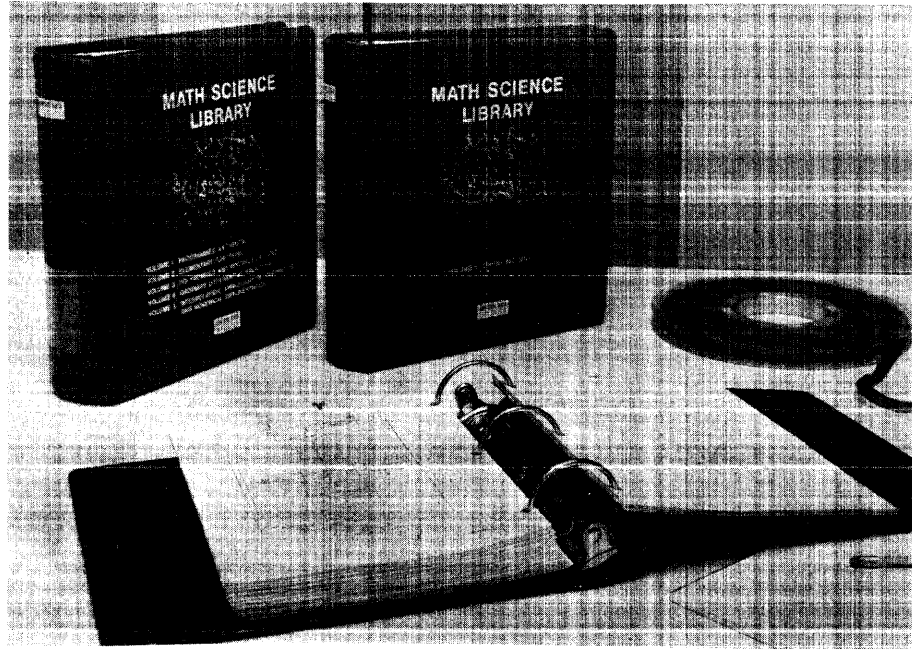

CONTROL DATA[®]
6000 COMPUTER SYSTEMS
7600 COMPUTER SYSTEM

FORTRAN EXTENDED REFERENCE MANUAL
6000 VERSION 3
7600 VERSION 1

MATH SCIENCE LIBRARY



The Math Science Library is the most efficient and comprehensive library of mathematical routines available. It consists of over 400 routines selected for:

- Speed
- Accuracy
- Reliability
- Maintainability
- Flexibility
- Storage conservation

The eight areas of computational mathematics that are covered are:

- Programmed arithmetic
- Elementary functions
- Polynomials and special functions
- Ordinary differential equations
- Interpolation, approximation and quadrature
- Linear algebra
- Probability, statistics, and time series
- Nonlinear equation solvers

The Math Science Library includes an 1800 page handbook, in eight volumes, guiding users to the routines most suitable for their objectives.

The Math Science Library is operational on CONTROL DATA CYBER 70, 6000, and 7600 systems.

PREFACE

This publication describes the features of the FORTRAN Extended language (version 3.0) for the CONTROL DATA® 6400/6500/6600/6700/7600 Computers. It is assumed that the reader has some knowledge of an existing FORTRAN language and CONTROL DATA computer systems. The language described herein is an extension of the ANSI FORTRAN language.

The FORTRAN compiler operates in conjunction with the Version 2 COMPASS assembly language processor under the control of three operating systems: SCOPE Version 3.3 and KRONOS® Version 2.0 for 6000 Series computers and SCOPE Version 1.1 for the 7600 computer. The FORTRAN processor makes optimum use of storage both during compilation and in generated machine language instructions. Implementation of this processor provides the capability of compilation and execution within a single job operation as well as the simultaneous compilation of several programs, utilizing the system's multi-programming features.

Control Data Corporation intends for the user of this product to exercise only those features, specifications, and parameters described in this document. Any use of adjunct code and/or undefined parameter values is done so at the user's risk.

Related manuals in which the FORTRAN user may find additional information are:

	Publication No.
SCOPE 3.3 Reference Manual	60305200
SCOPE 3.3 User's Guide	60252700
SIFT (FORTRAN Translator Program)	60358400
FORTRAN Extended Debug User's Guide	60329400
COMPASS 2 Reference Manual	60279900
7000 SCOPE 1 Reference Manual	60281200
KRONOS 2.0 Reference Manual	59150600

CONTENTS

PREFACE		iii
CHAPTER 1	PROPERTIES AND ELEMENTS OF FORTRAN	1-1
1.1	The FORTRAN Character Set	1-1
1.2	FORTRAN Statements	1-1
	Statements	1-1
	Continuations	1-2
	Comments	1-2
	Statement Label	1-2
	Identification Field	1-2
1.3	Symbolic Names	1-3
1.4	Data Types	1-3
1.5	Constants	1-3
	Integer	1-3
	Real	1-4
	Double Precision	1-4
	Complex	1-5
	Logical	1-5
	Hollerith	1-6
	Octal	1-6
1.6	Variables	1-7
	Variable Names	1-7
	Types of Variables	1-7
	Arrays	1-8
	Order of Array Storage	1-8
	Subscripted Variables	1-9
	Extended Core Storage	1-11
CHAPTER 2	EXPRESSIONS	2-1
2.1	Arithmetic Expressions	2-1
2.2	Relational Expressions	2-3
2.3	Logical Expressions	2-5
2.4	Masking Expressions	2-6
2.5	Evaluation of Expressions	2-8

CHAPTER 3	ASSIGNMENT STATEMENTS	3-1
	3.1 Arithmetic Assignment	3-1
	Mixed-Mode	3-2
	3.2 Logical Assignment	3-3
	3.3 Masking Assignment	3-3
CHAPTER 4	CONTROL STATEMENTS	4-1
	4.1 GO TO Statements	4-1
	Unconditional GO TO	4-1
	4.2 Assigned GO TO	4-1
	Computed GO TO	4-3
	4.3 IF Statements	4-4
	Arithmetic IF Three-Branch	4-4
	Arithmetic IF Two-Branch	4-5
	Logical IF	4-5
	Logical IF Two-Branch	4-6
	4.4 DO Statement	4-6
	DO Nests	4-7
	DO Loop Execution	4-8
	CONTINUE	4-12
	4.5 CALL	4-12
	RETURN	4-14
	4.6 Program Control	4-14
	STOP	4-14
	PAUSE	4-15
	END	4-15
CHAPTER 5	INPUT/OUTPUT STATEMENTS	5-1
	5.1 Modes of Input/Output	5-1
	5.2 I/O Lists	5-1
	5.3 Read/Write Statements	5-2
	5.4 Formatted Input/Output	5-2
	Read	5-2
	Input File	5-3
	Write	5-3
	Print/Punch	5-4
	Print Control	5-4
	5.5 Unformatted Input/Output	5-5
	Read	5-5
	Write	5-5
	5.6 Namelist Statement	5-6
	Input Data	5-7
	Output Data	5-9
	5.7 Rewind	5-9
	5.8 Backspace	5-9
	5.9 Endfile	5-10
	5.10 ECS I/O	5-10
	5.11 Mass Storage I/O	5-10

CHAPTER 6	FORMAT STATEMENTS	6-1
6.1	Format Declaration	6-1
	Field Descriptors	6-1
	Field Separators	6-2
6.2	Conversion Specification	6-2
	Iw Input	6-3
	Iw Output	6-3
	Ew.d Input	6-4
	Ew.d Output	6-7
	Fw.d Input	6-7
	Fw.d Output	6-8
	Gw.d Input	6-9
	Gw.d Output	6-9
	Dw.d Output	6-10
	Dw.d Input	6-10
	Ow Output	6-10
	Ow Input	6-10
	Aw Output	6-11
	Aw Input	6-11
	Rw Output	6-12
	Rw Input	6-12
	Lw Output	6-12
	Lw Input	6-12
	Complex Conversions	6-12
	nP Scale Factor	6-13
6.3	Editing Specifications	6-14
	nX	6-14
	nH	6-15
	New Record	6-16
	... ≠...≠	6-17
	Tn	6-17
6.4	Repeated Format Specifications	6-18
6.5	Variable Format	6-19
CHAPTER 7	AUXILIARY INPUT/OUTPUT STATEMENTS	7-1
7.1	Buffer Statements	7-1
	Buffer In	7-2
	Buffer Out	7-2
7.2	ENCODE/DECODE Statements	7-2
	Encode	7-3
	Decode	7-4
CHAPTER 8	SPECIFICATION AND DATA STATEMENTS	8-1
8.1	Dimensions	8-1
	Variable Dimensions	8-2
8.2	Common	8-3
	Labeled Common	8-3
	Unlabeled Common	8-4
	Arrangement of Common Blocks	8-4

	8.3	Equivalence	8-5
	8.4	External	8-7
	8.5	TYPE	8-7
	8.6	DATA	8-8
CHAPTER 9		PROGRAM FUNCTION, SUBROUTINE, BLOCK DATA, AND LIBRARY ROUTINES	9-1
	9.1	Main Program	9-1
	9.2	Subroutine Subprograms	9-3
		ENTRY Statement	9-6
		Library Subroutines	9-7
	9.3	Function Subprograms	9-8
		Statement Functions	9-8
		Intrinsic Function	9-9
		External Function	9-9
		External Function Reference	9-10
		Basic External Functions	9-11
	9.4	Block Data Subprogram	9-11
CHAPTER 10		OVERLAYS AND SEGMENTS	10-1
	10.1	Overlays	10-1
	10.2	Segments	10-3
		Segment Control Cards	10-4
		Sections	10-4
		Segments	10-5
CHAPTER 11		DEBUGGING FACILITY	11-1
	11.1	Format	11-2
	11.2	Arrays Statement	11-2
	11.3	Calls Statement	11-3
	11.4	Funcs Statement	11-5
	11.5	Stores Statement	11-6
	11.6	Gotos Statement	11-7
	11.7	Trace Statement	11-7
	11.8	Nogo Statement	11-8
	11.9	Deck Structure	11-9
	11.10	Debug Statement	11-14
	11.11	Area Statement	11-15
	11.12	Off Statement	11-16
	11.13	Printing Debug Output	11-17
CHAPTER 12		FORTRAN CONTROL CARD	12-1
	12.1	Control Card Format	12-1
	12.2	Source Input Parameter	12-1
	12.3	Binary (Object) Output Parameter	12-2
	12.4	List Parameter	12-2

	12.5	Error Traceback and Calling Sequence Parameter	12-3
	12.6	Update Parameter (Editing Parameters)	12-3
	12.7	Optimization Parameter	12-4
		Invariant Computations	12-5
		Register Assignment	12-6
	12.8	Rounded Arithmetic Parameter	12-9
	12.9	Debugging Mode Parameter	12-9
	12.10	Exit Parameter	12-9
	12.11	System Text File Parameter	12-9
	12.12	System Editing and I/O Reference Parameter	12-10
	12.13	Assembler Parameter	12-10
	12.14	Control Card Examples	12-10
	12.15	Small Buffers	12-11
	12.16	Reference Map Level	12-11
APPENDIX A		STANDARD SCOPE CHARACTER SETS	A-1
APPENDIX B		FORTRAN DIAGNOSTICS	B-1
APPENDIX C		CROSS REFERENCE MAP	C-1
APPENDIX D		LIBRARY SUBPROGRAMS	D-1
APPENDIX E		INTERMIXED COMPASS SUBPROGRAMS	E-1
APPENDIX F		STATEMENT FORMS	F-1
APPENDIX G		SYSTEM ROUTINE SPECIFICATIONS	G-1
APPENDIX H		DECK STRUCTURE	H-1
APPENDIX I		EXECUTION TIME I/O	I-1
APPENDIX J		SUBPROGRAM AND MEMORY STRUCTURE	J-1
APPENDIX K		FORTRAN-INTERCOM INTERFACE	K-1
INDEX			Index-1

FORTRAN CODING FORM										
PROGRAM		SAMPLE PROGRAM			CONTROL DATA		NAME			
ROUTINE					CORPORATION		DATE		PAGE	OF
TYPE	STATEMENT NO.	CONT.	FORTRAN STATEMENT							SERIAL NUMBER
			O = ZERO Ø = ALPHA O			I = ONE I = ALPHA I		2 = TWO Z = ALPHA Z		
C	Ø		$R_i A_i M_i I_i S_i T_i H_i E_i S_i Ø_i L_i U_i T_i I_i Ø_i N_i Ø_i F_i A_i N_i M_i D_i E_i G_i R_i E_i P_i Ø_i L_i Y_i N_i Ø_i M_i I_i A_i L_i B_i Y_i L_i E_i A_i S_i T_i - i S_i Q_i U_i A_i R_i S_i i M_i E_i T_i H_i Ø_i D_i$							
			$R_i E_i A_i L_i X_i (1,0,0)_i, Y_i (1,0,0)_i, W_i (2,1)_i, Z_i (1,1)_i, A_i (1,1)_i, B_i (1,1,1,2)_i$							P_i Ø_i L_i Y_i 1
	1		$F_i Ø_i R_i M_i A_i T_i (I, 2, I, 3 / 4, F, 1, 4, .7)_i$							P_i Ø_i L_i Y_i 2
	2		$F_i Ø_i R_i M_i A_i T_i (5, E, 1, 5, .6)_i$							P_i Ø_i L_i Y_i 3
			$R_i E_i A_i D_i (5, 1, 1)_i M_i, N_i (X_i (I)_i, I = 1, N_i)_i$							P_i Ø_i L_i Y_i 4
			$L_i W_i = 2 * M_i + 1$							P_i Ø_i L_i Y_i 5
			$L_i B_i = M_i + 2$							P_i Ø_i L_i Y_i 6
			$L_i Z_i = M_i + 1$							P_i Ø_i L_i Y_i 7
			$D_i Ø_i 5 J_i = 2, L_i W_i$							P_i Ø_i L_i Y_i 8
	5		$W_i (J_i) = 0.0$							P_i Ø_i L_i Y_i 9
			$W_i (1) = N_i$							P_i Ø_i L_i Y_i 10
			$D_i Ø_i 6 J_i = 1, L_i Z_i$							P_i Ø_i L_i Y_i 11
	6		$B_i (J_i) = 0.0$							P_i Ø_i L_i Y_i 12
			$D_i Ø_i 16 I_i = 1, N_i$							P_i Ø_i L_i Y_i 13
			$P_i = 1.0$							P_i Ø_i L_i Y_i 14
			$B_i (1) = B_i (1) + Y_i (I)$							P_i Ø_i L_i Y_i 15
			$D_i Ø_i 13 J_i = 2, L_i B_i$							P_i Ø_i L_i Y_i 16
			$P_i = X_i (I) * P_i$							P_i Ø_i L_i Y_i 17
			$W_i (J_i) = W_i (J_i) + P_i$							P_i Ø_i L_i Y_i 18
	13		$B_i (J) = B_i (J) + Y_i (I) * P_i$							P_i Ø_i L_i Y_i 19

Figure 1-1

**1.1
THE FORTRAN
CHARACTER SET**

Alphabetic:	A to Z	
Numeric:	0 to 9	
Special:	= equals) right parenthesis
	+ plus	, comma
	- minus	. decimal point
	* asterisk	\$ dollar sign
	/ slash	space (i.e., blank)
	(left parenthesis	

In addition, any character of the SCOPE character set (Appendix A) may be used in Hollerith information and in comments.

**1.2
FORTRAN
STATEMENTS**

FORTRAN source programs consist of an ordered set of statements from which the compiler generates machine instructions and constants. These statements describe a procedure to be followed during execution of the program.

The statements comprising the FORTRAN program are written in the following columns:

	<u>Column</u>	<u>Content</u>
Statements	{ 1-5	Statement label (optional)
	6	Blank or zero
	{ 7-72	FORTRAN statement
	{ 73-80	Identification field
Statement Continuations	{ 1-5	Ignored
	6	FORTRAN character other than blank or zero
	{ 7-72	Continued FORTRAN statement
	{ 73-80	Identification field
Comments	{ 1	C or \$ or *
	{ 2-80	Comments

Except in Hollerith constants, blanks may be used freely and are ignored by the compiler. A coding line may contain more than one FORTRAN statement if each statement is separated by the special character \$. The next column following \$ is interpreted the same as column 7 of a normal statement. A \$ may serve as a statement separator for all statements except FORMAT, END, or labeled statements.

Continuation

Any FORTRAN statement except a comment, END statement, or loader directive may be continued. A statement may be continued on as many as 19 lines, each denoted by a continuation character (any acceptable character other than blank or zero) in column 6 on the continuation card. A blank or zero in column 6 denotes the first line of a statement. Blank cards within the input deck are ignored by the compiler; however, a continuation card following a blank card is treated as a new statement. (See also chapter 11, Debugging Facility.)

Comment

In Column 1, a C, *, or \$ indicates a comment line. Comments do not affect the program; they can be written in columns 2 to 80 and placed anywhere in the program. When a comment occupies more than one line, each line must begin with C, *, or \$ in column 1. The continuation character in column 6 does not apply to comment cards. Comments can appear between continuation cards.

Statement Label

Statements are identified by unsigned integers which can be referred to from other sections of the program. A statement label (from 1-99999) may be placed anywhere in columns 1-5 of the initial line of a statement. Leading zeros are ignored. In any program unit, each statement label must be unique.

Identification Field

The FORTRAN Extended compiler is designed so that input lines may be greater than 80 characters long (e.g., when the input medium is a file produced by one of the source editing programs such as UPDATE). Only the first 72 characters are processed by the compiler and only the first 100 characters appear on the listing. Positions beyond 72 may be used for identification codes or sequencing.

1.3 SYMBOLIC NAMES

A symbolic name may be any alphabetic character followed by 0-6 alphanumeric characters. It may not contain special characters. Embedded blanks are ignored. Symbolic names are used for: subprogram and subroutine names, function names, variables, block data program, main program, input/output unit, common block, and namelist group names.

1.4 DATA TYPES

Each of the seven types of data has different significance. The types are: integer, real, double precision, complex, logical, octal, and Hollerith.

Integer type may assume only whole number values. For multiplication and division of integer operands, the result will be invalid if it exceeds $2^{47}-1$. For addition and subtraction, the full 60-bit word is used.

Real type data is carried in normalized floating point form. The magnitude of values of real type data is in the range 10^{322} to 10^{-293} with approximately 15 significant digits and 14 digit precision.

Double precision data is similar to type real, but it has approximately 29 significant digits.

Complex data consists of an ordered pair of real data. Each part has the same precision as real data. The first part is the real part, and the second is the imaginary part.

Logical data has only a true or false value. True is represented by any negative value, and false is represented by any positive value including +zero.

Octal data may consist of any value from 0-7...7 which can be represented in a maximum of 60 bits (20 octal digits).

Hollerith data consists of strings of characters. Blank characters are valid in a Hollerith string.

1.5 CONSTANTS

A constant is an unvarying quantity. The types of constants are the same as the types of data.

1.5.1 INTEGER

An integer constant is a string of up to 18 decimal digits with a magnitude no larger than $2^{59}-1$. If multiplication or division is specified, the operands and result should be less than $2^{47}-1$. Effectively, an integer constant string may contain up to 15 decimal digits with a maximum magnitude of $2^{47}-1$. It may not contain embedded commas. For example:

0	-2145637
67	45753576357
345	-77

The result of integer addition or subtraction must not exceed $2^{59}-1$. Integers used as subscripts and DO indexes are limited to $2^{17}-2$. The integer constant may be positive, zero, or negative (if unsigned, it is assumed to be positive) and must be within the allowed magnitude.

The maximum value of an integer constant as a result of a conversion from a real constant is $2^{47}-1$. The maximum value of an integer constant as a result of multiplication or division must not exceed $2^{47}-1$. If the value should exceed the magnitude allowed, the high order bits are lost.

1.5.2 REAL

A real constant may be represented by a string of up to 15 significant decimal digits. It contains a decimal point or an exponent representing a power of 10, or both. Real constants may be in the following forms:

n. .n n.n n.nE±s n.E±s nE±s .nE±s

n is the coefficient; E signifies that the succeeding datum is the exponent; and s is the base 10 exponent. The value of s must be in the range -308 to +337. The plus sign may be omitted if s is positive. The magnitude of non-zero absolute real values may be in the range 10^{-293} to 10^{322} , with up to 15 significant digits. If the range of the real constant is exceeded, the constant is considered zero and a compiler diagnostic is issued.

Examples:

3.E1	(means 3.0×10^1 ; or 30.)
3.1415768	31.41592E-01
314.07	.31415E01
-3.14159E+279	.31415E+01
30E02	-30E02

1.5.3 DOUBLE PRECISION

A double precision constant is written as a string of digits and represented internally by two words. The forms are:

.nD±s n.nD±s n.D±s nD±s

The D must always appear; the coefficient is n; s is the exponent of base 10.

The plus sign may be omitted for positive s. The range is the same as that of a real constant but is accurate to approximately 29 decimal digits. If the range is exceeded, a compiler diagnostic is issued.

Examples:

```

3.1415927D+1    3141.593D3
3.1416D0        31416.D-04
3131.593D-03   31416D02

```

1.5.4 COMPLEX

A complex constant is an ordered pair of signed or unsigned real constants, separated by a comma, and enclosed in parentheses (r1,r2). r1 represents the real part of the complex number; r2 represents the imaginary part. r1 and r2 must adhere to the magnitude specified for real constants. If this range is exceeded, a compiler diagnostic is provided. Diagnostics also occur when the pair contains integer constants, including (0,0).

Examples:

<u>FORTTRAN Representation</u>	<u>Complex Numbers</u>
(1.,6.55)	1. + 6.55i
(15.,16.7)	15. + 16.7i
(-14.09,1.6E-03)	-14.09 + .0016i
(0.,-1.)	0. - 1.0i

1.5.5 LOGICAL

Logical constants assume only the values of true or false. When the compiler generates a value for the constant .TRUE., it will generate a minus one; for the constant .FALSE., a zero is generated. Logical constants must be preceded and followed by a period and have the forms:

```

.TRUE. or .T.
.FALSE. or .F.

```

Example:

```

LOGICAL X1,X2
      :
      :
X1 = .T.
X2 = .FALSE.

```

1.5.6 HOLLERITH

A Hollerith constant is of the form hHf, hRf (right justified), or hLf (left justified). h is an integer constant whose value is greater than zero; f represents the Hollerith data and must contain exactly h characters. When the hHf form is used, if h is not a multiple of 10, the last word is left justified and blank filled. Incomplete words in the hRf and hLf forms are binary zero filled.

Blanks are significant in a Hollerith data string. Hollerith constants are stored internally in display code. (See Appendix A.)

Hollerith constants may be used in arithmetic expressions, DATA and CALL statements, and in function argument lists. If the constant is an operand of an arithmetic operation, an informative diagnostic to that effect is issued.

Examples:

```
6HCOGITO
4LERGO
3RSUM
3HSUM
```

The maximum number of characters allowed in a Hollerith constant depends on its usage. In an expression, h may not be greater than 10; in a DATA statement, h is limited only by the number of characters that can be contained in a maximum of 19 continuation lines. If more than 10 characters are given in a DATA statement for such a constant, only the last word will have the appropriate fill.

1.5.7 OCTAL

An octal constant consists of 1 to 20 octal digits followed by a B. The form is:

$$n_1 \dots n_i B$$

If the constant exceeds 20 digits, or if a non-octal digit appears, a fatal compiler diagnostic is issued. When fewer than 20 octal digits are specified, the digits are right justified and zero filled.

Example:

```
2374216B
7777776B
777000777000777B
```


1.6 VARIABLES

A variable is a symbolic representation of a quantity that may assume different values during execution of a program.

1.6.1 VARIABLE NAMES

A variable name may be any combination of 1 to 7 alphanumeric characters, must begin with an alphabetic character, and may contain embedded blanks. It may not contain special characters. For a main program, the program name may not appear as a symbolic name in any statement other than the PROGRAM statement.

1.6.2 TYPES OF VARIABLES

The type of a variable may be declared explicitly with the FORTRAN type declarations. (The type of the data is converted to the type of the variable.)

For example:

```
INTEGER ABC123, GNU12, CATXXX, FIREOUT, JOKER  
REAL ISPY, JASONII, KOOR47, NVRT, SAMPLE
```

If integer and real variables are not declared explicitly, the type is determined by the first character of the symbolic name. If the name begins with I, J, K, L, M, or N, the variable is assumed to be integer.

I15, JK26, KKK, LB02, NP456L, and MM are classed as integer variables and must adhere to all limitations stated for that type. Variables beginning with characters A-H and O-Z are considered to be real and must adhere to all limitations stated for that type.

Complex, logical, and double precision variables must be declared explicitly by a type declaration. The values which the variables represent must adhere to the limitations stated for the corresponding type of constant.

Octal and Hollerith data are interpreted as though they were typeless. They may be assigned to variables of any type, either with simple assignment statements or with input statements, and will undergo no type conversion in the process. When one operand of an arithmetic operation is an octal or Hollerith constant, it will be interpreted as if it agreed in type with the other operand. When both operands are octal or Hollerith constants, they are considered to be type integer. The same rules apply to relational expressions.

