDITING SPECIFICATIONS

s FORMAT (spec₁,...,j(spec_m,...),spec_n,...)

INPUT/OUTPUT FORMATS

INTRODUCTION

The FORMAT statement is used in conjunction with the input/output (I/O) statement in the transmission of data. The specification for the conversion to or from BCD is controlled by the FORMAT statement. The I/O statement specifies the I/O device and process--READ, WRITE, etc.-- and the list of data to be moved. The FORMAT statement specifies the form in which the data is to be moved. In binary input/output statements no FORMAT statement is used, as the data is transmitted in binary without conversion.

INPUT/OUTPUT LIST

The list portion of an I/O statement indicates the data items and the order, from left to right, of their transmission. List items may be array names, simple or subscripted variables, or an I/O statement implied DO loop. List items are separated by commas, and their order must correspond to the FORMAT specification referred to by the I/O statement.

Execution of the I/O statement is begun and terminated by the list. External records are always read or written until the list is satisfied, repeating the format scan as required.

STANDARD SUBSCRIPTS

Subscripts in an I/O list may be standard or nonstandard index functions. Standard index functions are of the form:

(c*I+d)	(I+d)
(I*c+d)	(c*I)
(d+I*c)	(I)
(d+c*I)	(c)
(c+I)	

c and d are unsigned integer constants, and I is a simple integer variable, previously defined, or defined within the I/O statement DO loop. The following are examples of FORTRAN READ statements:

```

READ 100,A,B,C,D
READ 100,B(3,4),A(I,J,7),H
READ 101,J,A(J),L,B(I,J)
READ 102,A(5*j+2,5*L-3,5*K),C,D,(I+7)

```

The following examples show standard subscripts within an I/O statement DO loop.

```

READ 100,A,B,C,(D(I),I=1,10),E(5,7),F(J),(G(I),H(I),I=2,6,2)
READ 100,I,J,K,((A(II+1,JJ+2,KK+3),II=1,I),JJ=1,J),KK=1,K
READ 100,(A(I,J),I=2,10,2),B(J,1),J=1,5),E,F,G(L+5,M-7)
READ 100,(C(4*j-3),J=1,4)
READ 100,(C(J),C(J+2),C(J+4),J=1,2)

```

**NONSTANDARD
SUBSCRIPTS**

Nonstandard subscripts are any arithmetic expression not defined above as standard. The expression is evaluated, and the result is truncated and used as the index.

```
WRITE (6,100) C(X+Y),C(T)
WRITE (6,100) C(COS(A)),C(MAX0(1,3)+1)
```

Limitations

- Nonstandard subscripts may *not* be used within an I/O statement implied DO loop. The diagnostic will be IMPROPER SUBSCRIPT IN AN INPUT OR OUTPUT STATEMENT.
- Subscripted subscripts are never allowed.
- More subscripts on a variable than specified in the DIMENSION statement for that variable will cause the diagnostic TOO MANY SUBSCRIPT INDICES.
- The DO variable in an I/O list DO may *not* use the same DO variable as a DO loop around the I/O statement. The diagnostic A DO VARIABLE IS USED IN AN OUTER DO LOOP appears.

**ARRAY
TRANSMISSION**

Part or all of an array can be represented for transmission as a single I/O list item by using DO loop notation in the WRITE statement. The general form of the implied DO loop is:

```
((A(I,J,K),I=m1,m2[,m3]), J=n1,n2[,n3]), K=p1,p2[,p3])
```

where m_i, n_i, and p_i are unsigned integer constants or simple integer variables. If m₃, n₃, or p₃ is omitted, the number used is equal to 1. I,J,K are subscripts of A.

All notation must be explicitly defined in the WRITE statement.

**ARRAY
TRANSMISSION**
(continued)

During execution, each subscript (index variable) is set to the initial index value: $I = m_1$, $J = n_1$, $K = p_1$. The first index variable defined in the list is incremented first, following the same rules as DO loop execution. When the first index variable reaches the maximum value, it is reset, the next index variable to the right is incremented, and the process is repeated until the last index variable has been incremented. If $m_1 > m_2$ initially, one card is read.

Rules

- An array name which appears without subscripts in an I/O list causes transmission of the entire array. The first index or subscript of the array varies most rapidly.
- A DO loop can be used to transmit a simple variable more than one time. For example, the list item $(A(K),B,K=1,5)$ causes the transmission of variable B five times. However, in the case $(B,(A(K),K=1,5))$, B is transmitted only once.
- A list of the form $K,(A(I),I=1,K)$ is permitted, and on input the input value of K is used as the value of K in the DO loop.

Examples

```

READ 100,(A(I),I=1,10)
READ 100,((A(JV,JX),JV=2,20,2),JX=1,30)
READ 100,(BETA(3*N+7),N=JONA,JONB,JONC)
READ 100,(((IMLST(I,J+1,K-2),I=1,125),J=2,N),K=IVAR,IVMAX,4)
READ 100,(A(I),B(I),I=1,10)

```

Nested DO loop list items are specified in the order of their indexing.

```
READ 100,((((A(I,J,K),B(I,L),C(J,N),I=1,10),J=1,5),K=1,8),
           L=1,15),N=2,7)
```

Data are transmitted in the following sequence:

```
A(1,1,1),B(1,1),C(1,2),A(2,1,1),B(2,1),C(1,2)...
...A(10,1,1),B(10,1),C(1,2), A(1,2,1),B(1,1),C(2,2)...
...A(10,2,1),B(10,1),C(2,2)...A(10,5,1),B(10,1),C(5,2)...
...A(10,5,8),B(10,1),C(5,2)...A(10,5,8),B(10,15),C(5,2)...
...A(10,5,8),B(10,15),C(5,7)
```

Although the above list is highly redundant, the items illustrate DO loop execution. The following list will transmit the array E(3,3) by columns:

```
READ 100,((E(I,J),I=1,3),J=1,3)
```

The following list will transmit the array E(3,3) by rows:

```
READ 100,((E(I,J),J=1,3),I=1,3)
```

The statements

```
DIMENSION ARRAY (3,4,7)
READ 100, ARRAY
```

are equivalent to the statements

```
DIMENSION ARRAY (3,4,7)
READ 100,(((ARRAY(I,J,K),I=1,3),J=1,4),K=1,7)
```

An I/O statement loop and a DO statement loop are similar, but not equivalent.

The I/O statement loop

```
READ 100,(A(I),I=1,3)
```

is the same as

```
READ 100,A(1),A(2),A(3)
```

**ARRAY
TRANSMISSION**
(continued)

The DO loop

DO 2 I=1,3
2 READ 100,A,(I)

is the same as

READ 100,A(1)
READ 100,A(2)
READ 100,A(3)

The same elements of A are output, but in an I/O loop only one read is generated, whereas in the DO loop three reads are performed.

**FORMAT
DECLARATION**

The formatted I/O statements require a FORMAT declaration which contains the conversion and editing information relating to the internal/external structure of the corresponding I/O list items. A FORMAT declaration has the following form:

s FORMAT (spec₁,...,j(spec_m,...),spec_n,...)

spec_i is a FORMAT specification, and j_k is an optional repetition factor which must be an unsigned integer constant. If j_k is omitted, the repeat will be continued to the end of the list. The FORMAT declaration is nonexecutable and may appear anywhere in the program. It is used by the I/O statement to convert internal/external structures. All FORMAT declarations must have a statement label in columns 1-5, as shown by s. The maximum number of FORMAT statements in any program unit is 64. (e.g. program, subroutine, or function subprogram.)

FORMAT SPECIFICATIONS

The data items in an I/O list are converted from external to internal or from internal to external representation according to FORMAT conversion specifications. Each item in the list must have a FORMAT specification associated with it. FORMAT specifications may also contain editing codes.

FORTRAN Conversion Specifications

Ew.d	Single-precision floating point with exponent
Fw.d	Single-precision floating point without exponent
Gw.d	General floating point
Dw.d	Double-precision floating point with exponent
Iw	Decimal integer
Ow	Octal integer
Aw	Alphanumeric
Rw	Alphanumeric
Lw	Logical
nP	Scaling factor

FORTRAN Editing Specifications

wX	Intraline spacing
wH	Heading and labeling
/	Begin new record (n/, advances n records)
...	Alphanumeric string included
'...'	Alphanumeric string included

Both w and d are unsigned integer constants: w specifies the field width in number of character positions in the external record, and d specifies the number of digits to the right of the decimal within the field.

Star (*) and apostrophe ('') are delimiters for an alphanumeric string to be included in the print line.

Note: Complex data items are converted on I/O according to a pair of consecutive Ew.d or Fw.d specifications.

**CONVERSION
SPECIFICATIONS**

Each item in the I/O list must have a FORMAT specification associated with it. One FORMAT specification will be used repeatedly for more than one item if only one FORMAT specification appears.

Ew.d INPUT

The E specification converts the number in the input field to a real number and stores it in the appropriate location in storage.

The total number of characters in the input field is specified by w. The field is scanned from left to right; blanks within the field are interpreted as zeros.

The integer subfield begins with a sign (+ or -) or a digit and may contain a string of digits. The integer field is terminated by a decimal point, a D, an E, a + or -, or the end of the input field.

The fraction subfield which begins with a decimal point may contain a string of digits. The field is terminated by a D, an E, a + or -, or the end of the input field.

The exponent subfield may begin with a D, an E, or a + or -. When it begins with D or E, the + is optional between E, or D and the string of digits of the subfield. The value of the string of digits in the exponent subfield must be in the range -293 to +322 in order to be printed. A number as small as 10^{-295} may be held in memory even though on output it prints as zero (see Constants in Chapter 2).

Ew.d INPUT
(continued)
Permissible Subfield Combinations

+1.6327E-04	integer fraction exponent
-32.7216	integer fraction
+328+5	integer exponent
.629+5	fraction exponent
+136	integer only
136	integer only
.07628431	fraction only
E-06 (interpreted as zero)	exponent only

Rules

- In the Ew.d specification, d acts as a negative power of ten scaling factor when an external decimal point is not present. The internal representation of the input quantity is

$$(\text{integer subfield}) \times 10^{-d} \times 10^{(\text{exponent subfield})}$$

For example, if the specification is E7.8, the input quantity 3267+05 is converted and stored as $3267 \times 10^{-8} \times 10^5 = 3.267$. The input quantity |3210000| would be stored as .0321.

- If a decimal point occurs in the input field, the decimal point overrides d. The input quantity 3.67294+5 read by an E9.d specification is always stored as 3.67294×10^5 , regardless of the value of d.
- When d does not appear, it is assumed to be zero (E9,E15).
- The field length specified by w in Ew.d should always be the same as the length of the input field containing the input number. When it is not, incorrect numbers may be read, converted, and stored. No diagnostic is provided in these cases.

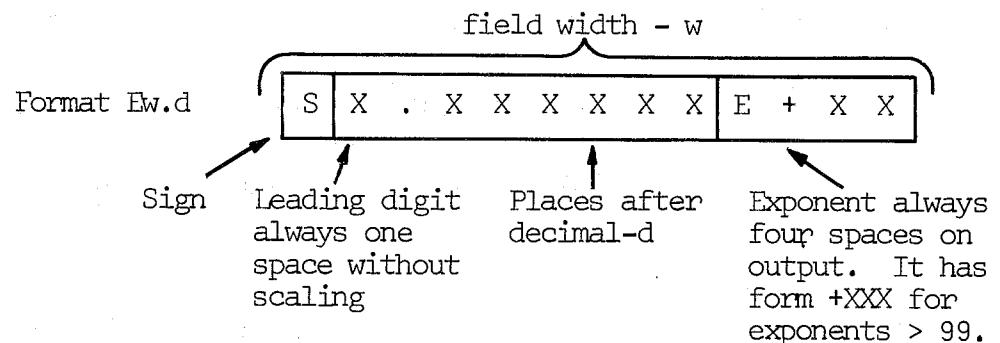
- An exponent greater than 322 or less than -295 will not be accepted on input. The diagnostic is EXPONENT TOO LARGE ON DATA INPUT.
- ILLEGAL DATA ENOUNTERED is printed for input fields, such as letters, that cannot be interpreted.

<u>Ew.d Input Field</u>	<u>Ew.d Format Specification</u>	<u>Converted Value</u>	<u>Remarks</u>
+143.26E-03	E11.2	.14326	All subfields present
-12.437629E+1	E13.6	-124.37629	All subfields present
8936E+004	E9.10	.008936	No fraction subfield; input number converted as $8936 \times 10^{-10+4}$
327.625	E7.3	327.625	No exponent subfield
4.376	E5	4.376	No d in specification
-.0003627+5	E11.7	-36.27	Integer subfield contains minus only
-.0003627E5	E11.7	-36.27	Integer subfield contains minus only
blanks	Ew.d	-0.	All subfields empty
1E1	E3.0	10.	No fraction subfield; input number converted as $1. \times 10^1$
E+06	E10.6	0.	No integer or fraction subfield; zero stored regardless of exponent field constants
1.bEbl	E6.3	10.	Blanks are interpreted as zeros

Ew.d OUTPUT

E conversion is used to convert real numbers in storage to the BCD character form for output. The field occupies w positions in the output record; the corresponding real number appears right-justified in the field in one of the following forms:

Sa.a...aE _{ee}	$0 \leq ee \leq 99$
Sa.a...a _{eee}	$100 \leq eee \leq 308$



a.a...a are the most significant digits of the integer and fractional part, and eee are the digits in the exponent. S is the sign, blank for plus on output. If d is zero or blank, the decimal point and digits to the right of the decimal do not appear as shown above. Field w must be wide enough to contain the significant digits, signs, decimal point, E, and the exponent. Generally, $w \geq d + 7$. Positive numbers need not reserve a space for the sign of the number. If the field width is less than the number of places specified for the decimal field, a diagnostic is printed at execution time.

If the field is not wide enough to contain the output value, an asterisk is inserted in the high-order position of the field. If the field is longer than the output value, the quantity is right-justified with blanks in the excess positions to the left.

Examples

PRINT 10,A
10 FORMAT (1Hb,E10.3) A contains -67.32
or +67.32

Result: -6.732E+01 or b6.732E+01

PRINT 10,A
10 FORMAT (1Hb,E13.3) A contains -67.32
or +67.32

Result: bbb-6.732E+01 or bbbb6.732E+01

PRINT 10,A
10 FORMAT (1Hb,E9.3)

A contains +67.32.
Since A is positive,
no provision is
necessary for the sign

Result: 6.732E+01

PRINT 10,A
10 FORMAT (1Hb,E10.4)

A contains -67.32. No
provision is made for
the sign, thus an *
indicates a bad format

Result: *.7320E+02

PRINT 100,A
100 FORMAT (1Hb,E10.2)

Where A=+∞
(37777777777777777777B)
and is output with an
E format

Result: RRRR

PRINT 100,A
100 FORMAT (1Hb,E10.2)

Where A=indefinite
(17777777777777777777B)
and A is printed with
an E format

Result: IIII

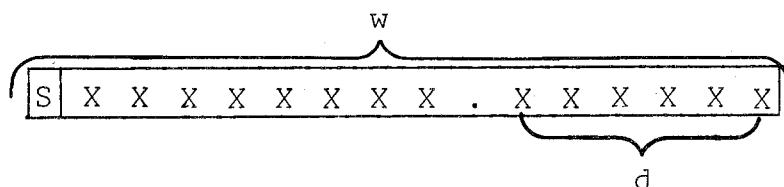
Fw.d INPUT

This specification is a modification of Ew.d. The input field consists of an integer and a fraction subfield. An omitted subfield is assumed to be zero. All rules listed under Ew.d apply.

<u>Fw.d Input Field</u>	<u>Fw.d Format Specification</u>	<u>Converted Value</u>	<u>Remarks</u>
367.2593	F8.4	367.2593	Integer and fraction field
37925	F5.7	.0037925	No fraction subfield; input number converted as 37925×10^{-7}
-4.7366	F7	-4.7366	No d in specification, but decimal in input field is used
.62543	F6.5	.62543	No integer subfield
.62543	F6.2	.62543	Decimal point in input field overrides d of specification
+144.15E-03	F11.2	.14415	Exponents are legitimate in F input and may have P-scaling

Fw.d OUTPUT

The field occupies w positions in the output record; the corresponding list item must be a floating point quantity, and it appears on output as a decimal number, right-justified in the field w, as



S is the sign. X represents the significant digits of the number to be printed or output. The number of decimal places to the right of the decimal is specified by d. If d is zero or omitted, the decimal point and digits to the right do not appear. If the number is positive, the plus sign is suppressed.

If the field is too short to accommodate the number, one asterisk appears in the high-order position of the output field.

If the field w is longer than required to accommodate the number, the number is right-justified with blanks occupying the excess field positions to the left.

Examples

PRINT 10,A A contains +32.694
10 FORMAT (1X,F7.3)

Result: b32.694

PRINT 11,A A contains +32.694
11 FORMAT (1X,F10.3)

Result: bbbb32.694

PRINT 12,A A contains -32.694;
12 FORMAT (1X,F6.3) no provision for
 minus sign

Result: *2.694

PRINT 13,A,A A contains .32694
13 FORMAT (1X,F4.3,F6.3)

Result: .327b0.327

PRINT 14,A,B,C,D A contains +0
14 FORMAT (1X,F10.4) B contains -0
 C contains +.000000001
 D contains -.000000001

Result: bbbb0.0000
 bbb-0.0000
 bbbb0.0000
 bbbb-.0000

Gw.d INPUT

Gw.d input specification is the same as Ew.d input specification.

Gw.d OUTPUT

The G conversion is a general conversion used for real data output. The floating point quantity is output as an F number or an E number depending on the magnitude of the number itself. w designates the field length as before. d, however, is the number of significant digits of the value to be represented unless the number reverts to a standard E format where d is the number of digits following the decimal point.

In the following table N is the magnitude of the number being converted. Four blanks are inserted within the field, right-justified if F format is used within a G specification.

<u>Range of Number</u>	<u>Type of Conversion</u>
.1 ≤ N < 1	F(w-4).d,4X
1 < N < 10	F(w-4.).(d-1),4X
⋮	⋮
$10^{d-2} \leq N < 10^{d-1}$	F(w-4).1,4X
$10^{d-1} \leq N < 10^d$	F(w-4).0,4X
Otherwise	Ew.d

<u>Internal Number</u>	<u>Gw.d Format Specification</u>	<u>Printer Output</u>
23.146	G10.3	23.1
.023	G10.3†	2.300E-02
123456789.2	G14.4†	1.2346E+08
123.456	G14.4	123.5
.001	G10.2†	1.00E-03
1.234E-1	G15.3	.123
1.23E+5	G15.6	123000
234.567	G10.5	234.57
"	G10.4	234.6
"	G10.3	235
"	G10.2†	2.35E+02
"	G10.1†	2.3E+02

† In these G specifications, the internal number does not conform to N as defined above so that d may *not* be used as the number of significant digits. Therefore, the format reverted to Ew.d.

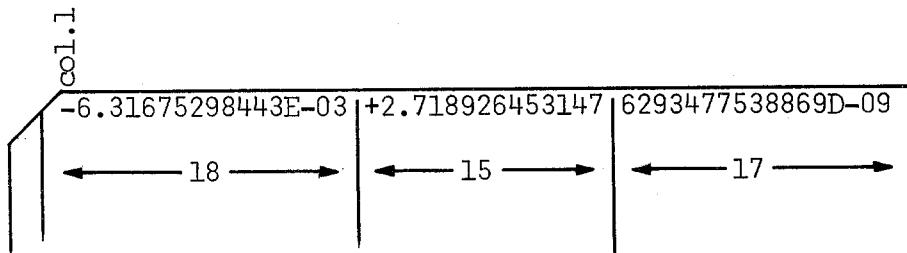
Dw.d INPUT

D conversion corresponds to Ew.d input. In double precision, 27 significant digits are permitted in the combined integer-fraction field. D or E is acceptable as the beginning of an exponent subfield on input. (Note: E is *not* acceptable in a replacement statement to define a double-precision quantity.)

Example

```
DOUBLE Z,Y,X
READ 1,Z,Y,X
1 FORMAT(D18.11,D15,D17.4)
```

Input Card:

**Dw.d OUTPUT**

The field occupies w positions of the output record. The corresponding list item, which must be a double-precision quantity, appears as a decimal number, right-justified in the field w, as one of the following:

SXX...X.X....X+eee
SX...X.X....XD+ee

100 ≤ eee ≤ 293
0 ≤ ee ≤ 99

S indicates the sign (blank for plus). D conversion corresponds to Ew.d output, but uses a double-precision (two-word) quantity. Single-precision information must *not* be output using the D specification.

Iw INPUT

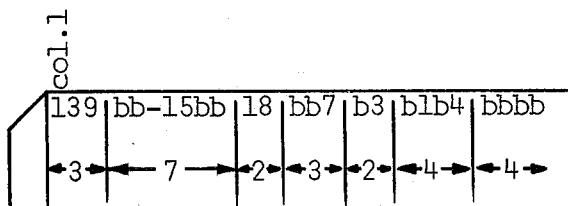
The field is w characters in length, and the corresponding list item may be a decimal integer constant or a logical constant.

The input field w which consists of an integer subfield may contain only the characters +, -, the digits 0 through 9, or blank. When a sign appears, it must precede the first digit in the field. Blanks are interpreted as zeros. Where the whole field is blank, the value is -0. The value is stored right-justified in the specified variable.

Example

```
READ 10,I,J,K,L,M,N,N1
10 FORMAT (I3,I7,I2,I3,I2,2I4)
```

Input Card:



In Storage:

- I contains 139
- J contains -1500
- K contains 18
- L contains 7
- M contains 3
- N contains 104
- N1 contains -0

Iw OUTPUT

I specification is used to output decimal integer values. The output quantity occupies w output record positions; it appears right-justified in the field w as



S is the sign. XXXXX are the decimal digits (maximum 18) of the integer. If the integer is positive, the plus sign is suppressed.

If the field w is larger than required, the output quantity is right-justified with blanks occupying excess positions to the left. If the field is too short, characters are printed starting from the right, with an asterisk in the leftmost position.

Example

```
PRINT 10,I,J,K  
10 FORMAT (1X,I8,I10,I5)
```

I contains -3762
J contains +4762937
K contains +13

Result: bbb-3762bbb4762937bbb13

Ow INPUT

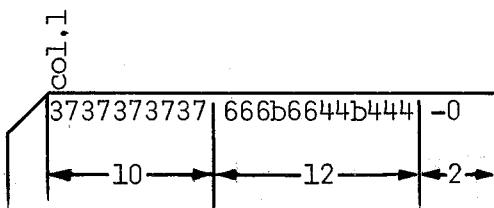
Octal integer values are converted under O specification. The field is w characters in length, and the corresponding list item must be an integer variable.

The input field w consists of an integer subfield only (maximum of 20 octal digits). The only characters that may appear in the field are +, -, blank, and 0 through 7. Only one sign is permitted and, if present, must precede the first digit in the field. Blanks are interpreted as zeros.

Example

```
TYPE INTEGER,P,Q,R
READ 10,P,Q,R
10 FORMAT (010,012,02)
```

Input Card:



In Storage:

```
P: 00000000037373737
Q: 0000000666066440444
R: 77777777777777777777
```

A negative number is represented in one's complement form.

A negative octal number is represented internally in 20-digit seven's complement form obtained by subtracting each digit of the octal number from seven. For example, if -703 is an input quantity, its internal representation is 7777777777777777074. That is:

$$\begin{array}{r}
 777777777777777777 \\
 - 0000000000000000703 \\
 \hline
 7777777777777777074
 \end{array}$$

Ow OUTPUT

O specification is used to output internal values as an octal representation of the binary word in memory. The output quantity occupies w output record positions, and it appears right-justified in the field as

XXXXXX

The X are octal digits. If w is less than 20, the right-most w digits appear, with an asterisk in the first digit to indicate that there are nonzero numbers left out. If w is greater than 20, the number is right-justified in the field with blanks to the left of the output quantity. A negative number is output in its ones-complement internal form. If w is 20, 20 digits appear.

Example

```
A=25252525256767676767B  
PRINT 100,A  
100 FORMAT(1X,020)  
PRINT 101,A,A  
101 FORMAT(1X,018/1X,025)  
  
Result: 252525252567676767  
*52525256767676767  
252525252567676767
```

Aw INPUT

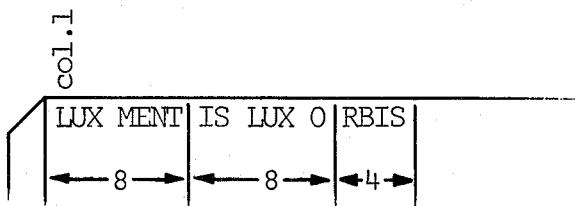
This specification accepts characters in the FORTRAN character set, including blank. The internal representation is display code; the field width is w characters.

If w exceeds 10, the input quantity is the rightmost 10 characters in the field. If w is 10 or less, the input quantity goes to the designated storage location as a left-justified BCD word; remaining spaces are blank filled.

Example

```
READ 10,Q,P,O
10 FORMAT (A8,A8,A4)
```

Input Card:



In Storage:

Q:	LUXbMENTbb
P:	ISbLUXbObb
O:	RBISbbbbbb

Octal Word:

Q:	14243055150516245555
P:	11235514253055175555
O:	22021123555555555555

Aw OUTPUT

A conversion is used to output alphanumeric characters. If w is 10 or more, the output quantity appears right-justified in the output field, blank filled to the left. If w is less than 10, the output quantity represents the leftmost w characters.

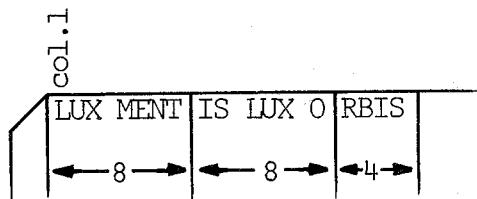
Rw INPUT

This specification is the same as the Aw input unless w is less than 10, in which case the input quantity goes to the designated storage location as a right-justified binary zero filled word.

Example

```
READ 10,Q,P,O
10 FORMAT (R8,R8,R4)
```

Input Card:



In Storage:

Q: ::LUXbMENT

P: ::ISbLUXb0

O: :::::::RBIS

Octal Word:

Q: 00001425305515051624

P: 00001123551425305517

O: 00000000000022021123

Rw OUTPUT

This specification is the same as the Aw output unless w is less than 10, in which case the output quantity represents the rightmost characters in the word.

Example

```
PRINT 100,B,C
100 FORMAT (1H0,R5,R3)
```

Printed Output:

PRSTUXYZ

In Storage:

B: KLMNOPRSTU

C: LTTEMP3XYZ

Lw INPUT

This specification accepts logical variables for input. w is the width of the external field. This field must contain T, F, TRUE or FALSE, preceded and/or followed by optional blanks.

Lw OUTPUT

Output specifications are given for logical variables. w is the width of the field where T or F may be preceded by w-1 blanks.

Example

```
TYPE LOGICAL L,K
L=.TRUE.
K=.FALSE.
PRINT 100, L,K
100 FORMAT (1H0,2L5)
```

Result: bbbbTbbbbF

A logical constant in a replacement statement may be .T., .F., .TRUE., .FALSE.. (See Logical Constants in Chapter 2.)

SCALE FACTOR

A scale factor may precede the D, E, and F conversion.

The scale factor is:

$$\text{External number} = \text{internal number} \times 10^n \text{ scale factor}$$

The scale factor applies to Fw.d on both input and output and to Ew.d and Dw.d on output only. A scaled specification is written in FORTRAN as:

nPDw.d
nPEw.d
nPFW.d
nP

Where n is a signed integer constant.

Rules

- P preceded by blank is interpreted as 0P.
- The scale factor is assumed to be zero if no other value has been given; however, once a value has been given, it holds for *all* D, E, and F specifications following the scale factor within the same FORMAT declaration. To nullify this effect in subsequent D, E, and F specifications, a zero scale factor, 0P, must precede a D, E, or F specification. When a FORMAT is repeated from a single I/O statement because the list is not exhausted, the last scale factor continues to apply to specifications at the beginning unless scale factors are included for each specification. Use (0PF10.5,1PE15.6) rather than (F10.5,1PE15.6).
- Scale factors on D or E input specifications are ignored.

- The scaling specification nP may appear independent of a D, E, or F specification, but it holds for all D, E, and F specifications that follow within the same FORMAT statement unless changed by another nP. For example, the specification (3P,3I9,F10.2) is the same as the specification (3I9,3PF10.2) when using the same FORMAT for repeated scans.

Fw.d SCALING

Input

The number in the input field is divided by 10^n and stored. For example, if the input quantity 314.1592 is read under the specification 2PF8.4, the internal number is $314.1592 \times 10^{-2} = 3.141592$.

Output

The number in the output field is the internal number multiplied by 10^n . In the output representation, the decimal point is fixed; the number moves to the left or right, depending on whether the scale factor is plus or minus. For example, the internal number 3.1415926538 may be represented on output under scaled F specifications as follows:

<i>Fw.d Format Specification</i>	<i>Output Representation</i>
F13.6	3.141593
1PF13.6	31.415927
3PF13.6	3141.592654
-1PF13.6	.314159

**Ew.d OR Fw.d
SCALING****Output**

The scale factor has the effect of shifting the output number left n places while reducing the exponent by n. Using 3.1415926538, some output representations corresponding to scaled E specifications are:

<i>Format Specifications</i>	<i>Output Representations</i>
E20.2	3.14 E+00
1PE20.2	31.42 E-01
2PE20.2	314.16 E-02
3PE20.2	3141.59 E-03
4PE20.2	31415.93 E-04
5PE20.2	314159.27 E-05
-1PE20.2	0.31 E+01

EDITING SPECIFICATIONS

wX SPECIFICATION

This specification may be used to include w blanks in an output record or to skip w characters on an input record to permit spacing of I/O quantities. Blank is represented as 1X. If w is not specified as in (F4.1,X,I5), it is interpreted as 1X. 0X is not permitted.

Examples

INTEGER A	A contains 7
PRINT 10,A,B,C	B contains 13.6
10 FORMAT (12,6X,F6.2,6X,E12.5)	C contains 1462.37

Results: b7bbbbbb13.60bbbbbbbl.46237E+03

Note: The initial characters of a line on the printer is interpreted as a carriage control command.

READ 502,I,S,NAM
502 FORMAT (4X,I6,3X,F8,2X,A10)

Input Card:

The diagram shows a rectangular input card. At the top left, there is a small vertical stack of three characters: a solid square (DC1), a dot (DC2), and a solid square (DC3). Below this, there is a horizontal line. To the left of the line, there is a small vertical tick mark. To the right of the line, the text "12Cbbb9999EEbl.21658bXXbbbbbbMARY" is printed, starting just after the tick mark.

In Storage:

I: 9999
S: 1.21658
NAM: MARY

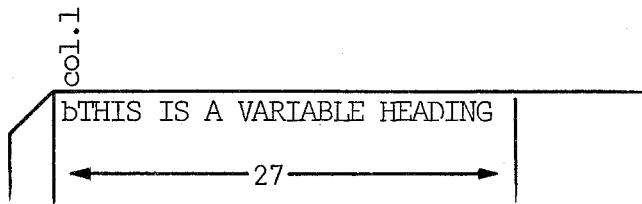
WH INPUT

The H specification may be used to read Hollerith characters into an existing H field within the FORMAT specification itself.