Examples:

```
REAL X, Y  
100 X= 20B  
200 Y= 10B + 10B
```

After execution of these statements, X and Y will not have the same value. X will have the real value 0.0, and Y will have the real value 16.0. The assignment is made to X in statement 100 without conversion, and the octal value 000000000000000020B is interpreted as a floating point number indistinguishable from 0.0, since X is real. In statement 200, however, because both operands of the addition are type octal, the result is considered to be type integer, and the integer 16 is converted to floating point before being assigned to Y.

```
I= 10HTYPEWRITER + 001B
PRINT 1, I
1  FORMAT (1X,A10)
```

In this example, the word TYPEWRITES will be output.

```
X= 10HTYPEWRITER + 001B
PRINT 1, X
1  FORMAT (1X,A10)
```

In this example, 0 ^ 5J, 9R/J- will be output, due to the conversion to floating point which will take place before assignment to X.

1.6.3 ARRAYS

An array is an ordered set of variables identified by a variable name. Each variable in the array is referred to by the array name followed by a subscript which indicates its relative position within the array. The entire array may be referenced by the array name without subscripts when used as an item in an input/output list or in a DATA statement.

Arrays may have one, two, or three dimensions and must be defined at the beginning of the program in a DIMENSION, COMMON, or type statement. When a reference is made to an array, if the subscripts exceed the magnitude of the dimensions declared initially, a position outside the array will be accessed. If the number of subscripts is greater than the number of dimensions defined, a diagnostic is issued.

1.6.4 ORDER OF ARRAY STORAGE

Arrays are stored in ascending storage locations, with the value of the first of their subscripts increasing most rapidly and the value of the last increasing least rapidly.

The following list shows the order of a three-dimension array A(3,2,3). The first subscript varies from 1 to 3, the second varies from 1 to 2, the third varies from 1 to 3.

	A(1,1,1)	A(2,1,1)	A(3,1,1)	A(1,2,1)	A(2,2,1)	A(3,2,1)
→	A(1,1,2)	A(2,1,2)	A(3,1,2)	A(1,2,2)	A(2,2,2)	A(3,2,2)
→	A(1,1,3)	A(2,1,3)	A(3,1,3)	A(1,2,3)	A(2,2,3)	A(3,2,3)

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

Given DIMENSION A(L, M, N), the location of A(i, j, k), with respect to the first element of the array, is given by $A + (i-1+L*(j-1+M*(k-1))) * E$.

E is the element length, the number of storage words required for each element of the array. For real, logical, and integer arrays, E = 1. For complex and double precision arrays, E = 2.

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:

$$\text{LocnA}(2,2,3) = \text{LocnA}(1,1,1) + (2-1+3*(2-1+3*(3-1))) * 1 = \text{LocnA}+22$$

1.6.5 SUBSCRIPTED VARIABLES

A subscripted variable is an alphanumeric identifier that is the name of an array followed by up to three subscript expressions representing a single element within the array. The elements of a subscript expression are separated by commas and the expression is enclosed in parentheses. Subscript expressions may be any legal arithmetic expression. If the number of subscript expressions used in a reference is less than the declared dimensionality, the compiler assumes missing subscripts have a value of one (see examples below). If the subscript list does not appear, all subscript expressions are assumed to be one, and an informative diagnostic is issued.

If the subscript expression is not integer, the value will be truncated to integer.

The value of the subscript must be greater than or equal to one and less than or equal to the maximum specified in the array specification statement, or the reference will be outside the array. If the reference is outside the bounds of the array, results are unpredictable.

FORTRAN Extended permits the following relaxation of the representation of subscripted variables:

Given $A(D_1, D_2, D_3)$, where the D_i are integer constants,
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 $A(1, 1, 1)^\dagger$

Similarly for

$A(D_1, D_2)$
 $A(I, J)$ implies $A(I, J)$
 $A(I)$ implies $A(I, 1)$
 A implies $A(1, 1)^\dagger$

and for $A(D_1)$

$A(I)$ implies $A(I)$
 A implies $A(1)^\dagger$

The elements of a single-dimension array $A(D_1)$ may not be referred to as $A(I, J, K)$ or $A(I, J)$. Diagnostics occur if this is attempted.

[†] Except in input/output lists and DATA statements.

1.6.6 EXTENDED CORE STORAGE (ECS)

An ECS variable must be defined explicitly by a type declaration. Each ECS variable occupies a 60-bit word and resides in extended core storage (ECS) in the 6000 series. The 7600 ECS variables reside in large core memory (LCM).

For the 7600, subroutines READEC and WRITEC will operate on LCM and small core memory (SCM) in the same manner as ECS and central memory in the 6000 series. ECS variables may appear in the source program only in the following circumstances:

- In a COMMON statement as an element of an ECS common block
- In a CALL or function reference as an actual parameter
- In a SUBROUTINE or FUNCTION statement as a dummy parameter
- In a type ECS statement
- In a DIMENSION statement

Only one common block may contain ECS variables, and all variables in the block must be of type ECS.

An expression is a constant, variable (simple or subscripted), function reference, or any combination of these separated by operators and parentheses. The four kinds of expressions in FORTRAN are: arithmetic and masking (Boolean) expressions which have numerical values, and logical and relational expressions which have truth values. Each kind of expression is associated with a group of operators and operands.

**2.1
ARITHMETIC
EXPRESSIONS**

An arithmetic expression is formed with arithmetic operators and arithmetic elements. Both the expression and its constituent elements identify values of one of the types integer, real, double precision, complex, octal, or Hollerith.

<u>Arithmetic Operators</u>	<u>Arithmetic Operands</u>
+ addition	Constants
- subtraction	Variables (simple or subscripted)
* multiplication	Evaluated functions
/ division	
** exponentiation	

Any unsigned constant, variable, or function reference is an arithmetic expression. If X is an expression, then (X) is an expression. If X and Y are expressions, then the following are expressions:

X + Y	X - Y
X * Y	X / Y
-X	X ** Y
+X	

An expression may not contain adjacent operators, such as X +/ Y. Omission of an operator, as for implied multiplication (X) (Y), for instance, is not valid and results in a compiler diagnostic.

The mode of an arithmetic expression is determined by the type specifications of the variables in the expression. The following table indicates how the mode is determined from the possible combinations of variables.

Table 1. Mixed Mode Arithmetic Expressions

+ - * /	Hollerith	Integer	Real	Double Precision	Complex	Octal
Integer	Integer	Integer	Real	Double Precision	Complex	Integer
Real	Real	Real	Real	Double Precision	Complex	Real
Double Precision	Double Precision	Double Precision	Double Precision	Double Precision	Complex	Double Precision
Complex	Complex	Complex	Complex	Complex	Complex	Complex
Octal	Integer	Integer	Real	Double Precision	Complex	Integer
Hollerith	Integer	Integer	Real	Double Precision	Complex	Integer

The following examples are valid expressions:

A
 3.14159
 B + 16.427
 (XBAR +(B(I, J+I, K) /3))
 -(C + DELTA * AERO)
 (B - SQRT(B**2*(4*A*C)))/(2.0*A)
 GROSS - (TAX*0.04)
 TEMP + V(M, MAXF(A, B))*Y**C/ (H-FACT(K+3))

The arithmetic operator denoting exponentiation (**) may be used to combine constants, variables, expressions, and subscripted variables. Rules governing the types of variables and constants used in the exponentiation operation are given on the following page:

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

The following examples illustrate how constants, variables, and expressions may be combined using the arithmetic operator, **.

Examples:

<u>Expression</u>	<u>Type</u>	<u>Result</u>
CVAB**(I-3)	Real**Integer	Real
D**B	Real**Real	Real
C**I	Complex**Integer	Complex
BASE(M, K)**2.1	Double Precision **Real	Double Precision
K**5	Integer** Integer	Integer
314D-02** 3.14D-02	Double Precision **Double Precision	Double Precision

2.2 RELATIONAL EXPRESSIONS

A relational expression has the value true or false; it contains two arithmetic expressions separated by a relational operator. The types of operands may be combined in the same manner as defined for arithmetic operators. Only the real part of complex elements are compared by relational operators, except for .EQ. and .NE.

Relational operators indicate comparison operations between operands and are enumerated below:

- .EQ. Equal to ($=$)
- .NE. Not equal to (\neq)
- .GT. Greater than ($>$)
- .GE. Greater than or equal to (\geq)
- .LT. Less than ($<$)
- .LE. Less than or equal to (\leq)

A relational expression has the form:

$$a_1 \text{ op } a_2$$

The a_1 are arithmetic expressions; op is an operator belonging to the above set.

A relation is true if a_1 and a_2 satisfy the relation specified by op; otherwise it is false. A false relational expression is assigned a positive value; a true relational expression is assigned a negative value. Relations are evaluated as illustrated in the relation $p.EQ.q$, which is equivalent to the question: Does $p - q = 0$? The difference is computed; and if it is zero, the relation is true; if the difference is not zero, the relation is false. Relational expressions are converted internally to arithmetic expressions according to the rules of mixed-mode arithmetic (Table 1). These expressions are evaluated to produce the truth value of the corresponding relational expressions.

The order of dominance of the operand types within an expression is the order stated for mixed mode arithmetic expressions.

In relational expressions, $+0$ is considered equal to -0 .

$a_1 \text{ op } a_2 \text{ op } a_3 \dots$ is not a valid expression. The relations $a_1 \text{ op } a_2$, $a_1 \text{ op } (a_2)$ are equivalent.

Examples:

A .GT. 16.	R(I) .GE. R(I-1)
R -Q(I)*Z .LE. 3.141592	K .LT. 16
B-C .NE. D+E	I .EQ. J(K)
	(I) .EQ. (J(K))

2.3 LOGICAL EXPRESSIONS

A logical expression is formed with logical operators and logical elements and has the value true or false. (The values have the same internal representation as for relational expressions, section 2.2.)

<u>Logical Operators</u>	<u>Alternate Form</u>
.OR. Logical disjunction	.O.
.AND. Logical conjunction	.A.
.NOT. Logical negation	.N.

A logical expression has the general form:

$$L_1 \text{ op } L_2 \text{ op } L_3 \dots$$

L_i are logical variables, logical constants, logical functions, logical expressions enclosed in parentheses, or relational expressions; and op is the logical operator .AND. indicating conjunction or .OR. indicating disjunction.

The logical operator that indicates negation appears in the form:

$$.NOT. L_1$$

Each expression is evaluated by scanning from left to right, with logical operations being performed according to the following hierarchy of precedence.

first .NOT.
then .AND.
then .OR.

A logical variable, logical constant, or a relational expression is, in itself, a logical expression. If L_1 , L_2 are logical expressions, then the following are logical expressions:

$$.NOT. L_1$$

$$L_1 .AND. L_2$$

$$L_1 .OR. L_2$$

If L is a logical expression, then (L) is a logical expression. If L_1 , L_2 are logical expressions and op is .AND. or .OR., then $L \text{ op } L_2$ is never legitimate. However, .NOT. may appear in combination with .AND. or .OR. only as follows:

L_1 .AND. .NOT. L_2
 L_1 .OR. .NOT. L_2
 L_1 .AND. (.NOT....)
 L_1 .OR. (.NOT....)

.NOT. may appear with itself only in the form .NOT. (.NOT. (.NOT. L))
 Other combinations cause compilation diagnostics.

If L_1 , L_2 are logical expressions, the logical operators are defined as follows:

.NOT. L_1	is false only if L_1 is true
L_1 .AND. L_2	is true only if L_1, L_2 are both true
L_1 .OR. L_2	is false only if L_1, L_2 are both false

Examples:

1. $B - C \leq A \leq B + C$
 is written $B - C$.LE. A .AND. A .LE. $B + C$
2. FICA greater than 176.0 and PAYNMB equal to 5889.0
 is written $FICA$.GT. 176.0 .AND. $PAYNMB$.EQ. 5889.0
3. An expression equivalent to the logical relationship P implies Q
 might be written in two ways:
 .NOT. (P.AND.(.NOT. Q))
 .N. (P.A.(.N. Q))

2.4 MASKING EXPRESSIONS

Masking expressions consist of masking operators and elements; they resemble logical operations in appearance only.

In a masking expression, 60-bit logical arithmetic is performed bit-by-bit on the operands within the expression. The operands may be any type variables, constants, or expressions, other than logical. No mode conversion is performed during evaluation. If the operand is complex or double precision, operations are performed on the real part, or higher order word. Although the masking operators are identical in appearance to the logical operators, their meanings are different. They are listed according to hierarchy. The following definitions apply:

.NOT. or .N. bit-by-bit logical negation
 .AND. or .A. bit-by-bit logical multiplication
 .OR. or .O. bit-by-bit logical addition

The operations are described below:

<u>p</u>	<u>v</u>	<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

If B_1 are masking expressions, variables or constants of any type other than logical, the following are masking expressions:

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

If B is a masking expression, then (B) is a masking expression .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 left to right.

A .AND. B .AND. C...
 A .OR. B .OR. C...

Masking expressions must not contain logical operands.

Examples:

A 77770000000000000000 octal constant
 D 00000000777777777777 octal constant
 B 00000000000000001763 octal form of integer constant
 C 20045000000000000000 octal form of real constant

.NOT. A is 00007777777777777777
A.AND. C is 20040000000000000000
A.AND..NOT. C is 57730000000000000000
B.OR..NOT. D is 77777777000000001763

The last expression could also be written as B.O. .N.D

2.5 EVALUATION OF EXPRESSIONS

Function references are evaluated before any other operation is performed.

Expressions are evaluated according to the following rules:

1. If the expression contains subexpressions enclosed by parentheses, all such subexpressions must be evaluated before the expression can be evaluated as a whole. The most deeply nested subexpression will be evaluated first. Evaluation of a parenthesized subexpression produces an intermediate result, which will be treated like any other operand.
2. If the expression consists of a series of terms (including parenthesized subexpressions) separated by operators, the order in which the indicated operations are performed is determined by the precedence of operators, as follows:

**	exponentiation	class 1
/ and *	division and multiplication	class 2
+ and -	addition and subtraction	class 3
relationals		class 4
.NOT.		class 5
.AND.		class 6
.OR.		class 7

The lowest class (1) of operations is performed first. Since unary plus and minus signs are treated as operations on an implied zero, they have a precedence of class 3. For example, +2 is treated as 0+2, and -3 is treated as 0-3.

3. When two or more operators of the same class appear in an expression, where parentheses do not indicate which operation is to be performed first, evaluation proceeds from left to right.
4. Array element subscript expressions and arguments to functions will be evaluated before the value of the array element or function reference is used as an intermediate result. If nesting of subscripts or function references occurs, those most deeply nested will be evaluated first.
5. The type of an evaluated expression depends on the types of its terms as explained in section 2.1. If dividing one integer by another produces a remainder, the result is truncated; 11/3 produces the result 3.

The evaluation of an expression with any of the following conditions is undefined:

Negative-value quantity raised to a real, double precision, or complex exponent

Zero-value quantity raised to a zero-value exponent

Infinite or indefinite operand

Element for which a value is not mathematically defined, such as division by zero

If the error traceback option is selected on the FTN card (Chapter 12), the first three conditions will produce informative diagnostics.

In the following examples, R indicates an intermediate result in evaluation. $A**B/C+D*E*F-G$ is evaluated:

$A**B \rightarrow R_1$
 $R_1/C \rightarrow R_2$
 $D*E \rightarrow R_3$
 $R_3*F \rightarrow R_4$
 $R_2-G \rightarrow R_5$
 $R_4+R_5 \rightarrow R_6$ evaluation completed

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

$A**B \rightarrow R_1$
 $C+D \rightarrow R_2$
 $R_1/R_2 \rightarrow R_3$
 $E*F \rightarrow R_4$
 $R_4-G \rightarrow R_5$
 $R_3*R_5 \rightarrow R_6$ evaluation completed

$H(13)+C(I,J+2)*(COS(Z))**2$ is evaluated:

$COS(Z) \rightarrow R_1$
 $R_1**2 \rightarrow R_2$

(Evaluation of the index function)

$R_2*C(I,J+2) \rightarrow R_3$
 $R_3+H(13) \rightarrow R_4$ evaluation completed

The following are examples of expressions with embedded parentheses:

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

$$C/D \rightarrow R_1$$

$$R_1 - E \rightarrow R_2$$

$$B + R_2 \rightarrow R_3$$

$$A * R_3 \rightarrow R_4 \quad \text{evaluation completed}$$

$(A*(\text{SIN}(X)+1.)-Z)/(C*(D-(E+F)))$ is evaluated:

$$\text{SIN}(X) \rightarrow R_1$$

$$R_1 + 1. \rightarrow R_2$$

$$A * R_2 \rightarrow R_3$$

$$R_3 - Z \rightarrow R_4$$

$$E + F \rightarrow R_5$$

$$D - R_5 \rightarrow R_6$$

$$C * R_6 \rightarrow R_7$$

$$R_4 / R_7 \rightarrow R_8 \quad \text{evaluation completed}$$

Statements are classified as executable or nonexecutable; executable statements specify actions. Assignment statements are executable. They assign values with four types of operations; arithmetic, logical, assign (Chapter 4), and masking.

3.1 ARITHMETIC ASSIGNMENT

The general form of the arithmetic assignment statement is $v = e$, where v is a variable, simple or subscripted, other than logical; and e is an arithmetic expression. The = indicates that v is assigned the value of the evaluated expression e . Mode conversion occurs if v is of a type different from e .

Examples:

```

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

Several variables may be assigned the value of the same expression with the following form:

$$v_1 = v_2 = \dots = v_m = e$$

The value of expression e is converted to the type of v_m and stored; v_m is then converted to the type of v_{m-1} and stored. The process is repeated until a value is stored in v_1 .

Example:

```

:
:
RATE=2.0
DATA=6.9
:
:
DATA=DATA1=LDATA=DATA2=DATA*RATE
    
```


The variable, DATA2, equals 13.8 from the expression DATA*RATE. LDATA equals 13 by real-to-integer conversion. DATA1 equals 13.0 by integer-to-real conversion; then DATA equals 13.0 by real-to-real assignment.

MIXED-MODE

The type of an evaluated expression is determined by the type of the dominant operand; however, this does not restrict the types that identifier v may assume. (v may not be logical). A complex expression may replace v, even if v is real. TABLE 2 on page 3-4 shows the v = e relationship for all standard modes. The mode of v determines the mode of the statement.

Examples:

Given:	C_1, A_1	Complex
	D_1, A_2	Double
	R_1, A_3	Real
	I_1, A_4	Integer

$$1. A_1 = C_1 * C_2 - C_3 / C_4 \quad (6.905, 15.393) = (4.4, 2.1) * (3.0, 2.0) - (3.3, 6.8) / (1.1, 3.4)$$

The expression is complex; the result of the expression is a two-word, floating point quantity. A_1 is complex, and the result replaces A_1 .

$$2. A_3 = C_1 \quad 4.4 = (4.4, 2.1)$$

The expression is complex. A_3 is real, therefore, the real part of C_1 replaces A_3 .

$$3. A_3 = C_1 * (0., -1.) \quad 2.1 = (4.4, 2.1) * (0., -1.)$$

The expression is complex. A_3 is real; the real part of the result of the complex multiplication replaces A_3 .

$$4. A_4 = \frac{R_1}{R_2} * (R_3 - R_4) + I_1 - \frac{(I_2 * R_5)}{(I_2 * R_5)} \quad 13 = 8.4 / 4.2 * (3.1 - 2.1) + 14 - (1 * 2.3)$$

The expression is real. A_4 is integer, the result of the expression evaluation, a real, is converted to an integer replacing A_4 .

$$5. \quad A_2 = D^{**2} * (D_2 + (D_3 * D_4)) + (D_2 * D_1 * D_2)$$

$$49.68 = 2.0D0^{**2} * (3.2D0 + (4.1D0 * 1.0D0)) + (3.2D0 * 2.0D0 * 3.2D0)$$

The expression is double precision. A_2 is double precision, the result of the expression evaluation, a double precision floating quantity replaces A_2 .

3.2

LOGICAL ASSIGNMENT

In the general form of the logical assignment statement,

$$v = e$$

v is a logical variable or subscripted logical variable, and e is a logical expression.

Examples:

```

LOGICAL A, B, C, D, E, LGA, LGB, LGC
REAL F, G, H
A = B.AND.C.AND.D
A = F.GT.G.OR.F.GT.H
5  A = .N.(A.A..N.B).AND.(C.O.D)
   LGA = .NOT.LGB
2109 LGC = E.OR.LGC.OR.LGB.OR.LGA.OR.(A.AND.B)

```

A multiple replacement statement of the following form is also allowed in logical assignment statements:

$$v_1 = v_2 = \dots v_m = e$$

3.3

MASKING ASSIGNMENT

In the masking assignment statement, $v = e$, e is a masking expression, v is a variable name and may be of any type other than logical. During the assignment, no mode conversion occurs, and the value of the expression is assigned to the first word of v if the type is double precision or complex with the least significant or imaginary part set to zero.

Examples:

```

INTEGER I, J, K, L, M, N(16)
REAL B, C, D, E, F(15)

N(2) = I.AND.J
B = C.AND.L
84  F(J) = I.OR..NOT.L.AND.F(J)
   N(1) = I.O.J.O.K.O.L.O.M
   I = .N.I

```

A multiple replacement statement of the following form is also allowed in masking assignment statements:

$$v_1 = v_2 = \dots v_m = e$$

Table 2 enumerates the assignment of e to v. These rules apply only for arithmetic assignment statements.

Table 2. Rules for Assignment of e to v

v Type	e Type	Assignment
Integer	Integer	Assign
Integer	Real	Fix and Assign
Integer	Double Precision	Fix and Assign
Integer	Complex	† Fix and Assign Real Part
Real	Integer	Float and Assign
Real	Real	Assign
Real	Double Precision	DP Evaluate and Real Assign
Real	Complex	† Assign Real Part
Double Precision	Integer	DP Float and Assign
Double Precision	Real	Real Evaluate, DP Assign
Double Precision	Double Precision	Assign
Double Precision	Complex	† DP Float Real Part and Assign
Complex	Integer	† Float and Assign to Real Part, I
Complex	Real	† Assign Real Part, I
Complex	Double Precision	† DP Evaluate and Real Assign to Real Part, I
Complex	Complex	Assign

† Prohibited combination under ANSI FORTRAN (but permitted in FORTRAN Extended).

Assign indicates transmission of resulting value, without change, to entity.

Real Assign indicates transmission to entity, of as much precision as a real value can contain.

DP Evaluate indicates evaluation of the expression according to rules of arithmetic expression evaluation.

Fix indicates truncation of any fractional part of the result and transformation to an integer value.

Float indicates transformation to a real value.

DP Float indicates transformation to a double precision value retaining, in the process, as much precision as a double precision value can contain.

Real Part refers to the real portion of the complex value.

I indicates the imaginary part of the complex value is set to zero.

Control statements alter the sequence of operations or affect the number of iterations of a program section. Control statement labels must be associated with executable statements within the same program unit. Control may not be transferred to a non-executable statement. See appendix F.

4.1 GO TO STATEMENTS

UNCONDITIONAL
GO TO GO TO *k*

When this statement is executed, control transfers to the statement identified by *k*.

Example:

```
GO TO 100
GO TO 9
```

4.2 ASSIGNED GO TO ASSIGN *k* TO *i*

k is a statement label and *i* is an integer variable. Execution of this statement and subsequent execution of an assigned GO TO statement using the value *i* causes the statement *k* to be executed next. The label must refer to an executable statement in the same program unit containing the ASSIGN statement. *k* must be the label of an executable statement.

The integer variable *i*, once used in an ASSIGN statement, may not be referenced in any statement other than an assigned GO TO or an ASSIGN statement until it has been defined in a replacement statement.

Example:

```
      :  
      :  
      ASSIGN 10 TO KLOK  
      :  
      :  
15  GO TO KLOK,(3, 12, 10, 20)  
      :  
      :  
12  CC = D+E -2*(F/G)  
      :  
      :  
10  D = SQRT(B**C*(1-E))  
      ASSIGN 20 TO KLOK  
      GO TO 15  
      :  
      :  
20  E = A+1.5
```

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

i is an integer variable, and k_i are statement labels; i must contain the value assigned by a preceding ASSIGN statement and it must be one of the statement labels in the list. At execution, control transfers to statement identified by k . If the value i is defined by other than an ASSIGN statement, a transfer is made to the absolute memory address represented by the low order 18 bits of i .

Example:

```
      :  
      :  
      ASSIGN 26 TO INDEX  
      :  
      :  
10  GO TO INDEX, (3, 45, 26, 78, 6)  
26  BASE (I) = BASE (I+1)*FACT*(CONST**2.0)  
      ASSIGN 45 TO INDEX  
      GO TO 10  
      :  
      :
```

COMPUTED GO TO

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

k_i are the statement labels and i is an integer variable. This statement acts as a many-branch GO TO; i is preset or computed prior to its use in the GO TO statement. Control transfers to k_i , if $1 \leq i \leq n$. If i is less than one or greater than n , a fatal error occurs. The comma separating the statement number list and the index is optional. i must not be specified by an ASSIGN statement.

Example:

```
      :  
      :  
      I=2  
      N=2  
      :  
      :  
      N=N*I  
      :  
      :  
      GO TO (100, 101, 18, 102, 103)N
```

Control transfers to the statement numbered 102.

Example:

```
      ISWICH=1  
      GO TO (10, 20, 30) ISWICH(control transfers to 10)  
      :  
      :  
      KSWICH=ISWICH+1  
      GO TO (11, 41, 31), KSWICH(control transfer is to statement 41)
```

Another form of the statement may be used where i is replaced by e :

GO TO (k_1, k_2, \dots, k_n), e

where e is a general expression of any type except logical.

The value of e is truncated and converted to integer and used in place of i .

Examples:

```
1.      :  
        :  
        K=2  
        X=4.6  
        :  
        :  
        GO TO (10, 110, 11, 12, 13), X/K
```

Control is transferred to statement 110 since the integer part of the expression X/K equals 2.