Example 1

```
READ 10
10 FORMAT (27Hbbbbbbbbbbbbbbbbbbbbbb)
```

Input Card



After the READ, the FORMAT statement labeled 10 contains the alphanumeric information read from the input card; a subsequent reference to statement 10 in an output statement acts as follows:

```
PRINT 10
```

Result: bTHIS IS A VARIABLE HEADING

Note: The initial character, a blank, is interpreted by printer as a carriage control command.

Example 2

```
READ 11
11 FORMAT (*      *)
```

Where the card is punched (bbVARIABLE), PRINT 11 produces the line bbVARIABLE, where the initial blank is for carriage initial on the printer.

wH OUTPUT

This specification provides for the output of 6-bit characters, including blanks, in the form of comments, titles, and headings. w is an unsigned integer specifying the number of characters to the right of the H that are transmitted to the output record; w may specify a maximum of 136 characters. In the case of records sent to the printer, the first character for carriage control is excluded from the maximum count of 136. H denotes a Hollerith field. The comma following the H specification is optional.

Examples

Note: The initial characters in each print record are interpreted as carriage control by the printer.

```
PRINT 20
20 FORMAT (28HbBLANKSbCOUNTbINbANbHbFIELD.)
```

Result: BLANKS COUNT IN AN H FIELD.

```
PRINT 30,A
30 FORMAT (6HbLMAX=,F5.2)
```

A contains 1.5
Comma is optional

Result: LMAX=bl.50

```
PRINT 504
504 FORMAT (*0 ASTERisks MAY BE USED TO ENCLOSE
A HOLLERITH FIELD ONLY IN A FORMAT
STATEMENT*)
```

Result: ASTERisks MAY BE USED TO ENCLOSE A HOLLERITH FIELD ONLY IN A FORMAT STATEMENT

```
TEMP = 32.
PRINT 504,TEMP
504 FORMAT (1H0*TEMPERATURE=*F3.0,* DEGREES*)
```

Result: TEMPERATURE= 32 DEGREES

```
PRINT 200,PE
200 FORMAT ('OFOR PE='F2,0)
```

PE contains 2.0

Result: FOR PE=b2

NEW RECORD

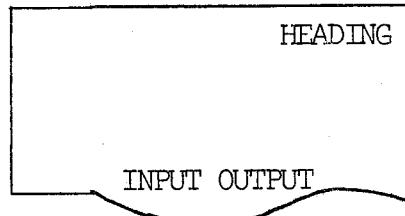
The slash (/) signals the end of a record anywhere in the specifications list. It need not be separated from the other list elements by commas; consecutive slashes may appear in a list. During output, the slash is used to skip lines, cards, or tape records. The maximum number of printed characters per line is 137, 136 characters printed plus 1 initial character for carriage control. One line on the line printer is a record. K(/) results in k-1 lines being taken on output.

During input, a slash (/) specifies that control passes to the next record or card. K(/) results in K-1 cards being skipped on input. One card has a maximum of 80 characters.

Example 1

```
PRINT 10
10 FORMAT (I6X,7HHEADING///6X,12HINPUT OUTPUT)
```

Printout:



(line 1)
(line 2)
(line 3)
(line 4)

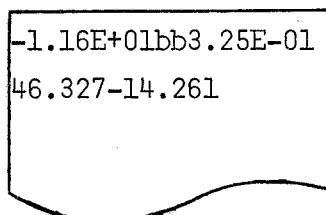
Each line corresponds to a BCD record. The second and third records are null and produce the line spacing illustrated.

Example 2

```
PRINT 11,A,B,C,D
11 FORMAT (2E10.2/2F7.3)
```

A contains -11.6
B contains .325
C contains 46.327
D contains -14.261

Printout:



(line 1)
(line 2)

Example 3

```
PRINT 11,A,B,C,D  
11 FORMAT (2E10.2/ /2F7.3)
```

Printout:

```
-1.15E+01bb3.25E-01  
46.327-14.261
```

(line 1)
(line 2)
(line 3)

Example 4

```
PRINT 15,(A(I),I=1,9)  
15 FORMAT (8HbRESULTS2(/)(3F8.2))
```

Printout:

```
RESULTS  
  
3.62 -4.03 -9.78  
-6.33 7.12 3.49  
6.21 -6.74 -1.18
```

(line 1)
(line 2)
(line 3)
(line 4)
(line 5)

Note: see section below on unlimited groups

REPEATED FORMAT SPECIFICATIONS

Any FORMAT specification may be repeated by using an unsigned integer constant repetition factor, k, as follows: K(spec), where spec is any conversion specification except nP. For example, to print two quantities, K,L:

```
PRINT 10,K,L  
10 FORMAT (I2,I2)
```

Specifications for K,L are identical; the FORMAT statement could use a repeat factor with I2.

```
10 FORMAT (2I2)
```

When a group of FORMAT specifications repeats itself, as in

```
FORMAT(E15.3,F6.1,I4,I4,E15.3,F6.1,I4,I4)
```

the use of k produces

```
FORMAT (2(E15.3,F6.1,2I4))
```

The parenthetical grouping of FORMAT specifications is called a repeated group. Repeated groups may be nested to two levels:

```
FORMAT (spec1,n(spec2)...)
```

A grouping (k(kspec)) such as

```
FORMAT(2(2E15.3,2F6.1,2I4))
```

is allowed. More than two levels in a repeated parenthesis group produces the diagnostic PAREN GROUP NOT CLOSED IN FORMAT STATEMENT.

UNLIMITED GROUPS

FORMAT specifications may be repeated without using a repetition factor. The innermost parenthetical group that has no repetition factor is unlimited and will be used repeatedly until the I/O list is exhausted. Parentheses are the controlling factors in repetition. The right parenthesis of an *unlimited* group is equivalent to a slash. Specifications to the right of an unlimited group can never be reached.

Example

Where A=1.2, B=2.3, I=1, J=2, K=3

```
PRINT 100,A,B,I,J,K,A,B
100 FORMAT(E16.3,F20.7,2(I4),(I3,F7.1),F8.2)
```

Result: 1.200E+00 2.3000000 1 2 3 1.2
R

The first two fields print according to E16.3 and F20.7. Since 2(I4) is a repeated parenthetical group, the next two fields are printed according to I4. The print fields follow (I3,F7.1), an unlimited group which does not have a repetition factor until the list elements are exhausted. F8.2 can never be reached, and B will be printed using I3, producing a range error. No diagnostic is given for format specifications to the right of an unlimited group.

VARIABLE FORMAT

FORMAT specifications may be defined at the time of program execution. The specification, including left and right parentheses but not the statement label or the word FORMAT, are read under A conversion or defined in a DATA statement and stored in an integer array. The name of the array containing the specifications may be used in place of the FORMAT statement labels in the associated I/O operation. The array name that appears specifies the location of the first word of the format information. This name may appear in an I/O statement as a format with a constant subscript or without subscripts. Variable subscripts are not allowed.

Assume the following FORMAT specifications:

(E12.2,F8.2,I7)

The information could be punched in an input card and read by a program such as

```
DIMENSION IFMT(2)
READ 100,(IFMT(I),I=1,2)
100 FORMAT (2A10)
```

IFMT contains the FORMAT specification as follows:

IFMT(1) contains (E12.2,F8.
IFMT(2) contains 2,I7)WWWW

A subsequent output statement uses the FORMAT specification as follows:

PRINT IFMT,A,B,I

This produces exactly the same result as fixed format in the following program:

```
PRINT 101,A,B,I
101 FORMAT (E12.2,F8.2,I7)
```

Another way to define variable format is with the DATA statement:

```
DIMENSION IF1(3),IF2(3),A(6),IP(3),TEMP(3)
DATA IF1/25H(1H ,2F6.3,I7,2E12.2,3I1)/,IF2/24H(1H ,I6,6X,
+3F4.1,2E12.2)/
```

The following output statement

```
PRINT IF1,(A(I),I=1,2),K,B,C,(IP(J),J=1,3)
```

is the same as

```
PRINT 102,(A(I),I=1,2),K,B,C,(IP(J),J=1,3)
102 FORMAT (1H ,2F6.3,I7,2E12.2,3I1)
```

The output statement

```
PRINT IF2,LA,(A(M),M=3,4),A(6),(TEMP(I),I=2,3)
```

is the same as the following one with fixed format:

```
PRINT 103,LA,(A(M),M=3,4),A(6),(TEMP(L),L=2,3)
103 FORMAT (1H I6,6X,3F4.1,2E12.2)
```


IX

INPUT/OUTPUT STATEMENTS

DATA TRANSFER
INPUT/OUTPUT TERMINOLOGY
READING AND WRITING
DATASET SUPPORT STATEMENTS
FILE POSITIONING
SUMMARY TABLE OF FORTRAN SYNTAX

Synchronous Input/Output Statements
Asynchronous Input/Output Statements
ENCODE/DECODE Statements
Dataset Support Statements
File Positioning Statements

INPUT/OUTPUT STATEMENTS

DATA TRANSFER

Data may be transferred from an external device to computer memory, from computer memory to an external device or from one place in computer memory to some other location in computer memory. Data transfers are accomplished in FORTRAN by using the input/output statements described in this chapter.

UNITS

The unit number in an input/output statement refers to an external device such as a 1/2" tape, a volume on a mass storage device, or the card punch or printer. In core-to-core transfers, the device referenced may also be a variable located in central memory. The unit is designated as u, or, in keyword form, UNIT=u.

SOURCE DATA

Data are transferred as a stream of bits that represent values or characters. The data are organized into records containing one or more values. Data records are separated by record marks. The records and record marks may then be organized into a file. An end-of-file mark may be used to mark the end of a collection of records, i.e., a file.

A more complete description of data sources, data paths and data access is available in two manuals, *The NCAR Terabit Memory System* (NCAR/TN-124+IA), and *The NCAR Data Storage System* (NCAR/TN-125+IA). This chapter includes all of the input/output statements with a description of the parameters included in each. More detailed descriptions of related conventions such as the Job Control Language (JCL) appear in the manuals referenced above.

THE I/O LIST

The form iolist indicates an input/output list in statements discussed in this section. The names of variables and array elements to be input or output are specified in the list.

Examples

```
WRITE (6,100) A,B,I
```

```
PRINT 100, (I,J),(T,U)
```

```
DIMENSION A(10),B(10)
```

```
WRITE (2)(A(I),I=1,10),B(J),J=1,10)
BUFFER IN (8,1)(A(I),B(10))
```

```
DIMENSION A(5),B(5)
```

```
READ (5,100)(I,J),A,(B(L),L=1,5)
```

An array may be input or output using implied DO notation. In the following example, N (in this case N is 5) elements of A are printed.

```
DIMENSION A(10)
```

```
N = 5
```

```
PRINT 100,(A(I),I=1,N)
```

This same list array may be input or output without DO notation by stating only the array name. In this case, all elements of A specified in the DIMENSION statement are output (10 in the following example):

```
DIMENSION A(10)
```

```
PRINT 100,A
```

RULES

- Constants may not appear in an I/O list.
- Nonstandard subscripts may not appear in I/O lists.
- An I/O list may be empty.

**INPUT/OUTPUT
TERMINOLOGY****SYNCHRONOUS
AND ASYNCHRONOUS**

Transfers may be classified as synchronous or asynchronous, depending on whether control is returned to the program before or after completion of the data transfer.

Synchronous data transfer occurs when control is returned to the executing program only after the input/output operation is completed. Asynchronous (often called buffered) data transfer occurs when control is returned to the FORTRAN program before input/output has been completed.

**CORE-TO-CORE
TRANSFER**

Data transfers such as ENCODE/DECODE that move data from one part of core to another core location are synchronous transfers, because the operation must be completed before control is returned to the next FORTRAN statement.

ACCESS TO DATA

Programmers may use direct or sequential access to files in FORTRAN statements. It is possible to access a file directly by name and its records directly by record number or to read datasets and records sequentially. In addition to the READ and WRITE statements, there are three FORTRAN statements used in support of these access methods: The OPEN, CLOSE and INQUIRE statements. The OPEN statement is used to connect old or new files to a unit or to change some of the attributes of connection to a file. The CLOSE statement disconnects a dataset from a unit. The use of the INQUIRE statement allows the programmer to determine the properties of a file or of connection to a file. The direct access capability and OPEN, CLOSE and INQUIRE statements are FORTRAN 77 features.

**KEYWORDS AND
KEYLISTS NOTATION**

Parameters used in these statements have the keyword letters designated in capitals followed by an equals sign and lower case letters, which symbolize a constant or variable which defines either an attribute or an inquiry value.

**FORMATTED
AND UNFORMATTED**

A data transfer is unformatted if there is no editing of the data to or from character code using a FORMAT statement. A data transfer with editing specified in a FORMAT statement is said to be formatted.

Synchronous and asynchronous input/output statements may use either direct or sequential access methods to transfer records and files that may be either formatted or not formatted.

Core-to-core transfers are always synchronous using sequential access and FORMAT statements.

<i>Control</i>	<i>Access Method</i>	<i>Editing</i>
Asynchronous	direct	formatted or unformatted
Asynchronous	sequential	formatted or unformatted
Synchronous	direct	formatted or unformatted
Synchronous	sequential	formatted or unformatted

**READING
AND WRITING**

The input/output statements are divided into various groupings in the following pages that describe the syntax. Data transfers from an external device to memory are referred to as reading. Writing describes output operations that transfer information from computer memory to an external device.

The syntax of these statements contains a list of keyword parameters within parentheses that describe the kind of input/output specified. Square brackets indicate optional items in the syntax.

The next two sections describe the generalized form of synchronous and asynchronous input/output on the NCAR system. Following that, some of the specific forms are described in more detail, e.g., the conventional, discrete forms that may be derived from the generalized forms. Other statements that are not generalizable are also included, like BUFFER IN and ENCODE/DECODE.

**GENERALIZED FORM
OF SYNCHRONOUS
READ/WRITE**

An input/output statement is called synchronous if control is not returned to the FORTRAN program until the read or write operation has been completed. While either direct or sequential access may be used, any unit specified for direct access must be opened for that type of access in a FORTRAN OPEN statement.

FORM

```
READ ([UNIT=]u[,FMT=]f)[,REC=rn][,ERR=s1][,END=s2]  
[,IOSTAT=ios]) iolist
```

```
WRITE ([UNIT=]u[,FMT=]f)[,REC=rn][,MODE=md][,ERR=s1]  
[,IOSTAT=ios]) iolist
```

Examples

```
READ (9,100)A,B
```

```
READ (UNIT=8,100)X,Y
```

```
READ (16,FMT=100,END=25)X
```

```
WRITE (UNIT=20,REC=13,ERR=99,END=98)X
```

```
WRITE (20,100,ERR=90,END=98,IOSTAT=NIO)X
```

INPUT PARAMETERS

u
UNIT=u

u is an expression specifying the unit number associated with a volume on a *VOLUME card and in a FORTRAN OPEN statement, if it appears. If the unit appears without the keyword, it must be the first parameter.

f
FMT=f

f is a FORMAT statement number. If the keyword FMT= is omitted, f must be the second parameter. If f appears in the parameter list, the records in the dataset are formatted; if f does not appear, the records are binary records (unformatted).

REC=rn

rn is an expression whose value is the record number specified for reading or writing using direct access. If the parameter REC=rn is specified, the END=s parameter may not be used. The record number specifier is not valid for sequential access.

INPUT PARAMETERS*(continued)*

MODE=md

md is an expression defining the mode of data in the record, and will be used to assist in conversion of data between computers. Once given in a write operation, subsequent operations will use the same value unless re-specified. This keyword is ignored in read operations.

- md 0 The data in the record is considered to be normal character data internal to the machine.
- 1 The data in the record is considered to be binary, bit-serial.
- 2 The data in the record is considered to be Binary Coded Decimal (BCD).
- 3 The data in the record is considered to be American Standard Code for Information Interchange (ASCII).
- 4 The data in the record is considered to be Expanded Binary Coded Decimal Interchange Code (EBCDIC).
- 5 The data in the record is considered to be in the binary integer format.
- 6 The data in the record is considered to be normalized floating point numbers.
- 7 The data in the record is DPC card images truncated to include only the initial non-blank characters in the image.

DEFAULT: MODE=0 for formatted; MODE=1 for binary.

ERR=s₁

s₁ is a statement label to which the program branches if an error condition occurs. If this parameter is omitted, the program terminates on an error condition. Without the use of this keyword parameter in conjunction with the INQUIRE statement, it is very difficult to determine the cause of an error. In using the error branch, the INQUIRE statement may be used to determine the type of error.

END=s₂

If an end-of-file is the last record read, a branch to statement s₂ is taken. If this parameter is omitted, execution proceeds to the next executable statement following the end file record. Use of END=s₂ is permitted only with sequential access.

OUTPUT PARAMETERS

IOSTAT=ios

ios is an integer variable or array element name indicating the completion status of the input/output operation.

ios=0 successful operation
ios>0 an error code appears in bits 0-11
ios<0 an end-of-file on a read or an end-of-tape on a write

Error codes are defined under STATUS in chapter 8 of *The NCAR Terabit Memory System*. Execution terminates on an error condition unless an ERR=s parameter appears in the list of keywords.

iolist

iolist is an input/output list containing the names of variables and array elements to be processed.

SYNTAX ERROR DIAGNOSTICS

END MUST REFER TO STATEMENT LABEL
ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE
FORMAT MUST BE INTEGER CONSTANT
ONLY 1ST OR 2ND ARG MAY BE DEFINED WITHOUT A KEYWORD
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED

**SPECIFIC FORMS
OF SYNCHRONOUS I/O**

A number of forms of synchronous I/O are available at NCAR. Some may be derived from the generalized forms in the previous section and are special cases of the generalized form. Some have their own special form, and remain from previous FORTRAN implementations to provide compatibility with past programming conventions. The section is subdivided into input/output, formatted and unformatted.

**SPECIFIC OUTPUT
STATEMENTS UNDER
FORMAT CONTROL**

Parameters in the forms used in this section are defined as follows:

f	format statement label or array with Hollerith format data
iolist	input/output list which may contain an implied DO
u	unit number, unsigned integer constant or variable

READ f[,iolist]

READ f[,iolist] reads one or more card images from the standard input unit, covering the information from left to right in accordance with FORMAT specification f, and stores the converted data in the storage locations named by iolist.

Example

```
READ 10,A,B,C
10 FORMAT (3F10.4)
```

The formatted READ statements transfer at least one record of information from a specified unit (u) to storage locations named by iolist, according to FORMAT specification (f). To read from the card reader, u may either be omitted or set to 5. The following two READ statements are equivalent:

Example

```
READ(5,10)A,B,C
READ 10,A,B,C
```

PRINT f[,iolist]

PRINT f[,iolist] transfers information from the storage locations given by iolist to the standard output unit. The information is transferred as line printer images, 136 characters or fewer per line, in accordance with the FORMAT declaration, f. The maximum record length is 136 characters, with the first character of every record used for carriage control. Characters in excess of the print line are lost; each new record starts a new print line.

<i>Carriage Control Character</i>	<i>Action</i>
Blank or any character other than the following	Single space before printing
0	Double space before printing
1	Eject page before printing causing the record to be printed at the top of a new page
+	Suppress spacing before printing, causing two successive records to be printed on the same line
-	Triple space

The characters 0, 1, + and - are not printed as first characters of a line since they are carriage control; any other character which is first in a line causes single spacing and is *not* printed.

Example

```

PRINT 50,A,B,C(I,J)
50 FORMAT (3X,8HMINIMUM=F17.7,2X,8HMAXIMUM=F8.8,
           2X,10HVALUE IS $F8.2)
      PRINT 51,(A(I),I=1,20)
51 FORMAT (*1VECTORA*/1H ,10F10.2))

```

PUNCH f[,iolist]

PUNCH f[,iolist] transfers information from the storage locations given by iolist identifiers to the standard punch unit. The information is transferred as card images, 80 characters or fewer per card, in accordance with the FORMAT declaration, f.

Example

```
PUNCH 52,ACCT,ISTAT,LOC,TE,PE,1  
52 FORMAT (F8,3X4A10,2F10.2,I5)
```

The above format assumes the following dimension statement:

```
DIMENSION ISTAT (2),LOC(2)
```

WRITE (u,f)[,iolist] The formatted output statements are forms which transfer information from storage locations given by iolist to a specified output unit (u) according to the FORMAT declaration, f. The unit number is 6 to designate the printer. If it is not 6, some external device such as tape or the mass storage device must be specified on a *ASSIGN or *VOLUME card. See *A User's Guide, Part II* (NCAR/TN-106+IA) for job control language information.

The record containing up to 136 characters is recorded on magnetic tape in even parity if u is assigned to a tape unit. The number of words in iolist output according to the FORMAT declaration (f) determines the number of records that are written on a unit.

On the NCAR system, unit 6 refers to the printer, and unit 5 to the card reader. If the user switches these unit numbers by mistake, no message is printed, but the I/O statement will not work.

If the programmer fails to allow for a printer control character, the first character of the output data is lost on the printed listing.

Example

```
      WRITE (6,53)A,B,C,D
53  FORMAT (1H0,4E21.9)
      WRITE (6,54)
54  FORMAT (32H THIS STATEMENT HAS NO DATA LIST)
```

**UNFORMATTED
INPUT STATEMENTS**

READ (u)[iolist]

The unformatted input statements are forms which transfer one record of information from a specified unit (u) to storage locations named by iolist. Since a FORMAT statement is not given, no character conversions are performed on the data; the data is read in binary mode, odd parity.

A record read by READ (u) in FORTRAN should have been written in binary (unformatted) mode by a corresponding WRITE statement in FORTRAN. However, the number of words in the list of READ *u)iolist may be fewer than the number of words in the corresponding WRITE statement. Only those variables read will be stored in the program.

If iolist is omitted, READ (u) spaces over one record. If iolist is shorter than the record written, READ (u)iolist spaces over the unread elements at the end of the record. (Unread list elements may appear only at the end of a list.)

Examples

```
DIMENSION C(264)
READ (10)C
DIMENSION BMAX(10),M2(10,5)
DO 9 I=1,10
9 READ (7)BMAX(I),(M2(I,J),J=1,5)
      READ(7)((A(I,J),I=1,100),J=1,50)
      READ (8)
```

UNFORMATTED OUTPUT STATEMENTS

The unformatted output statements are forms which transfer information from storage locations given by iolist to a specified output unit (u) in binary mode, odd parity. If iolist is omitted, the WRITE (u) statement acts as a do-nothing statement. One FORTRAN WRITE statement constitutes one record on tape.

Example

```
DIMENSION A(250),B(4000),AMAX(10),M(10,5)
WRITE (10),A,B
DO 5 I=1,10
 5 WRITE (9),AMAX(I),(M(I,J),J=1,5)
```

The number of words in iolist determines the number of physical records that are written on that unit for a given record in cases where the reference is to 1/2" tape

The conversion parameter (CONV=LG) on the *VOLUME card (see chapter 3 of *The NCAR Terabit Memory System* and pages 3.8-9 of *An Introduction to the NCAR Data Storage System*) inhibits writing the two control words when the disk system is used for staging. These control words will only appear on 1/2" magnetic tape. Units assigned to the drum or disk do *not* have this record format, nor does the use of CONV=LG on the *VOLUME card generate this format. Without CONV=LG on the *VOLUME card, the two extra control words are written, unless VSN=DISK, or VSN=DRUM.

Example

```
*ASSIGN,DISK=1
*VOLUME,2,VS N=DUM2,STAGE IN=NS,NM,STAGEOUT=ZT,CONV=BN
*VOLUME,3,VS N=DUM3,STAGEIN=NS,NM,STAGEOUT=ZT,CONV=BN,LG
*ASSIGN,B3141=4
      PROGRAM I04
      COMMON X(3000)
      DO 1 I=1,300
      1 X(I) = RANF(C.)
      DO 3 NUNIT=1,4
      REWIND NUNIT
      WRITE (NUNIT) X
      ENDFILE NUNIT
      3 CONTINUE
      STOP 99
      END
```

TAPE LABEL	PHYSICAL UNIT	CHANNEL ON CHAN 10, CHAN 11.	PARITY ERRORS	READ ERRORS	TAPE ACTIVITY SUMMARY			WORDS READ	FILE MARKS READ	WORDS WRITTEN	FILE MARKS WRITTEN
					WRITE ERRORS	RECORDS PER	WORDS READ				
DUM2	B		0	0	0	0	0	0	0	6	3012 1
DUM3	B		0	0	0	0	0	0	0	6	3000 1
1 B	B		0	0	0	0	0	0	0	6	3000 1
B3141	S		0	0	0	0	0	0	0	6	3012 1

**STRUCTURE OF
LOGICAL RECORDS**

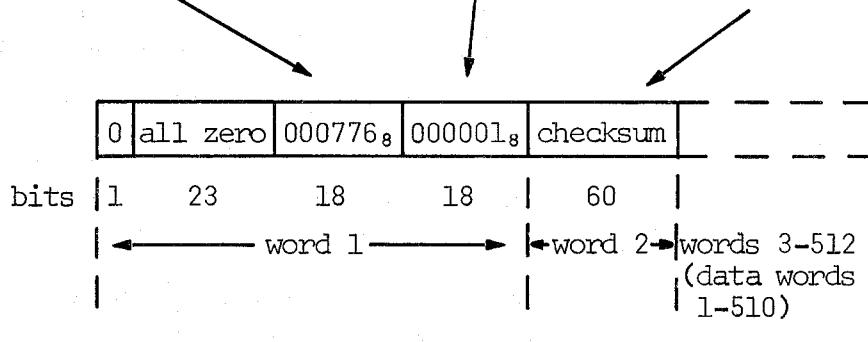
WRITE (8),AVE,NREC,((VAL(I,J)I=1,70),J=1,9)
 This statement will write a logical record having 63 words of data. The logical tape record is in the format shown in the following example:

Physical Record 1:

Count of data words in this physical record = 510_{10}

First physical record in this logical record

Checksum of the 510 data words (60 bit ACL... add with end-around carry)



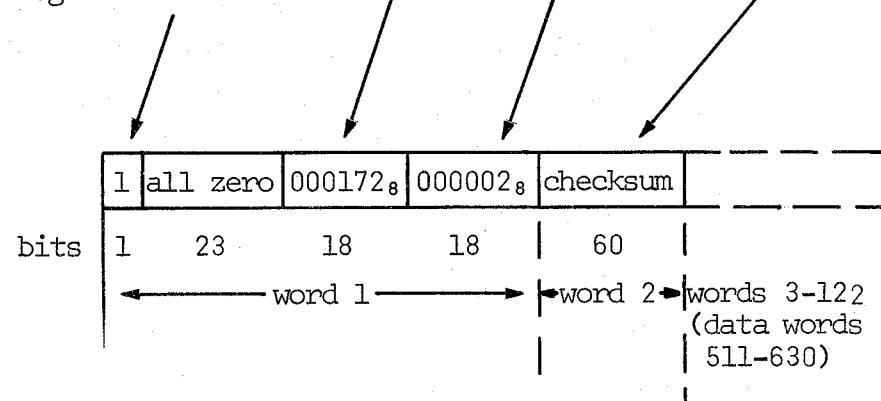
Physical Record 2:

Last physical record in this logical record

Data words = 122_{10}

Second physical record

Checksum of 122 data words



Since the maximum size of a physical record for an unformatted binary write on 1/2" tape is 510 data words, the logical record of 630 words is broken into two physical records. Due to the presence of control words, a total of 634 words is written onto tape, in this example (2+510+2+120).

**GENERALIZED FORM
OF ASYNCHRONOUS
READ/WRITE**

An input/output statement is called asynchronous if control is returned to the FORTRAN program before the read or write operation has been completed. Buffered or asynchronous input/output allows the user to initiate an I/O request and continue executing while it is being processed. Asynchronous requests are recognized by the presence of the keyword NWORDS in the keylist. This I/O form may be used with either direct or sequential access method. A single address, fwa, which is a variable or array element, locates the first word of the block of words to be processed. Asynchronous input/output is a local extension to the FORTRAN standard and is not portable. The alternate form is BUFFER IN/BUFFER OUT (see the next section). The system status word, ns, will be zero from the time the operation is initiated until it is completed.

FORM

```
READ ([UNIT=]u[,REC=rn],NWORDS=n[,NSTATE=ns]) fwa
```

```
WRITE ([UNIT=]u[,REC=rn][,MODE=md],NWORDS=n[,NSTATE=ns]) fwa
```

Examples

```
READ (10,NWORDS=100,NSTATE=NST) A(1)
```

```
WRITE (16,NWORDS=1000,REC=10,NSTATE=NSTA) X(1)
```

INPUT PARAMETERSu
UNIT=u

u is an expression specifying the unit number associated with a volume on a *VOLUME card and in a FORTRAN OPEN statement, if it appears. If the unit appears without the keyword, it must be the first parameter.

REC=rn

rn is an expression whose value is the record number of the record desired within the dataset connected for direct access. rn must be specified if the file is open for direct access. It is not valid for sequential access.

MODE=md

md is an expression defining the mode of data in the record, and will be used to assist in conversion of data between computers. Once given in a write operation, subsequent operations will use the same value unless re-specified. This keyword is ignored in read operations.

- md 0 The data in the record is considered to be normal character data internal to the machine.
- 1 The data in the record is considered to be binary, bit-serial.
- 2 The data in the record is considered to be Binary Coded Decimal (BCD).
- 3 The data in the record is considered to be American Standard Code for Information Interchange (ASCII).
- 4 The data in the record is considered to be Expanded Binary Coded Decimal Interchange Code (EBCDIC).
- 5 The data in the record is considered to be in the binary integer format.
- 6 The data in the record is considered to be normalized floating point numbers.
- 7 The data in the record is DPC card images truncated to include only the initial non-blank characters in the image.

DEFAULT: Mode=1.

NWORDS=n

n is an expression that, when evaluated, provides the number of words to be transmitted in the data transfer. In writing, n must be less than or equal to the record length specified in the OPEN statement. n is a required parameter in all asynchronous input/output statements.

OUTPUT PARAMETERS

NSTATE=ns

ns is an integer variable returned by the system to indicate the status of the current operation. This may be tested at any time after the read or write, and will contain zero if the operation is not yet complete. See the discussion of the status word in chapter 8 of *The NCAR Terabit Memory System*.

$ns > 0$ the count of words transferred is in bits 0-17 and bit 49 is set if an end-of-file was encountered.

$ns < 0$ the specific error code is in bits 48-58.

$ns = 0$ the operation is not complete.

The operation will be forced to completion by either an INQUIRE on this unit or a subsequent READ or WRITE on this unit.

fwa

fwa is a variable name which establishes the beginning of the buffer to be transferred. The length of the buffer is specified as n by the keyword NWORDS. Note that a list may not be given here, and will be flagged by the compiler as an error.

**SYNTAX ERROR
DIAGNOSTICS**

MODE MUST BE DEFINED FOR I/O STATEMENT
UNIT MUST BE DEFINED FOR I/O STATEMENT
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED

**ALTERNATE FORM
FOR ASYNCHRONOUS
(BUFFERED)
STATEMENTS**

Buffering data in and out of the computer while calculations are performed is an effective way to solve problems requiring large amounts of core. One odd- and one even-numbered logical unit should be assigned to the drum for buffered I/O using the drum.

Three characteristics of the buffer I/O statements are given below:

- The mode of transmission (0 or 1) in a buffer control statement must be specified by a parity indicator.
- The buffer control statements are not associated with a list; data transmission is from a first word address (fwa) to a last word address (lwa).
- A buffer control statement initiates data transmission and then returns control to the program, permitting the program to perform other tasks while data transmission is in progress. Before using any of the buffered data, the status of the buffer operation should be checked.

A magnetic tape written in odd parity must be buffered in odd parity; a tape written in even parity must be buffered in even parity. (While the buffer statement works for tapes made with other I/O statements, care should be taken to allow for differences in record format.)

PARAMETERS

Parameters in the forms used in this section are defined as follows:

u	unit number, unsigned integer constant or variable
p	parity; the recording mode interpretations for tapes and drums are: 0 selects even parity † 1 selects odd parity
s	a statement label
fwa	a variable identifier; first word of data block to be transmitted
lwa	a variable identifier; last word of data block to be transmitted. The fwa must be less than or equal to the lwa in the buffer statement. If not, the job will be terminated.

BUFFER IN (u,p)
(fwa,lwa)

The buffer transmits information from unit u in mode p to storage locations fwa sequentially through lwa.

BUFFER OUT (u,p)
(fwa,lwa)

This statement transmits information from storage locations fwa through lwa and writes one record on logical unit u in mode p. The record structure for buffered I/O is different from a binary unformatted WRITE (u). The record size is the length from fwa to lwa with no breakdown for smaller block sizes. It is essentially a long record WRITE depending entirely on fwa and lwa.

† The p parameter set to 0 will specify ASCII if a 9-track tape has been assigned. A p value of 0 is typically used for character data and 1 for numeric data.

**RECORD STRUCTURE
FOR BUFFER OUT
IN BINARY OR
CHARACTER MODE**

BUFFER OUT (KTAPE,1)(DATA(1),DATA(632))

This statement will write a record containing 632 data words.

[word₁] [word₂] [word₃] [word₆₃₂]

It is poor practice to generate excessively long tape records with BUFFER OUT statements.

K=LENGTHF(u)

The length function (type integer) is used to find the number of 60-bit words buffered in during the last I/O operation on unit u. It may be used with buffered I/O statements and is preceded by an IF(UNIT,u) test to ensure that the I/O is completed and there were no errors. LENGTHF(u) will force completion of an operation. However, if this unit is referenced and there were unchecked errors in the last operation, the job will be terminated.

STATUS STATEMENT

IF(UNIT,U)s₁,s₂,s₃,s₄

This statement checks the status of units used for BUFFER IN/BUFFER OUT operations. It will not work in conjunction with other I/O statements.

The transfer points s_i are interpreted as follows:

s₁ IF(UNIT,u)s₁,s₂,s₃,s₄

s₁ not ready

s₂ ready and no previous error

s₃ tape mark sensed on last operation

s₄ parity error sensed on last operation

IF(UNIT,u)s₁,s₂,s₃,s₄ As soon as the IF(UNIT,u)s₁,s₂,s₃,s₄ is encountered in a
(continued) FORTRAN program, the execution waits for the unit to be ready and data transfer to be completed. No branch is ever taken to statement s₁, as control will not return to the FORTRAN program until the unit is ready. The statement label "s₁" should appear as the label on the IF(UNIT,u) statement itself to prevent a MISSING STATEMENT LABEL diagnostic.

When true buffering is to be achieved, the code for calculations to be continued during buffering must appear between the buffer statement and the IF(UNIT,u) statement. These calculations should be long enough to use much of the wait time, or the program will be stoped at the IF(UNIT,u) test until I/O is completed.

Example

```
        :  
        BUFFER IN(3,1)(A(1),A(100))  
        (calculations not involving A)  
        :  
10 IF(UNIT,3)10,11,12,13  
        :
```

Note that blocking and staging by the system invoked when using the *VOLUME or *TLIB card may override the asynchronous code supplied by the user in a FORTRAN program. Consult the Library and Consultation Office if in doubt about the appropriate code to use.

INTERNAL FILES

ENCODE/DECODE STATEMENTS

The ENCODE/DECODE statements transfer information from one area of memory to another area of memory. They are similar to formatted READ/WRITE instructions since information is transferred under FORMAT specifications. No peripheral (i.e., I/O) device is used in ENCODE/DECODE. The array that references the information encoded or decoded is called an internal file.

Parameters

Parameters in the forms used in this section are defined as follows: (Let ICC equal Internal Character Code. The internal character code currently refers to the display code character set of the 7600, using 6-bit characters and 10 characters per word.)

cc	an unsigned integer constant or a simple integer variable (not subscripted) specifying the number of characters in the ICC record. When cc is not a multiple of 10, blanks will be used to fill the last word.
f	a statement number, a variable identifier, or a formal parameter representing the FORMAT statement.
v	a variable identifier or an array identifier which supplies the starting location of the ICC record. v is always in ICC whether ENCODE or DECODE is used. The identifier may have standard or nonstandard subscripts.
iolist	input/output list; may contain an implied DO.

**ENCODE/DECODE
STATEMENTS**
(continued)

Example

C WRITE IOLIST INTO INTERNAL FILE V
 ENCODE (cc,f,v)iolist

ENCODE is similar to a WRITE of the internal file v (in ICC).
 v is generated by the ENCODE from iolist (the list).

Example

C READ FROM INTERNAL FILE V INTO IOLIST
 DECODE (cc,f,v)iolist

DECODE is similar to READ. iolist (the list) is generated by the DECODE from the internal file v (in ICC).

Keep in mind that v is always the ICC array. The direction of the arrows shows whether v (ICC) is being encoded (defined) or decoded (broken down) by the FORMAT specification.

ENCODE(cc,f,v)[iolist] ENCODE is a data transfer from the list variables to some other array called the internal file. The transfer is done using a FORMAT specification so that the destination array contains the encoded list variables in internal character code.

ENCODE (10,100,IA)A
 100 FORMAT (*bA=*F7.2)

The number A is sent to IA using the FORMAT statement 100. IA is the internal file. The character count, 10, is the number of characters specified in the FORMAT statement including the alphanumeric information.

If A is equal to 1.0, then after the ENCODE

A is 17204000000000000B)
 (1.0 in floating point)

IA is 5501545555534573333B
 (A = 1 . 0 0)

ENCODE (cc,f,v)iolist is similar to PRINT f,iolist except that the destination is an internal file. The information from iolist is converted according to FORMAT f and stored, character by character, starting at the left end of the array specified by v, the internal file.

cc is the character count per record; "record" in this case is like line length in a normal PRINT statement. In the associated FORMAT statement, f, an end of "record" is indicated by a slash or by reaching the last right parenthesis with a repeat factor.

The FORMAT statement indicates how many characters are to be used in the conversion of each list variable, the type of conversion for that variable, and the record length of the statement.

RULES

- cc must be less than or equal to 150.
- The maximum number of characters specified in the longest record in the FORMAT statement f must be less than or equal to cc.
- If cc is not a multiple of 10, the encoded record will be blank filled to the next larger multiple of 10; however, only cc characters will actually undergo conversion.
- v must be dimensioned if the total number of characters encoded is greater than 10.

**ENCODE/DECODE
STATEMENTS**
(continued)

Example 1

```
DIMENSION L(2)
I=10HABCDEFGHIJ
J=10H1234567890
ENCODE (5,100,L)I,J
100 FORMAT(A5/A7)
```

L(1) =	A	B	C	D	E	b	b	b	b
L(2) =	1	2	3	4	5	b	b	b	b

The slash in the format indicates that two records are specified. Since the character count per record has a maximum of five, only five characters of J were converted, even though the FORMAT called for seven. Since cc is 5, each record was blank filled to 10, the next larger number which is a multiple of 10.

Example 2

```
DIMENSION L(4)
I = 10HABCDEFGHIJ
J = 10H1234567890
ENCODE (18,100,L)I,J
100 FORMAT(A5/A7)
```

L(1) =	A	B	C	D	E	b	b	b	b
L(2) =	b	b	b	b	b	b	b	b	b
L(3) =	1	2	3	4	5	6	7	b	b
L(4) =	b	b	b	b	b	b	b	b	b

Only the number of characters called for in the FORMAT (five and seven for each record) are actually converted. The record length (18) is not a multiple of 10, so blank fill is provided to the next larger multiple of 10 (20) for each record. Thus, L(2) and L(4) are blank words to complete the character count of 18 for each record.

Example 3

```
DIMENSION L(20),A(10),B(10)
DO 1 I=1,10
  A(I) = I
1  B(I) = I+20
  ENCODE (20,100,L)(A(I),B(I),I=1,10)
100 FORMAT(2HA=,F8.2,2HB=,F7.1)
```

L(1) =

A	=	b	b	b	b	1	.	0	0
---	---	---	---	---	---	---	---	---	---

L(2) =

B	=	b	b	b	2	1	.	0	b
---	---	---	---	---	---	---	---	---	---

⋮
L(19) =

A	=	b	b	b	1	0	.	0	0
---	---	---	---	---	---	---	---	---	---

L(20) =

B	=	b	b	b	3	0	.	0	b
---	---	---	---	---	---	---	---	---	---

There are 20 characters per record and 10 records to fill L from L(1) to L(20), two words per record. The Hollerith in the FORMAT is included in the character count.

**ENCODE/DECODE
STATEMENTS**
(continued)

ENCODE may be used to calculate a field definition in a FORMAT specification at object time. Assume that in the statement FORMAT(2A8,Im) the programmer wishes to specify m at some point in the program, subject to the restriction $2 \leq m \leq 9$.

Example 4

```
      IF(M.LT.10.AND.M.GT.1)1,2
1  ENCODE (8,100,IFMT)M
100 FORMAT (6H(2A8,I,I1,1H))
      :
      PRINT IFMT, IA,NA,J
```

M is tested to ensure it is within limits. If not, control goes to statement 2 which could be an error routine. If M is within limits, ENCODE first packs the six Hollerith characters (2A8,I, then, if M = 5, M is encoded to include 5 so that it is now (2A8,I5. Then the right parenthesis is added by using 1H). IFMT contains (2A8,I5) after the encode. IFMT must be type integer.

The PRINT statement will print IA and NA under specification A8, and the quantity J under specification I2, or I3, or ..., or I9 according to the value of M.

Example 5

```
DIMENSION LTIT(4)
      ENCODE (38,100,LTIT)IDATE,TE
100 FORMAT (*APRIL,*I2,*,1969,TEMPERATURE AT*F3.0,*DEGREES*)
```

A title for plotting using a graphics subroutine might be encoded inserting the appropriate day and temperature for each graph. Encoding a new IDATE and TE for each graph provides a variable title.

DECODE(cc,f,v)[iolist] DECODE is a data transfer from an ICC array to the specified list variables. The transfer is done according to the FORMAT statement; the ICC array is interpreted, and list variables are stored from the ICC array as binary numbers.

```
      DECODE(10,100,IA)A
100 FORMAT(3XF7.2)
```

The information in IA, the internal file, is internal character code.

```
5501545555534573333
b A = b b b l . 0 0
```

3X in FORMAT 100 skips the first three characters, bA-, then F7.2 is used to store a floating point l in A. cc, the character count for the format, is 10.

DECODE (cc,f,v)iolist is similar to READ f,iolist. The ICC information contained in cc consecutive characters starting at v is converted according to FORMAT f and stored in the list variables iolist.

The number of characters decoded per record is dependent on the list iolist and the format f.

Rules

- cc refers to the number of characters to be decoded per record. The record is the smallest number of consecutive computer words that can contain cc characters. cc must be less than or equal to 150.
- If the FORMAT specification is exhausted (i.e., coming to the last right parenthesis) before cc characters have been decoded or the list iolist is satisfied, a new record is started at the beginning of the FORMAT statement.
- A slash in the FORMAT specification initiates a new record.
- If the list iolist is satisfied before reaching the end of the FORMAT specification or before cc characters have been decoded, decoding will stop.
- An execution diagnostic will occur if DECODE attempts to process an illegal ICC code.
- If the list iolist contains a dimensioned variable with no specific indexing treatment, it is assumed to mean the whole array, as specified in the DIMENSION statement.

Examples

The following data applies to each of the four examples below:

Given: DIMENSION L(5),A(6)

L(1) =

1	2	3	4	5	6	7	8	9	0
---	---	---	---	---	---	---	---	---	---

L(2) =

1	1	.	4	A	B	C	7	2	.
---	---	---	---	---	---	---	---	---	---

L(3) =

0	9	8	7	6	5	4	3	2	1
---	---	---	---	---	---	---	---	---	---

L(4) =

2	3	.	7	C	X	Y	b	1	1
---	---	---	---	---	---	---	---	---	---

L(5) =

5	2	6	3	7	1	8	0	9	4
---	---	---	---	---	---	---	---	---	---

Example 1

C READ FROM INTERNAL FILE L INTO A
100 FORMAT(2F5.0,F4.1)
 DECODE (14,100,L)(A(I),I=1,6)

produces A(1) = 12345.
 A(2) = 67890.
 A(3) = 11.4
 A(4) = 9876.
 A(5) = 54321.
 A(6) = 23.7

Fourteen characters and three words are decoded per record.
Since there are six words in the array A, the format is
scanned twice.

**ENCODE/DECODE
STATEMENTS**
(continued)

Example 2

```
C      READ FROM INTERNAL FILE L INTO IOLIST
101 FORMAT(I2/F4.1)
      DECODE (10,101,L)(I,A,J,B,K)
```

produces I = 12
 A = 11.4
 J = 9
 B = 23.7
 K = 52

DECODE (4,101,L)(I,A,J,B,K) would produce identical results. Since five words are specified in the list, the scan repeats until the list is satisfied. Decoding stops after using I2 for K in the third repeat of the scan.

Example 3

```
C      READ FROM INTERNAL FILE L INTO K
102 FORMAT(3I2,F5.0,F3.1,A3)
      DECODE (17,102,L)K
```

produces K = 12

DECODE stops when the list is exhausted. Only one word is specified, K, so the scan stops after the first I2 in the list.

Example 4

```
C      READ FROM INTERNAL FILE L INTO IOLIST
102 FORMAT(3I2,F5.0,F3.1,A3)
      DECODE (17,102,L)I,J,K,X,Y,NAM
```

produces I = 12
 J = 34
 K = 56
 X = 78901.
 Y = 1.4

NAM = [A B C b b b b b b b]

The format is scanned completely once, since there are six variables specified in the list.

Generalized Form

Transfers from an iolist in memory to or from an internal file is also specified in FORTRAN 77. This is not yet implemented on the NCAR 7600 compiler, since the internal file requires a CHARACTER data type name in the generalized form.

DATASET SUPPORT STATEMENTS

The dataset support statements are OPEN, CLOSE, SETPASS and INQUIRE. These are used to perform various operations on files and to check the status of file characteristics. These statements, which are FORTRAN 77 constructs, are used in addition to the READ and WRITE statements.

THE OPEN STATEMENT

The OPEN statement is used to connect files to a unit. The unit is associated with a volume in the job control language. When an input/output statement appears referencing the unit associated with a volume, data are transferred to or from the dataset according to the attributes in the OPEN statement and the keyword specifiers in the I/O statement. A subsequent OPEN statement on the same unit for the same file may change some of the attributes and access methods described in the keyword parameters in the list for a previously connected dataset in a volume. Temporarily, there is a restriction on parallelism in the use of the OPEN statement. Two datasets on the same volume may be connected consecutively, but may not be connected at the same time. An implicit CLOSE is assumed between successive OPENS on the same unit.

Square brackets denote optional items; constant parameters of type character must be enclosed in apostrophes or preceded by nH.

FORM

OPEN (ident,keylist)

```
OPEN ([UNIT=]u[,FILE=dsi][,FILESEQ=n][,ERR=s][,FORM=fm]
      [,STATUS=sta][,RECL=rlen][,MAXREC=maxr][,ACCESS=acc]
      [,GEN=gen][,EXPDT=exp][,PASS=pass][,IOSTAT=ios]
      [,BLANK=blnk])
```

IDENT PARAMETERS

ident specifies the unit identifier, the dataset name, or the file sequence number. In all cases the unit must be specified. It may appear in any one of the following keyword identifier combinations:

- [UNIT=]u
- [UNIT=]u,FILE=dsi
- [UNIT=]u,FILESEQ=n
- [UNIT=]u,FILE=dsi,FILESEQ=n

u
UNIT=u

u is an expression specifying the unit. There are two forms for specifying the unit: UNIT=u or simply u. u is the unit number specified on the *VOLUME card. The parameter u must appear in the list and it must appear first in an OPEN statement if the UNIT= is omitted.

Unit numbers may range from 1 to 3552₁₀. Unit numbers greater than 3552 may be handled if the user understands the systems implications of the number opened. See a systems programmer for units greater than 3552.

FILE=dsi

dsi is a character constant or variable of 17 or fewer characters, beginning with a letter or number and dimensioned for two words. It is left-justified with no embedded blanks or commas, and specifies the dataset name for a new file. Unused character positions in the two words used for dsi should be blank-filled. If STATUS='OLD', dsi serves to identify an already existing file. Note that this parameter is optional. If the file status is NEW and this parameter is omitted, the system default label is used. If the file status is OLD and this parameter is omitted, positioning is by file sequence number.

DEFAULT: FILE=17HNCAR SYSTEMND1nnnn, where nnnn is the FILESEQ number.

IDENT PARAMETERS*(continued)***FILESEQ=n**

n is an integer constant or variable that specifies the sequence number of the dataset to be opened. It represents the current position of a file on a volume. The value of n may range from 1 to 600. Note that this parameter is optional. If both FILE=dsi and FILESEQ=n are omitted, the file is positioned as though FILESEQ=1.

**ERROR CONDITION
STATEMENT LABEL****ERR=s**

s is a statement label to which the program branches when an error condition occurs during the execution of the OPEN statement.

DEFAULT: The program terminates on an error condition.

KEYLIST PARAMETERS**FORM=fm**

fm is an alphanumeric constant or variable (dimensioned for 2) specifying whether the data is FORMATTED or UNFORMATTED.

DEFAULT: FORM='UNFORMATTED'

STATUS=sta

sta is an alphanumeric constant or a variable specifying whether the dataset is NEW, OLD, SCRATCH, or UNKNOWN. Datasets that are NEW have not been created prior to execution of the OPEN statement. At the end of each run, SCRATCH files are deleted.

DEFAULT: STATUS='UNKNOWN'

RECL=rlen rlen is an expression that establishes the maximum record length of all records in the dataset. rlen may not have a negative value. If given for an OLD file with a defined record length, the length must match. The value is undefined if not given for a NEW sequential file. For a NEW file to be used for direct access, length is a required parameter.

MAXREC=maxr maxr is an expression specifying the maximum number of records that may appear in a dataset. An error occurs if an attempt is made to reference a record number larger than maxr. This parameter may only appear with NEW files.

DEFAULT: MAXREC=1000 for direct access on a NEW file, and
MAXREC=undefined for sequential access.

ACCESS=acc acc is an alphanumeric constant or variable whose value is DIRECT or SEQUENTIAL. It defines the method for accessing records within the dataset.

DEFAULT: ACCESS='SEQUENTIAL'

EXPDT=exp exp is a character constant or variable. It contains the julian expiration date, consisting of two numeric characters for the year, followed by three numeric characters for the day within the year. Files will be deleted on or after the expiration date and data will not be recoverable.

DEFAULT: EXPDT=one year later than the creation date

KEYLIST PARAMETERS*(continued)* **\dagger PASS=pass**

pass is a security parameter supplied by the user which allows four kinds of access to a dataset. To establish passwords that are valid here, assign them using the FORTRAN SETPASS instruction. The four levels are:

owner	unlimited access
read/write	user may read and write the dataset, but is unable to change the password using the SETPASS instruction
read	access to the dataset is for reading only
write	access to the dataset is for writing only

Refer to the section on SETPASS for information on creating and maintaining passwords.

 \dagger BLANK=blnk

blnk is an alphanumeric constant or variable that is NULL or ZERO. NULL causes all blanks in an input field to be removed. ZERO causes all blanks to be zeros.

DEFAULT: In the current implementation, this parameter is always set to ZERO. Omit it.

RETURN PARAMETER**IOSTAT=ios**

ios specifies whether an error occurred during the execution of the OPEN statement.

ios=0 successful open
 ios \neq 0 error condition

\dagger This feature was not yet implemented at publication time.

**SYNTAX ERROR
DIAGNOSTICS**

ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE
OPEN STATEMENT-ERROR IN ACCESS. VALUE MUST BE DIRECT OR
SEQUENTIAL
OPEN STATEMENT-ERROR IN ERR. MUST BE STATEMENT LABEL
OPEN STATEMENT-ERROR IN EXPDT. MUST BE ALPHANUMERIC DATE
OPEN STATEMENT-ERROR IN FILE. MUST BE 17 CHARACTERS OR
LESS ALPHANUMERIC CONSTANT OR VARIABLE
OPEN STATEMENT-ERROR IN FILESEQ
OPEN STATEMENT-ERROR IN STATUS
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED

THE CLOSE STATEMENT

The CLOSE statement is used to disconnect a dataset from a given unit. Executing a CLOSE for a dataset that does not exist causes no action. An implicit CLOSE is assumed between two successive OPENS on the same unit.

FORM

CLOSE ([UNIT=]u[,STATUS=sta][,ERR=s][,IOSTAT=ios])

INPUT PARAMETERS

u
UNIT=u

u is an expression specifying the unit. There are two forms for specifying the unit: UNIT=u or simply u. u is the unit number specified on the *VOLUME card. The parameter u must appear in the list and it must appear first if the UNIT= is omitted.

Unit numbers may range from 1 to 3552₁₀. Unit numbers greater than 3552 may be handled if the user understands the systems implications of the number opened. See a systems programmer for units greater than 3552.

STATUS=sta

sta is an alphanumeric constant or variable assigned a value of KEEP or DELETE. If KEEP is specified, the dataset continues to exist following the CLOSE statement. Using DELETE as the parameter value, the dataset no longer exists for the executable program following the CLOSE statement. At the end of execution of a job, all units connected to named files are closed implicitly with a status of KEEP, except SCRATCH files, which are deleted.

DEFAULT: STATUS='KEEP', unless the status in the OPEN is SCRATCH, where the dataset is deleted when closed.

ERROR CONDITION STATEMENT LABEL

ERR=s

s is a statement label to which the program branches when an error condition occurs during execution of the CLOSE statement.

DEFAULT: If ERR= is omitted, the program terminates on an error condition.

RETURN PARAMETER

IOSTAT=ios

ios specifies whether an error occurred during the execution of the CLOSE statement.

ios=0 successful close
ios#0 error condition

SYNTAX ERROR DIAGNOSTICS

CLOSE STATEMENT--ERROR IN ERR. MUST BE STATEMENT LABEL
CLOSE STATEMENT--ERROR IN STATUS. MUST BE EITHER KEEP OR
DELETE

ERR MUST REFER TO STATEMENT LABEL

ERR STATEMENT LABEL OUT OF RANGE

UNIT NOT DEFINED

UNRECOGNIZABLE I/O STATEMENT

nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE

nTH KEYWORD DOUBLY DEFINED

THE INQUIRE STATEMENT

The attributes or properties of a dataset or a connection are requested by the programmer using the INQUIRE statement. Values of parameters defining these attributes used in a keyword list are returned to the program after the INQUIRE statement is executed. The form of the INQUIRE statement is:

FORM

```
INQUIRE (ident,keylist)
INQUIRE ([UNIT=]u[,FILE=dsi][,FILESEQ=n][,ERR=s][,IOSTAT=ios]
[,EXIST=ex][,OPENED=od][,NAMED=nmd][,NUMBER=un][,NAME=fn]
[,RECL=rlen][,MAXREC=maxr][,ACCESS=acc][,SEQUENTIAL=seq]
[,DIRECT=dir][,FORM=fm][,FORMATTED=fmt][,UNFORMATTED=unf]
[,NSTATE=ns][,MODE=md][,NEXTREC=nr][,SEQFIL=n][,GEN=gen]
[,EXPDT=exp][,VSN=name][,NUMREC=nr][,CRECL=c1][,GENTIME=gnt]
[,BLANK=blk][,CRDT=crdt])
```

IDENT PARAMETERS

ident specifies the unit identifier, the dataset name, or the file sequence number. In all cases the unit must be specified. It may appear in any one of the following keyword identifier combinations:

- [UNIT=]u
- [UNIT=]u,FILE=dsi
- [UNIT=]u,FILESEQ=n
- [UNIT=]u,FILE=dsi,FILESEQ=n

u
UNIT=u

u is an expression specifying the unit. There are two forms for specifying the unit: UNIT=u or simply u. u is the unit number specified on the *VOLUME card. The parameter u must appear in the list and it must appear first if the UNIT= is omitted.

FILE=dsi

dsi is a character constant or variable beginning with a letter or number and dimensioned for two words. It is left-justified with no embedded blanks or commas, and specifies the dataset name with 17 or fewer characters. Unused character positions in the two words reserved for dsi should be blank-filled.

FILESEQ=n

n is an expression designating the sequence number of the file or dataset. This identifier takes precedence over the FILE=dsi parameter in the following sense: if both FILE= and FILESEQ= are specified and do not match, the dataset with the specified FILESEQ number is used.

ERROR CONDITION STATEMENT LABEL

ERR=s

s is a statement label to which the program branches when an error condition occurs during the processing of the INQUIRE statement.

DEFAULT: If ERR= is omitted, the program terminates on an error condition.

KEYLIST PARAMETERS

keylist is a list of one or more of the following keyword forms. Values of dataset properties are returned in the variable names used in the INQUIRE statement. keylist items which provide properties of a file or a connection apply to the file located by a combination of ident parameters. If neither FILE= nor FILESEQ= are given and the file is open, the keylist items apply to the first file. FILESEQ takes precedence over the FILE= as a keylist identifier. If both FILE= and FILESEQ= are given and they do not match on the file dsi, EXIST= will be returned FALSE and other properties will be returned for the file in that file sequence position.

IOSTAT=ios

ios is an integer variable returned with the error condition that occurred during execution of the INQUIRE statement.

ios=0 successful inquiry

ios#0 error condition indicated in the right 12 bits of the word

Error codes are defined under STATUS in chapter 8 of *The NCAR Terabit Memory System*.

SUGGESTED FORMAT: 04

EXIST=ex

ex is a logical variable. It is returned with the value .TRUE. if there is a dataset with the specified name dsi on the connected volume. Otherwise, it is .FALSE.. If FILESEQ= is given as an identifier, only the file in that sequential position is checked.

SUGGESTED FORMAT: L2

OPENED=od

od is a logical variable which is returned with .TRUE. if the file dsi has been assigned or connected to a unit. If the dataset is not connected, od is returned with the value .FALSE..

SUGGESTED FORMAT: L2

NAMED=nmd

nmd is a logical variable returned with the value .TRUE. if the dataset has a user-supplied name, otherwise the value .FALSE. is returned.

SUGGESTED FORMAT: L2

NUMBER=un

un is an integer variable returned with the unit opened to the dataset name, dsi, if any. If no unit is connected to the file, un is undefined.

SUGGESTED FORMAT: I5

NAME=fn

fn is an alphanumeric array dimensioned for two words, and returned with the name for that file. fn is 17 characters in length and is left-justified with three characters of blank fill. If EXIST=.FALSE., fn will be different from dsi.
SUGGESTED FORMAT: 2A10

RECL=rlen

rlen is an integer variable returned with the maximum record length of a file that has been specified in an OPEN statement. If rlen has not been defined by the user in an OPEN statement, a system default of 0 will be returned.

SUGGESTED FORMAT: I7

MAXREC=maxr

maxr is an integer variable returned with the value of the maximum record number allowed in a file, if defined.

SUGGESTED FORMAT: I9

ACCESS=acc

acc is a variable returned with the alphanumeric value DIRECT or SEQUENTIAL, which applies to the current connection of that file. If there is no connection, acc is undefined.

SUGGESTED FORMAT: A10

KEYLIST PARAMETERS*(continued)***SEQUENTIAL=seq**

seq is an alphanumeric variable returned with:

- YES sequential access is allowed on this dataset when opened for sequential access
NO (not used at publication time)
UNKNOWN (not used at publication time)

SUGGESTED FORMAT: A10

DIRECT=dir

dir is an alphanumeric variable returned with:

- YES direct access is allowed on this dataset when opened for direct access
NO (not used at publication time)
UNKNOWN (not used at publication time)

SUGGESTED FORMAT: A10

FORM=fm

fm is an alphanumeric variable returned with a value of FORMATTED OR UNFORM for the current connection. If there is no connection, fm is undefined.

SUGGESTED FORMAT: A10

FORMATTED=fmt

fmt is an alphanumeric variable that specifies if the file may be connected for formatted input/output:

- YES allowed on all files
NO (not used)
UNKNOWN (not used)

SUGGESTED FORMAT: A10

UNFORMATTED=unf

unf is an alphanumeric variable that specifies if the file may be connected for unformatted input/output:

- YES allowed on all files
- NO (not used)
- UNKNOWN (not used)

SUGGESTED FORMAT: A10

NSTATE=ns

ns is an integer variable interpreted as a 60-bit octal number, returned with the contents of the system status cell for the previous input/output statement executed for the unit assigned to that dataset. See chapter 8 of *The NCAR Terabit Memory System* for the contents of the status cell.

SUGGESTED FORMAT: 020

MODE=md

md is a variable returned with the mode of the data referenced in the previous input statement executed on that unit:

- md 0 The data in the record is considered to be normal character data internal to the machine.
- 1 The data in the record is considered to be binary, bit-serial.
- 2 The data in the record is considered to be Binary Coded Decimal (BCD).
- 3 The data in the record is considered to be American Standard Code for Information Interchange (ASCII).
- 4 The data in the record is considered to be Expanded Binary Coded Decimal Interchange Code (EBCDIC).
- 5 The data in the record is considered to be in the binary integer format.
- 6 The data in the record is considered to be normalized floating point numbers.
- 7 The data in the record is DPC card images truncated to include only the initial non-blank characters in the image.

SUGGESTED FORMAT: I2

KEYLIST PARAMETERS*(continued)*

NEXTREC=nr

nr is an integer variable. The value returned in nr is the number of the record previously read or written on that unit plus one. If no reads or writes have been executed since the last REWIND or OPEN, NEXTREC=1.

SUGGESTED FORMAT: I9

SEQFIL=n

n is an integer variable returned with the current file sequence number.

SUGGESTED FORMAT: I4

GEN=gen

gen is an integer variable returned with the generation number. If the unit or file is not connected, the variable is undefined.