```
2.      :  
        :  
        BRANCH 2.3  
        INDEX=4  
        :  
        :  
        GO TO (23, 33, 43, 53, 63), INDEX*BRANCH
```

Cause execution to terminate since the integer part of the evaluated expression, $INDEX*BRANCH$, equals 9 and there is no ninth branch.

4.3 IF STATEMENTS

ARITHMETIC IF
THREE-BRANCH

IF (e) k_1, k_2, k_3

\underline{e} is an arithmetic expression of type integer, real, double precision, or complex, and \underline{k}_i are statement labels. For complex, only the real part is used in selecting the branch. Execution of the statement results in evaluation of \underline{e} and transfer of control as follows:

$e < 0$ to statement k_1
 $e = 0$ to statement k_2
 $e > 0$ to statement k_3

Example:

```
I = -2
K = 1
J = 3
:
:
1  IF (I*K*J)2,3,4  (control transfers to 2)
2  LDD=LDD+1
   GO TO (40,50,60)LDD
40  IF (X*Y*SIN(X))11,12,13
:
:
```

ARITHMETIC IF TWO-BRANCH

A second form of the Arithmetic IF statement; an arithmetic two-branch IF is allowed.

IF (e) k_1, k_2

e may be a masking or arithmetic expression; e is evaluated and control is transferred as follows:

$e \neq 0$	to statement k_1
$e = 0$	to statement k_2

Example:

```
IF (I*J*DATA(K))100,101
100 IF (I*Y*K)105,106
```

LOGICAL IF

IF (e) s

e is a logical expression and s is any executable statement except a DO statement or another logical IF statement. If the value of e is false, statement s is treated as if it were a CONTINUE statement. If the value of e is true, s is executed.

Example:

```
      :  
      :  
      B4=DATA(I)  
      :  
      :  
      YMAX=B(ILAST)  
      YMIN=B(IFRST)  
      :  
      :  
16   IF (B4.GE. YMIN.AND. B4. LE. YMAX) GO TO 109  
101  INDEX=INDEX+1  
      GO TO 110  
109  KDEX=KDEX+1  
      :  
      :
```

If B4 is satisfied by the condition, $YMIN \leq B4 \leq YMAX$, control transfers from statement 16 to 109. If the condition is not satisfied, execution resumes at statement 101.

LOGICAL IF TWO-BRANCH

Another form of the logical IF may be a two-branch statement:

```
      IF (e)  $k_1, k_2$ 
```

If the logical statement is true, the statement identified by statement label \underline{k}_1 is executed next, if false the statement \underline{k}_2 is executed.

4.4 DO STATEMENT

The DO statement makes it possible to repeat a sequence of statements and change the value of an integer control variable during the repetition. A DO statement takes one of the forms:

```
      DO n i= $m_1, m_2, m_3$       or      DO n i= $m_1, m_2$ 
```

The executable statement labeled \underline{n} is the terminal statement of the sequence to be repeated and must physically follow and be in the same program unit as the DO statement.

Example:

```

      :
      :
      DO 100L=300,400
      IF(B(L)) 101,100
101  B(L-100)=B(L)
100  CONTINUE
      :
      :

```

Statement n (100 in the example) may not be a GO TO of any form, arithmetic IF, RETURN, STOP, PAUSE, DO, two-branch logical IF, or a logical IF followed by any of the preceding statements.

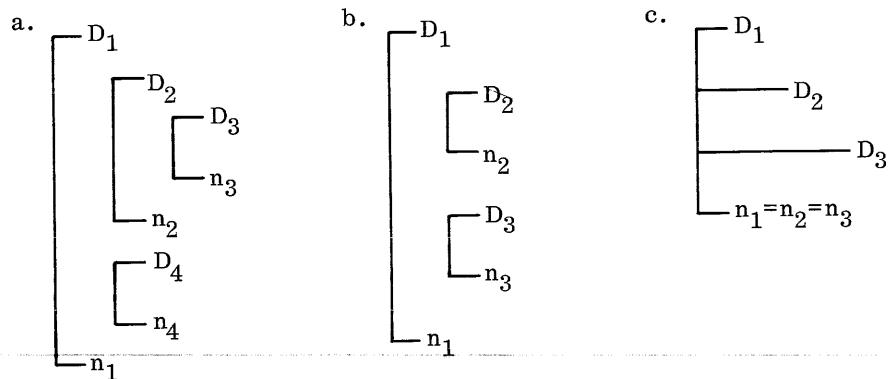
The simple integer variable i is the control variable; m_i are the indexing parameters; m_1 is the initial value of i , m_2 is the terminal value of i , and m_3 is the incrementing parameter. m_i may be either integer constants or simple integer variables. If m_3 is not specified, a value of one is implied. At execution of the DO statement. m_1 , m_2 and m_3 must be greater than zero. The range of each DO contains all executable statements between and including the first executable statement after the DO and the terminal statement identified by n .

DO NESTS

When a DO loop contains another DO loop, the grouping is called a DO nest. Nesting may be to 50 levels. Either the last statement of a nested DO loop must be the same as the last statement of the outer DO loop or it 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 appears before D_3 and n_1, n_2, \dots, n_m represent the corresponding limits of the D_i , then n_m must appear at or before n_{m-1} .

Examples:

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



The preceding diagrams would be coded as follows:

<pre> a. DO 1 I=1, 10, 2 ⋮ DO 2 J= 1 5 ⋮ DO 3 K=2, 8 ⋮ 3 CONTINUE ⋮ 2 CONTINUE ⋮ DO 4 L=1, 3 ⋮ 4 CONTINUE ⋮ 1 CONTINUE </pre>	<pre> b. DO 100 L= 2, LIMIT ⋮ DO 10 J= 1, 10 ⋮ 10 CONTINUE ⋮ DO 20 K=K1, K2 ⋮ 20 CONTINUE ⋮ 100 CONTINUE </pre>	<pre> c. DO 5 I=1, 5 DO 5 J=I, 10 DO 5 K=J, 15 ⋮ ⋮ 5 A = B*C </pre>
---	---	---

DO LOOP EXECUTION

The loop defined by a DO statement is executed as follows:

1. The control variable i is assigned the value represented by the initial parameter m_1 . The value of m_1 should be less than or equal to the value of the terminal parameter m_2 ; otherwise, the DO loop is executed only once. (The control variables of each nested DO loop must be unique.)
2. The range of the DO is executed.
3. After the DO is executed, the control variable is incremented by the value m_3 (or by one if m_3 is not specified).
4. If the value of the control variable i after it is incremented by m_3 is less than or equal to the value of the terminal parameter m_2 , execution of the range of the DO loop is repeated. When the value i is greater than the value of m_2 , the DO has been satisfied and the control variable i , becomes undefined (the value of i may be greater, less than or = to m_2 at the termination of the loop execution, therefore its value cannot be assumed).

5. If the DO is nested, the control variable i of the next outer DO is incremented by m_3 and execution continues repeating steps 4 and 5 until all the DO statements referencing this terminal statement are satisfied. After the last DO is satisfied, execution continues with the first executable statement following its terminal statement.
6. If m_1 , m_2 , or m_3 are constants which exceed $2^{17}-2$, a diagnostic notes the error and the control variable is used modulo $2^{17}-1$ for iteration of the DO loop.

Before the DO is satisfied, an exit may be made from its range through an IF or a GO TO statement. In this case, the control variable retains the value last assigned to it before the exit.

Example:

```

      :
      :
      DO 20 I=1,200
      IF(I-3) 20,10,10
20   CONTINUE
10   I9=I

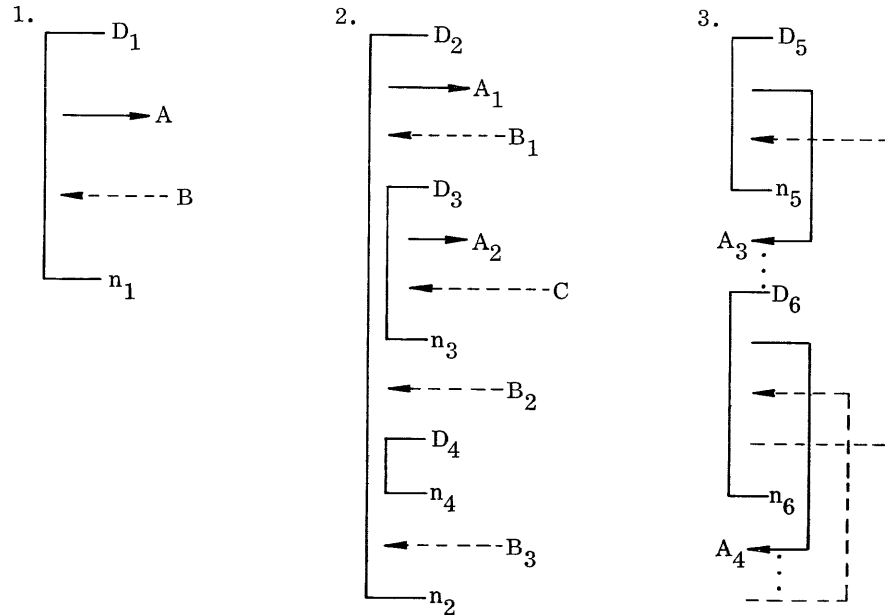
```

An exit from the range of the DO is made to statement 10 when the value of the control variable I is equal to 3. The value of the integer variable, I9 is equal to 3, since the last value assigned to I before the exit from the DO range was 3.

A DO has an extended range if both of the following conditions are satisfied:

1. A GO TO or an IF statement within the range of a DO nest transfers control outside the nest
2. A GO TO statement or an IF statement outside the nest causes control to re-enter a DO loop or nested set of DO loops as illustrated below.

Examples:



Example 1 shows an exit at point A. Any re-entry into D_1 may be made as illustrated at point B or at any subsequent point within the indicated loop.

Example 2 shows three nested loops with D_3 and D_4 being parallel. An exit is made at point A_1 , re-entry into D_2 may be made at points B_1 , B_2 , or B_3 . However, re-entry cannot be made into D_3 or D_4 because the control variables for those loops have not been defined. If an exit is made from point A_2 , re-entry may be made at C , B_1 , B_2 or B_3 but not at any other points within the other loops.

The third example illustrates the capability of specifying an extended range DO loop within the extended range of another loop. Loop D_5 has an extended range which is entered at point A_3 ; the loop D_6 , which also has an extended range beginning at point A_4 , is contained within the extended range of D_5 .

A GO TO or an IF statement may not cause control to pass into the range of a DO unless it is being executed as part of the extended range of that particular DO.

The control variable i and the parameters m_1 , m_2 , and m_3 may not be redefined during execution of the immediate or extended range of that DO. When parameters are redefined during execution, the results are unpredictable. An informative diagnostic is issued for redefinition during an immediate range.

When a statement is the terminal statement of more than one DO loop, the label of that terminal statement may not be used in any GO TO or IF statement in the nest, except in the range of the innermost DO.

Example:

```
DO 10 J=1, 50
DO 10 I=1, 50
DO 10 M=1, 100
  :
  :
  GO TO 10
  :
  :
10 CONTINUE
```

When the IF statement is used to bypass several inner loops, different terminal statements for each loop are required.

Example:

```
DO 10 K=1, 100
IF(DATA(K)-10.)20, 10, 20
20 DO 30 L=1, 20
IF(DATA(L)-FACT*K-10.)40, 30, 40
40 DO 50 J=1, 5
  :
  :
  GO TO (101, 102, 50), INDEX
101 TEST=TEST+1
  GO TO 104
103 TEST=TEST-1
  DATA(K)=DATA(K)*2.0
  :
  :
50 CONTINUE
30 CONTINUE
10 CONTINUE
  :
  :
  GO TO 104
102 DO 109 M=1, 3
  :
  :
109 CONTINUE
  GO TO 103
104 CONTINUE
```

CONTINUE

CONTINUE

This statement is most frequently used as the last statement of a DO loop to provide a 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.

Example:

```
      :  
      :  
      DO 10K = 1, 200  
      DATA(K)=DATB(K+1)  
10 CONTINUE
```

4.5 CALL

The CALL statement, which transfers control to a subroutine subprogram, may take one of the following forms:

```
CALL s (a1, a2, ..., an)  
CALL s  
CALL s (a1, a2, ..., an), RETURNS (b1, b2, ..., bm)  
CALL s, RETURNS (b1, b2, ..., bn)
```

s is the name of a subroutine and a_i are actual arguments which correspond to dummy arguments specified in the subroutine subprogram. b_i parameters indicate labels of statements in the current calling program or subprogram. The total number of parameters, a_i + b_i, should not exceed 63.

The arguments (a_i) appearing in the statement may be constants, variables, array element names, array names, the name of an external procedure, etc. (see p. 9-4). These arguments must correspond in number, order and type with those specified in the SUBROUTINE statement (see chapter 9 for an explanation of this statement).

The parameters b_i must be specified with the RETURNS if alternate exits are taken from the subroutine. If alternate exits are not taken, this specification may be omitted, and control returns to the statement immediately following the CALL. These parameters must also correspond to similar parameters specified in the subroutine.

The return of control from the designated subroutine completes the execution of the CALL statement.

Example:

```
PROGRAM MAIN (INPUT, OUTPUT)
  :
  :
10  CALL XCOMP(A, B, C), RETURNS(101, 102, 103, 104)
  :
  :
101 CONTINUE
  :
  :
    GO TO 10
102 CONTINUE
  :
  :
    GO TO 10
103 CONTINUE
  :
  :
    GO TO 10
104 CONTINUE
    END

SUBROUTINE XCOMP (B1, B2, G), RETURNS (A1, A2, A3, A4)
  IF(B1*B2-4.159) 10, 20, 30
10  CONTINUE
  :
  :
    RETURN A1
20  CONTINUE
  :
  :
    RETURN A2
30  CONTINUE
  :
  :
    IF (B1) 40, 50
40  RETURN A3
50  RETURN A4
    END
```

RETURN

RETURN or RETURN a

a is a formal parameter (as indicated in the RETURNS list).

Example:

```
          SUBROUTINE XYZ, (P, T, U), RETURNS(A, B, C)
          IF (P*T*U)1, 2, 3
1         CONTINUE
          :
          :
          RETURN A
2         CONTINUE
          :
          :
          RETURN B
3         RETURN C
          END
```

The statement, RETURN a, can appear only in a subroutine subprogram. Execution of this statement returns control to the statement number corresponding to a in the RETURN list.

A RETURN statement marks the logical end of a procedure (subroutine or function) subprogram and may appear only in a procedure subprogram. In a subroutine subprogram, a RETURN statement returns control to the next executable statement following the CALL statement of the current calling program. In function subprograms, a RETURN statement returns control to the statement containing the function reference.

4.6 PROGRAM CONTROL

STOP

STOP n or STOP

n is a string of 1-5 octal digits.

When a STOP statement is encountered, n is displayed in the dayfile, the executable program terminates and control returns to the monitor. If n is omitted, blanks are implied.

PAUSE

PAUSE n or PAUSE

n is a string of 1-5 octal digits.

When a PAUSE statement is encountered, the executable program halts and PAUSE n appears as a dayfile message on the display console. The operator can continue or terminate the program with an entry from the console. The program continues with the next statement. If n is omitted, blanks are implied.

END

END

This must be the final statement and marks the physical end of the program or subprogram. It is executable in the sense that it effects termination of a main program or acts as a RETURN in a SUBROUTINE or FUNCTION, but it should not be labeled.

The READ and WRITE input/output statements cause information to be transferred between internal storage and external devices.

5.1 MODES OF INPUT/OUTPUT

Input and output can be formatted or unformatted. Formatted information consists of strings of characters acceptable to the FORTRAN processor. Unformatted information consists of strings of binary word values in the form in which they normally appear in storage. The transmission of formatted information is always associated with a FORMAT statement, as described in chapter 6. Additionally, NAMELIST may be used for input/output as discussed in section 5.6.

5.2 I/O LISTS

The input list specifies the names of variables and array elements to which information is transmitted from the external device. The output list specifies the variables and array elements whose values are transmitted to the external device. Both lists may take any of the following forms.

If no list appears on input, a logical record is skipped. Only Hollerith information from the FORMAT statement can be output with a null (empty) output list.

A simple list consists of a variable name, an array name, an array element name, or a DO-implied list.

If an array name without any subscripts appears in a list, the entire array (not just the first word of the array) is read or written.

Multiple simple lists may appear, separated by commas, each of which may be enclosed in parentheses, such as: (...), (...).

A DO-implied list is a simple list followed by a comma and a DO-implied specification, all enclosed in parentheses.

A DO-implied specification takes one of the following forms:

$$i = m_1, m_2, m_3 \quad \text{or} \quad i = m_1, m_2$$

The elements \underline{i} , \underline{m}_1 , \underline{m}_2 and \underline{m}_3 have the same meaning for the DO statement. The range of DO-implied specification is that of the DO-implied list. For the input lists, \underline{i} , \underline{m}_1 , \underline{m}_2 and \underline{m}_3 may appear within that range only as subscripts.

Elements of a list are specified in order of occurrence from left to right. The elements of a DO-implied list must be specified for the initial cycle of the implied DO.

5.3 READ/WRITE STATEMENTS

The parameters used with the READ/WRITE statements are defined as follows:

- u Identifies the input/output unit; an integer constant or a simple integer variable.
- f Identifies the format specification; a FORMAT statement label or an array name. If \underline{f} is a statement label, the statement must appear in the same program unit as the input or output statement.
- k Input/output list indicating the data to be transferred.

5.4 FORMATTED INPUT/OUTPUT

The statements discussed in this section pertain to the transmission of data according to a FORMAT specification.

Information processed by the READ and WRITE statements is divided into records. Each time a READ or WRITE is executed at least one record is processed. It is not possible to read several parts of a single record with more than one READ statement.

READ

READ (u, f)k

READ (u, f)

READ f, k

This statement transmits data from the external device for which the logical unit number is the integer value of \underline{u} . Information contained on \underline{u} is scanned and converted in accordance with the format specification identified by \underline{f} . The values, as a result, are assigned to the element specified by the list, \underline{k} . However, if the list is omitted, this statement means the next logical record is bypassed (except for the case described on page 6-15 of reading Hollerith characters into an existing H field within a FORMAT statement).

Example:

```
      READ (2,10) (IDAT(I), I = 1, 10), C
10  FORMAT (2X, 10(I5,2X), F3.2)
      DO 30 K=1, 10
      READ (2,20) (B2(K, J), J=1, 5)
20  FORMAT (5(F10.2, 1X))
30  CONTINUE
```

INPUT FILE

READ f,k or READ f

This statement results in the input of records from the SCOPE INPUT file. The theory of operation is the same as that described for the formatted READ statement.

Example:

```
      READ 31, NAME, GREEN, HORNET
31  FORMAT (A10, F10.3, E20.2)
```

WRITE

WRITE (u,f)k or WRITE (u,f)

The above statements write formatted records on the logical unit specified by u. The parameters have the same meaning as described for the corresponding READ statement. The contents of the resulting records consist of the values of the list items in the order in which they appear in the list. The values represented by the list variables are converted according to the format specification, then transferred to the indicated output unit.

Example:

```
      WRITE (6, 10) L1, B1, L2, B2
10  FORMAT (2X, I5, 1X, F5.2, I5, F9.3)
      DO 20 J = 1, 10
      DO 20 K = 1, 10
20  WRITE (4, 26) DATA1(J, K), DATA1 (J, K)
26  FORMAT (2X, 15H THE VALUES ARE, 2F6.2)
```

If the list k in a formatted WRITE statement is omitted, the contents of the created record are dependent upon the corresponding FORMAT statement.

Example:

```
      WRITE (4, 27)
27  FORMAT (32H THIS COLUMN REPRESENTS X VALUES)
```

When the list k is specified for formatted input or output, the corresponding FORMAT declaration must contain at least one conversion specification other than Hollerith.

PRINT/PUNCH

PRINT f,k or PRINT f

The information specified by k is transferred as line printer images to the SCOPE OUTPUT file, 136 characters or less per line in accordance with FORMAT declaration f.

Example:

```
PRINT 20, DNAME
20  FORMAT (X,A10)
```

When the list designation is omitted, the statement has the form illustrated in the following example:

```
PRINT 20
20  FORMAT (31H THIS IS THE END OF THIS REPORT)
```

The first character of formatted records is not printed, but is used by the line printer to determine vertical spacing of records on a page. Appendix I, carriage control characters, lists the control options.

PUNCH f, k or PUNCH f

The information specified by k is transferred to the SCOPE PUNCH file as Hollerith images, 80 characters or less per card in accordance with FORMAT declaration f. If the card image is longer than 80 characters, a second card is punched with the excess characters. Omission of k is interpreted the same as for the PRINT statement.

Example:

```
PUNCH 30, JOHN
30  FORMAT (X,I7)
```

5.5 UNFORMATTED INPUT/OUTPUT

The statements discussed herein transmit data without a FORMAT designation.

READ READ (u)k or READ (u)

This form of the READ statement is classified as unformatted because of the omission of the f parameter in the statement form. Execution of the statement results in the sequential assignment of values, as they are read, to the variables appearing in the list k. If the sequence of values required by the list exceeds the length of the unformatted record the excess words retain their previous values. If the number of values specified in the list is less than the number of words in the logical record, the excess data is ignored. If list is omitted, READ (u) spaces over one record.

Examples:

```
READ (30)
READ (31) DATA1, DATA2, IDATA
READ (32) (SUM(K), K=1, 100)
READ (33) I, J, K, L, M, N
```

WRITE WRITE (u) k or WRITE (u)

This form of the WRITE statement creates the next record on the unit identified by u. The contents of the record are the sequence of values specified by the list k.

Examples:

```
WRITE (30) (DATA(I), I=1, 100)
WRITE (31) I, J, K, R
WRITE (32) PAY, COST, BAL
```

If the list is omitted from the statement, a null record is written on the output device. A null record is a record which consists of no data but contains all the other properties of a legitimate record.

Example:

```
WRITE (14)
```


5.6 NAMELIST STATEMENT

The NAMELIST statement permits the input and output of character strings consisting of names and values without a format specification.