SUGGESTED FORMAT: I4

EXPDT=exp

exp is an alphanumeric variable defining the julian expiration date, consisting of two numeric characters for the year, followed by three numeric characters for the day within the year. If the unit or file is not connected, the variable is undefined. The default value is one year later than the creation date.

SUGGESTED FORMAT: A5

VSN=name

name is a variable returned with the Volume Serial Number of the volume currently connected to the unit. It consists of 6 alphanumeric characters.

SUGGESTED FORMAT: A6

NUMREC=nr

nr is an integer variable returned with the number of records in the file being referenced.

SUGGESTED FORMAT: I9

- CRECL=cl cl is a variable returned with the total length of the record read in the previous input/output statement on that unit or file without regard to the number of words specified in the READ statement.
SUGGESTED FORMAT: I8
- GENTIME=gnt GENTIME returns the time the file was last read or written in the following format:
- | <i>Bit</i> | <i>Position</i> | <i>Description</i> |
|------------|-----------------|---------------------------------------|
| 59 - 45 | | Last time the file was read |
| 44 - 36 | | Last day of year the file was read |
| 35 - 30 | | Last year the file was read, 0 = 1976 |
| 29 - 15 | | Last time the file was written |
| 14 - 6 | | Last day of year the file was written |
| 5 - 0 | | Last year the file was written |
- SUGGESTED FORMAT: O20
- BLANK=blnk blnk is a variable returned with 'ZERO'. The 'NULL' value is not implemented at this time. BLANK='ZERO' interprets blanks as zeros in reading from the input file.
SUGGESTED FORMAT: A4
- CRDT=crdt crdt is an alphanumeric variable defining the julian creation date, consisting of two numeric characters for the year, followed by three numeric characters for the day within the year. If the unit or file is not connected, the variable is undefined.
SUGGESTED FORMAT: A5

**SYNTAX ERROR
DIAGNOSTICS**

ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE
INQUIRE STATEMENT-ERROR IN ACCESS. VALUE MUST BE DIRECT,
OR SEQUENTIAL
INQUIRE STATEMENT-ERROR IN BLANK
INQUIRE STATEMENT-ERROR IN CRDT
INQUIRE STATEMENT-ERROR IN ERR. MUST BE A STATEMENT LABEL,
OR PROGRAM TERMINATES
INQUIRE STATEMENT-ERROR IN EXPDT
INQUIRE STATEMENT-ERROR IN EXIST. MUST BE EITHER .TRUE. OR
.FALSE.
INQUIRE STATEMENT-ERROR IN FILE
INQUIRE STATEMENT-ERROR IN FILESEQ
INQUIRE STATEMENT-ERROR IN GEN
INQUIRE STATEMENT-ERROR IN GENTIME
INQUIRE STATEMENT-ERROR IN MAXREC
INQUIRE STATEMENT-ERROR IN MODE
INQUIRE STATEMENT-ERROR IN NAME
INQUIRE STATEMENT-ERROR IN NEXTREC
INQUIRE STATEMENT-ERROR IN NSTATE
INQUIRE STATEMENT-ERROR IN NUMBER
INQUIRE STATEMENT-ERROR IN OPENED
INQUIRE STATEMENT-ERROR IN RECL. MUST BE AN INTEGER VARIABLE
INQUIRE STATEMENT-ERROR IN SEQFIL
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED

**THE SETPASS
STATEMENT**

Passwords may be assigned to datasets within a volume using the SETPASS instruction. A dataset may have up to four levels of security using passwords. In order to add or change a password, the dataset must be currently opened; that is, connected to a unit. If SETPASS has never been used to assign a password, the dataset has unlimited access to all users. The four levels of security that may be established are:

owner	unlimited access
read/write	access for reading or writing the dataset, but no access for changing a password
read	access for reading only
write	access for writing only

FORM

```
SETPASS ([UNIT=]u,PASS=pass[,RWPASS=rwpass][,RPASS=rpass]
[,WPASS=wpass][,CPASS=cpass])
```

PARAMETERS

u
UNIT=u

u is an expression specifying the unit assigned to the dataset.

PASS=pass

pass is an 8-character alphanumeric constant or variable specifying the "owner's" password associated with the dataset identified by the parameter, u. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If the password has not yet been defined, the initial specification establishes pass as the owner's password. The owner password must be specified in assigning values to all other levels of passwords using the SETPASS instruction.

PARAMETERS*(continued)*

RWPASS=rpass

rpass is an 8-character alphanumeric constant or variable specifying access only for reading and writing the dataset connected to the unit, u. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If rpass is not defined, no access is allowed with the keyword RWPASS.

RPASS=rpass

rpass is an 8-character alphanumeric constant or variable specifying access for reading the dataset connected to the unit, u. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If rpass is not defined, the dataset may be read without the use of a password.

WPASS=wpass

wpass is an 8-character alphanumeric constant or variable specifying access to the dataset connected to the unit, u, only for writing. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If wpass is not defined, no access is allowed with the keyword WPASS.

CPASS=cpass

cpass is an 8-character alphanumeric constant or variable, specifying that a new owner password, cpass, be substituted for the current owner password, pass. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. Both keyword parameters, pass and cpass, must be entered, if the owner password is to be changed. CPASS=0 is used to remove *all* passwords.

Examples

SETPASS (9,PASS=8H12345678,RPASS=8H56781234)

If the owner password for the dataset assigned to unit 9 is 12345678, set or change the read only password to 56781234. wpass and rpass remain unchanged.

SETPASS (9,PASS=8H12345678,CPASS=8H87654321)

pass is changed to 87654321. rpass, rpass and wpass remain the same.

SETPASS (9,PASS=8H12345678,WPASS=8H4444444A,CPASS=8H9A998B88)

pass is changed to 9A998B88, wpass to 4444444A; rpass and rpass remain unchanged.

SETPASS (9,PASS=8H12345678,CPASS=0)

All passwords are removed.

SYNTAX ERROR DIAGNOSTICS

SETPASS-CPASS KEYWORD IN ERROR

SETPASS-PASS KEYWORD IN ERROR

SETPASS-RPASS KEYWORD IN ERROR

SETPASS-RWPASS KEYWORD IN ERROR

SETPASS-WPASS KEYWORD IN ERROR

UNIT NOT DEFINED

UNRECOGNIZABLE I/O STATEMENT

nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE

nTH KEYWORD DOUBLY DEFINED

**FILE POSITIONING
WITH SEQUENTIAL
ACCESS AND
SYNCHRONOUS
INPUT/OUTPUT**

File positioning statements are used only with sequential access to volumes. This access method is characterized as positional and order-dependent. The unit specified is assumed to contain a sequence of ordered files and records constituting the data. These files may be user-named or have names assigned by the system when used with the NCAR tape staging feature. Manipulation is basically positional. The following statements may be used to position sequential files.

```
BACKSPACE u
BACKSPACE ([UNIT=]u[,IOSTAT=ios][,ERR=s])
ENDFILE u
ENDFILE ([UNIT=]u[,IOSTAT=ios][,ERR=s])
REWIND u
REWIND ([UNIT=]u[,IOSTAT=ios][,ERR=s])
SKIPFILE u
SKIPFILE (u[,COUNT=m][,ERR=s])
BACKFILE u
BACKFILE (u[,COUNT=m][,ERR=s])
IF (EOF,u) s1,s2
```

The end-of-file record concept is a positional one and appropriately applies *only* to sequential files, where end-of-file records are encountered during a read. Changes to these statements are not anticipated in the near future. Some, that are non-standard conforming, may be removed from the compiler as users turn to the new syntax. File positioning statements may not be used on a file currently open.

In order to maintain the possibility of future growth in the area of data structures, and to conform to the standard, the file positioning statements have been disallowed in the random access FORTRAN statements implemented. An example is the REWIND u, which positions a sequential file or files to the volume load point. An OPEN statement will provide the user access to the beginning (first record) of a named dataset. A BACKSPACE will also cross dataset boundaries just as it has in the past. This is an extension to the standard. Notice that the END=s parameter may only be used with sequential files. For direct access, an ERR=s specifier would note that a user was attempting to cross a named dataset boundary.

**POSITIONING
STATEMENTS**

Parameters in the forms used in this section are defined as follows (square brackets indicate optional items):

u	unit number, unsigned integer constant or variable
s	a statement label
m	an integer constant greater than zero
ios	an integer variable

REWIND u or
 REWIND ([UNIT=]u
 [,IOSTAT=ios]
 [,ERR=s])

The magnetic tape on unit u is rewound to the load point. If the tape is already rewound, the statement acts as a do-nothing statement. Tapes must be rewound before the first read.

Keywords in the form are:

- UNIT=u u is an expression specifying the unit. There are two forms for specifying the unit, UNIT=u or simply u. u is the unit number specified on the *VOLUME card.
- IOSTAT=ios ios is an integer variable returned with the error condition that occurred during execution of the REWIND statement. If ios=0, the rewind was successful. If ios \neq 0, an error condition is indicated in the right 12 bits of the word.
- ERR=s Program control transfers to the FORTRAN statement numbered s if an error occurs.

BACKSPACE u or
BACKSPACE ([UNIT=]u
[,IOSTAT=ios]
[,ERR=s])

The magnetic tape on unit u is backspaced one logical record.
If a tape is at load point (rewound), this statement acts as
a do-nothing statement.

Keywords in the form are:

UNIT=u u is an expression specifying the unit.
There are two forms for specifying the
unit, UNIT=u or simply u. u is the unit
number specified on the *VOLUME card.

IOSTAT=ios ios is an integer variable returned with
the error condition that occurred during
execution of the BACKSPACE statement. If
ios=0, the backspace was successful. If
ios≠0, the error condition is indicated in
the right 12 bits of the word.

ERR=s Program control transfers to the FORTRAN
statement numbered s if an error occurs.

ENDFILE u or
ENDFILE ([UNIT=]u
[,IOSTAT=ios]
[,ERR=s])

This statement writes a tape mark on magnetic tape unit u.

Keywords in the form are:

UNIT=u u is an expression specifying the unit.
There are two forms for specifying the
unit, UNIT=u or simply u. u is the unit
number specified on the *VOLUME card.

IOSTAT=ios ios is an integer variable returned with
the error condition that occurred during
execution of the ENDFILE statement. If
ios=0, the endfile was successful. If
ios≠0, the error condition is indicated in
the right 12 bits of the word.

ERR=s Program control transfers to the FORTRAN
statement numbered s if an error occurs.

BACKFILE u or
 BACKFILE (u
 [,COUNT=m]
 [,ERR=s])

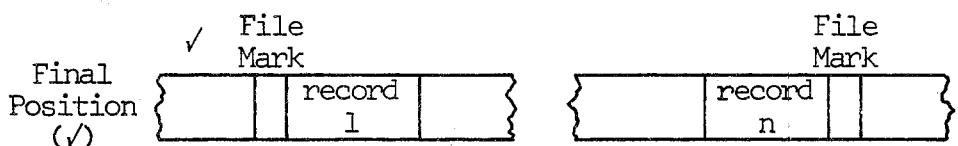
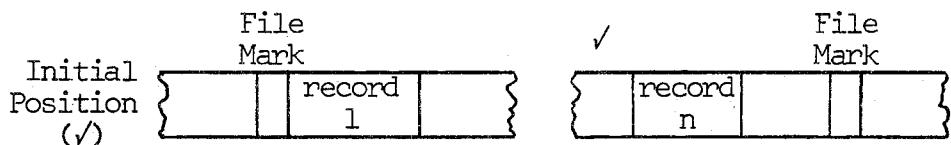
The magnetic tape on unit u is repositioned backward one or more files.

Keywords in the form are:

- u u is an expression specifying the unit, and must match the unit number on the VOLUME card.
- COUNT=m m is the number of files to be backspaced. If there is no preceding file, the unit is positioned at its initial point. If n=0, no repositioning occurs.
- ERR=s Program control transfers to the FORTRAN statement numbered s if an error occurs.

Example 1

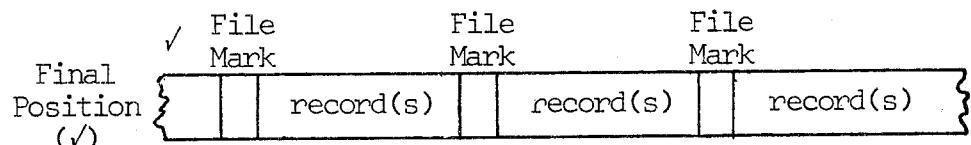
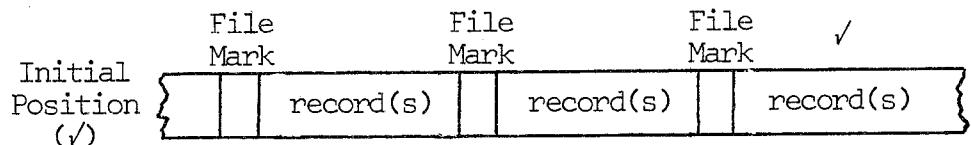
BACKFILE 9



Example 2

BACKFILE (9,COUNT=3,ERR=201)

BACKFILE (9,COUNT=3)



SKIPFILE u or
SKIPFILE (u
 $\text{[,COUNT=}m\text{]}$
 $\text{[,ERR=}s\text{]})$

The magnetic tape on unit u is repositioned forward one or more files.

Keywords in the form are:

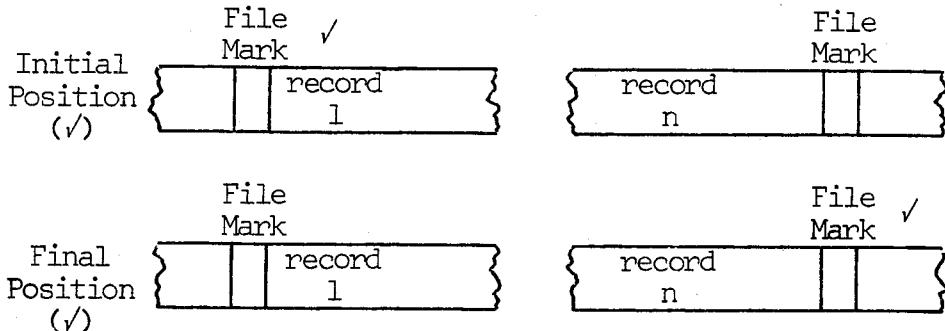
u u is an expression specifying the unit, and must match the unit number on the VOLUME card.

COUNT=m m is the number of files to be forward spaced. Do not use SKIPFILE if there are no more file marks on the tape. Proceeding to the end of tape may use large amounts of peripheral processor time.

ERR=s Program control transfers to the FORTRAN statement numbered s if an error occurs.

Example 1

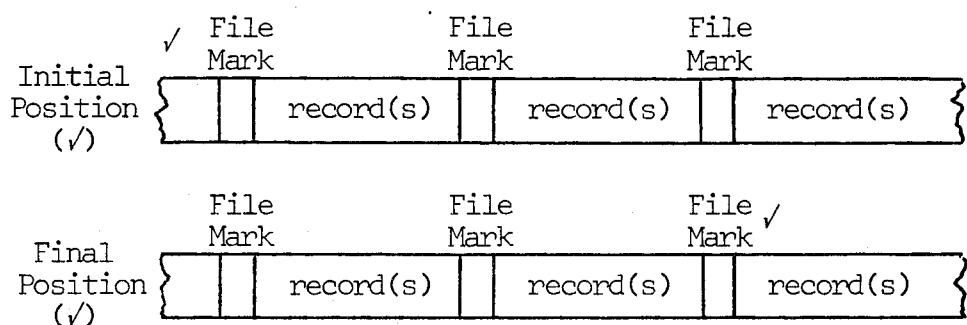
SKIPFILE 9



Example 2

SKIPFILE (9,COUNT=3,ERR=201)

SKIPFILE (9,COUNT=3)



IF(EOF,u)s₁,s₂

A check of the previous read operation is made to determine if an end-of-file has been encountered on unit u. If it has, control is transferred to statement s₁; if not, control is transferred to statement s₂. It may not be used to check an EOF on units 5 or 6, the standard reader and printer units. (Note that BUFFER IN can be used with even parity to read from the card reader. In this case the IF(UNIT,u) test may be used to check for an EOF.)

**EXAMPLE USING
SEQUENTIAL ACCESS**

```
*VOLUME,7,VSN=D0001,STAGEIN=NS,STAGEOUT=MD
*FORTRAN,FL
      PROGRAM DEM1
      DIMENSION B(6)
C
C      USE SEQUENTIAL ACCESS AND SYNCHRONOUS WRITE TO CREATE
C      A FILE OF ONE DATASET HAVING TEN RECORDS
C
      OPEN(UNIT=7,STATUS='NEW')
      DO 10 IREC=1,10
      READ(5,1000) B
      WRITE(UNIT=7) B
10    CONTINUE
      CLOSE(UNIT=7)
C
C      OPEN FILE WITH SEQUENTIAL ACCESS AND READ ALL RECORDS
C
      OPEN(UNIT=7,STATUS='OLD')
      DO 20 IREC=1,10
      READ(UNIT=7) B
20    CONTINUE
      CLOSE(UNIT=7)
C
C
      1000 FORMAT(6F10.1)
      END
```

COMMENTS

- Program DEM1 creates two copies of the same file, one on disk (during execution time) and one on MSD (during job termination phase).
- Many of the permissible arguments in the OPEN, CLOSE, READ and WRITE statements have not been specified in the above calls, thereby assuming default values.

In creating the above files, defaults are used to establish the file name, type of access, and mode of writing on unit 7. The following call is equivalent to the first OPEN call on program DEM1:

```
OPEN(UNIT=7,FILE=17HNCAR SYSTEMHD10001,FILESEQ=1,  
      FORM='UNFORMATTED',STATUS='NEW',RECL=6,  
      ACCESS='SEQUENTIAL',GEN=1,EXPDT=78093,BLANK='ZERO'
```

Because sequential access is used in program DEM1, specification of argument RECL is not required in the first OPEN call.

After execution, the status of the created file is 'KEEP', due to the status default in the last CLOSE call.

EXAMPLE USING DIRECT ACCESS

```

*VOLUME,7,VSN=D0002,STAGEIN=NS,CONV=LG
*FORTRAN,FL
      PROGRAM DEM2
      DIMENSION DATA(6)
C
C      USE DIRECT ACCESS AND SYNCHRONOUS WRITE TO CREATE
C      A FILE OF ONE DATASET HAVING FIVE RECORDS
C
      OPEN(UNIT=7,RECL=11,STATUS='NEW',MAXREC=25,ACCESS='DIRECT')
      DO 10 IREC=1,5
      READ(5,1000) IMONTH,IDAY,IYEAR,DATA
      WRITE(UNIT=7,REC=6-IREC) IMONTH,IDAY,IYEAR,DATA
10  CONTINUE
      CLOSE(UNIT=7)
C
C      ACCESS FOURTH RECORD DIRECTLY
C
      OPEN(UNIT=7,STATUS='OLD',ACCESS='DIRECT')
      READ(UNIT=7,REC=4) IMONTH,IDAY,IYEAR,DATA
      CLOSE(UNIT=7)
C
      1000 FORMAT(3 2,4X,6F10.2)
      END

```

COMMENTS

- RECL must be specified in the first OPEN call in program DEM2, since direct access and new status are used.
- In the first OPEN call, MAXREC is set to 25 even though only five records are written in this run. This is done in anticipation of adding 20 more records to the dataset at a later date. Similarly, RECL may be set to accommodate future expansion of record length.
- The RECL argument of the WRITE statement allows the records to be numbered independently of the order in which they are generated. In program DEM2, the numbering is reversed.

**EXAMPLE USING
DIRECT AND
SEQUENTIAL ACCESS
ON THE SAME UNIT**

```

*VOLUME,7,VSN=D0001,STAGEIN=MA,STAGEOUT=MD,CONV=LG
*FORTRAN,FL
      PROGRAM DEM3
      DIMENSION B(6)
C
C      OPEN FILE D0001 FROM PROGRAM DEM1 WITH DIRECT ACCESS
C
C      OPEN(UNIT=7,STATUS='OLD',ACCESS='DIRECT')
C      READ(UNIT=7,REC=4) B
C
C      ADD 1. TO EACH DATA ELEMENT IN RECORD 4 AND REWRITE
C      RECORD
C
C      DO 20 I=1,6
C      B(I)=B(I)+1.
20 CONTINUE
      WRITE(UNIT=7,REC=4) B
      CLOSE(UNIT=7)
C
C      OPEN FILE WITH SEQUENTIAL ACCESS AND SUM DATA ELEMENTS
C      OVER ALL RECORDS
C
C      OPEN(UNIT=7,STATUS='OLD',ACCESS='SEQUENTIAL')
C      SUM=0.
C      DO 30 IREC=1,10
C      READ(UNIT=7) B
C      DO 40 I=1,6
C      SUM=SUM+B(I)
40 CONTINUE
30 CONTINUE
      CLOSE(UNIT=7)
      WRITE(6,1000) SUM
C
1000 FORMAT(1H *SUM =*,F10.1)
END

```

COMMENTS

- The file with VSN=D0001 was created in program DEM1 using sequential access.
- Program DEM3 modifies the file created by program DEM1. To avoid the problem of the disk copy not matching the TMS-4 copy, STAGEOUT=MD is used on the *VOLUME card.

EXAMPLE OF USER CATALOG PROGRAM

```

1      *SEQUENCE,990934      7600 RUN      DATE 02/08/78      TIME 02/25/04
2      *JOB,8000,43430002,ADAMS      EDIT TEST
3      *VOLUME,9,VSN=P04323,STAGEIN=MA
INPUT -- *FORTRAN,FL

```

CARD NUMBER	APPROXIMATE PROGRAM LOCATION
1	0 PROGRAM DEM4
2	0 DIMENSION NAME(2)
3	0 DIMENSION IDATA(10)
4	0 LOGICAL EXIST,OPENED,NAMED
5	0 1GEN,EXPDT,VSN,CRECL,BLANK
6	0 PRINT 100
7	21 100 FORMAT(1H120X* INQUIRE CATALOG*)
8	21 DO 2 I=1,600
9	21 OPEN (UNIT=9,FILESEQ=I,ERR=90,IOSTAT=IOSTAT,ACCESS=#SEQUENTIAL#)
10	21 DO 20 II=1,I
11	44 READ(9,NWORDS=3) IDATA
12	52 20 CONTINUE
13	54 INQUIRE (UNIT=9,FILESEQ=I,ERR=91,IOSTAT=IOSTAT,EXIST=EXIST,
14	54 1OPENED=OPENED,NAMED=NAMED,NUMBER=NUMBER,NAME=NAME,RECL=RECL,
15	54 2MAXREC=MAXREC,ACCESS=ACCESS,SEQUENTIAL=SEQUENT,DIRECT=DIRECT,
16	54 3FORM=FORM,FORMATTED=FORMAT,UNFORMATTED=UNFORM,NSTATE=NSTATE,MODE=
17	54 4MCDE,NEXTREC=NEXTREC,SEQFIL=SEQFIL,GEN=GEN,EXPDT=EXPDT,
18	54 5VSN=VSN,NUMREC=NUMREC,CRECL=CRECL,BLANK=BLANK)
19	54 IF (.NOT.EXIST) GO TO 92
20	120 PRINT 102,NAME,NSTATE,NUMBER
21	133 102 FORMAT(///1x,2A10
22	133 2/1H #NSTATE=#020,2X#ON UNIT#I5)
23	133 PRINT 101,IOSTAT,EXIST,OPENED,NAMED,RECL,MAXREC,
24	133 1ACCESS,SEQUENT,DIRECT,FORM,FORMAT,UNFORM,MODE,
25	133 2NEXTREC,SEQFIL,GEN,EXPDT,VSN,NUMREC,CRECL,BLANK
26	212 101 FORMAT (1H0#IOSTAT=#04,7X#EXIST=#L2,10X#OPENED=#L2,12X#NAMED=#L2
27	212 1/1H #RECL=#I7,6X#MAXREC=#I9,2X#ACCESS=#A10,4X#SEQUENTIAL=#A10/
28	212 2H #DIRECT=#A10,1X#FORM=#A10,3X#FORMATTED=#A10,1X#UNFORMATTED=#A10
29	212 3/1H #MODE=#I2,11X#NEXTREC=#I9,1X#SEQFILE=#I4, 9X#GEN=#I4
30	212 4/1H #EXPDT=#A5, 7X#VSN=#A6, 8X#NUMREC=#I9 ,5X#CRECL=#I8,4X
31	212 5,#BLANK=#A4)
32	212 2 CONTINUE
33	214 90 STOP 90
34	216 91 STOP 91
35	217 92 STOP 92
36	220 END

LENGTH OF ROUTINE DEM4 354

VARIABLE ASSIGNMENTS

NAME -	0 IDATA -	2 EXIST -	221 OPENED -	222 NAMEDGE-	165555 EXPDT -	223
VSN -	224 CRECL -	225 BLANK -	226 I -	257 UNIT -	256 FILESEQ-	255
ERR -	254 IOSTAT -	253 ACCESS -	252 II -	251 NWORDS -	250 NAMED -	247
NUMBER -	246 RECL -	245 MAXREC -	244 SEQUENT-	243 DIRECT -	242 FORM -	241
FORMAT -	240 FORMAT -	237 UNFORMA-	236 UNFORM -	235 NSTATE -	234 MODE -	233
NEXTREC -	232 SEQFIL -	231 GEN -	230 NUMREC -	227		

SUBROUTINES CALLED

OUTPTC	Q8QAMB1 Q8QRDB	Q8QAMB2 STOP EXIT
COMPILE TIME = 63 MILLISECS		
INPUT -- *RUN,I		

-----+ EDITOR TERMINATING

0/100 SECONDS IS ELAPSED TIME

-----+

9.70

Input/Output Statements

EXAMPLE OF USER CATALOG PROGRAM

(continued)

PROGRAM SPACE IS 3253

ORIGIN	ENTRY POINTS AND LOCATIONS
4	DEM4 20
360	Q8QERR 360
373	Q8QIOU 373 IU903 616
627	Q8QAM83 627
740	EXIT 753 END 753 STOP 743
760	Q8QAM82 760
1054	K0DER 1054
2107	Q80AM81 2107
2403	OUTPTC 2403 OPTPTER 3116
3162	Q8QRDB 3162 Q8QWTE 3165

OVERALL CORE USE STATISTICS (DECIMAL)

THE MAXIMUM SCM IS 54272
THE PROGRAM CURRENTLY USES 1723
THE LOADING PROCESS USED 10193
THE MAXIMUM LCM IS 452622
THE PROGRAM CURRENTLY USES 54924

LCM AREA MAP BY BUFFER TYPE

MISC	186	SYSIOU	24	RANFO	64	RD	2240	PR	2240
DISK	8384	SSCR2	10432	DISK	16576	SYSLB0	256	SYSLB1	1024
SYSLB2	1024	SYSLB7	1124	SYSLB10	8192	SYSSAT	512	SYSDM1	1024
SYSSCM	1723								

INQUIRE CATALOG

PRCTC DATA

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAMED= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 2	SEQFILE= 1	GEN= 0
EXPDT=78288	VSN=P04323	NUMREC= 122	CRECL= 8 BLANK=D-5*

MCSTATS

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAMED= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 3	SEQFILE= 2	GEN= 0
EXPDT=78288	VSN=P04323	NUMREC= 16	CRECL= 8 BLANK=D-5*

MDCMPSTT

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAMED= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 4	SEQFILE= 3	GEN= 0
EXPDT=78288	VSN=P04323	NUMREC= 140	CRECL= 8 BLANK=D-5*

* The BLANK= specifier is not yet implemented.

JULY77A

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAME= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 5	SEQFILE= 4	GEN= 0
EXPDT=78288	VSN=P04323	NUMREC= 3410	CRECL= 8 BLANK=D-5*

MNTHDUMP

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAME= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 6	SEQFILE= 5	GEN= 0
EXPDT=78288	VSN=P04323	NUMREC= 992	CRECL= 8 BLANK=D-5*

MNTHDUMP2

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAME= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 7	SEQFILE= 6	GEN= 0
EXPDT=78288	VSN=P04323	NUMREC= 1055	CRECL= 8 BLANK=D-5*

CONCAT

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAME= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 8	SEQFILE= 7	GEN= 0
EXPDT=78291	VSN=P04323	NUMREC= 164	CRECL= 8 BLANK=D-5*

VOLCAT

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAME= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 9	SEQFILE= 8	GEN= 0
EXPDT=78297	VSN=P04323	NUMREC= 52	CRECL= 8 BLANK=D-5*

MNTHDMP3

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAME= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 10	SEQFILE= 9	GEN= 0
EXPDT=78301	VSN=P04323	NUMREC= 1205	CRECL= 8 BLANK=D-5*

DISPLAST

NSTATE=00000000740100000003 ON UNIT 9

IOSTAT=0000	EXIST= T	OPENED= T	NAME= T
RECL= 0	MAXREC= 0	ACCESS=SEQUENTIAL	SEQUENTIAL=YES
DIRECT=YES	FORM=UNFORM	FORMATTED=YES	UNFORMATTED=YES
MODE= 0	NEXTREC= 11	SEQFILE= 10	GEN= 0
EXPDT=78311	VSN=P04323	NUMREC= 277	CRECL= 8 BLANK=D-5*

*The BLANK= specifier is not yet implemented.

EXAMPLE OF USER CATALOG PROGRAM

(continued)

```
RPTGN
NSTATE=000000074010000003 ON UNIT 9

IOSTAT=0000 EXIST= T OPENED= T NAMED= T
RECL= 0 MAXREC= 0 ACCESS=SEQUENTIAL SEQUENTIAL=YES
DIRECT=YES FORM=UNFORM FORMATTED=YES UNFORMATTED=YES
MODE= 0 NEXTREC= 12 SEQFILE= 11 GEN= 0
EXPDT=78362 VSN=P04323 NUMREC= 320 CRECL= 8 BLANK=D-/*
```

```
RPTGN2
NSTATE=000000074010000003 ON UNIT 9

IOSTAT=0000 EXIST= T OPENED= T NAMED= T
RECL= 0 MAXREC= 0 ACCESS=SEQUENTIAL SEQUENTIAL=YES
DIRECT=YES FORM=UNFORM FORMATTED=YES UNFORMATTED=YES
MODE= 0 NEXTREC= 13 SEQFILE= 12 GEN= 0
EXPDT=79016 VSN=P04323 NUMREC= 434 CRECL= 8 BLANK=D-/*
```

```
RPTGN3
NSTATE=000000074010030003 ON UNIT 9

IOSTAT=0000 EXIST= T OPENED= T NAMED= T
RECL= 0 MAXREC= 0 ACCESS=SEQUENTIAL SEQUENTIAL=YES
DIRECT=YES FORM=UNFORM FORMATTED=YES UNFORMATTED=YES
MODE= 0 NEXTREC= 14 SEQFILE= 13 GEN= 0
EXPDT=79024 VSN=P04323 NUMREC= 682 CRECL= 8 BLANK=D-/*
```

```
RPTGN4
NSTATE=000000074010000003 ON UNIT 9

IOSTAT=0000 EXIST= T OPENED= T NAMED= T
RECL= 0 MAXREC= 0 ACCESS=SEQUENTIAL SEQUENTIAL=YES
DIRECT=YES FORM=UNFORM FORMATTED=YES UNFORMATTED=YES
MODE= 0 NEXTREC= 15 SEQFILE= 14 GEN= 0
EXPDT=79025 VSN=P04323 NUMREC= 724 CRECL= 8 BLANK=D-/*
STOP90
```

TERMINATION DATE = 02/08/78
 TERMINATION TIME = 02/25/08

TOTAL CPU TIME IN MILLISECONDS = 224
 PPU TIME IN MILLISECONDS = 3879
 PAGES PRINTED = 6
 TOTAL RESOURCES USED = .63

DISK BLOCK USAGE SUMMARY	
ODD UNITS	EVEN UNITS
DISK 0	DISK 1
4.95	4.95
LIMIT ON BLOCKS ALLOWED	5
MAXIMUM BLOCKS USED	
PLIB BLOCKS FOR THIS PROJECT	

*The BLANK= specifier is not yet implemented.