```
NAMELIST /y1/a1/y2/a2/.../yn/an
```

Each y is a NAMELIST group name consisting of 1-7 characters which must be unique within the program unit in which it is used. Each a is a list of the form b₁,b₂,...,b_n; each being a variable or array name.

In any given NAMELIST statement, the list a of variable names or array names between the NAMELIST identifier y and the next NAMELIST identifier (or the end of the statement if no NAMELIST identifier follows) is associated with the identifier y.

Examples:

```
PROGRAM MAIN
NAMELIST/NAME1/N1, N2, R1, R2/NAME2/N3, R3, N4, N1

SUBROUTINE XTRACT (A, B, C)
NAMELIST/CALL1/L1, L2, L3/CALL2/L3, P4, L5, B
```

A variable name or array name may be an element of more than one such list. In a subprogram, b may be a dummy parameter identifying a variable or an array, but the array may not have variable dimensions.

A NAMELIST group name may be defined only once in a program unit preceding any reference to it. Once defined, any reference to a NAMELIST name may be made in a READ, WRITE, PRINT, or PUNCH statement. The form of the input/output statements used with NAMELIST is as follows:

```
READ (u, x)
READ x
WRITE (u, x)
PRINT x
PUNCH x
```

u is an integer variable or integer constant denoting a logical unit, and x is a NAMELIST group name.

Example:

```
Assume A, I, and L are array names
:
NAMELIST /NAM1/A, B, I, J/NAM2/C, K, L
:
READ (5, NAM1)
:
WRITE (8, NAM2)
```

These statements result in the BCD (coded) input/outputs on the device specified as the logical unit of the variables and arrays associated with the identifiers, NAM1 and NAM2.

INPUT DATA

The current file on unit u is scanned up to an end-of-file or a record with a \$ in column 2 followed immediately by the name (NAM1) with no embedded blanks. Succeeding data items are read until a \$ is encountered.

The data item, separated by commas, may be in any of three forms:

```
v = c
a = d1, . . . , dj
a(n) = d1, . . . , dm
```

v is a variable name, c a constant, a an array name, and n is an integer constant subscript. d_i are simple constants or repeated constants of the form k*c, where k is the repetition factor.

Example:

```
DIMENSION Y(3, 5)
LOGICAL L
COMPLEX Z
NAMELIST /HURRY/I1, I2, I3, K, M, Y, Z, L
READ (5, HURRY)
```

and the input record:

```
$HURRY I1=1, L=. TRUE. , I2=2, I3=3. 5, Y(3, 5)=26, Y(1, 1)=11, 12. 0E1, 13, 4*14,  
Z=(1. , 2. ), K=16, M=17$
```

produces the following values in memory:

I1=1	Y(1, 2)=14. 0
I2=2	Y(2, 2)=14. 0
I3=3	Y(3, 2)=14. 0
Y(3, 5)=26. 0	Y(1, 3)=14. 0
Y(1, 1)=11. 0	K=16
Y(2, 1)=120. 0	M=17
Y(3, 1)=13. 0	Z=(1. , 2.)
	L=. TRUE.

The number of constants, including repetitions, given for an unsubscripted array name must equal the number of elements in that array. For a subscripted array name, the number of constants need not equal, but may not exceed, the number of array elements needed to fill the array.

$v=c$	variable v is set to c
$a=d_1, \dots, d_j$	the values d_1, \dots, d_j are stored in consecutive elements of array a in the order in which the array is stored internally.
$a(n)=d_1, \dots, d_m$	elements are filled consecutively starting at $a(n)$

The specified constant of the NAMELIST statement may be integer, real, double precision, complex of the form (c_1, c_2) , or logical of the form $.T.$, or $.TRUE.$, $.F.$, or $.FALSE.$. A logical or complex variable may be set only to a logical and complex constant, respectively. Any other variable may be set to an integer, real or double precision constant. Such a constant is converted to the type of its associated variable.

Constants and repeated constant fields may not include embedded blanks. Blanks, however, may appear elsewhere in data records.

A maximum of 150 characters per input record is permitted. More than one record may be used for input data. All except the last record must end with a constant followed by a comma, and no serial numbers may appear; the first column of each record is ignored.

The set of data items may consist of any subset of the variable names associated with x . These names need not be in order in which they appear in the defining NAMELIST statement.

OUTPUT DATA

When a NAMELIST group name is referenced in a WRITE (u,x), PRINT x, or PUNCH x statement, the entire list associated with that name is output as BCD information. Output consists of at least three records. The first record is a \$ in column 2 followed by the group identifier x; the last record is a \$ in column 2 followed by the letters END. Between these two records are as many records as necessary to output the current values of all variables in the list associated with x.

Each variable or array is output as a separate record, with no data appearing in column 1 of any record. Simple variables are output as v = c. Elements of dimensioned variables are output in the order in which they are stored internally. Logical constants appear as T and F. The data fields are made large enough to include all significant digits.

The records output by a WRITE (u,x) statement may be read by a READ (u,x) statement. The maximum length of a record written by a WRITE (u,x) statement is 130 characters. If unit u is the standard punch unit and a record to be output contains more than 80 characters, a second card is used for the record.

5.7

REWIND

REWIND u

This statement positions unit u at its initial point. If the statement is not applicable to the unit specified or u is at the initial point, the statement has no effect.

Example:

```
REWIND 31
REWIND L
```

5.8

BACKSPACE

BACKSPACE u

Execution of this statement positions unit u so that what had been the preceding user logical record becomes the next record. If the statement is not applicable to the unit specified or unit u is at the initial point, the statement has no effect.

Example:

```
BACKSPACE 40
BACKSPACE K
```

5.9 ENDFILE

ENDFILE u

When this statement is executed, an end-of-file record is written on unit u. The end-of-file record indicates a demarcation of a file.

Example:

```
ENDFILE 31
ENDFILE M
```

5.10 ECS I/O

The following statements result in data transmission between ECS (Extended Core Storage) and central memory.

```
CALL READEC (a,b,n)
CALL WRITEC (a,b,n)
```

- a Simple or subscripted variable located in central memory.
- b Simple or subscripted variable located in ECS common block.
- n Integer constant or integer expression.

When either statement is executed, n consecutive words of data are transmitted between central memory and ECS beginning at location a in central memory and b in Extended Core Storage.

5.11 MASS STORAGE I/O

Four object time subroutines control record transmission between central memory and a mass storage device. The references to these routines take the following forms:

```
CALL OPENMS (u,ix,l,p)          CALL WRITMS (u,fwa,n,i)
CALL READMS (u,fwa,n,i)        CALL STINDX (u,ix,l)
```

- u Logical unit number.
- ix First word address of the index (in central memory).
- l Length of the index; $l \geq 2$ (number of index entries)+1 for a name index; $l \geq$ number of index entries+1 for a number index.

Movement of data from disk to large core memory (LCM) is controlled by the SCOPE operating system; it is initiated when the LCM buffer pointers cross a threshold.

- p=1 Indicates the file is referenced through a name index, p=0 indicates a number index.
- fwa Central memory address of the first word of the record.
- n Number of central memory words to be transferred.
- f Record number or the address of a cell containing the record name (left justified display code with zero fill, 1 to 7 characters) or number.

OPENMS is used to open the mass storage file. This routine informs SCOPE that this file is a random access file. If the file exists, the master index is read into the area specified by the program. OPENMS must be called before READMS, WRITMS, and STINDX, and must be called only once per file.

The routines READMS and WRITMS perform the actual transfer of data to or from central memory.

STINDX is used to change the file index to the base specified in the CALL (See Appendix I for further information and examples concerning the use of these routines.)

The random access name must be left justified display code, from 1-7 characters long, with zero fill.

Random file processing employs a 30-bit index in 7000 SCOPE version 1 rather than an 24-bit index as in 6000 SCOPE Version 3.0.

The FORMAT statement is used in conjunction with the input/output of formatted records to indicate the manner of converting and editing information between the internal representation and the external character strings.

**6.1
FORMAT
DECLARATION**

FORMAT (q₁t₁z₁t₂z₂...z_{n-1}t_nq₂)

- q series of slashes (optional)
- t field descriptor or groups of field descriptors
- z field separator
- n may be zero

The FORMAT declaration is non-executable and may appear anywhere in the program. It must have a statement label in columns 1-5. FORMAT statements are analyzed for validity by the compiler. Diagnostics are provided.

FIELD DESCRIPTORS

The format field descriptors are:

- srEw.d Single precision floating point with exponent
- srFw.d Single precision floating point without exponent
- srGw.d Single precision floating point with or without exponent
- srDw.d Double precision floating point with exponent
- rIw Decimal integer conversion
- rLw Logical conversion
- rAw Alphanumeric conversion
- rRw Alphanumeric conversion
- rOw Octal integer conversion
- nHh₁h₂...h_n Hollerith character control
- nX Intraline spacing
- *...* or ≠...≠ Hollerith string delimiters
- Tn Column tabulation

E, F, G, D, I, L, A, R, O, H, X, and T are the conversion codes which indicate the type of conversion and editing.

w and n are non-zero integer constants which represent the field width in the external character string. n used with T indicates the beginning column position for subsequent information.

d is an integer constant which represents the number of digits in the fractional part of the external character strings (except for G conversion).

r is the repeat count. It is represented by an optional non-zero integer constant and indicates the repetition factor of the succeeding basic field descriptor.

s is optional and represents a scale factor.

h is one of the characters in the machine character set.

* or ≠ is used to delimit Hollerith strings. (≠ prints as ' on many printers.)

For all descriptors, the field width w or n must be specified. If d is not specified for w.d, it is assumed to be zero.

FIELD SEPARATORS

The two format field separators are the slash (/) and the comma (,). Series of slashes are another form of field separator. Field separators are used to separate field descriptors and groups of field descriptors. The slash is also used to specify demarcation of formatted records.

6.2 CONVERSION SPECIFICATION

Leading blanks are not significant in numeric input conversions; other blanks are treated as zeros. Plus signs may be omitted. An all blank field is considered to be minus zero, except for logical input, where an all blank field is considered to be FALSE. When an all blank field is read with a Hollerith input specification (R or A), each blank character will be translated into a display code 55 octal.

For the E, F, G, and D input conversions, a decimal point in the input field overrides the decimal point specification of the field descriptor.

The output field is right justified for all output conversions. If the number of characters produced by the conversion is less than the field width, leading blanks are inserted in the output field. The number of characters produced by an output conversion must not be greater than the field width. If the field width is exceeded, an asterisk is inserted in the leading position of the field.

Any output which is sent to the line printer uses the first character on the left for carriage control. Thus, the first character is lost and printing begins in the first print position using the second character. This applies only to line printers, not to other output devices.

lw INPUT

This specification, in conjunction with an input statement, designates a decimal integer constant; field length of w characters. The input field is an optionally signed integer or blank. When a sign appears, it must precede the first digit in the field. Blanks are interpreted as zeros. The value is stored right-justified in the specified variable.

Example:

```

          READ 10, I, J, K, L, M, N
10      FORMAT (I3, I7, I2, I3, I2, I4)

Stored Variable:  I    J    K  L  M  N
Input Card:      139bb-15bb18bb7b3b1b4
Field Width:     3    7    2  3  2  4
  
```

lw OUTPUT

I specification may also be used to indicate the output of decimal integer values. The output quantity occupies w output character positions, right justified:

ba...a

where b is a blank or minus sign if the integer is negative, a's are the digits (maximum 15) of the integer. If the integer is positive, the + sign is suppressed. If the field width w is larger than required, the output quantity is right justified with blank fill to the left. If the field is too short, characters are stored from the right; an asterisk occupies the leftmost position, with excess characters being discarded from the left. If the integer is greater than $2^{48}-1$, an X is printed in the field.

Example:

```

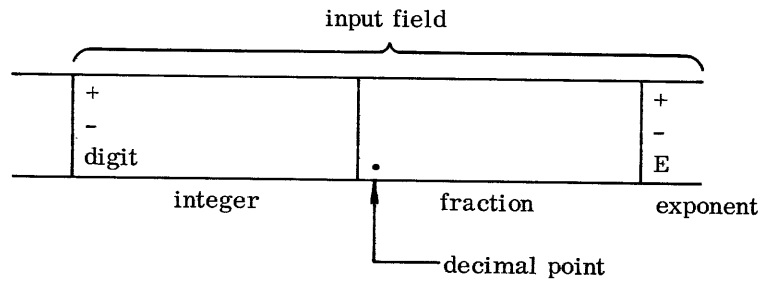
          PRINT 10, I, J, K          I contains -3762
10      FORMAT (I8, I10, I5)       J contains +4762937
                                         K contains +13

Result:  bbb-3762bbb4762937bbb13
          8      10      5
  
```

Ew.d INPUT

The E specification designates the conversion and storing of a number in the input field as a real number. The total number of characters in the input field is specified by w; this field is scanned from left to right; blanks are interpreted as zeros.

Subfield structure of the input field:



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, D, E, +, -, 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 D, E, +, -, or the end of the input field.

The exponent subfield may begin with D, E, + or - followed by an integer constant right adjusted in the field. When it begins with D, or E, a sign is optional between D or E and the string of digits of the subfield. The value of the string of digits in the exponent subfield must be less than 323.

Permissible subfield combinations:

+1.6327E-04	integer fraction exponent
-32.7216	integer fraction
+328+5	integer exponent
.629E-1	fraction exponent
+136	integer only
.07628431	fraction only
E-06 (interpreted as zero)	exponent only

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:

$$(\text{integer subfield}) \times 10^{-\underline{d}} \times 10^{(\text{exponent subfield})}$$

For example, if the specification is E7.8, the input quantity 3267+05 is converted and stored as: $3267 \times 10^{-8} \times 10^5 = 3.267$.

A decimal point in the input field overrides d. The input quantity 3.672+5 read by an E9.d specification is always stored as 3.672×10^5 . When d does not appear, it is assumed to be zero.

The field length specified by w in Ew.d should always be the same as the length of the field containing the input number. When it is not, incorrect numbers may be read, converted, and stored as shown below. The field w includes blanks, significant digits, signs, decimal point, E or D and the exponent.

Example:

```
READ 20, A, B, C
20  FORMAT (E9.3, E7.2, E10.3)
```

Input quantities on the card are in three contiguous fields columns 1 through 24:

$$\begin{array}{c} \underbrace{\hspace{2em}}_9 \quad \underbrace{\hspace{1em}}_5 \quad \underbrace{\hspace{3em}}_{10} \\ +6.47E-01 \ -2.36+5 \ .321E+02 \text{bb} \end{array}$$

The second specification (E7.2) exceeds the width of the second field by two characters.

Reading proceeds as follows:

$$\begin{array}{c} \underbrace{\hspace{2em}}_9 \quad \underbrace{\hspace{1em}}_7 \quad \underbrace{\hspace{3em}}_{10} \\ \boxed{+6.47E-01} \ -2.36+5 \ .321E+02\text{bb} \\ +6.47E-01 \ \boxed{-2.36+5} \ .321E+02\text{bb} \\ +6.47E-01 \ -2.36+5 \ \boxed{.321E+02\text{bb}} \end{array}$$

First, +6.47-01 is read, converted, and placed in location A. Next, -2.36+5 is read, converted, and placed in location B. The number actually desired was -2.36, but the specification error (E7.2 instead of E5.2) caused the two extra characters to be read. The number read (-2.36+5) is a legitimate input representation under the definitions and restrictions.

Finally, .321E+0200 is read, converted, and placed in location C. Here again, the input number is legitimate and is converted and stored, even though it is not the number desired.

The above example illustrates a situation where numbers are incorrectly read, converted, and stored, and yet there is no immediate indication that an error has occurred.

Examples:

<u>Input Field</u>	<u>Ew. d Input Specifi- cation</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 - only
-.0003627E5	E11.7	-36.27	Integer subfield contains - 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 sub- field; zero stored regardless of exponent field contents
1.bEb1	E6.3	10.	Blanks are interpreted as zeros

Ew.d OUTPUT

Real numbers in storage are converted to the BCD character form for output with the E conversion. The field occupies w positions in the output record; with the real number right justified in the form:

$$\begin{array}{l} \text{b.a...a}\pm\text{eee} \quad 100 \leq \text{eee} \leq 308 \\ \text{or} \\ \text{b.a...aE}\pm\text{ee} \quad 0 \leq \text{ee} \leq 99 \end{array}$$

b indicates no character position or minus sign; a's are the most significant digits of the value, and eee are the digits in the exponent. If d is zero or no character, the digits to the right of the decimal do not appear as shown above. Field w must be wide enough to contain the significant digits, sign (if negative), decimal point, E, and the exponent. Generally, $w \geq d+6$. Since positive numbers do not require a sign, space need not be reserved for one.

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 blank fill to the left. If the value being converted is indefinite, an I is printed in the field; if it is out of range, an R is printed.

Examples:†

```
PRINT 10,A          A contains -67.32 or +67.32
10 FORMAT (E10.3)
```

Result: b-.673E+02 or bb.673E+02

```
PRINT 10,A
10 FORMAT (E13.3)
```

Result: bbbb-.673E+02 or bbbbbb.673E+02

```
PRINT 10,A          A contains -67.32
10 FORMAT (E8.3)     no provision for - sign
```

Result: *.67E+02

```
PRINT 10,A
10 FORMAT (E10.6)
```

Result: *.6732E+02

Fw.d INPUT

This specification is the same as Ew.d input specification. It may be used for the transfer of real data that does not contain a decimal exponent.

†In the examples, the use of column 1 for carriage control has been ignored. The results demonstrate the way in which data is converted, not the way the line will appear when printed.

Fw.d OUTPUT

The field occupies w positions in the output record; the corresponding list item must be a floating point quantity, which appears as a decimal number, right justified

ba...a.a...a

b identifies a minus sign or no character position and a's represent the most significant digits of the number.

The number of decimal places to the right of the decimal is specified by d. If d is zero or omitted, digits to the right of the decimal point do not appear. If the number is positive, the + 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. Field w must be wide enough to contain significant digits, sign (if negative), and a decimal point. If the field is longer than required to accommodate the number, the number is right justified with blank fill to the left. If the value being converted is indefinite, an I is printed in the field; if it is out of range, an R is printed.

Examples:†

A contains +32.694

```
PRINT 10,A
10 FORMAT(F7.3)
```

Result: b32.694

```
PRINT 11,A
11 FORMAT(F10.3)
```

Result: bbbb32.694

A contains -32.694

```
PRINT 12,A
12 FORMAT(F6.3)    no provision for - sign
```

Result: *2.694

A contains .32694

```
PRINT 13,A,A
13 FORMAT(F4.3,F6.3)
```

Result: .327bb.327

†In the examples, the use of column 1 for carriage control has been ignored. The results demonstrate the way in which data is converted, not the way the line will appear when printed.

Gw.d INPUT

Gw.d input specification is the same as the Ew.d input specification.

Gw.d OUTPUT

The G conversion specifies the transfer of real data where w designates the field length and d denotes the number of significant digits of the value to be represented.

The method of representation in the external output string is a function of the magnitude of the real datum being converted. Let N be the magnitude of the internal datum. The following tabulation exhibits a correspondence between N and the equivalent method of conversion that will be effected:

Magnitude of Datum	Equivalent Conversion Effected
$0.1 \leq N < 1$	F(w-4).d,4X
$1 \leq N < 10$	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	sEw.d

The effect of the scale factor is suspended unless the magnitude of the datum to be converted exceeds the range that permits effective use of the F conversion. If the value being converted is indefinite, an I is printed in the field; if it is out of range, an R is printed.

When F conversion is used under Gw.d output specification, four blanks are inserted within the field, right justified. Therefore, for effective use of F conversion, w must be $\geq d+6$.

Examples:

```

      PRINT 101,XYZ    XYZ contains 77.132
101  FORMAT (G10.3)

      Result: bb77.1bbbb

```

```

      PRINT 101,XYZ    XYZ contains 1214635.1
101  FORMAT (G10.3)

      Result: bb.121E+07

```

Dw.d OUTPUT

D conversion corresponds to Ew.d output. The field occupies w positions of the output record, the list item is a double precision quantity which appears as a decimal number, right justified. If the value being converted is indefinite, an I is printed in the field; if it is out of range, an R is printed.

$$\begin{array}{l}
b.a\cdots a\pm eee \quad 100 \leq eee \leq 308 \\
\text{or} \\
b.a\cdots aD\pm ee \quad 0 \leq ee \leq 99
\end{array}$$

Dw.d INPUT

D conversion corresponds to E conversion except that the list variables must be double precision names. D is acceptable in place of E as the beginning of an exponent subfield.

Example:

```

DOUBLE Z,Y,X
READ1, Z,Y,X
1 FORMAT (D18.11,D15,D17.4)

```

Input Card:

$\underbrace{-6.31675298443D-03}_{18} + \underbrace{2.7189264531476293477528869D-09}_{15}$

Ow OUTPUT

O specification is used to output octal values. The output quantity occupies w output character positions right justified.

aa...a

The a's are octal digits. If w is 20 or less, the rightmost w digits appear. If w is greater than 20, the number is right justified in the field with blanks to the left of the output quantity. A value output under O specification undergoes no type conversion; a negative number is output in its one's complement internal form.

Ow INPUT

Octal values are input under O specification. The field is w characters in length.

The input field w consists of an integer subfield only (maximum of 20 octal digits) containing +, -, 0 through 7, or blank. Only one sign may precede the first digit in the field. Embedded blanks are interpreted as zeros.

Example:

```
INTEGER P, Q, R
READ 10, P, Q, R
10 FORMAT (O10, O12, O2)
```

Input Card: 3737373737666b6644b444-0

 └───┬───┬───┘
 10 12 2

In storage:

```
P 000000000037373737
Q 00000000666066440444
R 777777777777777777
```

A negative octal number is represented internally in one's complement form (20 digits) 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, 777777777777777777
 -000000000000000703
 ┌───────────────────┘
 777777777777777074

Aw OUTPUT

A conversion is used to output alphanumeric characters. If w is 10 or more, the quantity appears right justified in the output field, blank fill to left. If w is less than 10, the output quantity is represented by leftmost w characters.

Aw INPUT

This specification accepts FORTRAN characters including blanks. The internal representation is 6000 Series 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 is stored as a left justified BCD word; the remaining spaces are blank filled.

Example:

```
READ 10, Q, P, O
10 FORMAT (A8, A8, A4)
```

Input card: LUX MENTIS LUX ORBIS

 └───┬───┬───┘
 8 8 4

In storage: Q LUXbMENTbb
 P ISbLUXbObb
 O RBISbbbbbb

Rw OUTPUT This specification is similar to the Aw output with the following exception. If w is less than 10, the output quantity represents the rightmost characters.

Rw INPUT This specification is the same as the Aw input with the following exception. If w is less than 10, the input quantity is stored as a right justified binary zero filled word.

Example:

```
READ 10, Q, P, O
10 FORMAT (R8, R8, R4)
```

Input card: LUX MENTIS LUX ORBIS
 8 8 4

In storage: Q 00LUXbMENT
 P 00ISbLUXbO
 O 000000RBIS

Lw OUTPUT L specification is used to output logical values. The output field is w characters long, and the list item must be a logical element. A value of TRUE or FALSE in storage causes w-1 blanks followed by a T or F to be output.

Example:

```
LOGICAL I, J, K, L      I, K, L are negative (TRUE) and
PRINT 5, I, J, K, L    J is positive (FALSE)
5 FORMAT (4L3)
```

Result: bbTbbFbbTbbT

Lw INPUT This specification accepts logical quantities as list items. The field is considered true if the first non-blank character in the field is T or false if it is F. An all blank field is considered false. If the first non-blank character is neither T nor F, the field is considered false.

COMPLEX CONVERSIONS

The specification by which a complex variable is read or written requires the designation of two real field descriptors: the first designates the real part, the second the imaginary part. The field descriptors that may be used are: E (Ew.d), F (Fw.d), or G (Gw.d).

Example:

```
INTEGER A
COMPLEX CC           where A = 3762
PRINT 20,A,B,CC,D   B = 833.275
FORMAT (I5,F8.3,E10.4,E9.2,G11.5) CC = 36.292, -46.73
D = .62534
```

Results: b3762 b833.275 b.3629E+02b-.47E+02 b.62534bbbb

nP SCALE FACTOR

A scale factor that may be used with F, E, G, and D conversions is of the form:

```
nP
nPFw.d
nPEw.d
nPGw.d
nPDw.d
```

n, the scale factor, is a positive (unsigned) or negative integer constant.

A scale factor of zero is established when the format control is initiated; it holds for all F, E, G, and D field descriptors until another scale factor is encountered.

The scale factor n affects conversions as follows:

For F, E, G, and D input conversions (provided no exponent exists) in the external field) and F output conversions: External number = Internal number $\times 10^n$

For F, E, G, and D input, the scale factor has no effect if there is an exponent in the external field.

For E and D output, the basic real constant part of the output quantity is multiplied by 10^n and the exponent is reduced by n.

For G output, the effect of the scale factor is suspended unless the magnitude of the data to be converted exceeds the range that permits effective use of F conversion. If the effective use of the E conversion is required, the scale factor has the same effect as with E output.

Examples:

Using an internal number of 3.1415926538, some output representations with the use of a scale factor are:

<u>Specification</u>	<u>Output Representation</u>
E20.2	.31E+01
1PE20.2	3.14E+00
4PE20.2	3141.59E-03
7PE20.2	3141592.65E-06
-1PE20.2	.03E+02
5PF20.2	314159.27
-2PF20.4	.0314

6.3 EDITING SPECIFICATIONS

nX

This specification permits spacing of input/output quantities; it permits blanks to be inserted in an output record or n characters to be skipped in an input record. The designation of 0X is ignored and bX is interpreted as 1X. In the specification list, a comma following X is optional.

Examples:

```
INTEGER A                                A contains 7, B contains 13.6,  
PRINT 10, A, B, C                        C contains 1462.37  
10  FORMAT (I2, 6X, F6.2, 6X, E12.5)  
      Result: b7bbbbbbb13.60bbbbbbb.14624E+04
```

```
READ 11, R, S, T  
11  FORMAT (F5.2, 3X, F5.2, 6X, F5.2)
```

or

```
11  FORMAT (F5.2, 3XF5.2, 6XF5.2)
```

Input card: 14.62bb\$13.78bCOSTb15.97

```
In storage: R  14.62  
            S  13.78  
            T  15.97
```

nH

This specification provides for the input or output of 6-bit characters, including blanks, in the form of comments, titles, and headings. An unsigned integer n specifies the number of characters, maximum of 136 to the right of H that are transmitted to the output record; H denotes a Hollerith field; the comma following an H field is optional.

Examples:

Source program:

```
PRINT 20
20 FORMAT (28HbBLANKSbCOUNTbINbANbHbFIELD.)
```

produces output record:

bBLANKSbCOUNTbINbANbHbFIELD.

Source program:

```
PRINT 30,A           A contains 1.5
FORMAT (6HbLMAX=,F5.2) comma is optional
```

produces output record:

bLMAX = b1.50

The H specification may be used to read Hollerith characters into an existing H field within the FORMAT specification.

Example:

Source program:

```
READ 10
10 FORMAT (27Hbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb)
```

Input card:

bTHISbISbAbVARIABLEbHEADING

27 columns

After 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

produces the print line:

bTHISbISbAbVARIABLEbHEADING

NEW RECORD

The slash (/) indicates the end of the last record anywhere in the specification list. Consecutive slashes may appear and need not be separated from the other list elements by commas. During output, the slash is used to produce blank records. During input, it is used to bypass records. $k(/)$ is equivalent to $/_1/_2, \dots, /_k$.

Examples:

1. PRINT 10
10 FORMAT (6X, 7HHEADING///3X, 5HINPUT, 2X, 6HOUTPUT)

Printout:

```
bbbbbbHEADING          line 1
                        (blank) line 2
                        (blank) line 3
bbbINPUTbbOUTPUT      line 4
```

Each line corresponds to a BCD record. The second and third records are null and produce the line spacing illustrated.

2. PRINT 10, A, B, C, D
10 FORMAT (2E10.3/2F7.3)

In storage: A -11.6
 B .325
 C 46.327
 D -14.261

Printout:

```
b-.116E+02bb.325E+00
b46.327-14.261
```

3. PRINT 11, A, B, C, D
11 FORMAT (2E10.3//2F7.3)

Printout:

```
b-.116E+02bb.325E+00          line 1
                                (blank) line 2
b46.327-14.261                line 3
```

```

4.      DIMENSION X(3)
        PRINT 15, (X(I), I=1, 3)
15     FORMAT (8HbRESULTS2(/)(3F8.2))

```

Resultant lines:

bRESULTS			line 1
		(blank)	line 2
3.62	-4.03	-9.78	line 3

The same results may also be obtained by using the statement,
PRINT 15,X

... ≠...≠

Hollerith string delimiters are *...* and ≠...≠. All characters (including blanks) enclosed by a pair of delimiters are read or written. Each character may appear in a field delimited by the other. In an nH delimited specification, the * or ≠ (' for some printers) will be reproduced.

Example:

```

        PRINT 10
10     FORMAT (20X*THISbISbTHEbENDbOFbTHISbRUN*,T52*... HONEST*)

```

Result: (beginning in print position 20)

```

123456789012345678901234567890123456789012345678901234567890
                    THIS IS THE END OF THIS RUN    ...HONEST

```

Tn

This specification is used as a tabular column selection control. When Tn is used, the format pointer is skipped to column n and the next format specification is processed. n may be any unsigned integer, maximum of 136. If n = zero, column 1 is assumed. (If output is to a line printer, printing is left-shifted one character due to carriage control requirements.)

Using card input, if n > 80 the column pointer is moved to column n but a succeeding specification would read only blanks.

Examples:

```

1)      PRINT 60
        60  FORMAT (T80,*COMMENTS*, T60,*HEADING4*, T40
                  *HEADING3*, T20,*HEADING2*, T2,*HEADING1*)

```

Produces the following output: print positions are indicated by the upper line of numbers 1-80.