SUMMARY TABLE OF FORTRAN SYNTAX

Synchronous Input/Output Statements

```

READ ([UNIT=]u[ ,FMT=]f)[,REC=rn][,ERR=s][,END=s][,IOSTAT=ios]) iolist
WRITE ([UNIT=]u[ ,FMT=]f)[,REC=rn][,MODE=md][,ERR=s][,IOSTAT=ios]) iolist
READ f, iolist
PRINT f, iolist
PUNCH f, iolist
WRITE (u,f) iolist
READ (u) iolist

```

Asynchronous Input/Output Statements

```

READ ([UNIT=]u[ ,REC=rn],NWORDS=n[,NSTATE=ns]) fwa
WRITE ([UNIT=]u[ ,REC=rn][,MODE=md],NWORDS=n[,NSTATE=ns]) fwa
BUFFER IN (u,p)(fwa,lwa)
BUFFER OUT (u,p)(fwa,lwa)

```

ENCODE/DECODE Statements

```

ENCODE (cc,f,v) iolist
DECODE (cc,f,v) iolist

```

Dataset Support Statements

```

OPEN ([UNIT=]u[ ,FILE=dsi][,FILESEQ=n][,ERR=s][,FORM=fm][,STATUS=sta][,RECL=rlen]
      [,MAXREC=maxr][,ACCESS=acc][,GEN=gen][,EXPDT=exp][,PASS=pass][,IOSTAT=ios]
      [,BLANK=blnk])
CLOSE ([UNIT=]u[ ,STATUS=sta][,ERR=s][,IOSTAT=ios])
INQUIRE ([UNIT=]u[ ,FILE=dsi][,FILESEQ=n][,ERR=s][,IOSTAT=ios][,EXIST=ex][,OPENED=od]
         [,NAMED=nmd][,NUMBER=r][,NAME=fn][,RECL=rlen][,MAXREC=maxr][,ACCESS=acc]
         [,SEQUENTIAL=seq][,DIRECT=dir][,FORM=fm][,FORMATTED=fmt][,UNFORMATTED=unf]
         [,NSTATE=ns][,MODE=md][,NEXTREC=nr][,SEQFIL=n][,GEN=gen][,EXPDT=exp][,VSN=name]
         [,NUMREC=nr][,CRECL=c1][,GENTIME=gnt][,BLANK=blnk][,CRDT=crdt])
SETPASS ([UNIT=]u,PASS=pass[,RWPASS=rwpass][,RPASS=rpass][,WPASS=wpass]
K=LENGTHF(u)
IF(UNIT,u)s1,s2,s3,s4

```

SUMMARY TABLE OF FORTRAN SYNTAX
(continued)

File Positioning Statements

BACKSPACE u

BACKSPACE ([UNIT=]u[, IOSTAT=ios][,ERR=s])

ENDFILE u

ENDFILE ([UNIT=]u[, IOSTAT=ios][,ERR=s])

REWIND u

REWIND ([UNIT=]u[, IOSTAT=ios][,ERR=s])

SKIPFILE u

SKIPFILE (u[, COUNT=m][,ERR=s])

BACKFILE u

BACKFILE (u[, COUNT=m][,ERR=s])

IF(EOF,u)s₁,s₂

X

FORTRAN STATEMENT LIST

FORTRAN STATEMENT LIST

INTRODUCTION

The FORTRAN statement list includes allowable forms which statements may take using NCAR FORTRAN, CRAY-1 FORTRAN, FORTRAN 66 and FORTRAN 77. The list summarizes a FORTRAN vocabulary in each case. The statements have been subdivided into several categories: Program and subprogram statements, subprogram reference and transfer, arithmetic statement function, type declaration, storage allocation, replacement, branching and IF logic, looping and control statements, format, data input, data output, file positioning, core to core transfer, and dataset support.

The *CRAY FORTRAN Manual*, published by Cray Research, Inc., describes the FORTRAN compiler in use on the CRAY-1 computer. These statements are included for comparison with NCAR 7600 FORTRAN and the older and current standards. The symbols used by NCAR FORTRAN have been used in all of the statement lists. A table of symbol definitions appears at the end of this chapter.

10.2 FORTRAN Statement List

NCAR FORTRAN

CRAY FORTRAN

PROGRAM AND SUBPROGRAM STATEMENTS: (*non-executable*)

```
PROGRAM pgm
SUBROUTINE sub [(d1[,d2]...)]
[typ] FUNCTION fun (d1[,d2]...)
-----
BLOCK DATA [sub]
ENTRY en
EXTERNAL en1[,en2]...
```

```
PROGRAM pgm
SUBROUTINE sub [(d1[,d2]...)]
[typ] FUNCTION fun (d1[,d2]...)
-----
BLOCK DATA [sub]
-----
EXTERNAL en1[,en2]...
```

SUBPROGRAM REFERENCE AND TRANSFER: (*executable*)

```
CALL en [(a1[,a2]...)]  
RETURN  
fun (a1[,a2]...)
```

```
CALL en [(a1[,a2]...)]  
RETURN  
fun (a1[,a2]...)
```

ARITHMETIC STATEMENT FUNCTION: (definition)

`fun (d1[,d2]...) = exp` `fun (d1[,d2]...) = exp`

TYPE DECLARATION: (non-executable)

```
COMPLEX nlist
DOUBLE PRECISION nlist
REAL nlist
INTEGER nlist
LOGICAL nlist
IMPLICIT typ (let1[,let2]...)
[ ,typ (let3[,let4]...)]...
```

```
COMPLEX nlist
DOUBLE PRECISION nlist
REAL nlist
INTEGER nlist
LOGICAL nlist
IMPLICIT typ (let1[,let2]...)
[ ,typ (let3[,let4]...)]...
PARAMETER (param1=exp1[,param2=exp2]...)
```

FORTRAN 66**PROGRAM AND SUBPROGRAM STATEMENTS:** (*non-executable*)

 SUBROUTINE sub [(d₁[,d₂]...)]
 [typ] FUNCTION fun (d₁[,d₂]...)

 BLOCK DATA

 EXTERNAL en₁[,en₂]...

FORTRAN 77

PROGRAM pgm
 SUBROUTINE sub [(d₁[,d₂]...)]
 [typ] FUNCTION fun ([d₁[,d₂]...])
 CHARACTER [*len] FUNCTION fun ([d₁[,d₂]...])
 BLOCK DATA [sub]
 ENTRY en [(d₁[,d₂]...)]
 EXTERNAL en₁[,en₂]...

SUBPROGRAM REFERENCE AND TRANSFER: (*executable*)

CALL en [(a₁[,a₂]...)]
 RETURN
 fun (a₁[,a₂]...)

CALL en [(a₁[,a₂]...)]
 RETURN [exp]
 fun ([a₁[,a₂]...])

ARITHMETIC STATEMENT FUNCTION: (*definition*)

fun (d₁[,d₂]...) = exp

fun ([d₁[,d₂]...]) = exp

TYPE DECLARATION: (*non-executable*)

COMPLEX nlist
 DOUBLE PRECISION nlist
 REAL nlist
 INTEGER nlist
 LOGICAL nlist

COMPLEX nlist
 DOUBLE PRECISION nlist
 REAL nlist
 INTEGER nlist
 LOGICAL nlist
 IMPLICIT typ (let₁[,let₂]...)
 [,typ (let₁[,let₂]...)]...
 PARAMETER (param₁=exp₁[,param₂=exp₂]...)
 INTRINSIC fun₁[,fun₂]...
 CHARACTER [*len[,]]nam₁[,nam₂]...

NCAR FORTRAN**STORAGE ALLOCATION:** (*non-executable*)

DIMENSION v₁(n₁[,n₂][,n₃])
 [,v₂(n₁[,n₂][,n₃])]...
 COMMON [/blk₁/]nlist[/blk₂/nlist]...
 DATA klist₁/clist₁/
 [[,]klist₂/clist₂]...
 EQUIVALENCE (nlist)[,(nlist)]...

REPLACEMENT: (*executable*)

x = arithmetic expression
 x = relational/logical expression
 x = masking expression

ASSIGN s to lab

BRANCHING AND IF LOGIC: (*executable*)

GO TO s
 GO TO lab [, (s₁[,s₂]...)]
 GO TO (s₁[,s₂]...)[,]i
 IF(exp) s₁,s₂,s₃
 IF(exp) s₁,s₂
 IF(exp) st
 ELSE
 ELSE IF(exp) THEN
 END IF
 IF(exp) THEN

CRAY FORTRAN

DIMENSION v₁(n₁[,n₂][,n₃]...[,n₇])
 [,v₂(n₁[,n₂][,n₃]...[,n₇])]...
 COMMON [/blk₁/]nlist[/blk₂/nlist]...
 DATA klist₁/[j*]dlist₁/
 [,klist₂/[j*]dlist₂]...
 EQUIVALENCE (nlist)[,(nlist)]...

x = arithmetic expression
 x = relational/logical expression
 x = masking expression

ASSIGN s to lab

GO TO s
 GO TO lab [, (s₁[,s₂]...)]
 GO TO (s₁[,s₂]...)[,]i
 IF(exp) s₁,s₂,s₃
 IF(exp) 2₁,2₂
 IF(exp) st

FORTRAN 66**STORAGE ALLOCATION:** (*non-executable*)

DIMENSION v₁(n₁[,n₂][,n₃])
 [,v₂(n₁[,n₂][,n₃])]...
 COMMON [/blk₁/]nlist[/blk₂/]nlist]...
 DATA klist₁/[j*]dlist₁/
 [,klist₂/[j*]dlist₂]...
 EQUIVALENCE (nlist)[,(nlist)]...

FORTRAN 77

DIMENSION v₁(n₁[,n₂][,n₃]...[,n₇])
 [,v₂(n₁[,n₂][,n₃]...[,n₇])]...
 COMMON [/blk₁/]nlist[[,]/blk₂/]nlist]...
 DATA klist₁/clist₁/
 [[,]klist₂/clist₂]...
 EQUIVALENCE (nlist)[,(nlist)]...
 SAVE [a₁[,a₂]...]

REPLACEMENT: (*executable*)

x = arithmetic expression
 x = relational/logical expression

 ASSIGN s to lab

x = arithmetic expression
 x = relational/logical expression

 x = character expression
 ASSIGN s to lab

BRANCHING AND IF LOGIC: (*executable*)

GO TO s
 GO TO lab [, (s₁[,s₂]...)]
 GO TO (s₁[,s₂]), i
 IF(exp) s₁,s₂,s₃

 IF(exp) st

GO TO s
 GO TO lab [[,](s₁[,s₂]...)]
 GO TO (s₁[,s₂]...)[,]i
 IF(exp)s₁,s₂,s₃

 IF(exp) st
 ELSE
 ELSE IF(exp) THEN
 END IF
 IF(exp) THEN

10.6

FORTRAN Statement List

NCAR FORTRAN**LOOPING AND CONTROL STATEMENTS:** (*executable*)

DO s[,]i = exp₁,exp₂[,exp₃]
 CONTINUE
 PAUSE [string]
 STOP [string]
 END

CRAY FORTRAN

DO s i = m₁,m₂[,m₃]
 CONTINUE
 PAUSE [string]
 STOP [string]
 END

FORMAT: (*non-executable*)

FORMAT (spec)

FORMAT (spec)

DATA INPUT: (*executable*)

READ f[,iolist]
 READ (u,f)[iolist]
 READ (u)[iolist]
 READ (cilist)[iolist]
 READ (rlist) fwa
 BUFFER IN (u,p)(fwa,lwa)

READ f,iolist
 READ (u,f)iolist
 READ (u)iolist

 BUFFER IN (u,p)(fwa,lwa)

DATA OUTPUT: (*executable*)

PRINT f[,iolist]
 PUNCH f[,iolist]
 WRITE (u,f)[iolist]
 WRITE (u)[iolist]
 WRITE (cilist)[iolist]
 WRITE (wlist) fwa
 BUFFER OUT (u,p)(fwa,lwa)

PRINT f,iolist

 WRITE (u,f)iolist
 WRITE (u)iolist

 BUFFER OUT (u,ms)(fwa,lwa)

FORTRAN 66**LOOPING AND CONTROL STATEMENTS:** (*executable*)

```
DO s i = m1,m2[,m3]  
CONTINUE  
PAUSE [string]  
STOP [string]  
END
```

FORTRAN 77

```
DO s[,]i = exp1,exp2[,exp3]  
CONTINUE  
PAUSE [string]  
STOP [string]  
END
```

FORMAT: (*non-executable*)

```
FORMAT (spec)
```

```
FORMAT (spec)
```

DATA INPUT: (*executable*)

```
-----  
READ (u,f)iolist  
READ (u)iolist  
-----  
-----  
-----
```

```
READ f[,iolist]  
READ (u,f)[iolist]  
READ (u)[iolist]  
READ (cilist)[iolist]  
-----  
-----
```

DATA OUTPUT: (*executable*)

```
-----  
-----  
WRITE (u,f)iolist  
WRITE (u)iolist  
-----  
-----  
-----
```

```
PRINT f[,iolist]  
-----  
-----  
WRITE (u,f)[iolist]  
WRITE (u)[iolist]  
WRITE (cilist)[iolist]  
-----  
-----
```

NCAR FORTRAN**FILE POSITIONING:** (*executable*)

ENDFILE (alist)
 REWIND (alist)
 BACKSPACE (alist)
 BACKFILE (flist)
 SKIPFILE (flist)
 IF(EOF,u) s₁,s₂
 IF(UNIT,u) s₁,s₂,s₃,s₄

CRAY FORTRAN

ENDFILE u
 REWIND u
 BACKSPACE u

 IF(IEOF(u)) s₁,s₂,s₃
 IF(UNIT(u)) s₁,s₂,s₃

CORE TO CORE TRANSFER: (*executable*)

ENCODE (cc,f,v)[iolist]
 DECODE (cc,f,v)[iolist]

ENCODE (cc,f,v)[iolist]
 DECODE (cc,f,v)[iolist]

DATASET SUPPORT: (*executable*)

OPEN (olist)
 INQUIRE (iulist)

 CLOSE (cclist)
 SETPASS (passlist)

FORTRAN 66

FILE POSITIONING: (*executable*)

ENDFILE u

REWIND u

BACKSPACE u

CORE TO CORE TRANSFER: (*executable*)

FORTRAN 77

ENDFILE (alist)

REWIND (alist)

BACKSPACE (alist)

DATASET SUPPORT: (*executable*)

OPEN (olist)

INQUIRE (iulist)

INQUIRE (iflist)

CLOSE (clist)

Definition of Symbols in Statement Syntax

a	variable or array name; an actual argument expression
alist	a list of specifiers containing [UNIT=]u; may also contain IOSTAT=ios and/or ERR=s
blk	common block identifier
cc	character count
cilist	control information list
clist	list of constants or symbolic names of constants used in NCAR FORTRAN and FORTRAN 77
corlist	control information list with UNIT=nam (a character variable or array name) and no direct access
d	dummy argument
dlist	list of constants or symbolic names of constants used in FORTRAN 66 and CRAY FORTRAN
en	entry point name
exp	an expression
f	format statement label or array with Hollerith format data
flist	a list of specifiers: (u[,COUNT=n][,ERR=s])
fun	function name
fwa	first word address of I/O buffer area; an array element or array name
i	integer variable
iflist	file specifier list which may contain inquiry specifiers
iolist	an input/output list; may contain implied DO
iulist	external unit specifier which may also contain inquiry specifiers
ios	input/output status specifier
j*	a repeat factor
klist	variable, array identifiers and DO implied lists
lab	variable assigned a statement label
len	a specifier for the length of a character variable
let	a single letter or a range of single letters in alphabetic order

lwa	last word address of I/O buffer area; an array element
m	incrementation parameter of a DO statement
ms	made specifier for full or partial record processing
n	integer constant greater than zero, or symbolic name of such a constant
nam	character variable or array element
nlist	list of variables, array elements and array names
olist	specifier list for OPEN statement
p	parity
param	a symbolic name
passlist	password specifiers
pgm	a program name
rlist	asynchronous READ specifier list
s	statement label
st	a FORTRAN statement
spec	format specifiers
string	a digit string
sub	a subroutine name
typ	type: REAL, INTEGER, DOUBLE PRECISION, COMPLEX or LOGICAL
u	unit number, unsigned integer constant or variable
v	variable name
wlist	asynchronous WRITE specifier list
x	variable or array element

XI

DIAGNOSTICS

ERRORS FATAL TO EXECUTION

ERRORS FATAL TO COMPILEMENT

NONFATAL ERRORS

DIAGNOSTICS**COMPILEATION
DIAGNOSTICS**

Errors in FORTRAN program compilation are pointed out to the user by the compiler in diagnostic messages following a program or subprogram listing. They are found following the END card of the program or subprogram in which the error appears.

The form of the diagnostic message is

<i>Card Number</i>	<i>FORTRAN Error Message</i>	<i>Compiler Location</i>
n	SOME DIAGNOSTIC MESSAGE	m

The first column appearing on the FORTRAN program contains the card number. In the diagnostic message, n refers to the first card of the statement in error. m, the compiler location, is used for system debugging.

11.2

Diagnostics

COMPILE DIAGNOSTICS (continued)

Example

CARD NUMBER	APPROXIMATE PROGRAM LOCATION
1	0 PROGRAM NEWT
2	0 C
3	1 C PROGRAM NEWT ILLUSTRATES THE USE OF FORTRAN ERROR MESSAGES
4	0 C
5	0 101 READ 1001 * X,EPS,K
6	11 WRITE (6,1005) X,EPS,K
7	22 READ 1002, (A(I),I=1,K)
8	24 WRITE (6,1006) (A(I),I=1,K)
9	24 C
10	24 C DEFINE COEFF FOR DERIV
11	24 C
12	26 L = K-1
13	26 DO 102 I=1,K(J)
14	26 I2 = I-1
15	26 102 B(I) = I2*A(I)
16	26 C
17	26 C ITERATE MAXIMUM OF 20 TIMES
18	26 C
19	31 DO 104 N=1,30
20	31 FX = A(K)
21	31 DO 103 I=1,L
22	35 N = K-I
23	35 113 FX = X*FX+A(M)
24	35 FPX = R(K)
25	35 DO 103 I=2,L
26	53 M = K-I+1
27	53 FFX = X*FPX+B(M)
28	60 104 CONTINUE
29	65 WRITE (6,1007) N,FX,FFX,X
30	104 IF (ABS(FX) .LT. EPS) GO TO 103
31	116 NN = N
32	116 X = X-FX/FFX
33	105 CONTINUE
34	112 WRITE (6,1003)
35	115 GO TO 101
36	115 106 WRITE (6,1004) NN
37	123 GO TO 101
38	123 C
39	123 1001 FORMAT (2F10.2,I10)
40	123 1002 FORMAT (6F10.0)
41	123 1003 FORMAT (* DOES NOT CONVERGE*)
42	123 1004 FORMAT (*0MATRIX CONVERTED AT ITERATE NO=15*)
43	123 1005 FORMAT (0!FORX(0))
44	123 1006 FORMAT (0# FOR COEFF=*/*/(1H F10.2))
45	123 1007 FORMAT (1H 15.3E20.4)
46	123 C
47	123 END
LENGTH OF ROUTINE: NEWT 170	
VARIABLE ASSIGNMENTS	
X	= 140 EPS = 137 K = 136 I = 135 L = 134 I2 = 133
N	= 132 FX = 131 M = 130 FPX = 127 FFX = 126 NN = 125

SUBROUTINES CALLED
 INPUTC OUTPTC A EXIT
 COMPILE TIME = 236 MILISECS

THESE STATEMENT LABELS ARE MISSING

103

CARD NUMBER	FORTRAN ERROR MESSAGE	COMPILER LOCATION
7	PARENTHESIS USAGE OR DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL IN I/O DATA LIST	51105
8	PARENTHESIS USAGE OR DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL IN I/O DATA LIST	51105
13	A SUBSCRIPTED VARIABLE HAS NOT BEEN DIMENSIONED	61565
15	ILLEGAL USE OF A REPLACEMENT STATEMENT	52222
25	A DO VARIABLE IS USED IN AN OUTER DO LOOP	47402
28	A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO	47114
43	A FORMAT SPECIFICATION HAS NO FIELD WIDTH	52363
47	THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES	51610
47	THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES	51610
THE CARD NUMBER IS THE FIRST CARD OF THE STATEMENT IN WHICH THE ERROR OCCURRED		
THE COMPILER LOCATION IS FOR SYSTEM DEBUGGING		

In the example, the diagnostic for card number 7 points to

READ 1002,(A(I),I=1,K)

The message says that the I/O list is illegal. The reason is that A is not dimensioned. A nondimensioned variable is assumed to be a function subprogram and may not appear in an I/O list.

Card number 13 locates the statement

DO 102 I=1,K(J)

This is incorrect first because K has not been dimensioned, but further, a subscripted variable may not be a parameter in a DO. The dimension error came up first. The correction is not to dimension K, but to remove the subscript.

Card number 28 is 104 CONTINUE. The message says that an outer DO contains an inner DO not terminated. Statement 113 (card number 23) was mispunched: it should have been 103, which would terminate the DO 103 loop.

Often one FORTRAN mistake will generate more than one diagnostic. Card numbers 25, 28, and 47 relate to the same unterminated DO loop (DO 103 I=1,L).

The example shows that the compiler finds errors in a certain order. One error can cause several diagnostics, none of which necessarily point to the particular card that must be changed. A program is an ordered set of instructions which are interrelated, and the compiler looks at the program as a unified whole which should make sense. Whenever a statement appears which does not fit in with what has been previously set down, an error may be flagged.

11.4
Diagnostics

**COMPILEATION
DIAGNOSTICS**
(continued)

Some errors are caused by omission. Several errors in this program were caused by a missing DIMENSION statement.

A list of compiler diagnostics is included below. There are three kinds of errors:

- Those that cause compilation to stop ("fatal to compilation").
- Those that allow compilation to continue, but abort execution of the program ("fatal to execution").
- Those that are merely flags to sloppy programming and do not stop the job ("nonfatal").

**ERRORS FATAL TO
COMPILATION**

A VARIABLE NEEDS DIMENSIONING

AN EQUIVALENCED VARIABLE APPEARS WITH SUBSCRIPTS BUT HAS NOT BEEN DIMENSIONED OR REFERENCED VARIABLE IS NOT THE COMMON VARIABLE

AN UNDEFINED FUNCTION APPEARS WITHIN THIS DO LOOP

ARRAY SIZE DECLARATION ALLOWED IN EITHER DIMENSION OR TYPE STATEMENT, NOT BOTH

COMMON/EQUIVALENCE ERROR (EQVLST1 IN COMLST1)

COMPILER-BLOCK NAME TABLE OVERFLOW

COMPILER-COMMON TABLE OVERFLOW

COMPILER-DIMENSION LIST OVERFLOW

COMPILER-DIMENSION LIST OVERFLOW FROM EXTERNAL STATEMENT

COMPILER-DIMENSION LIST OVERFLOW FROM TYPE ROUTINE

COMPILER-DIMENSION VARIABLE LIST OVERFLOW

COMPILER-EQUIVALENCE LIST OVERFLOW

COMPILER ERROR DATA STATEMENT

COMPILER FAILURE

COMPILER-TYPE TABLE LIMIT EXCEEDED

COMPILER TABLE OVERFLOW

COMPILER TABLE OVERFLOW (PROCESS PI WLIS)

DIMENSION SIZE OF ZERO NOT ALLOWED

DIMENSION TOO LARGE FOR CORE

DIMENSION TOTAL .GT. 65K

EQUIVALENCE ATTEMPTS TO RE-ORDER COMMON

EQUIVALENCE OVERLAPS BLOCK ORIGIN

EQUIVALENCE OVERLAPS BLOCKS

FORMAT POINTER TABLE OVERFLOW

FTN-4 DATA STATEMENT FORMAT ERROR

FTN-4 DATA STATEMENT PRE-SCAN TABLE OVERFLOW

GREATER THAN THREE DIMENSIONS IN DIMENSION STATEMENT

ERRORS FATAL TO**COMPILEATION***(continued)*

ILLEGAL EXPONENT

ILLEGAL PROGRAM CARD

IMPROPER DIMENSIONED VARIABLE LIST

IMPROPER FORMAT OF PROGRAM STATEMENT

IMPROPER SUBROUTINE OR FUNCTION STATEMENT OR PARAMETER ERROR

MULTIPLE EQUIVALENCE BETWEEN SAME NAME VARIABLES IS
AMBIGUOUSNON-INTEGER EXPRESSION NOT ALLOWED IN ASA FORTRAN
STANDARDS. EXECUTION MAY CONTINUE.

PARENTHESIS MISSING FROM DIMENSIONED TYPE STATEMENT

PROGRAM CARD, SUBROUTINE CARD, OR END CARD MISSING

PROGRAM LENGTH EXCEEDS MEMORY SIZE (1777777B)

PROGRAM OR SUBROUTINE CARD MISSING

PROGRAM OR SUBROUTINE CARD NOT FIRST CARD

STORAGE DEFINITIONS EXCEED COMPILER CAPACITY

TABLE OVERFLOW (ASF)

THIS EQUIVALENCE CAUSES A REORIGIN OF THE COMMON BLOCK

TOO MANY CONSTANTS

TOO MANY INDEX VARIABLES

TOO MANY PARAMETERS IN PARAMETER LIST

TRIED TO EQUATE DIFFERENT VARIABLE TYPES

ERRORS FATAL TO EXECUTION

A CONSTANT IN A COMPLEX PARAMETER IS NOT A REAL NUMBER
A DATA LIST ENTRY MAY NOT BE DEFINED BY FORMAL PARAMETER
A DO EXPRESSION CANNOT BE EVALUATED CORRECTLY
A DO LOOP IMPROPERLY DEFINED
A DO LOOP LIMIT EXCEEDS MACHINE CAPACITY
A DO LOOP MAY NOT TERMINATE AT THE END STATEMENT
A DO LOOP TERMINATES AT THIS STATEMENT
A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO
A DO OR DO-IMPLYING LOOP OUT OF RANGE
A DO VARIABLE IS USED IN AN OUTER DO LOOP
A DO VARIABLE NOT SET IN DATA STATEMENT
A EXPRESSION FOR A DO PARAMETER IS IMPROPERLY FORMED
A FUNCTION NAME WAS NOT USED AS A REPLACEMENT STATEMENT
A MULTIPLY OPERATOR HAS BEEN INSERTED
A NUMBERED STATEMENT HAS A CONTINUATION MARK
A PARAMETER ARGUMENT MUST BE DEFINED AS A CONSTANT
A PARAMETER NAME WAS PREVIOUSLY DEFINED AS A DIMENSIONED VARIABLE
A PREVIOUS DO TERMINATES ON THIS DO STATEMENT
A REPETITION COUNT IN A DATA STATEMENT CONSTANT LIST MUST BE TYPE INTEGER
A REPLACEMENT CANNOT BE FORMED
A RIGHT PARENTHESIS MISSING IN DATA STATEMENT
A SIGN ON A REPETITION COUNT IN A CONSTANT LIST IS MEANINGLESS
A SUBSCRIPTED VARIABLE HAS NOT BEEN DIMENSIONED
A VARIABLE IN DATA STATEMENT HAS VARIABLE DIMENSIONS
A ZERO INCREMENT IN A DO OR DO-IMPLYING LOOP
AN ARRAY NAME PREVIOUSLY USED
AN ENTRY STATEMENT MAY NOT OCCUR INSIDE A DO LOOP
AN EQUAL SIGN MUST FOLLOW THE DO VARIABLE
AN IDENTIFIER HAS MORE THAN SEVEN CHARACTERS
AN IMPLIED DO VARIABLE MUST BE A NON-SUBSCRIPTED INTEGER VARIABLE
AN INDEX IN DATA LIST VARIABLE IS NOT DEFINED BY AN IMPLIED DO VARIABLE

**ERRORS FATAL TO
EXECUTION**
(continued)

AN OCTAL CONSTANT HAS MORE THAN 20 DIGITS
AN OPERATOR IS MISSING OR USED IMPROPERLY
.AND., .OR. MUST BE FOLLOWED BY (, .NOT., OR OPND
ARGUMENT NOT DEFINED IN I/O LIST
ARGUMENTS IN IMPLICIT STATEMENT MUST BE A SINGLE ALPHABETIC LETTER
ARGUMENTS IN IMPLICIT STATEMENT MUST BE ALPHABETIC CHARACTER BETWEEN A AND Z
ARITHMETIC STATEMENT FUNCTION DOUBLY DEFINED
ARITHMETIC STATEMENT FUNCTION PARAMETER LIST NOT ENDED
ARITHMETIC STATEMENT FUNCTION PARAMETERS DO NOT AGREE IN NUMBER
ASF INVOLVES ITSELF
ASF PARAMETER ERROR
ATTEMPT TO DOUBLY TYPE A VARIABLE
ATTEMPT TO PRESET DATA INTO BLANK COMMON

BLANK COMMON MAY NOT BE ASSIGNED BY A DATA STATEMENT
BLANK USED AS A BLOCK NAME MORE THAN ONCE
BRANCH DESTINATION INVALID OR A SEPARATOR IS MISSING
BUFFER TAPE MODE PARAMETER MUST BE SIMPLE INTEGER VARIABLE
OR INTEGER CONSTANT

CALLED SUBPROGRAM AND VARIABLE USE SAME NAME
CANNOT FIND END OF IMPLIED DO
CANNOT FIND LIMITING / IN CONSTANT LIST
CLOSE STATEMENT-ERROR IN (STATUS). MUST BE EITHER KEEP OR
DELETE
COMMA IN DATA STATEMENT MISSING
COMMA MISSING IN EQUIVALENCE STATEMENT
COMMA MISSING IN OVERLAY STATEMENT
COMPILER - CANNOT PROCESS COMMON STATEMENTS
COMPILER ERROR DATA STATEMENT
COMPILER - TOO MANY ARITHMETIC STATEMENT FUNCTION ENTRIES
COMPILER - TOO MANY FUNCTIONS

COMPILER - TOO MANY IDENTIFIERS
COMPILER - TOO MANY INDEX FUNCTIONS
COMPILER - TOO MANY NON-STANDARD INDICES
CONSTANT LIST SYNTAX ERROR PROBABLY OPERATOR ERROR
CONSTANT LIST SYNTAX ERROR, PROBABLY TOO LONG
CONSTANT OUT OF RANGE
CONTENTS OF DATA AND CONSTANT LIST MUST AGREE IF EITHER IS TYPE COMPLEX
COUNT VALUE PREVIOUSLY DEFINED

DATA LIST AND CONSTANT LIST MUST BE ONE TO ONE
DATA LIST CONTAINS A FORMAL PARAMETER
DATA LIST CONTAINS THE FUNCTION SUBROUTINE NAME
DATA LIST SYNTAX ALLOWS ONLY VARIABLE, ARRAY, OR SUBSCRIPTED VARIABLE
DATA TO TYPE LOGICAL VARIABLE
DATA TO VARIABLE DIMENSIONED ARRAY
DECLARATIVE STATEMENTS MUST PRECEDE ALL ARITHMETIC STATEMENTS
DIMENSIONED ARRAY TOO LARGE FOR CORE
DO INDEX IN DATA STATEMENT NOT STANDARD
DO NOT LABEL AN ENTRY STATEMENT
DOUBLE PRECISION NUMBER TO SINGLE PRECISION VARIABLE
DOUBLY DEFINED VARIABLE NAME IN COMMON
DOUBLY DEFINED VARIABLE OR DIMENSION IN COMMON
DUPLICATE BLOCK NAME
DUPLICATE STATEMENT NUMBER

**ERRORS FATAL TO
EXECUTION**
(continued)

ENCODE COUNT PARAMETER MUST BE SIMPLE INTEGER VARIABLE OR
INTEGER CONSTANT

END CARD MISSING

END OF FILE ERROR FROM INPUT FILE

EQUAL SIGN IN DATA STATEMENT MISSING

EQUAL SIGN MUST DELIMIT KEYWORD

ERROR EXIT PREVIOUSLY SET

ERROR IN ASF SETUP

ERROR STATEMENT LABEL TOO LARGE

EXCESS LEFT PARENTHESIS

EXPRESSION TOO LARGE

FIRST ELEMENT OF BOOLEAN EXP MUST BE OPERAND, (, OR .NOT.

FIRST WORD OF SIGMA STRING NOT AN IDENTIFIER

FORMAL PARAMETER ERROR IN EQUIVALENCE

FORMAL PARAMETER IN COMMON DECLARATION

FORMAT MAY NOT HAVE A VARIABLE INDEX

FORMAT MUST BE SIMPLE INTEGER VARIABLE OR INTEGER CONSTANT

FUNCTION FORMED INCORRECTLY

.GT. # DIMENSIONS FOR VARIABLE IN DATA STATEMENT

IF FORMED INCORRECTLY

IF STATEMENT FORMAT ERROR

ILLEGAL CHARACTER APPEARS IN A CONSTANT

ILLEGAL COMPLEX NO. OR SUBSCRIPTED SUBSCRIPTS

ILLEGAL CONSTANT TYPE

ILLEGAL EXPONENT

ILLEGAL FORMAT IN TYPE STATEMENT

ILLEGAL MARK IN COLUMN SIX

ILLEGAL REPLACEMENT APPEARS IN AN EXPRESSION

ILLEGAL REPLACEMENT IN ARITHMETIC STATEMENT

ILLEGAL SEQUENCE OR USE OF OPERATORS

ILLEGAL SUBSCRIPT IN I/O DATA LIST

ILLEGAL TYPE DECLARED

ILLEGAL USE OF A PERIOD
ILLEGAL USE OF A REPLACEMENT STATEMENT
ILLEGAL USE OF AN OPERATOR
ILLEGAL USE OF FORMAL PARAMETER IN DATA STATEMENT
ILLEGAL VARIABLE LIST FORMAT IN DATA STATEMENT
ILLEGAL VARIABLE NAME IN TYPE STATEMENT
IMPLICIT MUST BE DEFINED BEFORE ALL STATEMENTS EXCEPT
PARAMETER
IMPLIED DO DEPTH EXCEEDED
IMPROPER INDEXING IN A DO STATEMENT
IMPROPER LENGTH OF A HOLLERITH CONSTANT
IMPROPER SUBSCRIPT IN AN INPUT OR OUTPUT STATEMENT
IMPROPER TERMINATION OF OVERLAY STATEMENT
IMPROPER USE OF A FUNCTION NAME
IMPROPER USE OF A PROGRAM, SUBROUTINE, OR FUNCTION NAME
IMPROPER USE OF FUNCTION NAME
IMPROPERLY FORMED IMPLIED DO
IN COMPUTED GO-TO ONLY INTEGER, REAL, OR DOUBLE PRECISION
EXPRESSIONS ARE VALID
INCORRECT ARITHMETIC SUB-EXP IN BOOLEAN EXP
INCORRECT FORM FOR THE ENTRY STATEMENT
INCORRECT LOGICAL EXPRESSION
INCORRECT RELATIONAL SUB-EXPRESSION
INPUT OF DATA INTO A CONSTANT IS ILLEGAL
INQUIRE STATEMENT - ERROR IN (EXIST). MUST BE EITHER .TRUE.
OR .FALSE.
INQUIRE STATEMENT - ERROR IN (NAME)
INTRINSIC FUNCTION HAS TOO MANY ARGUMENTS
INTRINSIC FUNCTION HAS WRONG NUMBER OF ARGUMENTS
INTRINSIC OR LIBRARY FUNCTION CAN NOT BE DECLARED EXTERNAL
IT IS NOT STANDARD TO INCLUDE DATA FROM LABELED COMMON OTHER
IN BLOCK DATA

**ERRORS FATAL TO
EXECUTION**
(continued)

LABEL MUST BE INTEGER CONSTANT
LEFT PAREN MUST START ARGUMENT LIST
LEFT PAREN OR COMMA MISSING IN DATA STATEMENT
LEFT PARENTHESIS MISSING IN EQUIVALENCE STATEMENT
LEFT PARENTHESIS MISSING IN OVERLAY STATEMENT
LEFT SIDE REPLACEMENT SYMBOL IN PARAMETER MUST BE A SYMBOLIC NAME
LETTER SEQUENCE IN IMPLICIT STATEMENT NOT IN ALPHABETIC ORDER
LIST ITEMS AND CONSTANTS NOT 1 TO 1 CORRESPONDENCE
LOGICAL CONNECTIVE MUST BE FOLLOWED BY (OR AN OPERAND
LOGICAL OPERATOR INCORRECTLY USED
LOGICAL SUB-EXPRESSION MAY NOT BEGIN WITH AN OPERATOR

MAIN PROGRAMS SHOULD NOT CONTAIN ENTRY STATEMENTS
MASKING EXPRESSIONS MAY NOT CONTAIN ARITHMETIC OPERATORS
MASKING OPNDS MUST BE REAL OR INTEGER
MISSING OR MISPLACED SLASH IN DATA STATEMENT
MIXED MODE-TYPE 5 AND/OR 6 AND/OR 7
MODE MUST BE DEFINED FOR BUFFER STATEMENT
MORE THAN 64 PARAMETERS IN CALL OR FUNCTION
MULTIPLE DATA TO NON-DIMENSIONED VARIABLE

NAME NOT STARTING WITH AN ALPHABETIC CHARACTER
NEGATIVE DO COUNT NOT ALLOWED IN IMPLIED DO
NO PARENTHESIS AFTER ARRAY NAME IN A DIMENSION STATEMENT
NO SEPARATOR IN COMMON STATEMENT
NO SLASH (/) SEPARATOR IN BLOCK DESIGNATION
NO STATEMENT TO DEFINE COUNT
NO TERMINAL PARENTHESIS IN DIMENSION STATEMENT
NO TERMINAL RIGHT PARENTHESIS IN COMMON STATEMENT
NON-CONSTANT DATA IN DATA STATEMENT
NON-CONSTANT IN DATA STATEMENT DO LOGIC
NON-CONSTANT OVERLAY NUMBER
NON-CONSTANT SUBSCRIPT IN COMMON STATE
NON-CONSTANT SUBSCRIPT IN EQUIVALENCE

NON-DIMENSIONED VARIABLE IN DATA LIST
NON-NCAR STANDARD INDEXING
.NOT. MUST BE FOLLOWED BY (OR OPERAND
NOT STANDARD TO MIX TYPE LOGICAL DATA AND CONSTANT
NUMBERED COMMON NAME .GT. 99999999

OPEN STATEMENT - ERROR IN (ACCESS). VALUE MUST BE DIRECT,
SEQUENTIAL, OR MIXED
OPEN STATEMENT - ERROR IN (EXPDT). MUST BE ALPHANUMERIC DATE
OPEN STATEMENT - ERROR IN (FILE). MUST BE 17 CHARACTERS OR
LESS ALPHANUMERIC CONSTANT OR VAR
OPEN STATEMENT - ERROR IN (FILESEQ)
OPEN STATEMENT - ERROR IN (FORM)
OPEN STATEMENT - ERROR IN (STATUS)
OPERAND MAY BE FOLLOWED BY OPERATOR OR)
OUTPUT OF TYPE 5,6, OR 7 PROHIBITED
OVERLAY NUMBER .GT. 20B

PARAMETER NAME HAS BEEN PREVIOUSLY DEFINED
PAREN GROUP NOT CLOSED IN PREAMBLE LIST
PARENTHESIS USAGE OR DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL
IN I/O DATA LIST
PARENTHEICAL EXPRESSION VOID
PARITY IN I/O STATEMENT NOT 0 OR 1
POSSIBLE COMPILER ERROR
POSSIBLE MACHINE ERROR

REPEAT COUNT IN DATA STATEMENT OUT OF RANGE
REPLACE NOT A VARIABLE
REPLACEMENT VARIABLE MUST BE REAL OR INTEGER
RETURN NOT ALLOWED IN MAIN PROGRAM
RIGHT PARENTHESIS MISSING IN EQUIVALENCE STATEMENT OR
ILLEGAL SUBSCRIPT

**ERRORS FATAL TO
EXECUTION**

(continued)

SEGMENT NUMBER .GT. 20B
SEGMENT NUMBER MUST BE A CONSTANT
STATEMENT ENDS WITH AN ASTERISK
STATEMENT LABEL INCORRECT
STATEMENT LABELS IN GO TO REQUIRE INTEGER VARIABLES
STATEMENT NUMBER TOO LARGE
STATEMENT TOO LONG
SUBROUTINE NAME USED AS A VARIABLE
SUBSCRIPT ON A NON-DIMENSIONED VARIABLE
SUBSCRIPT VALUE EXCEEDS MACHINE LIMITS. LOOK FOR VARIABLE
WITH SUBSCRIPT SIZE .GT. 60000
SUBSCRIPTED SUBSCRIPTS NOT ALLOWED
SUBSCRIPTED VARIABLE IN DATA STATEMENT NOT DIMENSIONED
SUCCESSIVE COMMAS OR IMPROPER CONSTANT IN PARAMETER LIST
SYNTAX ERROR IN COMPUTED GO-TO. COMMA OR PAREN LIKELY
SYNTAX OF IMPLICIT STATEMENT IS NOT CORRECT

TAPE PARAMETER MUST BE SIMPLE INTEGER VARIABLE OR INTEGER
CONSTANT
THE ARGUMENT MUST BE AN INTEGER
THE IF STATEMENT UNRECOGNIZED
THE LEFT HAND SIDE OF A REPLACEMENT STATEMENT IS MISSING
THE NESTING CAPACITY OF THE COMPILER HAS BEEN EXCEEDED
THE OBJECT OF A COMPUTED GO TO MUST BE AN INTEGER CONSTANT
OR VARIABLE
THE OBJECT OF AN ASSIGN OR ASSIGNED GO MUST BE A SIMPLE
INTEGER VARIABLE
THE PARAMETER TO THIS STATEMENT MUST BE AN INTEGER CONSTANT
OR VARIABLE
THE PARAMETERS OF A DO LOOP MUST BE AN UNSIGNED INTEGER
CONSTANT OR SIMPLE INTEGER VARIABLE
THE RUNNING SUBSCRIPT IN A DO LOOP MUST BE A SIMPLE INTEGER
VARIABLE
THE TERMINAL LABEL OF A DO MUST BE AN INTEGER CONSTANT
THIS FORMAT HAS NO STATEMENT NUMBER
THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES

TOO MANY ARGUMENTS IN PREAMBLE LIST
TOO MANY CHARACTERS IN AN IDENTIFIER
TOO MANY RELATIONAL OPERATORS
TOO MANY SUBSCRIPT INDICES
TYPE COMPLEX IN IMPLICIT IS MULTIPLY DEFINED
TYPE DECLARATION NOT RECOGNIZED IN IMPLICIT STATEMENT
TYPE DOUBLE IN IMPLICIT IS MULTIPLY DEFINED
TYPE LOGICAL IN IMPLICIT IS MULTIPLY DEFINED
TYPE REAL OR INTEGER IN IMPLICIT IS MULTIPLY DEFINED

UNDEFINED SEPARATOR IN COMMON STATEMENT
UNIT MUST BE DEFINED FOR BUFFER STATEMENT
UNIT NUMBER INCORRECT
UNIT NUMBER MUST BE FOLLOWED BY A)
UNIT VALUE NOT INTEGER VARIABLE OR INTEGER CONSTANT
UNRECOGNIZABLE I/O STATEMENT
UNRECOGNIZED DELIMITER IN PARAMETER = EXPECTING A COMMA OR EQUALS SIGN
UNRECOGNIZED KEYWORD, SCAN TERMINATED
UNRECOGNIZED STATEMENT

VARIABLE DIMENSION ERROR
VARIABLE DIMENSIONED IDENTIFIER NOT IN FORMAL PARAMETER LIST
VARIABLE EQUIVALENCED TO ITSELF + N
VARIABLE IDENTIFIER IN EXTERNAL STATEMENT

WRONG FORMAT OF I/O STATEMENT, DATA LIST WAS NOT YET PROCESSED

ZERO IS NOT A VALID STATEMENT NUMBER

1ST (through 31ST) EXPRESSION NOT RECOGNIZABLE
1ST (through 31ST) POSITION KEYWORD UNRECOGNIZABLE

NONFATAL ERRORS	A FORMAT SPECIFICATION HAS NO FIELD WIDTH ARGUMENT NOT DEFINED IN I/O LIST
	E OR F OR D DECIMAL FIELD UNDEFINED ENCODE/DECODE APPEARS WITHOUT A LIST END MUST REFER TO STATEMENT LABEL ERR MUST REFER TO STATEMENT LABEL ERR STATEMENT LABEL OUT OF RANGE
	FORMAT MUST BE INTEGER CONSTANT FORMAT NOT TERMINATED BY RIGHT PARENTHESIS
	IN EW.D OR DW.D FORMAT SPECIFICATION, W SHOULD BE .GE. D+7 INQUIRE STATEMENT - ERROR IN (ACCESS). VALUE MUST BE DIRECT, SEQUENTIAL, OR MIXED
	INQUIRE STATEMENT - ERROR IN (EXPDT)
	INQUIRE STATEMENT - ERROR IN (EXISTS). MUST BE EITHER .TRUE. OR .FALSE.
	INQUIRE STATEMENT - ERROR IN (FILESEQ)
	INQUIRE STATEMENT - ERROR IN (GEN)
	INQUIRE STATEMENT - ERROR IN (MODE)
	INQUIRE STATEMENT - ERROR IN (NAMED). IF FILE HAS A NAME, (NAMED) MUST BE .TRUE.
	INQUIRE STATEMENT - ERROR IN (NEXTREC)
	INQUIRE STATEMENT - ERROR IN (NSTATE)
	INQUIRE STATEMENT - ERROR IN (NUMBER)
	INQUIRE STATEMENT - ERROR IN (OPENED). MUST BE EITHER .TRUE. OR .FALSE.
	INQUIRE STATEMENT - ERROR IN (RECL). MUST BE AN INTEGER VARIABLE
	INQUIRE STATEMENT - ERROR IN (SEQFIL)
	INVALID CHARACTER IN A FORMAT SPECIFICATION
	OPEN STATEMENT - ERROR IN (ERR). MUST BE STATEMENT LABEL

SETPASS - CPASS KEYWORD IN ERROR
SETPASS - PASS KEYWORD IN ERROR
SETPASS - RPASS KEYWORD IN ERROR
SETPASS - WPASS KEYWORD IN ERROR

THE FIRST ELEMENT IN A FORMAT STATEMENT MUST BE A (
THERE IS NO PATH TO THIS STATEMENT
THERE MUST BE A SEPARATOR BETWEEN FIELD SPECIFICATIONS
THIS CONSTANT STATEMENT PRECEDES DO

UNIT NOT DEFINED

2 BRANCH IF(UNIT) TEST WILL CAUSE UNPREDICTABLE RESULTS IN
CASE OF PARITY ERROR, ADVISE USING 4 BRANCH TEST

**COMPILER TABLE
SIZE LIMITS**

Several of the compiler diagnostics above indicate that a compiler table limit has been exceeded; e.g., COMPILER-TOO MANY FUNCTIONS. The following table gives the table sizes for both the FORTRAN and FORTRAN1 compilers. This table shows that certain of these errors can be eliminated by using the FORTRAN1 compiler. This is done by replacing *FORTRAN control cards with *FORTRAN1.

Table 11-1. Compiler Table Size Limits †

DESCRIPTION	LIMIT	
	FORTRAN	FORTAN1
COMMON block names	64	64
Constants/routine	592	2048
Dimensioned variables/routine	284	512
D0-loop nests	50	50
Equivalenced variables/routine	192	192
Format statements/routine	64	256
Functions/routine	256	512
Index functions/routine	224	224
Index variables/routine	192	192
Nonstandard indices/routine	32	128
Parameters in parameter list	64	64
Typed variables/routine	128	128
Variables in COMMON blocks	284	512

† These table size limits are current as of the publication of this manual. They are subject to change.

XII

APPENDICES

APPENDICES

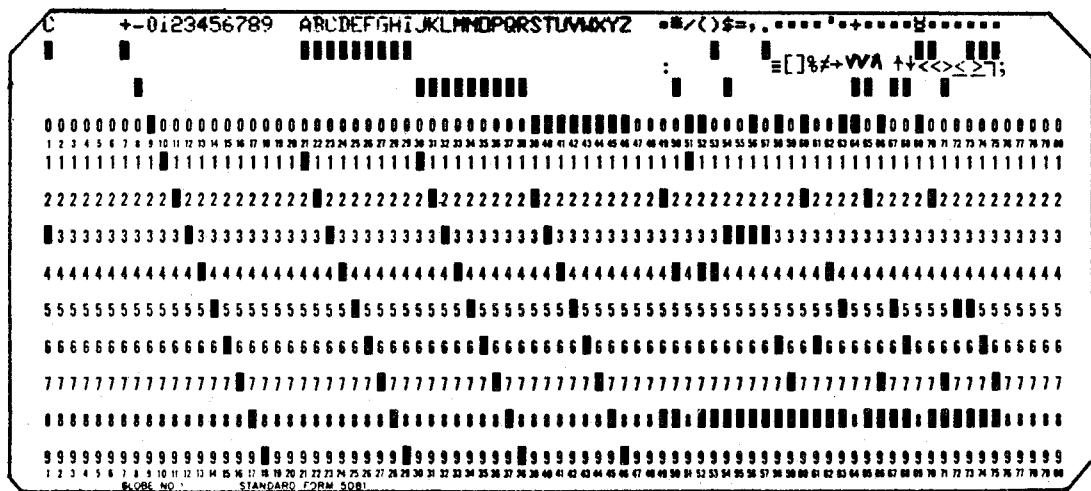
Appendix A:	7600 Series Character Set	12.2
Appendix B:	Table of Powers of Two	12.4
Appendix C:	Octal-Decimal Integer Conversion Table	12.5
Appendix D:	Internal Floating-Point Decimal Table	12.9
Appendix E:	Decimal/Binary Position Table	12.10
Appendix F:	Computer Word Structure of Constants--7600	12.11
Appendix G:	Computer Word Structure of Constants--CRAY-1	12.12
Appendix H:	FORTRAN Optimization Modifiers	12.13

APPENDIX A: CONTROL DATA 7600 SERIES CHARACTER SET

<i>Source Language Character</i>	<i>Display Code (DPC)</i>	<i>External BCD Code</i>	<i>Punch Position in a Hollerith Card Column</i>
:	00	00	8-2
A	01	61	12-1
B	02	62	12-2
C	03	63	12-3
D	04	64	12-4
E	05	65	12-5
F	06	66	12-6
G	07	67	12-7
H	10	70	12-8
I	11	71	12-9
J	12	41	11-1
K	13	42	11-2
L	14	43	11-3
M	15	44	11-4
N	16	45	11-5
O	17	46	11-6
P	20	47	11-7
Q	21	50	11-8
R	22	51	11-9
S	23	22	0-2
T	24	23	0-3
U	25	24	0-4
V	26	25	0-5
W	27	26	0-6
X	30	27	0-7
Y	31	30	0-8
Z	32	31	0-9
0	33	12	0
1	34	01	1
2	35	02	2
3	36	03	3
4	37	04	4
5	40	05	5
6	41	06	6
7	42	07	7

12.3
Appendices

<u>Source Language Character</u>	<u>Display Code (DPC)</u>	<u>External BCD Code</u>	<u>Punch Position in a Hollerith Card Column</u>
8	43	10	8
9	44	11	9
+	45	60	12
-	46	40	11
*	47	54	11-8-4
/	50	21	0-1
(51	34	0-8-4
)	52	74	12-8-4
\$	53	53	11-8-3
=	54	13	8-3
space	55	20	space
,	56	33	0-8-3
.	57	73	12-8-3
≡	60	36	0-8-6
[61	17	8-7
]	62	32	0-8-2
%	63	16	8-6
#	64	14	8-4
→	65	35	0-8-5
▼	66	52	11-0 +
^	67	37	0-8-7
↑	70	55	11-8-5
↓	71	56	11-8-6
<	72	72	12-0 ++
>	73	57	11-8-7
≤	74	15	8-5
≥	75	75	12-8-5
¬	76	76	12-8-6
:	77	77	12-8-7



+ 11-0 and 11-8-2 are equivalent

++ 12-0 and 12-8-2 are equivalent

APPENDIX B: TABLE OF POWERS OF TWO

2^n	n	2^n
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 868 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 630 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 000 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625

APPENDIX C: OCTAL-DECIMAL INTEGER CONVERSION TABLE

		0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	
0000	0000	0000	0001	0002	0003	0004	0005	0006	0007	0400	0256	0257	0258	0259	0260	0261	0262	0263	
to	to	0010	0008	0009	0010	0011	0012	0013	0014	0410	0264	0265	0266	0267	0268	0269	0270	0271	
0777	0511	0020	0016	0017	0018	0019	0020	0021	0022	0420	0272	0273	0274	0275	0276	0277	0278	0279	
{Octal}	(Decimal)	0030	0024	0025	0026	0027	0028	0029	0030	031	0430	0280	0281	0282	0283	0284	0285	0286	0287
		0040	0032	0033	0034	0035	0036	0037	0038	039	0440	0288	0289	0290	0291	0292	0293	0294	0295
		0050	0040	0041	0042	0043	0044	0045	0046	047	0450	0296	0297	0298	0299	0300	0301	0302	0303
		0060	0048	0049	0050	0051	0052	0053	0054	055	0460	0304	0305	0306	0307	0308	0309	0310	0311
Octal	Decimal	0070	0056	0057	0058	0059	0060	0061	0062	0063	0470	0312	0313	0314	0315	0316	0317	0318	0319
10000 -	4096	0100	0064	0065	0066	0067	0068	0069	0070	0071	0500	0320	0321	0322	0323	0324	0325	0326	0327
20000 -	8192	0110	0072	0073	0074	0075	0076	0077	0078	0079	0510	0328	0329	0330	0331	0332	0333	0334	0335
30000 -	12288	0120	0080	0081	0082	0083	0084	0085	0086	0087	0520	0336	0337	0338	0339	0340	0341	0342	0343
40000 -	16384	0130	0088	0089	0090	0091	0092	0093	0094	0095	0530	0344	0345	0346	0347	0348	0349	0350	0351
50000 -	20480	0140	0096	0097	0098	0099	0100	0101	0102	0103	0540	0352	0353	0354	0355	0356	0357	0358	0359
60000 -	24576	0150	0104	0105	0106	0107	0108	0109	0110	0111	0550	0360	0361	0362	0363	0364	0365	0366	0367
70000 -	28672	0160	0112	0113	0114	0115	0116	0117	0118	0119	0560	0368	0369	0370	0371	0372	0373	0374	0375
		0170	0120	0121	0122	0123	0124	0125	0126	0127	0570	0376	0377	0378	0379	0380	0381	0382	0383
		0200	0128	0129	0130	0131	0132	0133	0134	0135	0600	0384	0385	0386	0387	0388	0389	0390	0391
		0210	0136	0137	0138	0139	0140	0141	0142	0143	0610	0392	0393	0394	0395	0396	0397	0398	0399
		0220	0144	0145	0146	0147	0148	0149	0150	0151	0620	0400	0401	0402	0403	0404	0405	0406	0407
		0230	0152	0153	0154	0155	0156	0157	0158	0159	0630	0408	0409	0410	0411	0412	0413	0414	0415
		0240	0160	0161	0162	0163	0164	0165	0166	0167	0640	0416	0417	0418	0419	0420	0421	0422	0423
		0250	0168	0169	0170	0171	0172	0173	0174	0175	0650	0424	0425	0426	0427	0428	0429	0430	0431
		0260	0176	0177	0178	0179	0180	0181	0182	0183	0660	0432	0433	0434	0435	0436	0437	0438	0439
		0270	0184	0185	0186	0187	0188	0189	0190	0191	0670	0440	0441	0442	0443	0444	0445	0446	0447
		0300	0192	0193	0194	0195	0196	0197	0198	0199	0700	0448	0449	0450	0451	0452	0453	0454	0455
		0310	0200	0201	0202	0203	0204	0205	0206	0207	0710	0456	0457	0458	0459	0460	0461	0462	0463
		0320	0208	0209	0210	0211	0212	0213	0214	0215	0720	0464	0465	0466	0467	0468	0469	0470	0471
		0330	0216	0217	0218	0219	0220	0221	0222	0223	0730	0472	0473	0474	0475	0476	0477	0478	0479
		0340	0224	0225	0226	0227	0228	0229	0230	0231	0740	0480	0481	0482	0483	0484	0485	0486	0487
		0350	0232	0233	0234	0235	0236	0237	0238	0239	0750	0488	0489	0490	0491	0492	0493	0494	0495
		0360	0240	0241	0242	0243	0244	0245	0246	0247	0760	0496	0497	0498	0499	0500	0501	0502	0503
		0370	0248	0249	0250	0251	0252	0253	0254	0255	0770	0504	0505	0506	0507	0508	0509	0510	0511
1000	0512	1000	0512	0513	0514	0515	0516	0517	0518	0519	1400	0768	0769	0770	0771	0772	0773	0774	0775
to	to	1010	0520	0521	0522	0523	0524	0525	0526	0527	1410	0776	0777	0778	0779	0780	0781	0782	0783
1777	1023	1020	0528	0529	0530	0531	0532	0533	0534	0535	1420	0784	0785	0786	0787	0788	0789	0790	0791
{Octal}	(Decimal)	1030	0536	0537	0538	0539	0540	0541	0542	0543	1430	0792	0793	0794	0795	0796	0797	0798	0799
		1040	0544	0545	0546	0547	0548	0549	0550	0551	1440	0800	0801	0802	0803	0804	0805	0806	0807
		1050	0552	0553	0554	0555	0556	0557	0558	0559	1450	0808	0809	0810	0811	0812	0813	0814	0815
		1060	0560	0561	0562	0563	0564	0565	0566	0567	1460	0816	0817	0818	0819	0820	0821	0822	0823
		1070	0568	0569	0570	0571	0572	0573	0574	0575	1470	0824	0825	0826	0827	0828	0829	0830	0831
		1100	0576	0577	0578	0579	0580	0581	0582	0583	1500	0832	0833	0834	0835	0836	0837	0838	0839
		1110	0584	0585	0586	0587	0588	0589	0590	0591	1510	0840	0841	0842	0843	0844	0845	0846	0847
		1120	0592	0593	0594	0595	0596	0597	0598	0599	1520	0848	0849	0850	0851	0852	0853	0854	0855
		1130	0600	0601	0602	0603	0604	0605	0606	0607	1530	0856	0857	0858	0859	0860	0861	0862	0863
		1140	0608	0609	0610	0611	0612	0613	0614	0615	1540	0864	0865	0866	0867	0868	0869	0870	0871
		1150	0616	0617	0618	0619	0620	0621	0622	0623	1550	0872	0873	0874	0875	0876	0877	0878	0879
		1160	0624	0625	0626	0627	0628	0629	0630	0631	1560	0880	0881	0882	0883	0884	0885	0886	0887
		1170	0632	0633	0634	0635	0636	0637	0638	0639	1570	0888	0889	0890	0891	0892	0893	0894	0895
		1200	0640	0641	0642	0643	0644	0645	0646	0647	1600	0896	0897	0898	0899	0900	0901	0902	0903
		1210	0648	0649	0650	0651	0652	0653	0654	0655	1610	0904	0905	0906	0907	0908	0909	0910	0911
		1220	0656	0657	0658	0659	0660	0661	0662	0663	1620	0912	0913	0914	0915	0916	0917	0918	0919
		1230	0664	0665	0666	0667	0668	0669	0670	0671	1630	0920	0921	0922	0923	0924	0925	0926	0927
		1240	0672	0673	0674	0675	0676	0677	0678	0679	1640	0928	0929	0930	0931	0932	0933	0934	0935
		1250	0680	0681	0682	0683	0684	0685	0686	0687	1650	0936	0937	0938	0939	0940	0941	0942	0943
		1260	0688	0689	0690	0691	0692	0693	0694	0695	1660	0944	0945	0946	0947	0948	0949	0950	0951
		1270	0696	0697	0698	0699	0700	0701	0702	0703	1670	0952	0953	0954	0955	0956	0957	0958	0959
		1300	0704	0705	0706	0707	0708	0709	0710	0711	1700	0960	0961	0962	0963	0964	0965	0966	0967
		1310	0712	0713	0714	0715	0716	0717	0718	0719	1710	0968	0969	0970	0971	0972	0973	0974	0975
		1320	0720	0721	0722	0723	0724	0725	0726	0727	1720	0976	0977	0978	0979	0980	0981	0982	0983
		1330	0728	0729	0730	0731	0732	0733	0734	0735	1730	0984	0985	0986	0987	0988	0989	0990	0991
		1340	0736	0737	0738	0739	0740	0741	0742	0743	1740	0992	0993	0994	0995	0996	0997	0998	0999
		1350	0744	0745	0746	0747	0748	0749	0750	0751	1750	1000	1001	1002	1003	1004	1005	1006	1007