```

1           19           39           59           79
HEADING1   HEADING2   HEADING3   HEADING4   COMMENTS

```

```

2)         WRITE (31,10)
          10  FORMAT (T20,*LABELS*)

```

The first 19 characters of the output record are skipped and the next six characters, LABELS, are written on output unit number 31 beginning in character position 20.

```

3)         READ (20,20)
          20  FORMAT (T10,*COLUMN1*)

```

The first nine characters of the input record are skipped and the next seven are read from input file 20; these seven characters replace COLUMN1, the data in storage.

6.4 REPEATED FORMAT SPECIFICATIONS

FORMAT specifications may be repeated by using an unsigned integer constant repetition factor k as follows: $k(\text{spec})$. For example, to print the array Y:

```

          PRINT 10, (Y(I), I=1, 9)
          10  FORMAT (3(3F8.3))

```

is equivalent to:

```

          PRINT 10, (Y(I), I=1, 9)
          10  FORMAT (9F8.3)

```

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))
```

If no group repetition factor is specified, a basic group (repetition factor of one) is assumed. If, however, the format control proceeds to the last outer right parenthesis of the format specification, a test is made to determine if another list element is specified. If not, control terminates. However, if another list element is specified, the format control demands a new record start and control reverts to that group repeat specification terminated by the last preceding right parenthesis, or if none exists, then to the first left parenthesis of the format specification.

Further groupings may be formed by enclosing field descriptors, field separators, or basic groups within parentheses, and a group repetition factor may be specified for these groupings. The parentheses enclosing the format specification are not considered as group delimiting parentheses.

FORMAT statement specifications may be nested to a depth of two. For instance:

```
10  FORMAT(1H0,3E10.3/(I2,2(F12.4,F10.3))/D28.17)
```

6.5

VARIABLE FORMAT

FORMAT specifications may be indicated at the time of program execution. The specification, including left and right parentheses but not the statement label or the word FORMAT, must be Hollerith data stored in an array. The name of the array containing the specifications may be used in place of the FORMAT statement labels in the associated input/output operation. The array name specifies the location of the first word of the FORMAT information and may appear with or without a subscript.

Examples:

- 1) Assume the following FORMAT specifications:

```
(E12.2,F8.2,I7,2E20.3,F9.3,I4)
```

This information can be punched in an input card and read by the statements of the program such as:

```
DIMENSION IVAR(3)
READ 1, (IVAR(I), I=1, 3)
1  FORMAT (3A10)
```

The elements of the input card are placed in storage as follows:

```
IVAR(1):  (E12.2,F8.
IVAR(2):  2,I7,2E20.
IVAR(3):  3,F9.3,I4)
```

A subsequent output statement in the same program can refer to these FORMAT specifications as:

```
PRINT IVAR,A,B,I,C,D,E,J
```

This produces exactly the same result as the program:

```
PRINT 10,A,B,I,C,D,E,J
10  FORMAT (E12.2,F8.2,I7,2E20.3,F9.3,I4)
```

```
2) DIMENSION LAIS1(3), LAIS2(2), A(6), LSN(3), TEMP(3)
DATA LAIS1/21H(2F6.3, I7, 2E12.2, 3I1)/LAIS2/20H(I6, 6X, 3F4.1, 2E12.2)/
```

Output statement:

```
PRINT LAIS1, (A(I), I=1, 2), K, B, C, (LSN(J), J=1, 3)
```

which is the same as:

```
PRINT 1, (A(I), I=1, 2), K, B, C, (LSN(J), J=1, 3)
1 FORMAT (2F6.3, I7, 2E12.2, 3I1)
```

Output statement:

```
PRINT LAIS2, LA, (A(M), M=3, 4), A(6), (TEMP(I), I=2, 3)
```

which is the same as:

```
PRINT2, LA, (A(M), M=3, 4), A(6), (TEMP(I), I=2, 3)
2 FORMAT (I6, 6X, 3F4.1, 2E12.2)
```

```
3) DIMENSION LAIS(3), VALUE(6)
DATA LAIS/26H(I3, 13HMEANbVALUEbIS, F6.3)/
```

Output statement:

```
WRITE (10, LAIS)NUM, VALUE(6)
```

which is the same as:

```
WRITE(10, 10)NUM, VALUE(6)
10 FORMAT(I3, 13HMEANbVALUEbIS, F6.3)
```

7.1 BUFFER STATEMENTS

Some of the characteristics of buffered input/output are given below:

1. The mode of transmission (BCD or binary) is tacitly implied by the form of the input/output control statements. In a buffer control statement, parity must be specified by a parity indicator.
2. The input/output control statements are associated with a list and in BCD transmission, with a FORMAT statement. The buffer control statements do not have a list; data transmission is to or from one area in storage.
3. An input/output control statement does not return control to the program until completion of the operation. A buffer control statement initiates data transmission, then returns control to the program, permitting the program to perform other tasks while data transmission is in progress. Before buffered data is used, status of the buffer operation should be checked through the UNIT function (Appendix I). Failure to perform a status check renders the result of the last buffer operation unpredictable.

In the following discussion, the parameters indicated are defined as follows:

- u Logical unit number; an integer constant or variable in the range 1 to 99.
- p Recording mode; an integer constant or variable which may assume the values of zero, designating even parity (coded mode), or 1 indicating odd parity (binary mode). The recording mode, p, is inoperative for mass storage units on the 7600, but it still must be specified. Tape parity is controlled by the input/output stations servicing the 7600.
- fwa First word address of the block of data to be transmitted.
- lwa Last word address of the block of data to be transmitted. This address must be greater than or equal to fwa.

A unit referenced in a BUFFER statement may not be referenced in other input/output statements except REWIND, BACKSPACE and ENDFILE.

BUFFER IN

BUFFER IN (u,p) (fwa,lwa)

This statement transfers information from unit u in mode p to storage locations fwa through lwa. Only one logical record is read for each BUFFER IN statement.

BUFFER OUT

BUFFER OUT (u,p) (fwa,lwa)

This statement initiates output of data from locations fwa through lwa onto unit u. One logical record is written for each BUFFER OUT statement.

A more detailed discussion of these statements is given in Appendix I.

7.2

ENCODE/DECODE STATEMENTS

The ENCODE/DECODE statements are comparable to the formatted WRITE/READ statements; however, no peripheral equipment is involved. Information is transferred under FORMAT specifications from one area of storage to another. The parameters in these statements are defined as follows:

- n Unsigned integer constant or a simple integer variable (not subscripted) specifying the number of characters in the record. n may be an arbitrary number of BCD characters.
- f Statement number or array identifier representing the FORMAT statement.
- id Identifier of a variable or an array which supplies the starting location of the BCD record.
- k Input/output list.

The first record begins with the leftmost character position specified by id and continues until n BCD characters have been transferred (10 BCD characters per computer word).

Each succeeding record begins with a new computer word, the integral number of computer words allocated for each record is $\frac{n+9}{10}$.

Further information on these statements is given in Appendix I.

ENCODE

ENCODE (n,f,A)k

The list of variables, k, is transmitted according to the FORMAT f and stored, n BCD characters per record, starting at location A. If n is not a multiple of 10, the remainder of the word is blank filled. If the I/O list k and the specification list f translate more than n characters per record, an execution diagnostic occurs.

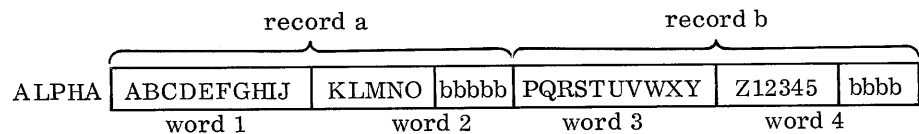
Examples:

A(1) = 10HABCDEFGHJ
 A(2) = 10HKLMNO
 B(1) = 10HPQRSTUVWXYZ
 B(2) = 10HZ12345

1. n = multiple of 10

```
ENCODE (20,1,ALPHA)A,B
1 FORMAT (A10,A5/A10,A6)
```

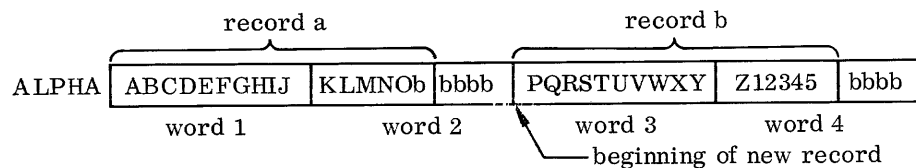
Result:



2. n ≠ multiple of 10

```
ENCODE (16,1,ALPHA)A,B
1 FORMAT (A10,A6)
```

Result:



3. ENCODE can be used to rearrange and change the information in a record. The following example also illustrates that it is possible to encode an area into itself and that encoding will destroy information previously contained in an area.

```

PROGRAM ENCO2(OUTPUT)
I = 10RBCDEFGHIJK
IA = 1H1
ENCODE (8, 10, D)I, IA, I
10 FORMAT (A3, A1, R4)
PRINT 11, I
11 FORMAT (O20)
END

```

Print-out is:

```
02030434101112135555
```

The display code equivalent is:

```
BCD1HIJKbb
```

4. ENCODE may be used to calculate a field definition in a FORMAT specification at object time. Assume that in the statement `FORMAT (2A10, Im)` the programmer wishes to specify `m` at some point in the program, subject to the restriction $2 \leq m \leq 9$. The following program permits `m` to vary.

```

IF(M. LT. 10. AND. M. GT. 1)1, 2
1 ENCODE (10, 100, SPECMAT)M
100 FORMAT (7H(2A10, I, I1, 1H))
:
:
PRINT SPECMAT, A, B, J

```

`M` is tested to insure it is within limits. If not, control goes to statement 2 which could be an error routine. If `M` is within limits, ENCODE packs the integer value of `M` with the characters: `(2A10, D)`. This packed FORMAT is stored in SPECMAT. SPECMAT contains `(2A10, Im)`.

`A` and `B` will be printed under specification `A10`, and the quantity `J` under specification `I2`, or `I3`, or ... or `I9` according to the value of `m`.

DECODE

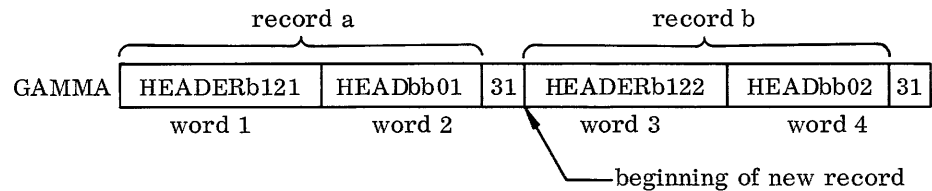
DECODE (n,f,A)k

The information in n consecutive BCD characters (starting at address A) is transmitted according to the FORMAT and stored in the list variables. If the record ends with a partial word the balance of the word is ignored. However, if the number of characters specified by the I/O list and the specification list f is greater than n (record length) per record, an execution diagnostic occurs. If DECODE attempts to process an illegal BCD character for a given conversion specification, a FATAL error 78 diagnostic will occur.

Examples:

1. n ≠ multiple of 10

```
DECODE (18,1,GAMMA) A6,B6
1  FORMAT (A10,A8)
```



Result:

```
A6(1) = HEADERb121
A6(2) = HEADbb01bb
B6(1) = HEADERb122
B6(2) = HEADbb02bb
```

2. The following illustrates one method of packing the partial contents of two words into one word. Information is stored in core as:

```
LOC(1)SSSSxxxxxx
:
:
LOC(6)xxxxxddd
10 BCD ch/wd
```

To form SSSSddddd in storage location NAME:

```
DECODE(10,1,LOC(6))TEMP
1  FORMAT (5X,A5)
ENCODE(10,2,NAME) LOC(1),TEMP
2  FORMAT(2A5)
```

The DECODE statement places the last 5 BCD characters of LOC(6) into the first 5 characters of TEMP. The ENCODE statement packs the first 5 characters of LOC(1) and TEMP into NAME.

With the R specification; the program may be shortened to:

```
        ENCODE (10, 1, NAME) LOC(1), LOC(6)
1      FORMAT (A5, R5)
```


DIMENSION, COMMON, EQUIVALENCE, EXTERNAL, and TYPE statements, are called specification statements. Specification statements are nonexecutable statements which describe the characteristics, allocation and arrangement of data. The ordering of specification statements is immaterial, but they must appear before any statement function definition, DATA, NAMELIST, or executable statements in the program.

**8.1
DIMENSION**

Information necessary to allocate storage for arrays may be provided by the DIMENSION statement.

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

Each v_j is a symbolic name and i_j is the corresponding subscript. Each i_j may consist of one, two, or three integer constants designating the dimensionality for the array and defining the maximum value which a subscript may assume in a subsequent array reference.

Example:

DIMENSION A(20, 2, 5)
DIMENSION MATRIX(10, 10, 10), VECTOR(100)

An array name may not contain a subscript which assumes a value during execution that is less than one or larger than the maximum length specified in the DIMENSION statement. If such a condition exists, an element beyond the array may be referenced. However, a subscript expression which assumes the value zero renders a result which is undefined.

The maximum value a subscript may attain is indicated below:

<u>Dimen-</u> <u>sionality</u>	<u>Subscript</u> <u>Declarator</u>	<u>Subscript</u>	<u>Subscript</u> <u>Value</u>	<u>Maximum</u> <u>Subscript</u> <u>Value</u>
1	(A)	(a)	a	A
2	(A, B)	(a, b)	a+A*(b-1)	A*B
3	(A, B, C)	(a, b, c)	a+A*(b-1) +A*B*(c-1)	A*B*C

a, b, c are subscript expressions.
A, B, C are dimensions.

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. A maximum of $2^{17}-1$ elements may be reserved in any one array. If the maximum is exceeded, a diagnostic is issued.

Example:

```
COMPLEX CELL  
DIMENSION CELL (20,10)
```

The number of elements in the array CELL is 200. Since two words are used to contain a complex element, 400 words are reserved. This is also true for double precision arrays. For real, logical, 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, an informative diagnostic is issued and the first dimensions encountered are retained.

VARIABLE DIMENSIONS

If an entry in a declarator subscript is an integer variable name, the array is variable, and the variable names are called variable dimensions. Such an array may appear only in a procedure subprogram. The dummy argument list of the subprograms must contain the array name and the integer names that represent the variable dimensions. The values of the actual parameter list of the reference must be defined prior to calling the subprogram and may not be redefined or undefined during execution of the subprogram. The maximum size of the actual array may not be exceeded. Every array in an executable program requires at least one associated constant array declaration through subprogram references.

Example:

```
SUBROUTINE XMAX (DATA, K, J)  
DIMENSION DATA (K, 6, J)
```

In a subprogram, a symbolic name that appears in a COMMON statement may not identify a variable array.

DIMENSION statements must appear before any statement function definition, executable, DATA, or NAMELIST statements in the program.

8.2 COMMON

The COMMON statement reserves blocks of storage for variables or arrays appearing in one calling program or subprogram which may be shared and referenced with variables or arrays of other subprograms. The areas of common storage are specified by the statement form:

```
COMMON /x1/a1/.../xn/an
```

Each a is a non-empty set of variable names, array names or array declarators such as, v(i) illustrated for the DIMENSION statement, and each x is a block name. Block names may be symbolic names or integer constants in the range 0 to 9999999, but may not exceed seven characters in length.

Example:

```
COMMON/BLOCK1/A, T(10, 15)/BLOCK2/E, G, Q
```

The list of variable names (A and T or E, G, and Q) may not be dummy parameters. The entries A and T are defined to be in the block labeled BLOCK1 and E, G, and Q are in the block labeled BLOCK2. These blocks are referred to as labeled common. However, if the block name is omitted as in:

```
COMMON/H/D, C, F//U, L, P(12, 12)
```

or

```
COMMON S, V, Z, X, M
```

the list of variables following the empty block name specification are placed in unlabeled or blank common. In the two above examples, D, C, and F are in the block H, whereas U, L, P, S, V, Z, X, and M are defined in unlabeled common.

LABELED COMMON

Any labeled common block may be referred to by any number of programs or subprograms which comprise an executable program. References are made by block name which must be identical in all references. The definition of all labeled common blocks need not be made within any one program, but must be made in the program unit in which the data is needed.

The length of a common block in a program unit is the sum of the storage required for the elements defined by the COMMON statement. The length of labeled common blocks with the same label should be the same.

Example:

```
SUBROUTINE A          SUBROUTINE B
REAL B, W, X(20)      COMPLEX G, F(10)
COMMON/BLKA/V, W, X   COMMON/BLKA/G, F
```

Both references to the COMMON block, BLKA, correspond in size. That is, both subprograms define the block as containing 22 words; subroutine A specifies 22 items of real type and the specification in B indicates 11 items of complex type.

Reference may be made to the name of a labeled common block more than once in any program or subprogram. Multiple references may occur in a single COMMON statement, or the block name may be specified in any number of individual COMMON statements. In both cases, the processor links together all variables into a single labeled common block.

UNLABELED COMMON All variables defined in unlabeled or blank common blocks are assigned together; that is, only one section of the storage allocated for common is assigned to such variables. These variables are always referred to by an unlabeled COMMON statement (block name is omitted).

Unlike labeled common, the sizes specified in various program units to be executed together need not be the same. Size is measured in terms of storage units.

Example:

```
SUBROUTINE ALPHA      SUBROUTINE BETA
COMMON E, F, G(20, 10) COMMON H, A, D, S
      ⋮                ⋮
```

Subroutine ALPHA defines an area of 202 words in unlabeled common, BETA uses only 4 words or a maximum of 8 words of the storage already defined.

ARRANGEMENT OF COMMON BLOCKS

The properties of common block names as used in all of the program units of an executable program are as follows:

Each subprogram using a common block assigns the allocation of words in the block. The entities used within the block may differ as to name, type, and number of storage units although the block identifier itself must remain the same.

When a block is labeled and the entities are defined for the block, the values of identifiers in the corresponding positions (counted by the number of preceding storage units) are the values referenced through COMMON declaration in the executable program. The order of entities in the labeled common block is significant throughout the executable program.

Example:

```
PROGRAM MAIN (INPUT, OUTPUT)
  :
  :
  COMMON A1, A2, L1/B1/B2, B3
  :
  :
  CALL CALL1(S, T, Z)
  :
  :
  END
  SUBROUTINE CALL1(X, Y, Z)
  :
  :
  COMMON A1, D, M/B1/F, G
  :
  :
  END
```

A double precision or a complex entity consists of two logical consecutive storage units: a logical, real, or integer entity is one storage unit.

If any common block elements are type ECS, all the elements of that block must be type ECS. No type ECS elements may appear in the blank common block.

COMMON statements must appear before any statement function definitions, executable, DATA, or NAMELIST statements in the program unit.

8.3 EQUIVALENCE

An EQUIVALENCE statement permits storage to be shared by two or more entities, it does not imply equality of entities. Each element in a given list is assigned the same storage (or part of the same storage) by the processor

EQUIVALENCE (k_1), (k_2), ..., (k_n)

Each k is a list of the form:

a_1, a_2, \dots, a_m

Each a is an array name, a variable name, or an array element name (but not a dummy argument or an ECS variable or array element), the subscripts may contain only constants. m is greater than or equal to two. The number of subscript expressions of an array element name must correspond to the dimensionality of the array declarator, or it must be one.

EQUIVALENCE may not be used to reorder COMMON nor reposition the base. The base of an equivalence group is the element with the smallest address; or, if in common, it is the beginning element of the common block. When an element of an array is referred to in an EQUIVALENCE statement, the relative locations of the other array elements are, thereby, defined also.

The effect of an EQUIVALENCE statement upon common assignment may be the lengthening of a common block beyond the last assignment for that block made by a COMMON statement.

An element or array is brought into COMMON if it is equivalenced to an element in COMMON. Two elements in COMMON must not be equivalenced to each other.

The following examples illustrate changes in block lengths as the result of EQUIVALENCE declaration.

Given: Arrays A and B

Sa subscript of A

Sb subscript of B

Examples:

1. A and C in common, B not in common

Sb \leq Sa is a permissible subscript arrangement

Sb $>$ Sa is not

Block 1

origin	A(1)		COMMON/1/A(4), C
	A(2)	B(1)	DIMENSION B(5)
	A(3)	B(2)	EQUIVALENCE (A(3), B(2))
	A(4)	B(3)	
	C	B(4)	
		B(5)	

EQUIVALENCE statements must appear before any statement function definitions, executable, DATA, or NAMELIST statements in the program unit.

8.4 EXTERNAL

The EXTERNAL statement defines variable names to be external procedure names. This feature permits external procedure names to be passed as arguments to another external procedure; the names must be defined in an EXTERNAL statement in the program unit in which it is used.

```
EXTERNAL v1, v2, . . . , vn
```

v_i are declared to be external procedure names.

Example:

```
EXTERNAL NAME1, NAME2, NAME3
      :
      :
      CALL SUB(A, B, NAME2)
      SUBROUTINE SUB(X, Y, IFUNC)
```

The user is also allowed to define an Intrinsic function name in an EXTERNAL declaration. This re-definition of an intrinsic function name causes the processor to consider any subsequent reference as an external function reference; the user must supply the procedure.

EXTERNAL statements must appear before any statement function definitions, executable, DATA, or NAMELIST statements.

8.5 TYPE

The TYPE declaration provides the processor with information concerning the structure of variable and function identifiers. Six variable types may be declared by the statement:

```
t v1, v2, . . . , vn
```

t may be INTEGER, REAL, DOUBLE PRECISION (or DOUBLE), COMPLEX, LOGICAL or ECS optionally preceded by the characters TYPE. Each v is a variable name, array name, function name, or an array name with its dimensions which assumes the type indicated by t.