APPENDIX C: OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	2000	1024
2000	1024	1025	1026	1027	1028	1029	1030	1031	2400	1280	1281	1282	1283	1284	1285	1286	1287
2010	1032	1033	1034	1035	1036	1037	1038	1039	2410	1288	1289	1290	1291	1292	1293	1294	1295
2020	1040	1041	1042	1043	1044	1045	1046	1047	2420	1296	1297	1298	1299	1300	1301	1302	1303
2030	1048	1049	1050	1051	1052	1053	1054	1055	2430	1304	1305	1306	1307	1308	1309	1310	1311
2040	1056	1057	1058	1059	1060	1061	1062	1063	2440	1312	1313	1314	1315	1316	1317	1318	1319
2050	1064	1065	1066	1067	1068	1069	1070	1071	2450	1320	1321	1322	1323	1324	1325	1326	1327
2060	1072	1073	1074	1075	1076	1077	1078	1079	2460	1328	1329	1330	1331	1332	1333	1334	1335
2070	1080	1081	1082	1083	1084	1085	1086	1087	2470	1336	1337	1338	1339	1340	1341	1342	1343
2100	1088	1089	1090	1091	1092	1093	1094	1095	2500	1344	1345	1346	1347	1348	1349	1350	1351
2110	1096	1097	1098	1099	1100	1101	1102	1103	2510	1352	1353	1354	1355	1356	1357	1358	1359
2120	1104	1105	1106	1107	1108	1109	1110	1111	2520	1360	1361	1362	1363	1364	1365	1366	1367
2130	1112	1113	1114	1115	1116	1117	1118	1119	2530	1368	1369	1370	1371	1372	1373	1374	1375
2140	1120	1121	1122	1123	1124	1125	1126	1127	2540	1376	1377	1378	1379	1380	1381	1382	1383
2150	1128	1129	1130	1131	1132	1133	1134	1135	2550	1384	1385	1386	1387	1388	1389	1390	1391
2160	1136	1137	1138	1139	1140	1141	1142	1143	2560	1392	1393	1394	1395	1396	1397	1398	1399
2170	1144	1145	1146	1147	1148	1149	1150	1151	2570	1400	1401	1402	1403	1404	1405	1406	1407
2200	1152	1153	1154	1155	1156	1157	1158	1159	2600	1408	1409	1410	1411	1412	1413	1414	1415
2210	1160	1161	1162	1163	1164	1165	1166	1167	2610	1416	1417	1418	1419	1420	1421	1422	1423
2220	1168	1169	1170	1171	1172	1173	1174	1175	2620	1424	1425	1426	1427	1428	1429	1430	1431
2230	1176	1177	1178	1179	1180	1181	1182	1183	2630	1432	1433	1434	1435	1436	1437	1438	1439
2240	1184	1185	1186	1187	1188	1189	1190	1191	2640	1440	1441	1442	1443	1444	1445	1446	1447
2250	1192	1193	1194	1195	1196	1197	1198	1199	2650	1448	1449	1450	1451	1452	1453	1454	1455
2260	1200	1201	1202	1203	1204	1205	1206	1207	2660	1456	1457	1458	1459	1460	1461	1462	1463
2270	1208	1209	1210	1211	1212	1213	1214	1215	2670	1464	1465	1466	1467	1468	1469	1470	1471
2300	1216	1217	1218	1219	1220	1221	1222	1223	2700	1472	1473	1474	1475	1476	1477	1478	1479
2310	1224	1225	1226	1227	1228	1229	1230	1231	2710	1480	1481	1482	1483	1484	1485	1486	1487
2320	1232	1233	1234	1235	1236	1237	1238	1239	2720	1488	1489	1490	1491	1492	1493	1494	1495
2330	1240	1241	1242	1243	1244	1245	1246	1247	2730	1496	1497	1498	1499	1500	1501	1502	1503
2340	1248	1249	1250	1251	1252	1253	1254	1255	2740	1504	1505	1506	1507	1508	1519	1510	1511
2350	1256	1257	1258	1259	1260	1261	1262	1263	2750	1512	1513	1514	1515	1516	1517	1518	1519
2360	1264	1265	1266	1267	1268	1269	1270	1271	2760	1520	1521	1522	1523	1524	1525	1526	1527
2370	1272	1273	1274	1275	1276	1277	1278	1279	2770	1528	1529	1530	1531	1532	1533	1534	1535
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	3000	1536
3000	1536	1537	1538	1539	1540	1541	1542	1543	3400	1792	1793	1794	1795	1796	1797	1798	1799
3010	1544	1545	1546	1547	1548	1549	1550	1551	3410	1800	1801	1802	1803	1804	1805	1806	1807
3020	1552	1553	1554	1555	1556	1557	1558	1559	3420	1808	1809	1810	1811	1812	1813	1814	1815
3030	1560	1561	1562	1563	1564	1565	1566	1567	3430	1816	1817	1818	1819	1820	1821	1822	1823
3040	1568	1569	1570	1571	1572	1573	1574	1575	3440	1824	1825	1826	1827	1828	1829	1830	1831
3050	1576	1577	1578	1579	1580	1581	1582	1583	3450	1832	1833	1834	1835	1836	1837	1838	1839
3060	1584	1585	1586	1587	1588	1589	1590	1591	3460	1840	1841	1842	1843	1844	1845	1846	1847
3070	1592	1593	1594	1595	1596	1597	1598	1599	3470	1848	1849	1850	1851	1852	1853	1854	1855
3100	1600	1601	1602	1603	1604	1605	1606	1607	3500	1856	1857	1858	1859	1860	1861	1862	1863
3110	1608	1609	1610	1611	1612	1613	1614	1615	3510	1864	1865	1866	1867	1868	1869	1870	1871
3120	1616	1617	1618	1619	1620	1621	1622	1623	3520	1872	1873	1874	1875	1876	1877	1878	1879
3130	1624	1625	1626	1627	1628	1629	1630	1631	3530	1880	1881	1882	1883	1884	1885	1886	1887
3140	1632	1633	1634	1635	1636	1637	1638	1639	3540	1888	1889	1890	1891	1892	1893	1894	1895
3150	1640	1641	1642	1643	1644	1645	1646	1647	3550	1896	1897	1898	1899	1900	1901	1902	1903
3160	1648	1649	1650	1651	1652	1653	1654	1655	3560	1904	1905	1906	1907	1908	1909	1910	1911
3170	1656	1657	1658	1659	1660	1661	1662	1663	3570	1912	1913	1914	1915	1916	1917	1918	1919
3200	1664	1665	1666	1667	1668	1669	1670	1671	3600	1920	1921	1922	1923	1924	1925	1926	1927
3210	1672	1673	1674	1675	1676	1677	1678	1679	3610	1928	1929	1930	1931	1932	1933	1934	1935
3220	1680	1681	1682	1683	1684	1685	1686	1687	3620	1936	1937	1938	1939	1940	1941	1942	1943
3230	1688	1689	1690	1691	1692	1693	1694	1695	3630	1944	1945	1946	1947	1948	1949	1950	1951
3240	1696	1697	1698	1699	1700	1701	1702	1703	3640	1952	1953	1954	1955	1956	1957	1958	1959
3250	1704	1705	1706	1707	1708	1709	1710	1711	3650	1960	1961	1962	1963	1964	1965	1966	1967
3260	1712	1713	1714	1715	1716	1717	1718	1719	3660	1968	1969	1970	1971	1972	1973	1974	1975
3270	1720	1721	1722	1723	1724	1725	1726	1727	3670	1976	1977	1978	1979	1980	1981	1982	1983
3300	1728	1729	1730	1731	1732	1733	1734	1735	3700	1984	1985	1986	1987	1988	1989	1990	1991
3310	1736	1737	1738	1739	1740	1741	1742	1743	3710	1992	1993	1994	1995	1996	1997	1998	1999
3320	1744	1745	1746	1747	1748	1749	1750	1751	3720	2000	2001	2002	2003	2004	2005	2006	2007
3330	1752	1753	1754	1755	1756	1757	1758	1759	3730	2008	2009	2010	2011	2012	2013	2014	2015
3340	1760	1761	1762	1763	1764	1765	1766	1767	3740	2016	2017	2018	2019	2020	2021	2022	2023
3350	1768	1769	1770	1771	1772	1773	1774	1775	3750	2024	2025	2026	2027	2028	2029	2030	2031
3360	1776	1777	1778	1779	1780	1781	1782	1783	3760	2032	2033	2034	2035	2036	2037	2038	2039
3370	1784	1785	1786	1787	1788	1789	1790	1791	3770	2040	2041	2042	2				

APPENDIX C: OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

		Octal								Octal										
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7			
(Octal)	4000	2048	4000	2048	2049	2050	2051	2052	2053	2054	2055	4400	2304	2305	2306	2307	2308	2309	2310	2311
	to	to	4010	2056	2057	2058	2059	2060	2061	2062	2063	4410	2312	2313	2314	2315	2316	2317	2318	2319
	4777	2559	4020	2064	2065	2066	2067	2068	2069	2070	2071	4420	2320	2321	2322	2323	2324	2325	2326	2327
	(Decimal)		4030	2072	2073	2074	2075	2076	2077	2078	2079	4430	2328	2329	2330	2331	2332	2333	2334	2335
	4040	2080	2081	2082	2083	2084	2085	2086	2087		4440	2336	2337	2338	2339	2340	2341	2342	2343	
	4050	2088	2089	2090	2091	2092	2093	2094	2095		4450	2344	2345	2346	2347	2348	2349	2350	2351	
	4060	2096	2097	2098	2099	2100	2101	2102	2103		4460	2352	2353	2354	2355	2356	2357	2358	2359	
	4070	2104	2105	2106	2107	2108	2109	2110	2111		4470	2360	2361	2362	2363	2364	2365	2366	2367	
	10000 - 4096																			
	20000 - 8192		4100	2112	2113	2114	2115	2116	2117	2118	2119	4500	2368	2369	2370	2371	2372	2373	2374	2375
30000 - 12288			4110	2120	2121	2122	2123	2124	2125	2126	2127	4510	2376	2377	2378	2379	2380	2381	2382	2383
40000 - 16384			4120	2128	2129	2130	2131	2132	2133	2134	2135	4520	2384	2385	2386	2387	2388	2389	2390	2391
50000 - 20480			4130	2136	2137	2138	2139	2140	2141	2142	2143	4530	2392	2393	2394	2395	2396	2397	2398	2399
60000 - 24576			4140	2144	2145	2146	2147	2148	2149	2150	2151	4540	2400	2401	2402	2403	2404	2405	2406	2407
70000 - 28672			4150	2152	2153	2154	2155	2156	2157	2158	2159	4550	2408	2409	2410	2411	2412	2413	2414	2415
			4160	2160	2161	2162	2163	2164	2165	2166	2167	4560	2416	2417	2418	2419	2420	2421	2422	2423
			4170	2168	2169	2170	2171	2172	2173	2174	2175	4570	2424	2425	2426	2427	2428	2429	2430	2431
			4200	2176	2177	2178	2179	2180	2181	2182	2183	4600	2432	2433	2434	2435	2436	2437	2438	2439
			4210	2184	2185	2186	2187	2188	2189	2190	2191	4610	2440	2441	2442	2443	2444	2445	2446	2447
			4220	2192	2193	2194	2195	2196	2197	2198	2199	4620	2448	2449	2450	2451	2452	2453	2454	2455
			4230	2200	2201	2202	2203	2204	2205	2206	2207	4630	2456	2457	2458	2459	2460	2461	2462	2463
			4240	2208	2209	2210	2211	2212	2213	2214	2215	4640	2464	2465	2466	2467	2468	2469	2470	2471
			4250	2216	2217	2218	2219	2220	2221	2222	2223	4650	2472	2473	2474	2475	2476	2477	2478	2479
			4260	2224	2225	2226	2227	2228	2229	2230	2231	4660	2480	2481	2482	2483	2484	2485	2486	2487
			4270	2232	2233	2234	2235	2236	2237	2238	2239	4670	2488	2489	2490	2491	2492	2493	2494	2495
			4300	2240	2241	2242	2243	2244	2245	2246	2247	4700	2496	2497	2498	2499	2500	2501	2502	2503
			4310	2248	2249	2250	2251	2252	2253	2254	2255	4710	2504	2505	2506	2507	2508	2509	2510	2511
			4320	2256	2257	2258	2259	2260	2261	2262	2263	4720	2512	2513	2514	2515	2516	2517	2518	2519
			4330	2264	2265	2266	2267	2268	2269	2270	2271	4730	2520	2521	2522	2523	2524	2525	2526	2527
			4340	2272	2273	2274	2275	2276	2277	2278	2279	4740	2528	2529	2530	2531	2532	2533	2534	2535
			4350	2280	2281	2282	2283	2284	2285	2286	2287	4750	2536	2537	2538	2539	2540	2541	2542	2543
			4360	2288	2289	2290	2291	2292	2293	2294	2295	4760	2544	2545	2546	2547	2548	2549	2550	2551
			4370	2296	2297	2298	2299	2300	2301	2302	2303	4770	2552	2553	2554	2555	2556	2557	2558	2559
(Octal)	5000	2560	2561	2562	2563	2564	2565	2566	2567		5400	2816	2817	2818	2819	2820	2821	2822	2823	
	to	to	5010	2568	2569	2570	2571	2572	2573	2574	2575	5410	2824	2825	2826	2827	2828	2829	2830	2831
	5777	3071	5020	2576	2577	2578	2579	2580	2581	2582	2583	5420	2832	2833	2834	2835	2836	2837	2838	2839
	(Decimal)		5030	2584	2585	2586	2587	2588	2589	2590	2591	5430	2840	2841	2842	2843	2844	2845	2846	2847
	5040	2592	2593	2594	2595	2596	2597	2598	2599		5440	2848	2849	2850	2851	2852	2853	2854	2855	
	5050	2600	2601	2602	2603	2604	2605	2606	2607		5450	2856	2857	2858	2859	2860	2861	2862	2863	
	5060	2608	2609	2610	2611	2612	2613	2614	2615		5460	2864	2865	2866	2867	2868	2869	2870	2871	
	5070	2616	2617	2618	2619	2620	2621	2622	2623		5470	2872	2873	2874	2875	2876	2877	2878	2879	
											5500	2880	2881	2882	2883	2884	2885	2886	2887	
											5510	2888	2889	2890	2891	2892	2893	2894	2895	
											5520	2896	2897	2898	2899	2900	2901	2902	2903	
											5530	2904	2905	2906	2907	2908	2909	2910	2911	
											5540	2912	2913	2914	2915	2916	2917	2918	2919	
											5550	2920	2921	2922	2923	2924	2925	2926	2927	
											5560	2928	2929	2930	2931	2932	2933	2934	2935	
											5570	2936	2937	2938	2939	2940	2941	2942	2943	
											5600	2944	2945	2946	2947	2948	2949	2950	2951	
											5610	2952	2953	2954	2955	2956	2957	2958	2959	
											5620	2960	2961	2962	2963	2964	2965	2966	2967	
											5630	2968	2969	2970	2971	2972	2973	2974	2975	
											5640	2976	2977	2978	2979	2980	2981	2982	2983	
											5650	2984	2985	2986	2987	2988	2989	2990	2991	
											5660	2992	2993	2994	2995	2996	2997	2998	2999	
											5670	3000	3001	3002	3003	3004	3005	3006	3007	
											5700	3008	3009	3010	3011	3012	3013	3014	3015	
											5710	3016	3017	3018	3019	3020	3021	3022	3023	
											5720	3024	3025	3026	3027	3028	3029	3030	3031	

APPENDIX C: OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7		
6000	3072	3073	3074	3075	3076	3077	3078	3079	6400	3328	3329	3330	3331	3332	3333	3334	3335	6000	3072	to	to	6777	3583	(Octal)	(Decimal)
6010	3080	3081	3082	3083	3084	3085	3086	3087	6410	3336	3337	3338	3339	3340	3341	3342	3343	6010	3072	to	to	6777	3583	(Octal)	(Decimal)
6020	3088	3089	3090	3091	3092	3093	3094	3095	6420	3344	3345	3346	3347	3348	3349	3350	3351	6020	3072	to	to	6777	3583	(Octal)	(Decimal)
6030	3096	3097	3098	3099	3100	3101	3102	3103	6430	3352	3353	3354	3355	3356	3357	3358	3359	6030	3072	to	to	6777	3583	(Octal)	(Decimal)
6040	3104	3105	3106	3107	3108	3109	3110	3111	6440	3360	3361	3362	3363	3364	3365	3366	3367	6040	3072	to	to	6777	3583	(Octal)	(Decimal)
6050	3112	3113	3114	3115	3116	3117	3118	3119	6450	3368	3369	3370	3371	3372	3373	3374	3375	6050	3072	to	to	6777	3583	(Octal)	(Decimal)
6060	3120	3121	3122	3123	3124	3125	3126	3127	6460	3376	3377	3378	3379	3380	3381	3382	3383	6060	3072	to	to	6777	3583	(Octal)	(Decimal)
6070	3128	3129	3130	3131	3132	3133	3134	3135	6470	3384	3385	3386	3387	3388	3389	3390	3391	6070	3072	to	to	6777	3583	(Octal)	(Decimal)
6100	3136	3137	3138	3139	3140	3141	3142	3143	6500	3392	3393	3394	3395	3396	3397	3398	3399	6100	3072	to	to	6777	3583	(Octal)	(Decimal)
6110	3144	3145	3146	3147	3148	3149	3150	3151	6510	3400	3401	3402	3403	3404	3405	3406	3407	6110	3072	to	to	6777	3583	(Octal)	(Decimal)
6120	3152	3153	3154	3155	3156	3157	3158	3159	6520	3408	3409	3410	3411	3412	3413	3414	3415	6120	3072	to	to	6777	3583	(Octal)	(Decimal)
6130	3160	3161	3162	3163	3164	3165	3166	3167	6530	3416	3417	3418	3419	3420	3421	3422	3423	6130	3072	to	to	6777	3583	(Octal)	(Decimal)
6140	3168	3169	3170	3171	3172	3173	3174	3175	6540	3424	3425	3426	3427	3428	3429	3430	3431	6140	3072	to	to	6777	3583	(Octal)	(Decimal)
6150	3176	3177	3178	3179	3180	3181	3182	3183	6550	3432	3433	3434	3435	3436	3437	3438	3439	6150	3072	to	to	6777	3583	(Octal)	(Decimal)
6160	3184	3185	3186	3187	3188	3189	3190	3191	6560	3440	3441	3442	3443	3444	3445	3446	3447	6160	3072	to	to	6777	3583	(Octal)	(Decimal)
6170	3192	3193	3194	3195	3196	3197	3198	3199	6570	3448	3449	3450	3451	3452	3453	3454	3455	6170	3072	to	to	6777	3583	(Octal)	(Decimal)
6200	3200	3201	3202	3203	3204	3205	3206	3207	6600	3456	3457	3458	3459	3460	3461	3462	3463	6200	3072	to	to	6777	3583	(Octal)	(Decimal)
6210	3208	3209	3210	3211	3212	3213	3214	3215	6610	3464	3465	3466	3467	3468	3469	3470	3471	6210	3072	to	to	6777	3583	(Octal)	(Decimal)
6220	3216	3217	3218	3219	3220	3221	3222	3223	6620	3472	3473	3474	3475	3476	3477	3478	3479	6220	3072	to	to	6777	3583	(Octal)	(Decimal)
6230	3224	3225	3226	3227	3228	3229	3230	3231	6630	3480	3481	3482	3483	3484	3485	3486	3487	6230	3072	to	to	6777	3583	(Octal)	(Decimal)
6240	3232	3233	3234	3235	3236	3237	3238	3239	6640	3488	3489	3490	3491	3492	3493	3494	3495	6240	3072	to	to	6777	3583	(Octal)	(Decimal)
6250	3240	3241	3242	3243	3244	3245	3246	3247	6650	3496	3497	3498	3499	3500	3501	3502	3503	6250	3072	to	to	6777	3583	(Octal)	(Decimal)
6260	3248	3249	3250	3251	3252	3253	3254	3255	6660	3504	3505	3506	3507	3508	3509	3510	3511	6260	3072	to	to	6777	3583	(Octal)	(Decimal)
6270	3256	3257	3258	3259	3260	3261	3262	3263	6670	3512	3513	3514	3515	3516	3517	3518	3519	6270	3072	to	to	6777	3583	(Octal)	(Decimal)
6300	3264	3265	3266	3267	3268	3269	3270	3271	6700	3520	3521	3522	3523	3524	3525	3526	3527	6300	3072	to	to	6777	3583	(Octal)	(Decimal)
6310	3272	3273	3274	3275	3276	3277	3278	3279	6710	3528	3529	3530	3531	3532	3533	3534	3535	6310	3072	to	to	6777	3583	(Octal)	(Decimal)
6320	3280	3281	3282	3283	3284	3285	3286	3287	6720	3536	3537	3538	3539	3540	3541	3542	3543	6320	3072	to	to	6777	3583	(Octal)	(Decimal)
6330	3288	3289	3290	3291	3292	3293	3294	3295	6730	3544	3545	3546	3547	3548	3549	3550	3551	6330	3072	to	to	6777	3583	(Octal)	(Decimal)
6340	3298	3297	3298	3299	3300	3301	3302	3303	6740	3552	3553	3554	3555	3556	3557	3558	3559	6340	3072	to	to	6777	3583	(Octal)	(Decimal)
6350	3304	3305	3306	3307	3308	3309	3310	3311	6750	3560	3561	3562	3563	3564	3565	3566	3567	6350	3072	to	to	6777	3583	(Octal)	(Decimal)
6360	3312	3313	3314	3315	3316	3317	3318	3319	6760	3568	3569	3570	3571	3572	3573	3574	3575	6360	3072	to	to	6777	3583	(Octal)	(Decimal)
6370	3320	3321	3322	3323	3324	3325	3326	3327	6770	3576	3577	3578	3579	3580	3581	3582	3583	6370	3072	to	to	6777	3583	(Octal)	(Decimal)
7000	3584	3585	3586	3587	3588	3589	3590	3591	7400	3840	3841	3842	3843	3844	3845	3846	3847	7000	3584	to	to	7777	4095	(Octal)	(Decimal)
7010	3592	3593	3594	3595	3496	3497	3598	3599	7410	3848	3849	3850	3851	3852	3853	3854	3855	7010	3584	to	to	7777	4095	(Octal)	(Decimal)
7020	3600	3601	3602	3603	3604	3605	3606	3607	7420	3856	3857	3858	3859	3860	3861	3862	3863	7020	3584	to	to	7777	4095	(Octal)	(Decimal)
7030	3608	3609	3610	3611	3612	3613	3614	3615	7430	3864	3865	3866	3867	3868	3869	3870	3871	7030	3584	to	to	7777	4095	(Octal)	(Decimal)
7040	3616	3617	3618	3619	3620	3621	3622	3623	7440	3872	3873	3874	3875	3876	3877	3878	3879	7040	3584	to	to	7777	4095	(Octal)	(Decimal)
7050	3624	3625	3626	3627	3628	3629	3630	3631	7450	3880	3881	3882	3883	3884	3885	3886	3887	7050	3584	to	to	7777	4095	(Octal)	(Decimal)
7060	3632	3633	3634	3635	3636	3637	3638	3639	7460	3888	3889	3890	3891	3892	3893	3894	3895	7060	3584	to	to	7777	4095	(Octal)	(Decimal)
7070	3640	3641	3642	3643	3644	3645	3646	3647	7470	3896	3897	3898	3899	3900	3901	3902	3903	7070	3584	to	to	7777	4095	(Octal)	(Decimal)
7100	3648	3649	3650	3651	3652	3653	3654	3655	7500	3904	3905	3906	3907	3908	3909	3910	3911	7100	3584	to	to	7777	4095	(Octal)	(Decimal)
7110	3656	3657	3658	3659	3660	3661	3662	3663	7510	3912	3913	3914	3915	3916	3917	3918	3919	7110	3584	to	to	7777	4095	(Octal)	(Decimal)
7120	3664	3665	3666	3667	3668	3669	3670	3671	7520	3920	3921	3922	3923	3924	3925	3926	3927	7120	3584	to	to	7777	4095	(Octal)	(Decimal)
7130	3672	3673	3674	3675	3676	3677	3678	3679	7530	3928	3929	3930	3931	3932	3933	3934	3935	7130	3584	to	to	7777	4095	(Octal)	(Decimal)
7140	3680	3681	3682	3683	3684	3685	3686	3687	7540	3936	3937	3938	3939	3940	3941	3942	3943	7140	3584	to	to	7777	4095	(Octal)	(Decimal)
7150	3688	3689	3690	3691	3692	3693	3694	3695	7550	3944	3945	3946	3947	3948	3949	3950	3951	7150	3584	to	to	7777	4095	(Octal)	(Decimal)
7160	3696	3697	3698	3699	3700	3701	3702	3703	7560	3952	3953	3954	3955	3956	3957	3958	3959	7160	3584	to	to	7777	4095	(Octal)	(Decimal)
7170	3704	3705	3706	3707	3708	3709	3710	3711	7570	3960	3961	3962	3963	3964	3965	3966	396								

APPENDIX D: INTERNAL FLOATING-POINT DECIMAL TABLE

<i>Floating Point</i>					<i>Decimal</i>	
1574	7571	3207	5355	1751	1.0	E - 25
1615	5716	2410	3111	1021	1.0	E - 20
1636	4401	6574	7563	5260	1.0	E - 15
1656	6676	3376	6353	6755	1.0	E - 10
1677	5174	2654	2161	5507	1.0	E - 05
1702	6433	3427	2616	1031	1.0	E - 04
1706	4061	1156	4570	6517	1.0	E - 03
1711	5075	2312	1727	0243	1.0	E - 02
1713	6314	6314	6314	6314	5.0	E - 02
1714	6314	6314	6314	6314	1.0	E - 01
1717	4000	0000	0000	0000	5.0	E - 01
1720	4000	0000	0000	0000	1.0	E + 00
1722	5000	0000	0000	0000	5.0	E + 00
1723	5000	0000	0000	0000	1.0	E + 01
1725	6200	0000	0000	0000	5.0	E + 01
1726	6200	0000	0000	0000	1.0	E + 02
1731	7640	0000	0000	0000	1.0	E + 03
1735	4704	0000	0000	0000	1.0	E + 04
1740	6065	0000	0000	0000	1.0	E + 05
1761	4520	1371	0000	0000	1.0	E + 10
2002	7065	7651	1432	0000	1.0	E + 15
2023	5327	4353	6132	6142	1.0	E + 20
2044	4105	4521	3024	0023	1.0	E + 25

Note: A negative number is represented as the 7's complement of a positive number.

Example

7777	7777	7777	7777	7777
+ .01 is 1711	5075	3412	1727	0243
- .01 is 6066	2702	4365	6050	7534

APPENDIX E: DECIMAL/BINARY POSITION TABLE

Largest Decimal Integer	Decimal Digits Req'd*	Number of Binary Digits	Largest Decimal Fraction
1	1	1	.5
3		2	.75
7		3	.875
15		4	.937 5
31	2	5	.968 75
63		6	.984 375
127		7	.992 187 5
255	3	8	.996 093 75
511		9	.998 046 875
1 023		10	.999 023 437 5
2 047	4	11	.999 511 718 75
4 095		12	.999 755 859 375
8 191		13	.999 877 929 687 5
16 383		14	.999 938 964 843 75
32 767	5	15	.999 969 482 421 875
65 535		16	.999 984 741 210 937 5
131 071		17	.999 992 370 605 468 75
262 143	6	18	.999 996 185 302 734 375
524 287		19	.999 998 092 651 367 187 5
1 048 575		20	.999 999 046 325 683 593 75
2 097 151	7	21	.999 999 523 162 841 796 875
4 194 303		22	.999 999 761 581 420 898 437 5
8 388 607		23	.999 999 880 790 710 449 218 75
16 777 215		24	.999 999 940 395 355 244 609 375
33 554 431	8	25	.999 999 970 197 677 612 304 687 5
67 108 863		26	.999 999 985 098 838 806 152 343 75
134 217 727		27	.999 999 992 549 419 403 076 171 875
268 435 455	9	28	.999 999 996 274 709 701 538 085 937 5
536 870 911		29	.999 999 998 137 354 850 769 042 968 75
1 073 741 823		30	.999 999 999 068 677 425 384 521 484 375
2 147 483 647	10	31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 767 169 356 346 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 678 173 065 185 546 875
17 179 869 183		34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 367	11	35	.999 999 999 970 896 169 543 266 296 386 718 75
68 719 476 735		36	.999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471		37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943	12	38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 169 921 875
1 099 511 627 775		40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103	13	42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415		44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663	14	46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327		47	.999 999 999 999 992 894 572 542 398 998 141 288 757 324 218 75

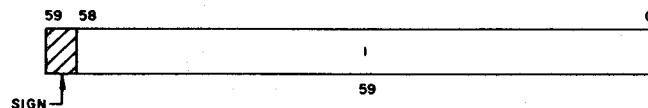
* Larger numbers within a digit group should be checked for exact number of decimal digits required.

Examples of Use

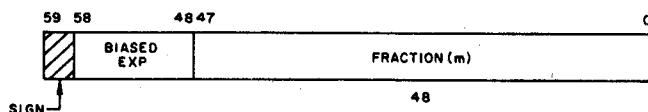
- Q. What is the largest decimal value which can be expressed by 36 binary digits? A. 68,719,476,735.
- Q. How many decimal digits will be required to express a 22-bit number? A. 7 decimal digits.