A TYPE statement may be used to override or confirm implicit typing; it must be used to declare entities to be double precision, complex, logical or ECS; it may also supply dimension information.

Example:

```
INTEGER ACBS, AFDS, ITRC
TYPE COMPLEX CC, F
```

The TYPE declaration is non-executable and must precede any statement function definitions, executable, DATA, or NAMELIST statements in a given program unit. Any variable defined by a TYPE statement may not be re-defined in another TYPE statement; when such a condition does exist, a diagnostic occurs and the processor assumes the type as declared when first encountered.

8.6 DATA

The DATA, data initialization, statement is used to define initial values of variable or array elements not located in blank COMMON.

```
DATA vlist1/dlist1/, vlist2/dlist2/, ..., vlistn/dlistn/
```

vlist List of array names, array elements, variable names, or implied DO loops separated by commas. Only one array name can be used within an implied DO nest. The implied DO control parameters must be constants. Subscripts used to identify an array element must be integer constants.

dlist One or more of the following forms separated by commas:

```
constant
(constant list)
rf*constant
rf*(constant list)
rf(constant list)
```

constant list List of constants separated by commas

rf Integer constant. The constant or constant list is repeated the number of times indicated by rf.

Example:

```
DATA X, Y, Z/32.5, -7.4, 3./, S, T/1.5E3, .TRUE./
```

Entries in the list are separated by commas. Hollerith constants may also be included.

Example:

```
DIMENSION AMASS(10,10,10), A(10), B(5)
DATA (AMASS(6,K,3),K=1,10)/4*(-2.,5.139),6.9,10./
DATA (A(I),I=5,7)/2*(4.1),5.0/
DATA B/5*0.0/
```

ARRAY AMASS:

```
AMASS(6,1,3) = -2.
AMASS(6,2,3) = 5.139
AMASS(6,3,3) = -2.
AMASS(6,4,3) = 5.139
AMASS(6,5,3) = -2.
AMASS(6,6,3) = 5.139
AMASS(6,7,3) = -2.
AMASS(6,8,3) = 5.139
AMASS(6,9,3) = 6.9
AMASS(6,10,3) = 10.
```

ARRAY A:

```
A(5) = 4.1
A(6) = 4.1
A(7) = 5.0
```

ARRAY B:

```
B(1) = 0.0
B(2) = 0.0
B(3) = 0.0
B(4) = 0.0
B(5) = 0.0
```

A one-to-one correspondence is necessary between the list items and the constants which establish their initial value.

Example:

```
DIMENSION K(10), A(2)
DATA A/2.0/
```

The value 2.0 is stored in A(1), however, in A(2), there is no definite value.

When the number of list elements exceeds the range of the implied DO, the excessive list elements are not stored.

Example:

```
DIMENSION B(10)
DATA(B(J),J=1,5)/4*1.23,6*1.24/
```

The excessive values 5*1.24 are discarded.

If a list item is an array name with no control subscripts or parameters, the constant list defines the values in the array to the maximum dimensional length or until the constant list is exhausted.

An initially defined variable or array element may not be in blank common.

An alternate form of the data initialization statement has the form:

```
DATA (r1=d1), (r2=d2), . . . , (rn=dn)
```

Each r is an array element name that may have from one to three control subscripts or a list of names of variables and array elements (each of which may be a single integer variable) and from one to three integer constant control parameters.

Each d is a list of constants and optionally signed constants, any of which may be preceded by j*. The constants may be grouped by parentheses and optionally preceded by j*; j is an integer constant.

Example:

```
DIMENSION D3(4), POQ(5, 5)
DATA (D3 = 5., 6., 7., 8.), (((POQ(I, J), I=1, 5), J=1, 5)=25*0)
```

which initializes:

D3(1) = 5.

D3(2) = 6.

D3(3) = 7.

D3(4) = 8.

and sets the entire POQ array to zero.

DATA statements must appear after all specification statements in a program unit.

The type of the DATA value is determined by the form of the constant, not the type of the list variables.

PROGRAM FUNCTION, SUBROUTINE, BLOCK DATA, AND LIBRARY ROUTINES

9

A FORTRAN Extended program consists of a main program with or without subprograms. Subprograms are separate programs that are executed only when called and may be defined by the programmer or be preprogrammed and contained in the processor or system library.

9.1 MAIN PROGRAM

The first statement of the main program must be one of the following forms; it may begin anywhere after column 6.

```
PROGRAM s  
PROGRAM s (f1,f2,...,fn)
```

s is a symbolic name of the main program, f_i are 1 to 6 character names of all input/output files required by the main program and its subprograms.

The arguments must satisfy the following conditions within the program and its subprograms at compile time.

File name INPUT must appear if the READ f,k statement is included.

File name OUTPUT must appear if any PRINT statement is included: also needed for printing of execution time diagnostics.

File name PUNCH must appear if any PUNCH statement is included.

File name TAPE i (i is an integer constant 1-99) must appear if any input/output statement involving unit i appears in the program. If i is a variable, there must be a file name TAPE i for each value i may assume.

Files may be equivalenced at compile time. For example,

```
(INPUT, OUTPUT, TAPE1 = INPUT, TAPE2=OUTPUT)
```

All input normally provided by TAPE1 is to be extracted from INPUT and all listable output normally recorded on TAPE2 is to be transmitted to the OUTPUT file.

In the list of parameters, equivalenced file names must follow those to which they are made equivalent.

File buffers may be assigned a non-standard size at compile time; (OUTPUT=400, TAPE4=OUTPUT). If buffer size is not indicated, 1025 is assumed. For 7600, if no parameter is given, an SCM buffer size of 2568 is assumed and an 8192 word LCM buffer is assigned. The LCM buffer size is not variable under SCOPE 1.1. If the buffer is explicitly assigned a length, the assignment must appear with the first reference to the file on the program card. The length may be specified in octal (with the trailing B) or in decimal.

If the PROGRAM card is omitted, the FORTRAN processor assumes a program name of START. when it encounters a statement that is not a comment card. Input/output buffers and files for the program are equated to the standard SCOPE system files INPUT and OUTPUT.

The equivalencing of files causes associated buffer and file names to be equivalenced.

Example:

```

PROGRAM HELLO (TAPE1, TAPE2=TAPE1)
  :
  :
N=1
WRITE (N) A
  :
  :
END

PROGRAM HELLO (TAPE1, TAPE2=TAPE1)
  :
  :
N=2
WRITE(N) A
  :
  :
END

```

The file name resulting from both programs is TAPE 1.

The file names declared on the program card are only names that may result from I/O references within the program. If no parameters appear on the control card which calls a program into execution, the non-equivalenced declared names will be taken as the SCOPE file names to be accessed. If parameters do appear on the control card which calls a program into execution, each parameter will be the SCOPE file name to be accessed by the corresponding program declared name. In a program headed by the program card

PROGRAM name (f_1, f_2, \dots, f_n)

which is called into execution by the control card

LGO(p_1, p_2, \dots, p_n)

(where each p_i may be null), a reference to the declared name f_i will access the SCOPE file f_i if p_i is null; otherwise, the SCOPE file p_i will be accessed. Only non-equivalenced program declared names may have a corresponding p_i specified on the control card which calls the program into execution:

Examples:

1. If a program is headed by the card

```
PROGRAM PROG (TAPE1,OUTPUT,TAPE2=OUTPUT)
```

and is called into execution with

```
LGO.
```

every reference to unit 1 within PROG will access the SCOPE file TAPE1, every print statement and every reference to unit 2 will access the SCOPE file OUTPUT.

If PROG is called into execution with the control card

```
LGO(INPUT,LOAD)
```

every reference to unit 1 within PROG will access the SCOPE file INPUT; every print statement or reference to unit 2 will access the SCOPE file LOAD.

Calling PROG into execution with the control card

```
LGO(,LOAD)
```

will act the same as using

```
LGO.
```

in the former case, there is an illegal attempt to change an equivalenced declared name (the attempt is ignored).

2. If INPUT is to be changed to TAPE10, and the program is headed by the card:

```
PROGRAM ELKE (OUTPUT,TAPE61=OUTPUT,INPUT)
```

this is achieved by

```
LGO(,TAPE10)
```

9.2 SUBROUTINE SUBPROGRAMS

A subroutine is an external computational procedure defined by FORTRAN statements which is identified by a SUBROUTINE statement and may or may not return values to the calling program. The statement may have any one of the following forms:

```
SUBROUTINE s (a1,a2, . . . ,an) or SUBROUTINE s  
SUBROUTINE s (a1,a2, . . . ,an), RETURNS (b1,b2, . . . ,bm)  
or  
SUBROUTINE s, RETURNS (b1,b2, . . . ,bm)
```

s is the symbolic name of the subroutine, a_i are the dummy arguments (these may be variable names, array names or external procedure names), and b_i are variable names containing statement labels which indicate alternate exits from the subroutine. SUBROUTINES and FUNCTIONS are restricted to a maximum of 63 dummy arguments.

Example:

<u>Calling Program</u>	<u>Subprogram</u>
⋮	SUBROUTINE PGM1 (X, Y, Z),
⋮	RETURNS (M, N)
CALL PGM1 (A, B, C),	U=V*W+T**2
RETURNS (5, 10)	X=Y*Z
⋮	20 IF (U+X) 25, 30, 35
⋮	25 RETURN M
5 B=SQRT(A*C)	30 RETURN N
⋮	35 Z=Z+(X*Y)
⋮	RETURN
10 CALL PGM2 (D, E)	END
⋮	

The above example illustrates the different types of returns which may be made from a subroutine subprogram. If the RETURNS list is omitted from the CALL statement in the calling program, a return of the form RETURN a may not be made. However, the converse is permitted; a normal return via the RETURN statement may be made to the calling program if the RETURNS list is specified in the CALL statement.

Subroutine subprograms are constructed with the following restrictions:

Symbolic name of the subroutine must not appear in any other statement in this subprogram.

Symbolic names of the dummy arguments may not appear in an EQUIVALENCE, COMMON, or DATA statement in the subprogram.

Subroutine subprograms do not require a RETURN statement if the procedure is completed upon executing the END statement. When the end line is encountered, a return is implied.

Subroutine subprograms may contain any statements except BLOCK DATA, FUNCTION, or another SUBROUTINE statement.

Execution of a subroutine begins with the first executable statement of the subprogram. Continuation is sequential unless a GO TO, IF, RETURN, STOP or terminal statement of a DO is encountered, in which case execution proceeds as indicated.

A reference to a subroutine is made by a CALL statement. The actual arguments, which constitute the argument list, must agree in order, number, and type with the corresponding dummy arguments in the defining program; otherwise the results are unpredictable. The use of Hollerith constants and octal constants, which are interpreted as though they were typeless (page 1-7), as actual arguments is an exception to the rule requiring agreement of type. An actual argument in a subroutine reference may be one of the following:

- Constant
- Variable name
- Array element name
- Array name
- Name of an external procedure
- ECS variable name
- ECS array name
- Any other expression

Several restrictions and rules govern the correspondence of actual arguments in the calling program to dummy arguments in the subprogram:

If an argument in the calling program is an external function or subroutine name, the corresponding dummy argument must be used in the same manner.

An argument in the calling program must be a variable name, an array element name, or an array name if it corresponds to a dummy argument which is defined or redefined in the subprogram.

The association of arguments in the calling program is made by name to dummy arguments appearing in executable statements, function definition statements, or those used as adjustable dimensions in the subprogram. However, if an argument takes the form of an expression (any other expression), the association is by value rather than by name.

An argument which is an array element name containing variables in the subscript expression may be replaced by the same argument with a constant subscript with an equivalent value.

If a subroutine reference causes a dummy argument in the referenced subroutine to become associated with another dummy argument in the same subroutine or with an entity in common, a definition of either entity within the subroutine is prohibited.

Example:

Assume X = 3 and Y = 2

```
1) CALL SUBA (X, X)      SUBROUTINE SUBA (A, B)
                           :
                           :
                           A = Y
                           Z = B

2) COMMON X              SUBROUTINE SUBB (B)
   CALL SUBB (X)         COMMON A
                           :
                           :
   END                   A = Y
                           Z = B
                           END
```

In the above examples, the first two statements in the subroutine set X = Y then Z = X resulting with X = 2 and Z = 2. However, if the statements are reversed the results obtained will be different; Z = X then X = Y, the numeric values resulting are Z = 3 and X = 2.

ENTRY STATEMENT

This statement provides alternate entry points to a function or subroutine subprogram.

ENTRY name

Name is an alphanumeric identifier which may appear within the subprogram only in the ENTRY statement. Each entry identifier must appear in a separate ENTRY statement. The formal parameters, if any, appearing with the FUNCTION or SUBROUTINE statement do not appear with the ENTRY statement. They are assumed to be the same as those of the FUNCTION or SUBROUTINE in which the ENTRY statement is located. ENTRY may appear anywhere within the subprogram except within a DO; ENTRY statements cannot be labeled. The first executable statement following ENTRY becomes an alternate entry point to 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 imbedded. The name may appear in an EXTERNAL statement and, if a function entry name, in a TYPE statement.

The ENTRY name type must agree with the function name type. The name may not be given a type explicitly in the defining program; it assumes the same type as the name in the FUNCTION statement.

Examples:

```
FUNCTION JOE(X, Y)
10 JOE=X+Y
RETURN
ENTRY JAM
IF (X.GT.Y) 10, 20
20 JOE=X-Y
RETURN
END
```

This could be called from the main program as follows:

```
      :
      :
      Z = A+B-JOE(3. *P, Q-1)
      :
      :
      R = S+JAM(Q, 2. *P)
```

LIBRARY SUBROUTINES Library subroutine subprograms may be referred to by any program with a CALL statement. i must be an integer variable or constant, j is an integer variable.

CALL SLITE (i) Turn on sense light i. If i = 0, turn all sense lights off. i is 0 to 6; if i > 6 or < 0, an informative diagnostic is given and all sense lights remain unchanged.

CALL SLITET (i, j) If sense light i is on, j = 1, if sense light i is off, j = 2; then turn sense light i off. i is 1 to 6. If i > 6 or < 0, an informative diagnostic is given and all sense lights remain unchanged and j=2.

CALL SSWTCH (i, j) If sense switch i is on (down), j = 1, if sense switch i is off (up), j = 2, i is 1 to 6. If i > 6 or < 0, an informative diagnostic is given and all sense switches remain unchanged and j = 2.

CALL EXIT Terminate program execution and return control to the operating system.

CALL REMARK (H) Place a message, not to exceed 80 characters, in the dayfile. H is a Hollerith specification or an array containing character data. When an array name is used the message should be terminated by at least 12 bits of binary zeros in the right end of a 60 bit word.

CALL DISPLA(H, k) Same as REMARK except the value of the second parameter is appended to the message. k may be either a real or integer expression. The entire message including space for the value of k should not exceed 80 characters. (If H is 40 characters or less this maximum will not be exceeded.)

CALL RANGET(n) Obtain current generative value of RANF between 0 and 1. n is a symbolic name.

CALL RANSET(n) Initialize generative value of RANF. n is real.

CALL DUMP (a₁,b₁,f₁, . . . , a_n,b_n,f_n)
 CALL PDUMP (a₁,b₁,f₁, . . . , a_n,b_n,f_n)

Dump storage on the OUTPUT file in the indicated format. If PDUMP was called, return control to the calling program; if DUMP was called, terminate program execution and return control to the monitor. a_i and b_i identifiers indicate the first word and the last word of the storage area to be dumped; 1 ≤ n ≤ 20. The dump format indicators are as follows:

- f = 0 or 3 octal dump
- f = 1 real dump
- f = 2 integer dump
- f = 4 octal dump; this implies that a_i and b_i are statement numbers that have been defined by an ASSIGN statement.

CALL STRACE Provides subroutine calling traceback information from the subroutine which calls STRACE back to the main program. Traceback information is written to the file DEBUG. To obtain traceback information interspersed with the source program DEBUG should be equivalenced to OUTPUT in the PROGRAM statement. (Refer to Section 11.13 STRACE.)

9.3 FUNCTION SUBPROGRAMS

STATEMENT FUNCTIONS Statement function definitions must precede the first executable statement of the program or subprogram and must follow any specification statements. The name of a statement function must not appear in an EXTERNAL statement, nor as a variable name or an array name in the same program or subprogram. A statement function applies only to the program or subprogram containing the definition; it is defined by a statement of the form:

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

f is the statement function name, e is any expression. a_i are variable names which are dummy arguments indicating type, number, and order of arguments; they may be the same as variable names of the same type appearing elsewhere in the program unit. n may not exceed 63. f and e must be both logical or both non-logical.

Examples:

1. LOGICAL C, P, EQV
EQV(C, P) = (C .A. P) .O. (.N. C .A. .N. P)
2. COMPLEX D, F
D(A, B) = (3.2, 0.9)*EXP(A)*SIN(B) + (2.0, 1.)*EXP(Y)*COS(B)
3. GROS(R, HRS, OTHERS) = R*HRS + R*.5*OTHERS

INTRINSIC FUNCTION

The symbolic names of the intrinsic functions (built-in functions) have special meaning and type as described in Appendix D. An intrinsic function is referenced when the name of the function appears in an arithmetic or logical expression. The actual arguments, which constitute the argument list, must agree in type, number, and order with the specification in Appendix D and may be any expression of the specified type.

Examples:

- 1) DATA(I)=DATA(I+1)*((FLOAT(MAX)/K(I))/DATA(I))
- 2) IB(J)=IFIX(B(J))

The intrinsic functions SIGN, ISIGN, and DSIGN are defined when the value of the second argument is zero, such that the sign of the second argument is taken as positive (negative) for +0(-0). However, the functions AMOD and MOD are not defined when the second argument is zero; division by zero renders the results undefined.

EXTERNAL FUNCTION

An external function is defined externally to the program or subprogram that references it. An external procedure defined by FORTRAN statements headed by a FUNCTION statement is called a function subprogram.

t FUNCTION $f(a_1, a_2, \dots, a_n)$ or FUNCTION $f(a_1, a_2, \dots, a_n)$

t is INTEGER, REAL, DOUBLE, DOUBLE PRECISION, COMPLEX, LOGICAL, or it is omitted.

f is the symbolic name of the function. If t is omitted the type of the function is derived from f according to the type rules of implicit definition.

a_i are the dummy arguments; each may be a variable name, an array name, or an external procedure name. $1 \leq i \leq 63$.

The function name f must appear as a variable in the defining subprogram. During every execution of the subprogram, the variable must be defined, and once defined, it may be referenced or redefined. The value of the variable when a RETURN statement is executed is the value of the function. The function name f must not appear in any non-executable statement other than the FUNCTION statement in the function subprogram.

The dummy argument names may not appear in an EQUIVALENCE, COMMON, or DATA statement in the function subprogram. The function subprogram may define or redefine one or more of its arguments so as to effectively return results in addition to the value of the function.

A function subprogram may contain any statement except BLOCK DATA, SUBROUTINE, another FUNCTION STATEMENT, or any statement that directly or indirectly references the function being defined.

When the END line is reached, a return is implied.

Example:

```
FUNCTION GRATER(A, B)
  IF(A. GT. B)1, 2
1 GRATER=A-B
  RETURN
2 GRATER=A+B
END
```

EXTERNAL FUNCTION REFERENCE

An external function is referenced when it is used as an operand in an arithmetic or logical expression. The actual arguments must agree in order, number, and type with the corresponding dummy arguments in the defining program.

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

f is a symbolic name of the function, a_i are the actual arguments. An actual argument name in an external function reference may be one of the following:

- Variable
- Array element
- Array name
- External procedure reference
- Constant
- ECS variable
- ECS array
- ECS array element
- Any other expression

The rules governing the association of arguments in the function call to dummy arguments in the function are the same as those enumerated for subroutine subprograms.

Examples:

- 1) $W(I, J) = FA + FB - GRATER(C - D, 3. * AX / BX)$
- 2)

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

The reference to the function PHI in example 2 may be:

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

The replacement statement in the function PHI will produce the same result as if it had been written $PHI = SIN(Q(K))$.

BASIC EXTERNAL FUNCTIONS

The basic external functions are described in Appendix D. These functions are referred to in the section, External Function Reference. Arguments may not be used for which a result is not mathematically defined and they may not be of a type other than that specified.

9.4 BLOCK DATA SUBPROGRAM

Initialization of data to be stored in labeled common may be accomplished by the specification of a BLOCK DATA subprogram which begins with a statement of the form:

```
BLOCK DATA
      or
BLOCK DATA d
```

d is the symbolic name of the BLOCK DATA subprogram. This parameter must be specified if the subprogram is to be included in a SEGMENT (6000 series only).

The BLOCK DATA subprogram contains only specification and DATA statements; executable statements are prohibited. Only the DATA, COMMON, DIMENSION, EQUIVALENCE, and TYPE statements associated with the data being defined are accepted; data may not be entered into an unlabeled (blank) common block. If an entry for a common block is given an initial value in such a subprogram, a complete set of specification statements for the entire block must be included, even though some of the elements of the block do not appear in DATA statements.

Example:

```
BLOCK DATA
COMMON/MAX/DATA(3),AA, BB/MIN/A, B, C, LAX
REAL LAX
INTEGER BB
COMPLEX A
DOUBLE PRECISION C
DATA LAX/145.12/, (DATA(I), I=1, 3)/1.1, 2*9.3/, BB/1256/, A, B, C/
      (2.0, 1.0), 13.6, 172.5432D06/
END
```

Initial values may be entered into more than one block in a single subprogram.

Programs that exceed available memory may be divided into independent parts which may be called and executed as needed. Such programs can be divided into segments or overlays. However, the segment feature is not supported by 7600 LOADER and is not available to 7600 FORTRAN Extended users under SCOPE version 1.

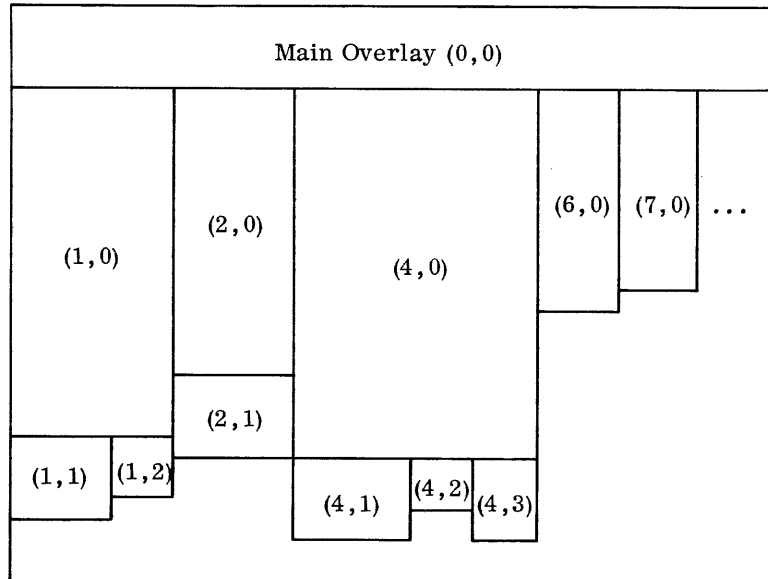
Segments are groups of subprograms that are loaded in relocatable form when requested, giving the user explicit control over established inter-program links. An overlay is a program combined with its subprograms which is converted to absolute form and written on mass storage prior to execution. During execution, overlays are called into memory and executed as requested. OVERLAY and SEGMENT loader control cards must start in column 7 or later and appear between subprograms or before the first program. Compiler processing places them in the desired position on the binary output file.

10.1 OVERLAYS

Each overlay is numbered with an ordered pair of numbers (I, J), each in the range 0-77₈. I denotes the primary level and J the secondary level. An overlay with a non-zero secondary level is called a secondary overlay. It is associated with and subordinate to the primary overlay which has the same primary level and a zero secondary level. The initial or main overlay which always remains in memory has levels (0,0). The significance of this distinction appears in the order in which overlays are loaded.

Overlay level numbers, such as (0,1), (0,2), (0,3)... are illegal. Primary overlays all have their origin at the same point immediately following the main overlay (0,0). The origin of secondary overlays immediately follows the primary overlay. For any given program, all overlay identifiers must be unique. The loading of any primary overlay destroys any previous primary overlay. For this reason, no primary overlay may load other primary overlays. Secondary overlays may be loaded only by the associated primary overlay or main overlay. Thus two levels of overlays are available to the programmer.

Example:



Overlays (1,1) and (1,2) are secondary to overlay (1,0)

Overlay (2,1) is secondary to overlay (2,0)

Overlay (2,1) may not be called from (1,0) or (1,1) or (1,2) but only from (2,0) or (0,0)

Overlays (1,0), (2,0), (4,0)... may be called only from the main overlay (0,0)

OVERLAY CONTROL CARDS

OVERLAY (lfn, l1, l2, Cnnnnnn)

- lfn File name on which overlay is to be written; first overlay card must have a named lfn. Subsequent cards may omit it, indicating that the overlays are related and are to be written in the same lfn. A different lfn on subsequent cards results in generation of overlays to the new lfn.
- l1 Primary level number in octal.
- l2 Secondary level number in octal. l1, l2 for the first overlay card must be 0,0.
- Cnnnnnn Optional parameter consisting of letter C and six-digit octal number. If this parameter is present, the overlay is loaded nnnnnn words from the start of blank common. This provides a method of changing the size of blank common at execution time. Cnnnnnn cannot be included on the overlay 0,0 loader directive. If this parameter is omitted, the overlay is loaded in the normal manner.