APPENDIX F: COMPUTER WORD STRUCTURE OF CONSTANTS--7600

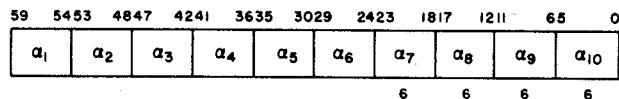
INTEGER



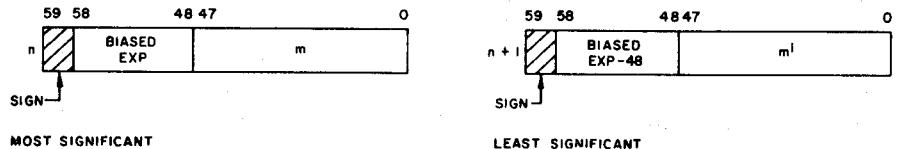
REAL



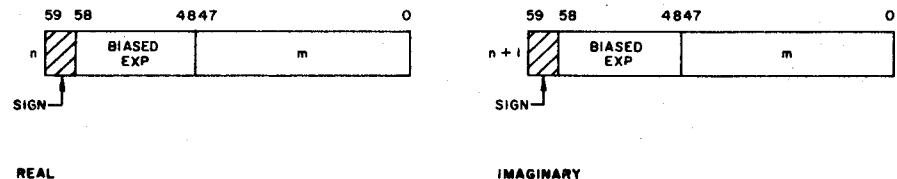
HOLLERITH BCD AND DISPLAY CODE



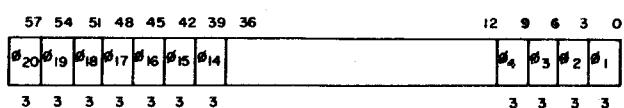
DOUBLE-PRECISION



COMPLEX

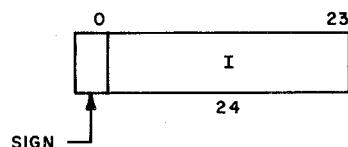


OCTAL

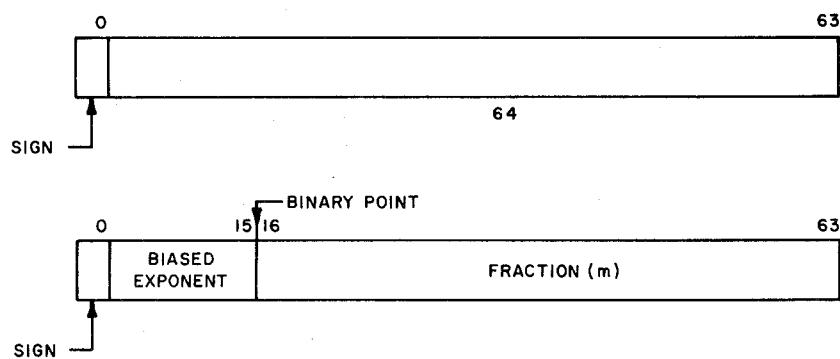


APPENDIX G: COMPUTER WORD STRUCTURE OF CONSTANTS--CRAY-1

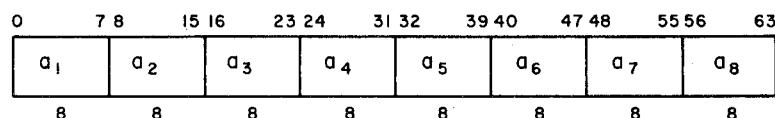
INTEGER



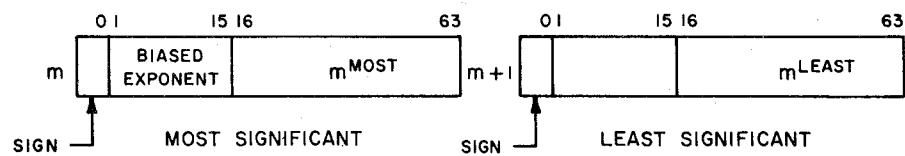
REAL



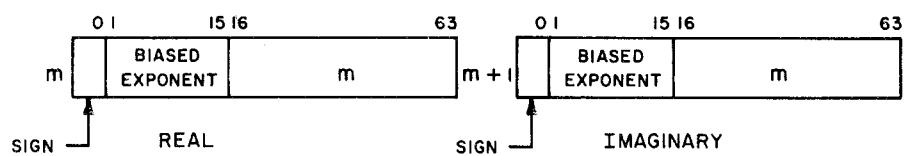
CHARACTER CODE--ASCII



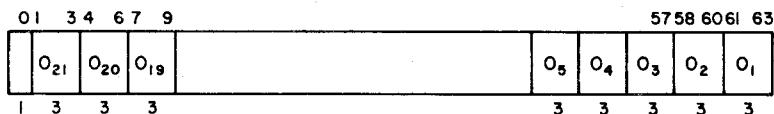
DOUBLE-PRECISION



COMPLEX



OCTAL



APPENDIX H: FORTRAN COMPILE MODIFIERS

*FORTRAN,0	Apply 14, 15, 16 to prevent reordering of arithmetic sequence.
*FORTRAN,1	Do not remove constant statements from DO loops.
*FORTRAN,2	Do not remove constant subexpressions from DO loops.
*FORTRAN,3	Do not precalculate addresses in B-registers.
*FORTRAN,4	Do not optimize variant global functions.
*FORTRAN,13	Do not set B-boxes to constants.
*FORTRAN,14	Do not optimize divide operators. (This option turns off 15 as well.)
*FORTRAN,15	Do not optimize multiply operators.
*FORTRAN,16	Do not optimize add and subtract operators.
*FORTRAN,18	DO loop optimization is removed for modifiers 1,2,3,4,13.
*FORTRAN,60	Do not optimize any of the program whatsoever.
*FORTRAN,25	Compile DO-loops according to FORTRAN 77 standards.
*FORTRAN,26	Compile DATA statements according to FORTRAN 77 standards.

XIII

INDEX

INDEX

A

Absolute value, 7.19
 Access, 9.5, 9.66-67
 direct, example of, 9.69
 direct and sequential,
 example of, 9.68
 sequential, example of, 9.68
 ACCESS=acc, 9.41, 9.49
 Actual parameters, 7.4
 Address, relative, 5.16
 Algorithms, in-line, 7.18
 Allocation
 array, 2.20
 storage, 5.1, 5.28, 10.4-5
 Alphanumeric
 constant, 2.11
 conversion, 8.23
 identifier, 2.2
 labels, 2.16
 names, 2.16
 Alternate entry points, 7.28-29
 Apostrophe delimiters, 8.8
 Appendices, 12.1
 Argument
 dummy, 5.36
 list, 7.4
 Arithmetic evaluation, 3.3
 Arithmetic IF statement, 6.5
 Array, 2.17, 2.19
 allocation, 2.20
 double precision, 2.21
 elements, 2.19-20
 integer, 2.21
 mapping, 5.12
 name, 5.11
 structure, 2.20
 transmission, 8.3-4
 ASSIGN statement, 6.3
 Assigned GO TO statement, 6.2

Asynchronous

statement, alternate form for, 9.23
 data transfers, 9.4
 READ/WRITE, 9.19
 Argument list, 7.3
 Aw on input and output, 8.23

B

BACKFILE statement, 9.62
 BACKSPACE statement, 9.61
 Base 8 system, 2.6
 Binary, 9.18, 9.25
 BLANK COMMON, 5.16, 5.21, 5.23
 Blank specification, 8.29
 BLANK=blnk, 9.42, 9.53
 BLK parameter in COMMON statement, 5.20
 BLOCK DATA statement, 7.26
 Block, 6.30-32
 common; *see* Common block
 ELSE, 6.32
 IF, 6.28-29
 IF/THEN/ELSE, 6.28
 Branching and IF logic, 10.4-5
 BUFFER IN statement, 9.24
 BUFFER OUT statement, 9.24-25

C

CALL EXIT statement, 6.27
 CALL statement, 7.8
 Calling program, argument list for, 7.4
 Card
 blank, 1.4
 program, 7.2
 punched, 1.4
 Character mode, in BUFFER OUT, 9.25

C (continued)

Character set
FORTRAN, 2.1
7600 series, 12.2

Characters
alphabetic, 2.1
Hollerith, 2.11
maximum
in ENCODE, 9.29
in output record, 8.31
number of, in a variable, 2.13

numeric, 2.1
special, 2.1
string of, in PAUSE statement, 6.26

Classes of variables, 2.16

clist; list in DATA statement, 5.35
correspondence between nlist and, 5.36

CLOSE statement, 9.44
format, 9.44
error diagnostics for, 9.45

Code, display, 2.11

Coding
forms, FORTRAN, 1.1
line, 1.2

Combined expressions, 3.17, 3.19

Comment
information, 1.3
option, special, 1.4

Common
blank, 5.16, 5.21, 5.23
block, 5.16
changing length of, 5.22
length of, 5.22-23

declaration, 5.14
blank, 5.21
unused variables in, 5.22

equivalence, 5.28-29
variables, 5.20

COMMON statement, 5.20

Compilation
diagnostics, 11.1
errors fatal to, 11.5
errors in FORTRAN program, 11.1

Compiler, 1.1
errors, 11.3
table size limits, 11.18

Complementing in masking expressions, 3.15

Complex
constants, 2.10
number, imaginary part of, 2.10
number, real part of, 2.10
variable, 2.15
type of, 2.15
word length of, 2.15

COMPLEX nlist, 5.2

Computed GO TO statement, 6.3

Connectives, logical, 3.13

Constant
alphanumeric, 2.11
changing the value of, 5.7
double-precision, 2.9
expression, 5.8
Hollerith, 2.11
integer, 2.4

Continuation, 1.2-3

CONTINUE statement, 6.26

Conversion, 7.19
alphanumeric, 8.23
and editing information, 8.7-8
general, 8.16
logical, 8.25
octal, 8.21
specifications, 8.9
table, octal-decimal, 12.5

Core
dimensioned array too large for, 5.10
size, programs approaching maximum, 5.18
to core transfer, 9.4, 10.8-9

Count, iteration, in DO statement, 6.40

CPASS=cpass, 9.56

CRAY-1 FORTRAN, 10.1
Manual, 10.1

CRDT=crdt, 9.53

CRECL=cl, 9.53

D

Data

access to, 9.5
block, 7.26
declaration, 5.35-36
input, 10.6-7
mode of, definition, 9.10
output, 10.6-7
source, 9.2
transfer, 9.1
 formatted and unformatted, 9.6
 synchronous and asynchronous,
 9.4
transmission of, 8.1
DATA statement, 5.35
 assigning initial values in, 5.36
 blank common and, 5.21
 block, 7.26
 implied DO loop in, 5.37
 variable format in, 8.37
Dataset support, 10.8-9
statements, 9.38
Decimal
 /binary position table, 12.10
 internal floating-point table,
 12.9
Deck, arrangement of FORTRAN, 7.35
Declaration
 blank common, 5.21
 common, 5.14
 unused variables in, 5.22
 data, 5.35
 rules for, 5.36
 dimension, 5.10
 equivalence, 5.24
 rules for, 5.27
 format, 8.7
 type, 5.1, 10.2-3
 and storage allocation, 5.1
 nlist parameter in, 5.2
 rules for, 5.2
DECODE statement, 9.27, 9.33
examples, 9.36
parameters, 9.27
Default, modifying, 5.4
Delimiters, star and apostrophe, 8.8

Diagnostics, 11.1

ARRAY NAME PREVIOUSLY USED, 5.11
ATTEMPT TO CHANGE LENGTH, 5.22
ATTEMPT TO DOUBLY TYPE A VARIABLE, 5.2
compilation, 11.1
DIMENSIONED ARRAY TOO LARGE FOR CORE, 5.10
DUPLICATE BLOCK NAME, 5.21
error, in subscript use, 2.19
form of, 11.1
GREATER THAN THREE DIMENSIONS, 5.10
ILLEGAL USE OF REPLACEMENT STATEMENT, 4.5
incorrect DO forms, 6.11
rules and, in ENTRY statement, 7.30
SCM DIRECT RANGE, 5.12
syntax error, 9.22
 in CLOSE statement, 9.45
 in INQUIRE statement, 9.54
 in OPEN statement, 9.43
 in READ/WRITE statement, 9.11
 in SETPASS statement, 9.57
UNRECOGNIZED STATEMENT, 6.3
VARIABLE DIMENSIONED IN FORMAL PARAMETER
LIST, 5.13
Diagram, flow, for DO loop, 6.43
Digits, octal, 8.22
Dimension
 declaration, 5.10
 specification, 5.11
DIMENSION statement, 5.10
 diagnostic, 5.10
Dimensioned
 array diagnostic, 5.10
 identifier diagnostic, 5.13
variables, 5.12
Dimensions
 assigned through parameter list, 5.13
 greater than three, in DIMENSION
 statement, 5.10
variable, 5.13
 correct usage of, 7.33
 in subprograms, 6.25, 7.33
Direct
 access
 comments, 9.67
 examples of, 9.67-9.68
 range diagnostic, SCM, 5.12
DIRECT=dir, 9.50
Display code, 2.11

D (*continued*)

DO-loop
active and inactive, 6.42
diagnostics of incorrect, 6.11
execution, 6.13, 6.39
flow diagram for, 6.43
FORTRAN 77, 6.37
general form of, 8.3
illegal entry in, 6.18
implied, in DATA statement, 5.37
incrementation processing, 6.41
nests, 6.15
 levels of, 6.16
 limitations of, 6.17
 optimization of, 6.21
 defining statements of, 6.20
operations, sequence of, 6.39
optimization, 6.2
optimizer, turning off, 6.21
programming sample of, 6.12
range, 6.37
 extended, 6.18
 execution of, 6.41
rules, 6.38
structure, 6.37
subscripts
 nonstandard, 6.18
 standard, 6.18
 within, 6.18
terminal statement in, 6.10, 6.38
transfer out of, 6.15
DO statement, 6.9, 6.37
 execution, 6.40
 form of, 6.9, 6.37
 iteration count in, 6.40
 largest possible value in, 6.9
parameter
 incrementation, 6.10, 6.40
 indexing, 6.9
 initial, 6.9
 terminal, 6.40
 rules in, 6.9
Dominant operand type, 3.18-19
 mixed-mode, 4.1
 order, 3.18
Double-precision
 arrays, 2.21
 constants, 2.8-9
 nlist, 5.2
 variable, 2.15

Dummy arguments, 5.36
DUPLICATE BLOCK NAME diagnostic, 5.21
Dw.d input and output, 8.18

E

Editing
 conversion, 8.7-8
 information, 8.7
 specifications, 8.8, 8.29
Elements, array
 reference of, 2.19
 storage locations, 2.20
ELSE block, 6.32
ELSE IF statement, 6.31
ELSE statement, 6.32
ENCODE statement, 9.27-8
 characters in, 9.29
 examples, 9.36
 parameters, 9.27
ENDFILE statement, 9.61
Entry
 in DO-loop, illegal, 6.18
 points, alternate, 7.28-29
ENTRY statement, 7.28, 7.30
Equivalence
 common, 5.28-29
 declaration, 5.24, 5.27
 group, 5.24
EQUIVALENCE statement, 5.24
Equivalent storage, 5.27
Error
 compilation, 11.1
 fatal, 11.5
 nonfatal, 11.16
 compiler, 11.3
 condition
 in OPEN statement, 9.40
 statement label, 9.40, 9.45, 9.47
diagnostic
 READ/WRITE, 9.11
 subscript use, 2.19
syntax, 9.11
 CLOSE statement, 9.45
 INQUIRE statement, 9.54
 OPEN statement, 9.43
 READ/WRITE, 9.11
 SETPASS statement, 9.57
 range, 2.5, 2.8
ERR=s, 9.40, 9.45, 9.47

Evaluation
arithmetic, 3.3
logical, 3.12
standard, 3.3
with optimization, 3.6
Ew.d
input, 8.9
output, 8.12
scaling, 8.28
summary table, 8.10
Executable statement, 5.1
Execution
DO-loop, 6.13, 6.39
range, 6.41
of statements
DO, 6.40
ELSE, 6.32
ELSE IF, 6.31
END IF, 6.33
IF/THEN, 6.30
sequence, program, 6.1
EXIST=ex, 9.48
exp parameter in
IF statement, 6.5-7
PARAMETER statement, 5.7
EXPDT=exp, 9.41, 9.52
Exponent in E specification, 8.9
Expression, 3.1-2
arithmetic, 3.2
constant, 5.8
evaluation of, 3.2
logical, 3.11
evaluation of, 3.11
in IF statement, 3.14
masking, 3.14
complementing in, 3.15
evaluation of, 3.16
logical arithmetic in, 3.14
logical sum of operands in, 3.15
operands, 3.15
evaluation of, 3.16
mixed mode, 3.17
evaluating, 3.18
parentheses in, 3.20
parenthetical, 3.3
simple and combined, 3.17
with ** operator, 3.19
External records, 8.2
EXTERNAL statement, 7.5, 7.31

F
Factor
repetition, 8.34
scale, 8.26
False value logical evaluation, 3.12
Field
identification, 1.3
short, in FORMAT, 8.15
FILE=dsi, 9.39, 9.47
FILESEQ=n, 9.40, 9.47
Floating-point decimal table, internal, 12.9
Flow diagram for DO-loop, 6.43
Formal parameters, 5.13, 5.36, 7.3
Format, 10.6-7
control, 9.12
declaration, 8.7
input/output, 8.1
specifications, 8.8, 8.34-35
variable, 8.7, 8.36
FORMAT statement
form of, 8.7
use of slash (/) in, 8.32
Formatted data transfers, 9.6
FORMATTED=fmt, 9.50
Forms, FORTRAN coding, 1.1
FORM=fm, 9.40, 9.50
*FORTRAN,1, 6.21
*FORTRAN,2, 6.22
*FORTRAN,3, 6.23
*FORTRAN,4, 6.23
*FORTRAN,13, 6.24
*FORTRAN,18, 6.24
*FORTRAN,60, 6.24
Function, 7.1
fwa, 9.21
Fw.d
input, 8.14
output, 8.14
scaling, 8.27-28
summary table, 8.14

G
GEN=gen, 9.52
GO TO statement, 6.2
assigned, 6.2
computed, 6.3
many-branch, 6.2-3
Gw.d input and output, 8.16

H

Hierarchy of arithmetic evaluation, 3.3
Hollerith character, 2.11
constant, 2.11-12

I

ident parameters, 9.39, 9.46
Identification field, 1.3
Identifier, 2.2
 alphanumeric, 2.2
 function, 7.12
 statement, 2.3
 variable dimensioned, 5.13
 zero as, 5.20
If
 block, 6.28-29
 logic, 10.4-5
IF statement, 6.5
 arithmetic, 3-branch, 6.5
 logical, 3.13-14
 relational, 6.7-8
IF, ELSE; *see* ELSE IF
IF/THEN statement, 6.30
IF/THEN/ELSE statement, 6.28
 sample, 6.35
Imaginary part of complex number, 2.10

Implicit typing, 5.4-6
IMPLICIT statement, 2.14, 5.4
Implied DO-loop, 5.37
Inactive DO-loop, 6.42
Incrementation processing in DO-loop, 6.10, 6.40-41

Indefinite conditions, integer, 2.5

Index

 check, 5.11
 values, 5.11
 variable, 8.4
Infinity, positive and negative, 2.7
Initial values, 5.35-36
Input, 8.27
 data, 10.6-7
 parameters, 9.9, 9.20, 9.44

Input/Output
 formats, 8.1
 list, 8.2, 9.3
 statements, 9.1, 9.16
 synchronous, 9.12, 9.58
 terminology, 9.4
INQUIRE statement, 9.46
 error diagnostics in, 9.54
 keylist parameters in, 9.48
Integer
 arrays, 2.21
 constants, 2.3-4
 conversion, octal-decimal, 12.5
 indefinite conditions, 2.5
 multiply, 2.5
 nlist, 5.2
 overflow, 2.5
 type, 5.3
 variable, 2.14
Internal files, 9.27
In-line, 7.1
 algorithms, 7.18
IOSTAT=ios, 9.42, 9.45, 9.48
Iteration count, 6.40
Iw input and output, 8.19-20

J

Job, 7.1

K

Keylist parameters, 9.40, 9.48
Keywords, 9.5

L

Label
 alphanumeric, 2.16
 statement, 1.3, 2.3
 error condition, 9.40, 9.45, 9.47
Left-justified Hollerith constant, 2.11
Length
 common block, 5.22-23
 Hollerith constant, 2.11

let parameter, 5.4
Level, nesting, 6.15-16, 6.28
Line, coding, 1.2
Loader, 5.23
Local variable, 5.14
Location, 7.20
 storage, 5.24
 of array elements, 2.20
 variable name of, 2.13
Logic, IF, 10.4-5
Logical
 arithmetic, 3.1, 3.14
 connectives, 3.13
 evaluation, 3.12
 expressions, 3.11, 3.14
 functions, 7.22
 nlist, 5.2
 records, 9.18
 sum, 3.15
 type, 5.3
 variables, 2.16, 3.14
Loop
 control processing, 6.41
 DO; *see* DO loop
Looping and control statements, 10.6-7
Lw input and output, 8.25

M

Machine program, 1.1
Main program, 7.2
Mapping, array, 5.12
Masking, 3.1
 expressions, 3.14-16
 statement, 3.16
Maximum
 arguments, 7.20
 characters in
 ENCODE, 9.29
 output record, 8.31
 core size, 5.18
 record length, 9.13
MAXREC=maxr, 9.41, 9.49
Minimum arguments, 7.20
Mixed mode, 3.1
 expressions, 3.17-18
 replacement statements, 4.1

Mode
 character, 9.25
 data, 9.10, 9.20
 mixed, 3.1; *see also* mixed mode
 of variables in masking statement, 3.16
MODE=md, 9.20, 9.51
Modular design, 7.1
Modulus, 7.21
Multiple replacement statements, 4.4
Multiply, integer, 2.5

N

NAMED=nmd, 9.49
Names
 alphanumeric, 2.16
 symbolic, 2.2
 variable, 2.15, 5.10
NAME=fn, 9.49
NCAR
 forms of synchronous I/O, 9.12
 FORTRAN, 10.1
 printer/card reader units, 9.14
Negative
 infinity, 2.7
 subscripts, 5.12
Nested DO loops, 6.20-6.21
Nesting
 capacity, 6.17
 levels, 6.15-16, 6.20
NEXTREC=nr, 9.52
nHf, 2.11
nlist
 and clist, 5.36
 double precision, 5.2
 integer, 5.2
 logical, 5.2
parameter
 in COMMON statement, 5.20
 in EQUIVALENCE statement, 5.24
 in type declaration, 5.2
real, 5.2
 subscripts in, 5.3
Nonexecutable statement, 5.1
Nonfatal errors, 11.16
Nonstandard subscript, 2.18, 8.3
 in DO loops, 6.18
 limitations of, 8.3
Nonzero test in relational IF statement, 8.3

N (*continued*)

NSTATE=ns, 9.21, 9.51
Number
 complex, 2.10
 indefinite, 2.7
 real, 2.7
 unit, 9.39
NUMBER=un, 9.49
Numeric characters, 2.1
NUMREC=nr, 9.52
NWORDS=n, 9.21

O

Octal
 constants, 2.6
 conversion, 8.21
 /decimal conversion, 12.5
 digits, 8.22
 variable, 2.15
Odd parity, 9.23
OPEN statement, 9.38
 error condition in, 9.40
 error diagnostics in, 9.43
OPENED=od, 9.48
Operand
 in arithmetic expressions, 3.2
 in masking expressions, 3.15
 type, 3.18-19
Operator, 3.19
 in arithmetic expressions, 3.2
 relational, 3.11
Optimization
 DO loop, 6.2, 6.21
 evaluation with, 3.6
 of arithmetic sequences, 3.6
Optimizer, DO loop, 6.21
Ordering, statement, 5.1

Output, 8.27-28
Aw, 8.23
data, 10.6-7
Dw.d, 8.18
Ew.d, 8.12
Fw.d, 8.14
Gw.d, 8.16
Iw, 8.20
Lw, 8.25
Ow, 8.22
parameters, 9.11, 9.21
real data, 8.16
Rw, 8.24
statements, 9.12, 9.17
wH, 8.31
Overflow, integer, 2.5
Overlay, 5.23
Ow input and output, 8.21-22

P

param parameter, 5.7
Parameter, 9.24, 9.55
actual, 7.4
communication of, 7.3
diagnostic, 5.13
example of, 5.9
formal, 5.36, 7.3
ident, 9.44, 9.46
in COMMON statement, 5.20
in DO statement
 exp₁, exp₂, exp₃, 6.10
 initial, 6.40
 indexing, 6.9
 m₁, m₂, m₃, 6.40
 terminal, 6.40
in ENCODE/DECODE, 9.27
in EQUIVALENCE statement, 5.24
in IMPLICIT statement, 5.4
in PARAMETER statement, 5.7
in TYPE declaration, 5.2
input, 9.9, 9.20, 9.44
keylist, 9.40, 9.48
output, 9.11, 9.44
return, 9.42, 9.45
PARAMETER statement, 5.7
 rules for using, 5.8
Parentheses, use of, in evaluating
 expressions, 3.20
Parenthetical expressions, 3.3

Parity, 9.23
PASS=pass, 9.42, 9.55
PAUSE statement, 6.26
Place holder, 5.14
Points, entry; *see* Entry points
Positioning
 file, 9.58, 10.8-9
 statements, 9.60
Positive
 difference, 7.21
 infinity, 2.7
Powers of two, table of, 12.4
Precision, double; *see* Double precision
PRINT statement, 9.13
Printer unit, 9.14
Program, 7.1
PROGRAM statement, 7.2
PUNCH statement, 9.14
Punched cards, 1.4

R

Range
 direct, 5.12
 error, 2.5, 2.8
 in DO loop, 6.37, 6.41
 extended, 6.18
 of constants, 2.7
 double precision, 2.8
 integer, 2.4
 of exponent in E specification, 8.9
 of unit numbers, 9.39
READ statement, 9.12, 9.16
 asynchronous, 9.19
 diagnostics, 9.11
 format of, 9.8
 synchronous, 9.8
Reader units, 9.14
Real
 constant, 2.7
 data, 8.16
 number, 2.7
 nlist, 5.2
 part of complex number, 2.10
 type, 5.3
 variables, 2.15
RECL=rlen, 9.41, 9.49

Record
 end of, 8.32
 external, 8.2
 length, 9.13
 logical, 9.18
 new, 8.32
 output, 8.31
 structure, 9.25
 tape, 9.18
REC=rn, 9.20
Reentry into a loop, 6.19
Reference
 function, 7.13-14
 identifier, 5.24
 label, statement, 6.9
 of array elements, 2.19
 of entire array, 2.19
 subprogram, 10.2-3
Relational
 IF statement, 6.7-8
 operators, 3.11
Relative address, 5.16
Repetition factor, 8.34
Replacement, 10.4-5
statements, 4.1
 diagnostic, 4.5
 function identifier, 7.12
 function reference, 7.14
 logical variables in, 3.14
 mixed mode, 4.1
 multiple, 4.4
Reserved computer words, 5.10
Return parameter, 9.42, 9.45
RETURN, 7.27
REWIND statement, 9.60
Right-justified Hollerith constant, 2.11
RPASS=rpass, 9.56
Rw input and output, 8.24
RWPASS=rwpass, 9.56

S

Scale factor, 8.26
Scaling, Fw.d and Ew.d, 8.27-28
SCM DIRECT RANGE diagnostic, 5.12
SEQFIL=n, 9.52
Sequential access, 9.65-68
 file positioning with, 9.58
SEQUENTIAL=seq, 9.50

S (*continued*)

Set, character; *see* Character set
SETPASS statement, 9.55-57
Share, 5.24
Shifting, 7.22
Sign, 7.21
Significant part, 2.9
Simple
 expressions, 3.17, 3.19
 variable, 2.16
Size
 core, maximum, 5.18
 limits, compiler table, 11.18
SKIPFILE statement, 9.63
Slash (/) in FORMAT statement, 8.32
Source
 data, 9.2
 program, 1.1
Specification
 blank, 8.29
 conversion, 8.8-9
 dimension, 5.11
E, 8.9-10
 editing, 8.8, 8.29
 format, 8.8, 8.34-35
H, 2.11, 8.30
 scaled, 8.26
 statements, 5.1
wX, 8.29
Standard
 evaluation, 3.3
 FORTRAN, 2.1
 subscripts, 2.17, 6.18, 8.2
Star (*) delimiter, 8.8
STATUS=sta, 9.40, 9.44
STOP statement, 6.27
Storage
 allocation, 5.1, 5.28, 10.4-5
 equivalent, 5.27
 location, 2.20, 5.24
String substitution, 5.8
Subfield combinations in FORMAT,
 8.10
Subroutine, 7.1, 7.7
 subprogram, 7.7, 7.28

Subscript, 2.17
 error diagnostic in use of, 2.19
nonstandard, 2.18, 8.3
 in DO loop, 6.18
 limitations of, 8.3
standard, 2.17, 8.2
 in DO loop, 6.18
 in I/O list, 8.2
 negative, 5.12
 subscripted, 2.18
Subscripted
 subscript, 2.18
 variable, 2.17-18, 5.10
Substitution, string, 5.8
Support, dataset; *see* Dataset support
Symbolic
 name, 2.2, 5.7
 representation of a variable, 2.13
Synchronous
 data transfers, 9.4
 input/output, 9.12, 9.58
 READ/WRITE, 9.8
Syntax
 error diagnostics; *see* Diagnostics
 FORTRAN, 9.70
 statement, definition of symbols in, 10.10

T

Tables
 compiler, 11.18
 decimal/binary position, 12.10
 Ew.d summary, 8.10
 FORTRAN syntax, 9.70
 Fw.d summary, 8.14
 octal/decimal integer conversion, 12.5
 powers of two, 12.4
 variable assignments, 5.16
Tape
 odd and even parity in writing, 9.23
 1/2" tape recorded in binary unformatted,
 9.18
Terminal
 parameter in DO statement, 6.40
 statement in DO loop, 6.10, 6.20, 6.38
Termination statements, 6.27
Tests in relational IF statement, 6.7
Three-branch arithmetic IF statement, 6.5

Transfer
control, 7.8
to a function, 7.13
core-to-core, 9.4, 10.8-9
data, 9.1
formatted and unformatted, 9.6
synchronous and asynchronous,
9.4
out of DO loop, 6.15
subprogram reference and, 10.2-3
unconditional, of statement, 6.2
Transmission
array, 8.3-4
data, 8.1
Two-branch relational IF statement,
6.7
typ parameter in IMPLICIT statement,
5.4
Type
complex variable, 2.15
declaration, 5.1-2, 10.2-3
dominant, 4.1
double-precision variable, 2.15
Hollerith constant, 2.12
integer, 5.3
logical, 5.3
octal variable, 2.15
operand, dominant, 3.18-19
real, 5.3
variable, 2.14
Typing, implicit, 5.4-6

U

Unconditional GO TO statement, 6.2
Unformatted
binary, 9.18
data transfers, 9.6
input statements, 9.16
output statements, 9.17
UNFORMATTED=unf, 9.51
UNIT=u, 9.20, 9.39, 9.44, 9.47, 9.55
UNRECOGNIZED STATEMENT diagnostic, 6.3
Unsigned integer constants, 2.3
User catalog program, example, 9.69

V

Value
absolute, 7.19
false, logical evaluation, 2.13
index, 5.11
initial, 5.35
in DATA statement, 5.36
true, logical evaluation, 3.12
Variable
complex, 2.15
dimension, 5.13
correct usage of, 7.33
in a subprogram, 6.25, 7.33
DO, 6.37
double-precision, 2.15
format, 8.36-37
in IMPLICIT statement, 2.14
in TYPE statement, 2.14
index, 8.4
integer, 2.14
multiply-subscripted, in EQUIVALENCE
statement, 5.24
name
in DIMENSION statement, 5.10
octal, 2.15
of location, 2.13
number of characters in, 2.13
octal, 2.15
simple, 2.16, 2.18, 5.10
subscripted, 2.16
symbolic representation of, 2.13
value of, 2.13
VSN=name

W

WH input and output, 8.30-31
Word
computer, length on common block, 5.22
length on common block, 5.22
reserved for an array, 5.10
structure of constants, 12.11
length, 2.14-15
WPASS=wpass, 9.56
WRITE statement, 9.14
asynchronous, 9.19
diagnostics in, 9.11
form of, 9.8
synchronous, 9.8

13.12
Index

W (*continued*)

Writing a tape, 9.23
wX specification, 8.29

X

Y

Z

Zero

as an identifier, 5.20
test in relational IF statement,
6.7