Overlays are called by: `CALL OVERLAY (fn, I, J, p, l)`

OVERLAY is a FORTRAN execution time subroutine which translates the FORTRAN call into a call to the loader.

- fn Variable name of a location containing the name of the file (left justified display code) that contains the overlay.
- I Primary level of overlay; integer value 1-63
- J Secondary level of overlay; integer value 0-63
- p Recall parameter. Checks whether overlay specified was last one loaded. If p equals 6HRECALL, the overlay is not reloaded if it is already in memory.
- l Load parameter. Used to determine which value of the fn will be used. l may be any value. If l is present and non-zero, the overlay designated by fn will be loaded from the system library; otherwise, it will be loaded from the file designated by fn.

Prior to execution of this call which causes loading and execution of the overlay, the overlay must have been made absolute and written on file fn. When an END statement in the main program of an overlay is encountered, control returns to the statement following the CALL OVERLAY which initialized execution of the overlay in question.

Numbers in the OVERLAY card are octal; thus to call OVERLAY (SAM, 1, 11) the FORTRAN statement CALL OVERLAY(3HSAM, 1, 9, 0) or CALL OVERLAY(3HSAM, 1, 11B, 0) must be used.

10.2 SEGMENTS†

A segment is a group of subprograms (possibly one) which are loaded together when specified by the programmer. Segments are loaded at levels from 0-77₈. Level zero is reserved for the initial or main segment. Level zero, which must contain a PROGRAM, remains in memory during execution.

The following definitions apply to segments.

Entry point. A named location within a subprogram that can be referenced by another program – created by the SUBROUTINE, FUNCTION and ENTRY statements.

External reference. A reference within a program or subprogram to the entry point of some other subprogram – created by explicit CALL statements, function references, I/O statements, etc.

Link. The connection established between an external reference and an entry point when the programs are loaded into memory.

Unsatisfied external. An external reference for which no matching entry point can be found, and therefore no link established.

† The segment feature is not available to FORTRAN Extended users under 7000 SCOPE Version 1 or KRONOS 2.0.

When the segment is loaded, external references will be linked to entry points in previously loaded segments (those at a lower level). Similarly, entry points in the segment are linked to unsatisfied external references in previously loaded segments. Unsatisfied external references in the segment remain unsatisfied; subsequent segment loading may include entry points to satisfy the external references. Unsatisfied external references will be satisfied, if possible, from the system library.

If a segment is to be loaded at a requested level which is less than or equal to the level of the last loaded segment, all segments at levels down to and including the requested level will be removed/delinked. Delinking a segment at a given level requires that the linkage of external references in lower levels to entry points in the delinked segment be destroyed so that the external references are unsatisfied once again.

Once the delinking is complete, the segment is loaded. Only one occurrence of a given subprogram or entry point is necessary since all levels may eventually link to the subprogram. However, a user may force loading of a subprogram by explicitly naming it in another segment at a higher level. Thereafter, all external references in higher levels are linked to the new version. In this manner, a subprogram and/or entry point can effectively replace an identical one already loaded at a lower level. However, once a linkage is established, it is not destroyed unless the segment containing the entry point is removed.

Example:

The SIN routine is loaded in a segment at level 1. The user wishes to try an experimental version of SIN. He loads a segment containing the new SIN at level 2. Segments loaded at level 3 or higher will now be linked to SIN at level 2 until a new level 2 or a new SIN is loaded.

Common blocks may be loaded with any segment. Labeled common may not be cross-referenced in segments. Maximum blank common length is established in the first segment which declares blank common.

10.2.1 SEGMENT CONTROL CARDS †

SECTIONS

This card defines a section within a segment. Segments are loaded by user calls during execution or by monitor (MTR) during initial load.

SECTION (sname, pn₁, pn₂, . . . , pn_n)

sname Name of section (7 alphanumeric characters maximum).

pn_i Names of subprograms in the section. If more than one card is necessary to define a section, additional cards with the same sname may follow consecutively.

† The segment feature is not available to FORTRAN Extended users under 7000 SCOPE Version 1 or KRONOS 2.0.

All subprograms within a section are loaded whenever the named section is loaded. All section cards must appear prior to the SEGMENT and SEGZERO cards which refer to the named sections.

SEGMENTS †

All programs using segments must contain a SEGZERO card prior to any of the binary text.

SEGZERO (sn, pn₁, pn₂, . . . , pn_n)

sn Segment name

pn₁ Names of subprograms or section names which make up main or zero level segment. Defining other segments in a similar manner reduces the list of subprograms in the loader call.

SEGMENT (sn, pn₁, pn₂, . . . , pn_n)

The parameters are defined as in SEGZERO. In a segment, all programs must reside on the same file. A segment defined in the user's program need not be defined by a SEGMENT card; however, a SEGZERO card is always required.

Segments may be loaded by the statement:

CALL SEGMENT (fn, e, a, lib, m)

fn Variable name of location which contains the file name (left justified display code) from which the segment load takes place.

e Level of the segment load.

a Variable name of array containing a list of SEGMENTS, SECTIONS and/or SUBPROGRAMS to be loaded with this call. In this list, the name must be in left justified display code, and the list must be terminated by a zero entry. An initial list entry of zero signals a segment load of all subprograms remaining on the file fn.

lib If zero or blank, unsatisfied externals are to be satisfied, if possible, from the system library.

m If zero or blank, a map of the segment load is not produced. lib and m need not be specified.

Once the named subprograms are loaded control returns to the statement following the CALL SEGMENT. The programmer is free to call on the loaded subprograms as desired.

†The segment feature is not available to FORTRAN Extended users under 7000 SCOPE Version 1 or KRONOS 2.0.

The debugging mode of compilation, along with the source cross-reference map selection, is provided specifically to aid in the development or conversion of programs. In the debugging mode of compilation, a programmer can establish a record of selected operations as they are performed in the execution of his program. This mode facilitates debugging from a source listing, and perhaps a source cross-reference map should core dumps be required; it makes their interpretation much easier.

Features provided with the debugging mode of compilation:

Array bounds checking

Program flow tracing

Call and return tracing

Function call and value returned tracing

Stores checking

Assigned GO TO checking

Partial execution of routines containing fatal errors

The debugging mode is selected by the option D on the FTN control card (Appendix C). In this mode, debugging selection cards are recognized. If this mode is not specified, debugging selection cards are treated as comments.

In the debugging mode, a program is compiled so that specified checks can be performed during execution; however, execution will stop when a fatal error is detected.

When a program is compiled in debug mode, 12000₈ words will be required beyond the minimum field length for non-debug mode compilation. To execute, 2500₈ words beyond the minimum will be required.

11.1 FORMAT

Debugging statements are punched in columns 7-72, as in the normal FORTRAN statement. In addition:

Columns 1 and 2 of each statement must contain the characters C\$

A continuation line must be flagged by a character in column 6 (any FORTRAN character other than blank or zero). Columns 3-5 must be blank.

P E	NO.					N T.	Ø = ALPHA O																											
	1	2	3	4	5		6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29				
	C	\$					(s	t	a	t	i	e	m	e	n	t)																
	C	\$				*	(c	o	n	t	i	n	u	a	t	i	o	n															

† If required

The restriction on the number of debug continuation lines is the same as for FORTRAN continuation lines. When FORTRAN Extended is not in debug mode or when the program is used with another FORTRAN compiler, the debug cards will be treated as comment cards. Since even working programs sometimes exhibit new bugs, it could be advantageous to retain the debugging statements in a program once checkout is complete.

In the following pages, excerpts from an actual printout of a working program are used in conjunction with typewritten examples to illustrate the debugging messages. A sample working program is reproduced in full at the end of the chapter. (The reference map level specified on the FTN card was R-1.)

11.2 ARRAYS STATEMENT

The ARRAYS statement initiates subscript bounds checking on specified arrays. Warning messages appear on the output if the address calculated by the array indexing function is not within the storage allocated for the array.

C\$ ARRAYS(a₁, a₂... a_n)

C\$ ARRAYS

Printout: C\$ ARRAYS(A1)

(a₁... a_n) are the names of the arrays for which subscript bounds are to be checked. If array names are not given, all arrays in the program unit are checked.

The C\$ ARRAYS statement does not provide checking of individual subscripts, only checking of the address computed from all the subscripts.

When ARRAYS statement is used, a bounds check is made each time an element of an array is referenced. Bounds checking is not performed for array references in an input/output list. If the element is not within the overall bounds of the array, a message is printed with the job output, as shown in the following example. After printing a message for an out of bounds array reference, the reference is allowed to occur.

/DEBUG/	SAMPLE	AT LINE	11-	THE SUBSCRIPT VALUE OF	6	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND OF	5
/DEBUG/		AT LINE	11-	THE SUBSCRIPT VALUE OF	0	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND OF	5
/DEBUG/		AT LINE	13-	THE SUBSCRIPT VALUE OF	5	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND OF	5
/DEBUG/		AT LINE	14-	THE SUBSCRIPT VALUE OF	0	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND OF	5
Identifies a debugging message	Program unit name containing subscript reference	Line number of reference		Value of subscript in reference		Name of array being referenced	Actual dimension limits of array	

11.3 CALLS STATEMENT

This statement traces calls to and returns from specified subroutines.

C\$ CALLS(a₁, . . . , a_n)

C\$ CALLS

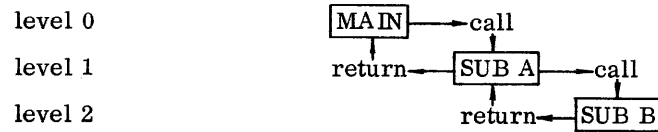
Printout: C\$ CALLS(SUB1, SLITE)

The subroutine names for which call tracing is to be performed are indicated by (a₁ . . . a_n). If this parameter is not specified, all subroutine calls are traced. Nonstandard returns are also traced.

The message produced for each call and return is printed with the job output as follows:

/DEBUG/	SAMPLE	AT LINE	23-	ROUTINE SUB1	CALLED AT LEVEL	0
/DEBUG/		AT LINE	24-	ROUTINE SUB1	RETURNS TO LEVEL	0
/DEBUG/		AT LINE	25-	ROUTINE SLITE	CALLED AT LEVEL	0
/DEBUG/		AT LINE	26-	ROUTINE SLITF	RETURNS TO LEVEL	0
Identifies a debugging message	Program unit name containing reference	Line number containing call or return		Name of subroutine called or returned	Indicates call or return status and level number	

A main program is at level 0; a subroutine or function called by the main program is at level 1; another subprogram called by a subprogram is at level 2, etc. Calls are shown in order of ascending level number, returns in order of descending level number.



For example, subroutine SUB A is called at level 1 and a return is made to level 0. SUB B is called at level 2 and a return is made to level 1.

**11.4
FUNCS
STATEMENT**

Function tracing is similar to call tracing except that functions return a value that often is of concern to the programmer.

C\$ FUNCS(a₁, . . . , a_n)

C\$ FUNCS

Printout: C\$ FUNCS(FUN1, IABS, EXP)

The function names for which function tracing is to be performed are indicated by (a₁, . . . , a_n). If no names are listed, all functions are traced. Functions used in array subscripts in input/output lists and statement functions are not traced. A message is issued for each use of a function; it is printed with the job output as shown below.

/DEBUG/	SAMPLE	AT LINE	33- REAL	FUNCTION FUN1	CALLLED AT LEVEL 0		
/DEBUG/		AT LINE	33- REAL	FUNCTION FUN1	RETURNS A VALUE OF	7743.000000	AT LEVEL 0
/DEBUG/		AT LINE	35- INTEGER	FUNCTION IABS	CALLLED AT LEVEL 0		
/DEBUG/		AT LINE	35- INTEGER	FUNCTION IABS	RETURNS A VALUE OF	8242	AT LEVEL 0
/DEBUG/		AT LINE	37- REAL	FUNCTION EXP	CALLLED AT LEVEL 0		
/DEBUG/		AT LINE	37- REAL	FUNCTION EXP	RETURNS A VALUE OF	23414063123	AT LEVEL 0
Identifies a debug- ging message	Program unit using functions	Line number containing the function usage	Function type	Function name	Level number of using program unit including call or return status	Value returned by function	Level to which value is being returned

**11.5
STORES
STATEMENT**

The STORES statement is used to record changes in value of specified simple variables resulting from arithmetic assignment statements. Variables altered as a result of use in an input list or a subroutine (function) parameter list are not detected. Stores checking is not performed on the control variable of a DO loop; stores checking is not performed when a variable is changed as a result of a store into an equivalenced variable.

Stores checking cannot be specified for individual elements of an array.

C\$ STORES(c₁,c₂,...,c_n)

Printout:

```
C$ STORES(A1, AGAIN, I, A2.EQ.5.0, IAGAIN.LE.10)
C$ STORES(C.EQ.(1., 1.), L.VALID., D.NE.10.004)
C$ STORES(G.RANGE., TR.EQ..FALSE.)
```

(c₁,...c_n) can be variable names or relational expressions in the form:

variable name .relational operator. constant

or expressions with checking operators in the form:

variable name .checking operator.

The checking operators are:

- RANGE prints when the value is out of range
- INDEF prints when the value is indefinite
- VALID prints for either out of range or indefinite

If variable names are used, a message is issued each time a new value is stored in a variable or array element. If the relational or checking expression is used, a message is issued only when the stored value satisfies the relation. The two components of the relational expression should be of the same type, because no type conversion takes place before evaluation of the relational expression. For example, the integer value 5 is not considered equal to the real value 5.0. The message will contain:

```
/DEBUG/ SAMPLE AT LINE 48- THE NEW VALUE OF THE VARIABLE A1 IS 1.000000000
/DFBUG/ AT LINE 48- THE NEW VALUE OF THE VARIABLE A1 IS 2.000000000
/DEBUG/ AT LINE 48- THE NEW VALUE OF THE VARIABLE A1 IS 3.000000000
/DFBUG/ AT LINE 48- THE NEW VALUE OF THE VARIABLE A1 IS 4.000000000
/DEBUG/ AT LINE 48- THE NEW VALUE OF THE VARIABLE A1 IS 5.000000000
/DFBUG/ AT LINE 51- THE NEW VALUE OF THE VARIABLE AGAIN IS 3.141590000
/DEBUG/ AT LINE 53- THE NEW VALUE OF THE VARIABLE A2 IS 5.000000000
/DFBUG/ AT LINE 54- THE NEW VALUE OF THE VARIABLE IAGAIN IS 10
/DEBUG/ AT LINE 54- THE NEW VALUE OF THE VARIABLE IAGAIN IS 9
/DFBUG/ AT LINE 54- THE NEW VALUE OF THE VARIABLE IAGAIN IS 8
/DEBUG/ AT LINE 54- THE NEW VALUE OF THE VARIABLE IAGAIN IS 7
/DFBUG/ AT LINE 54- THE NEW VALUE OF THE VARIABLE IAGAIN IS 6
```

~~~~~	~~~~~	~~~~~	~~~~~	~~~~~
Identifies a debug message	Name of program unit	Line number of reference	Name of variable, and message	New value of variable

## 11.6 GOTOS STATEMENT

This statement checks the validity of the selected statement labels in an assigned GO TO.

C\$ GOTOS

The statement label assigned to the integer variable is compared with statement labels in the list. A message is printed when the label value is not in the list, but the transfer of control is allowed to occur.

/DEBUG/	SAMPLE	AT LINE	94-	ASSIGNED GOTO INDEX CONTAINS THE ADDRESS 087061.	NO MATCH FOUND IN STATEMENT LABEL ADDRESS LIST
Identifies a debug- ging message	Name of program unit	Line number of assigned go to		Address of assigned go to statement label	Message

## 11.7 TRACE STATEMENT

When the TRACE statement is used, a message is produced for each intra-program transfer of control at a level less than or equal to the level specified by lv.

C\$ TRACE(lv)

C\$ TRACE

Printout: C\$ TRACE(3)

If lv = 0, tracing will occur only outside DO loops; if lv = n, tracing will occur up to and including level n in a DO nest; if no level is specified, zero level is implied. If a DO loop is not satisfied, the transfer back to the start of the loop is not traced. Transfers resulting from nonstandard returns are not traced. (These may be checked using C\$ CALLS.) When tracing is selected and an out-of-bound computed GO TO is executed, the value of the incorrect index is printed before the job is terminated.

Flow tracing will follow these types of program flow control:

Simple GO TO

Computed GO TO

Assigned GO TO

Arithmetic IF

True side of logical IF

The output message will contain the following:

/DEBUG/	SAMPLE	AT LINE	71-	CONTROL WILL BE TRANSFERRED TO STATEMENT 503	AT LINE	73
/DEBUG/		AT LINE	73-	CONTROL WILL BE TRANSFERRED TO STATEMENT 504	AT LINE	75
/DEBUG/		AT LINE	75-	CONTROL WILL BE TRANSFERRED TO STATEMENT 505	AT LINE	77
/DEBUG/		AT LINE	77-	CONTROL WILL BE TRANSFERRED TO STATEMENT 506	AT LINE	78

Identifies a debug- ging message	Program unit name	Line number from which control trans- ferred	Statement number to which control was transferred	Line number of statement to which control was transferred
-------------------------------------------	----------------------	-------------------------------------------------------	------------------------------------------------------	--------------------------------------------------------------------

## 11.8 NOGO STATEMENT

The NOGO statement suppresses partial execution of a compiled routine whenever a fatal compilation error occurs during compilation.

C\$ NOGO

If the NOGO statement is not present and the debugging mode is in effect, the program executes until a fatal error is encountered; at which point, the following message is issued:

FATAL ERROR ENCOUNTERED DURING PROGRAM  
EXECUTION DUE TO COMPILATION ERROR.

Partial execution is not permitted for only three classes of errors:

Errors in the declarative statements

Missing DO loop terminators

Missing FORMAT statement numbers

## 11.9 DECK STRUCTURE

Debugging statements may be interspersed with FORTRAN statements in the source deck of a program unit (main program, subroutine, function). The debugging statements apply to the program unit in which they appear. Inclusion of interspersed debugging statements will change the FORTRAN generated line numbers for a program (figure 11-1).

Debugging statements also may be grouped to form a debugging deck beginning with a C\$ DEBUG card. Debugging decks may be placed in a job in one of the following ways:

As an external debugging deck in a separate file named by the D parameter on the FTN card. When no name is specified by the D parameter, the INPUT file is assumed. (Figure 11-2.)

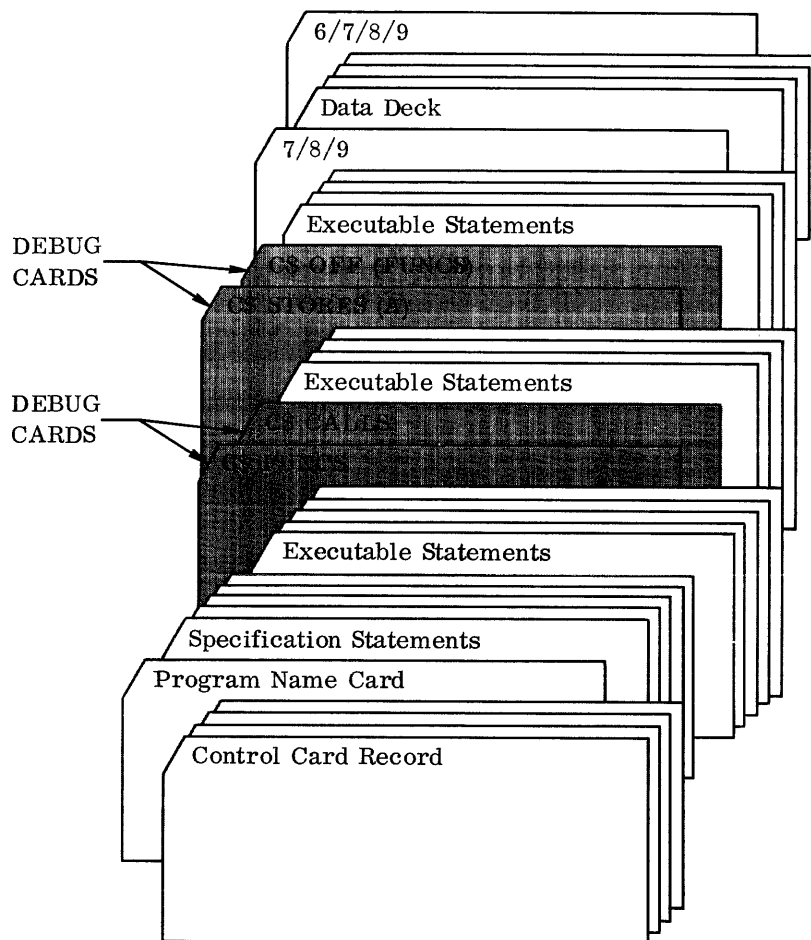
Immediately preceding the first source deck in the compiler input record (External Packet, figure 11-3).

Immediately after a program header card (PROGRAM, SUBROUTINE, or FUNCTION statement) (Internal Packet, figure 11-4).

The range of a debugging statement depends on its position:

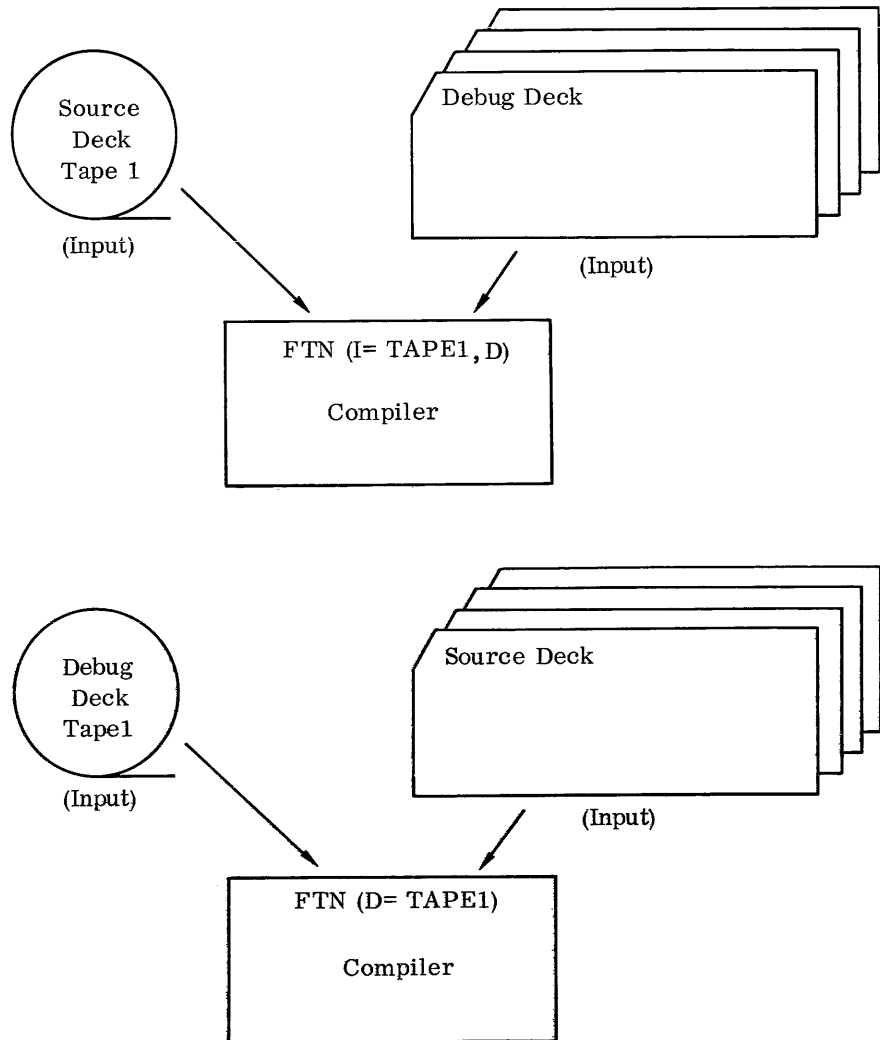
<u>Location</u>	<u>Range</u>
External File	Any or all program units
External Packet	Any or all program units
Internal Packet	Routine containing the packet
Interspersed	Routine containing the specifications

Note: In the following illustrations, it is assumed that a 7/8/9 card terminates each Control Card Record.



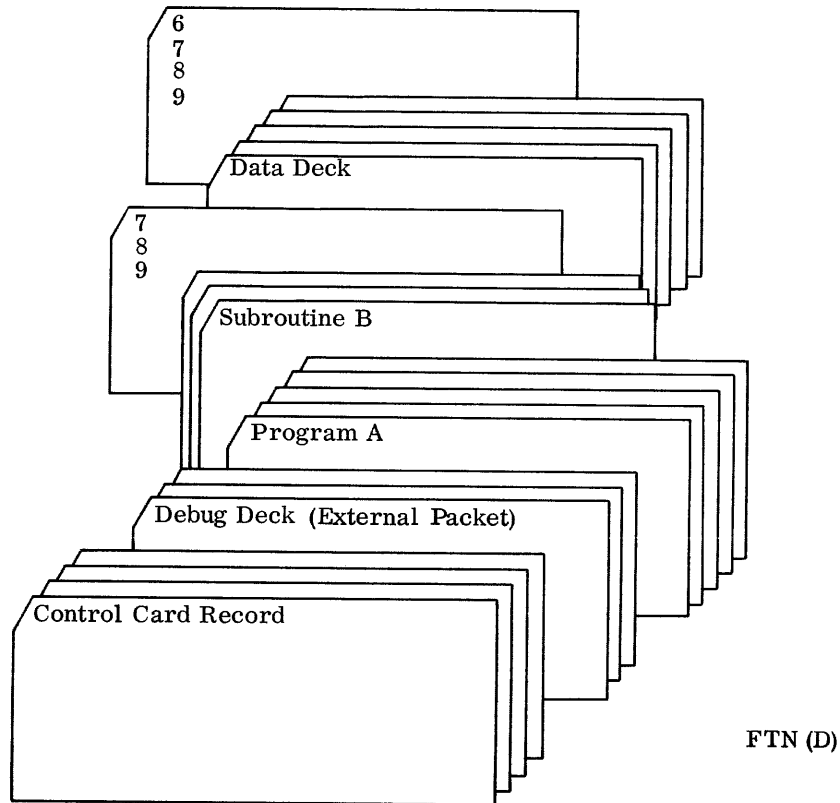
**SAMPLE DEBUG AID POSITION:** As individual debug cards interspersed in a program unit. The debug cards are inserted into the program where they will be activated. This positioning is especially useful when a new program is to be run for the first time and the accuracy of specific areas, such as array bounds, is in doubt.

Figure 11-1. Sample Debug Aid Position



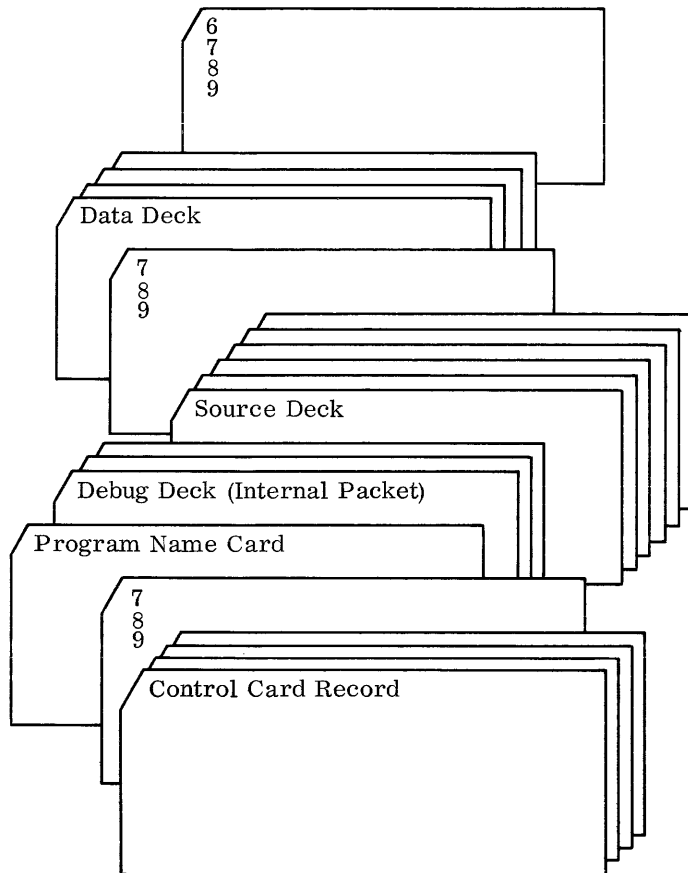
**SAMPLE DEBUG AID POSITIONS:** Debug deck placed on a separate file (external debug deck) named by the D parameter on the FTN control card, and called in during compilation. With these positions, all program units will be debugged (unless limiting bounds are specified in the deck). This positioning is particularly useful when several jobs can be debugged using the same debugging deck.

Figure 11-2. Sample Debug Aid Positions



**SAMPLE DEBUG AID POSITION:** As a deck, placed immediately in front of the first source line (when the D file is the same as the source input file). All program units (here, Program A and Subroutine B) will be debugged (unless limiting bounds are specified in the debug deck). This positioning is particularly useful when a program is to be run for the first time, since it ensures that all program units will be debugged.

Figure 11-3. Sample Debug Aid Position



FTN (D)

**SAMPLE DEBUG AID POSITION:** As a deck, placed immediately after the program header card and before any specification statements. All statements in the program unit will be debugged (unless limiting bounds are specified in the debug deck), but no statements in other program units will be debugged. This positioning is especially useful when the job is composed of several program units known to be free of bugs and one unit that is new or is known to have bugs.

Figure 11-4. Sample Debug Aid Position



## 11.10 DEBUG STATEMENT

A debug deck must begin with a DEBUG statement written in either of the forms:

```
C$  DEBUG
```

```
C$  DEBUG(name1, . . . , namen)
```

The program unit names, to which the debugging deck applies, must be enclosed in parentheses.

In an internal debugging deck, the DEBUG statement must appear immediately after the PROGRAM, SUBROUTINE, or FUNCTION statement heading the routine to which the debugging deck applies. Any names specified in the DEBUG statement, other than the name of the enclosing routine, are ignored.

In a single external debugging deck, whether on the job INPUT file or not, the DEBUG statement may contain a list of the program unit names to which the deck applies. If no name appears, the debugging deck applies to all program units being compiled.

When more than one C\$ DEBUG card occurs in an external debugging deck, this card specifies the routines to which the debugging specifications between it and the next C\$ DEBUG or non-debugging card apply.

This debug deck specifies arrays checking in all routines, stores checking on the variable CHI in routines CHISQ, STATP, and calls checking in routine MAIN.

```
C$  DEBUG
```

```
C$  ARRAYS
```

```
C$  DEBUG(CHISQ,STATP)
```

```
C$  STORES(CHI)
```

```
C$  DEBUG(MAIN)
```

```
C$  CALLS
```

## 11.11 AREA STATEMENT

The AREA statement allows a region smaller than a program unit to be debugged. All debugging statements that apply to the program areas designated by the AREA statement must follow that statement. Each succeeding AREA statement cancels the preceding program area designations.

When debugging statements are interspersed within the source deck, areas are defined by the position of the debug statements and the AREA statement is unnecessary. AREA statements interspersed in a source deck will be ignored.

The AREA statement can be written in two forms:

```
C$ AREA(bounds1), . . . , (boundsn)
```

for use in a debugging deck with the statement:

```
C$ DEBUG
```

or

```
C$ AREA/name1/(bounds1), . . . /namen/(boundsn)
```

for use in a debugging deck with the statement:

```
C$ DEBUG(name1, . . . , namen)
```

```
C$ DEBUG
```

The second form of the AREA statement must be used in an external debugging deck.

In the second form of the AREA statement, the /name_i/parameter designates the program units to which the bounds following it apply. If a (name_i) list appears on the C\$ DEBUG card, the /name_i/ parameter must be present and name_i must be included in the list. Otherwise, the C\$ AREA statement and its associated debugging specifications are ignored. For an external debugging deck the /name_i/ field must be present when using either form of the C\$ DEBUG statement.

The (bounds) parameter may be written in one of the following forms:

(from field) indicates line position to be debugged

(from field, to field) defines a range of line positions which may be in one of the following:

nnnn FORTRAN statement label

Lnnnn	Program line number as printed on the source listing (source listing line numbers will change when debugging cards are interspersed in the program.)
id.n	Legal UPDATE line identifier, from the source line, where id = information in columns 73-79; must begin with an alphabetic character and contain no special characters; and n = columns 82-86. (80-81 are blank.)
*	First line in the from field Last line in the to field

A comma must be used to separate the line numbers, and embedded blanks are not permitted.

```

C$  DEBUG(CHISQ)
C$  AREA/CHISQ/(210,400)
C$  ARRAYS(SVAL,RMS)
C$  DEBUG(CHISQ,STATP)
C$  AREA/CHISQ/(210,*)/STATP/(L20,L47),
C$  *   (570,L94)
C$  STORES(CHI)
C$  DEBUG(MAIN)
C$  AREA/MAIN/(MAIN.2,MOD1.13)
C$  CALLS

```

## 11.12 OFF STATEMENT

C\$ OFF statements are effective only on interspersed debug directives. In a debugging deck, the C\$ OFF statement is ignored.

```

C$  OFF(x1,x2,...,xn)
C$  OFF

```

```

Printout: C$  OFF
          C$  OFF(STORES,ARRAYS,GOTOS)

```

The C\$ OFF statement deactivates subsequent references to debugging options previously activated by interspersed specifications except for C\$ NOGO. If a parameter list is specified, only the options in the list are deactivated. Debugging options activated subsequent to the C\$ OFF statement and options activated by packet specifications will function normally. The C\$ OFF statement is effective at compile time only.

**11.13  
PRINTING  
DEBUG OUTPUT**

All debug messages produced by the object routines are written to a file named DEBUG. The file DEBUG will be printed at job termination time, since it has a print disposition. If the programmer wants to intersperse debug information with his output, he should equate DEBUG to OUTPUT on his program card. A FET and buffer will be supplied automatically at load time if the programmer does not declare the DEBUG file on his program card. For overlay jobs the buffer and FET will be placed in the lowest level of overlay containing debugging. If this overlay level will be overwritten by a subsequent overlay load, the debug buffer will be cleared before it is overwritten.

All object time printing is performed by seven debug routines coded in FORTRAN. These routines are called by code generated when debugging is selected on items such as arrays, calls, stores, etc. The seven routines and their functions are:

<u>ROUTINE</u>	<u>FUNCTION</u>
BUGARR	Checks array subscripts
BUGCLL	Prints messages when subroutines are called. Return
BUGFUN	Prints messages when functions are called. Return
BUGGTa	Prints a message if the target of an assigned GO TO is not in the list.
BUGSTO	Performs stores checking
BUGTRC	Flow trace printing except for true sides of logical IF
BUGTRT	Flow trace printing for true sides of logical IF.

## STRACE

Traceback information from a current subroutine level back to the main level is available through a call to STRACE, STRACE is an entry point in the object routine BUGCLL. A program need not specify the D option on the FTN card to use the STRACE feature.

STRACE output is written on the file DEBUG; to obtain traceback information interspersed with the source program DEBUG should be equivalenced to OUTPUT in the PROGRAM statement.

```
PROGRAM MAIN (OUTPUT,DEBUG=OUTPUT)
CALL SUB1
END
```

```
SUBROUTINE SUB1
CALL SUB2
RETURN
END
```

```
SUBROUTINE SUB2
I = FUNC1(2)
RETURN
END
```

```
FUNCTION FUNC1 (K)
FUNC1 = K ** 10
CALL STRACE
RETURN
END
```

Output from STRACE:

```
/DEBUG/ FUNC1 AT LINE 3- TRACE ROUTINE CALLED
          FUNC1 CALLED BY SUB2 AT LINE 2, FROM 1 LEVELS BACK
          SUB2 CALLED BY SUB1 AT LINE 2, FROM 2 LEVELS BACK
          SUB1 CALLED BY MAIN AT LINE 2, FROM 3 LEVELS BACK
```

```

PROGRAM      SAMPLE      DEEUG      TRACE      CCC 6600 FTN V3.0-P240 OPT=0 01/14/71 16.58.14.      PAGE      1
C$          PROGRAM SAMPLE(OUTPUT, DEBUG=OUTPUT)
C$          DEBUG
C$          AREA(1,100)
C$          ARRAYS(A1)
05          DIMENSION A1(5), A2(5)
           1 PRINT 99
           99 FORMAT(//* MESSAGES SHOULD FOLLOW FOR REFERENCES TO A1(0) AND
           /*/* A1(6), FOR BOTH LOADS AND STORES. THERE SHOULD BE NO MESSAGE
           /*/* FOR A2.*/)
10          DO 100 I = 1,4
           A1(2+I) = A1(4-I) = I
           A2(2+I) = A2(4-I) = I
           AGAIN = A1(2+I)
           AGAIN = A1(4-I)
15          AGAIN = A2(2+I)
           100 AGAIN = A2(4-I)
C$          CALLS(SUB1,SLITE)
           201 PRINT 299
           299 FORMAT(//* TWO MESSAGES SHOULD FOLLCH, ONE FOR A CALL OF SUB1 WITH
           /*/* ARGUMENT 7743, AND ONE FOR A CALL OF SLITE WITH ARGUMENT 1.
           /*/* THERE SHOULD BE NO MESSAGES FOR CALLS OF SUB2 AND SLITET.*/)
           CALL SUB1(7743)
           CALL SUB2(8242)
           CALL SLITE(1)
25          300 CALL SLITET(1,I)
C$          FUNCS(FUN1,IABS,EXP)
           301 PRINT 399
           399 FORMAT(//* MESSAGES SHOULD FOLLOW FOR CALLS OF FUN1 WITH ARGUMENT
           /*/* 7743, IABS WITH ARGUMENT 8242, AND EXP WITH ARGUMENT 3.14159.
           /*/* THERE SHOULD BE NO MESSAGES FOR CALLS OF FUN2, ABS, OR ALOG.*/)
           IAGAIN = FUN1(7743)
           IAGAIN = FUN2(7743)
           IAGAIN = IABS(8242)
           AGAIN = ABS(8242.)
35          AGAIN = EXP(3.14159)
           400 AGAIN = ALOG(3.14159)
C$          STORES(A1,AGAIN,I,A2.EQ.5.,IAGAIN.LE.10)
           401 PRINT 499
           499 FORMAT(//* MESSAGES SHOULD FOLLOW FOR STORES INTO A1(1), A1(2),
           /*/* A1(3), A1(4), A1(5), I, AGAIN, A2(1), IAGAIN, IAGAIN, IAGAIN,
           /*/* IAGAIN, AND IAGAIN. THE VALUES STORED IN THE RESPECTIVE
           /*/* VARIABLES SHOULD BE 1., 2., 3., 4., 5., 5, 3.14159, 5.,
           /*/* 10., 9, 8, 7, 6. THERE SHOULD BE NO OTHER STORES MESSAGES.*/)
           DO 402 I = 1,10
           A1(I) = I
           IF(I.EQ.5)GO TO 403
           402 CONTINUE
           403 AGAIN = 3.14159
           DO 500 I = 1,10
           A2(1) = 4. + I
50          500 IAGAIN = 16 - I
C$          AREA(501,600)
C$          TRACE(3)
           501 PRINT 599
           599 FORMAT(//* MESSAGES SHOULD FOLLOW FOR TRANSFERS OF CONTROL

```

```

PROGRAM      SAMPLE  DEEUG  TRACE          CCC 6600 FTM V3.0-P240 OPT=C  01/14/71  16.58.14.    PAGE      2

        **/* FROM 502 TO 503, 503 TO 504, 504 TO 505, AND 505 TO 506.
        **/* THERE SHOULD BE NO OTHER CONTROL TRANSFER MESSAGES.*/
        DO 510 I = 1,2
60       DO 511 J = 1,2
        DO 512 K = 1,2
        DO 513 L = 1,2
        DO 514 M = 1,2
        GO TO 507
        514 CONTINUE
65       507 GO TO (508,508,508,508),L
        513 CONTINUE
        509 ASSIGN 503 TO L
        502 GO TO L, (503,506)
        512 CONTINUE
70       503 GO TO (504,504), 1
        511 CONTINUE
        504 GO TO 505
        510 CONTINUE
75       505 GO TO 506
        506 CONTINUE
        600 CONTINUE
C$      OFF
        A1(1) = 1.
        GO TO 601
80       601 AGAIN = FUN1(1)
        I = 1
        GO TO (602,602),I
        602 CALL SUB1(7743)
        AGATN = 3.7
85       C$      GOTOS
        701 PRINT 799
        799 FORMAT(**/* WILL NOW ATTEMPT AN ASSIGNED GO TO. SHOULD ISSUE
        **/* MESSAGE.*/)
90       ASSIGN 6327 TO IGO
        GO TO IGO, (601,602)
        800 CONTINUE
        6327 PRINT 6328
        6328 FORMAT(**/* END OF SAMPLE DEBUG PROGRAM.*/)
        END

```

```

PROGRAM      SAMPLE  DEBUG  TRACE          CEC 6600 FTN V3.0-P240 OPT=0 01/14/71 16.58.14.    PAGE      3
SYMBOLIC REFERENCE MAP

ENTRY POINTS
2026 SAMPLE

VARIABLES      SN  TYPE      RELOCATION
2723 AGAIN     REAL          2732 A1      REAL      ARRAY
2737 A2        REAL      ARRAY  2722 I        INTEGER
2724 IAGAIN    INTEGER       2731 IG      INTEGER
2725 J         INTEGER       2726 K        INTEGER
2727 L         INTEGER       2730 M        INTEGER

FILE NAMES     MODE
0  DEBUG              0  OUTPUT  FMT

EXTERNALS      TYPE  ARCS
ABS            REAL   1      ALOG      REAL     1
EXP            REAL   1      FUN1      REAL     1
FUN2           REAL   1      IABS     INTEGER  1
SLITE          REAL   1      SLITET   2
SUB1           REAL   1      SUB2     1

STATEMENT LABELS
0 1            INACTIVE  2471 99    FMT              0 100
0 201          INACTIVE  2510 299  FMT              0 300    INACTIVE
0 301          INACTIVE  2534 399  FMT              0 400    INACTIVE
0 401          INACTIVE          0 402              2205 403
2561 499      FMT          0 500              0 501    INACTIVE
0 502          INACTIVE  2267 503  2303 504
2311 505          2314 506  2241 507
2254 508          0 510              0 511
0 512          0 513              0 514
2623 599      FMT          0 600    INACTIVE  2317 601
2331 602          0 701    INACTIVE  2647 799    FMT
0 800          INACTIVE  2350 6327  2660 6328    FMT

STATISTICS
PROGRAM LENGTH  722B  46E
BUFFER LENGTH  2022B 1042

```

```

SUBROUTINE SUB1  DEBUG  TRACE          CDC 6600 FTN V3.0-P240 OPT=0 01/14/71 16.58.14.    PAGE      1
      SUBROUTINE SUB1(I)
      END

```



```

SUBROUTINE SUB1      DEEUG   TRACE          CCC 6600 FTN V3.0-P240 OPT=0 01/14/71 16.58.14.    PAGE    2
SYMBOLIC REFERENCE MAP
ENTRY POINTS
2 SUB1
VARIABLES          SN TYPE          RELOCATION
0 I                INTEGER *UNUSED  F.P.
STATISTICS
PROGRAM LENGTH      7B          7

```

```

SUBROUTINE SUB2      DEEUG   TRACE          CCC 6600 FTN V3.0-P240 OPT=0 01/14/71 16.58.14.    PAGE    1
SUBROUTINE SUB2(I)
END

```

```

SUBROUTINE SUB2      DEEUG   TRACE          CCC 6600 FTN V3.0-P240 OPT=0 01/14/71 16.58.14.    PAGE    2
SYMBOLIC REFERENCE MAP
ENTRY POINTS
2 SUB2
VARIABLES          SN TYPE          RELOCATION
0 I                INTEGER *UNUSED  F.P.
STATISTICS
PROGRAM LENGTH      7B          7

```

```

FUNCTION FUN1        DEEUG   TRACE          CCC 6600 FTN V3.0-P240 OPT=0 01/14/71 16.58.14.    PAGE    1
FUNCTION FUN1(I)
FUN1 = I
END

```

```

FUNCTION   FUN1   DEEUG   TRACE           CCC 666E FTN V3.0-P240 OPT=0 01/14/71 16.58.14.   PAGE   2
SYMBOLIC REFERENCE MAP

ENTRY POINTS
 2 FUN1

VARIABLES  SN TYPE          RELOCATION      0  I      INTEGER          F.F.
 12 FUN1   REAL

STATISTICS

PROGRAM LENGTH   138   11

```

```

FUNCTION   FUN2   DEEUG   TRACE           CCC 666E FTN V3.0-P240 OPT=0 01/14/71 16.58.14.   PAGE   1
FUNCTION FUN2(I)
FUN2 = I
END

```

```

FUNCTION   FUN2   DEEUG   TRACE           CCC 666E FTN V3.0-P240 OPT=0 01/14/71 16.58.14.   PAGE   2
SYMBOLIC REFERENCE MAP

ENTRY POINTS
 2 FUN2

VARIABLES  SN TYPE          RELOCATION      0  I      INTEGER          F.F.
 12 FUN2   REAL

STATISTICS

PROGRAM LENGTH   138   11

```

```

CORE MAF 16.58.37. NORMAL CONTROL
---TIME---LCAD MODF --L1--L2---TYPE-----USER---+---CALL-----FWA LOAD--LWA LCAD--BLNK CCM--LENGTH--
FWA LOADER 054771 FWA TABLES 052456
-PROGRAM---ADDRESS- --Labeled---COMMON--
SAMPLE 000100
SUB1 003:44
SUB2 003:53
FUN1 003:62
FUN2 003:75
GETBA 003110
SIO$ 003127
SYSTEM$ 004502
ACGOER$ 005465
BUGARR$ 005500
BUGCLL$ 005570
BUGCTL$ 006:77
BUGFUN$ 006134
BUGGTA$ 006553
BUGSTO$ 006571
BUGTRC$ 007715
DBGFET$ 010147
KODER$ 011173
OUTPTC$ 012567
TRAGEX$ 012663
ABSS$ 012712
IABSS$ 012715
ALNLCGE 012720
ALOG$ 012757
EXP$ 013:11
EXPE 013:53
LEGVAR$ 013117
LOCFS$ 013124
SLITE$ 013126
SLITET$ 013150
    
```

} DEBUG object time routines

----UNSATISFIED EXTERNALS-----

REFERENCES

MESSAGES SHOULD FOLLOW FOR REFERENCES TO A1(0) AND A1(6), FOR BOTH LOADS AND STORES. THERE SHOULD BE NO MESSAGE FOR A2.

/DEBUG/	SAMPLE	AT LINE	11-	THE SUBSCRIPT VALUE CF	6	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND CF	5
/DEBUG/		AT LINE	11-	THE SUBSCRIPT VALUE CF	0	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND CF	5
/DEBUG/		AT LINE	13-	THE SUBSCRIPT VALUE CF	6	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND CF	5
/DEBUG/		AT LINE	14-	THE SUBSCRIPT VALUE CF	0	IN ARRAY A1	EXCEEDS DIMENSIONED BOUND CF	5

TWO MESSAGES SHOULD FOLLOW, ONE FOR A CALL OF SUB1 WITH ARGUMENT 7743, AND ONE FOR A CALL OF SLITE WITH ARGUMENT 1. THERE SHOULD BE NO MESSAGES FOR CALLS OF SUB2 AND SLITET.

/DEBUG/	SAMPLE	AT LINE	22-	ROUTINE SUB1	CALLED AT LEVEL	0
/DEBUG/		AT LINE	23-	ROUTINE SUB1	RETURNS TO LEVEL	0
/DEBUG/		AT LINE	24-	ROUTINE SLITE	CALLED AT LEVEL	0
/DEBUG/		AT LINE	25-	ROUTINE SLITE	RETURNS TO LEVEL	0

MESSAGES SHOULD FOLLOW FOR CALLS OF FUN1 WITH ARGUMENT 7743, TABS WITH ARGUMENT 8242, AND EXP WITH ARGUMENT 3.14159. THERE SHOULD BE NO MESSAGES FOR CALLS OF FUN2, ABS, OR ALOG.

/DEBUG/	SAMPLE	AT LINE	31-	REAL FUNCTION FUN1	CALLED AT LEVEL	0
/DEBUG/		AT LINE	31-	REAL FUNCTION FUN1	RETURNS A VALUE OF	7743.000000 AT LEVEL 0
/DEBUG/		AT LINE	32-	INTEGER FUNCTION IABS	CALLED AT LEVEL	0
/DEBUG/		AT LINE	32-	INTEGER FUNCTION IABS	RETURNS A VALUE OF	8242 AT LEVEL 0
/DEBUG/		AT LINE	35-	REAL FUNCTION EXP	CALLED AT LEVEL	0
/DEBUG/		AT LINE	35-	REAL FUNCTION EXP	RETURNS A VALUE OF	23.14063123 AT LEVEL 0

MESSAGES SHOULD FOLLOW FOR STORES INTO A1(1), A1(2), A1(3), A1(4), A1(5), I, AGAIN, A2(1), IAGAIN, IAGAIN, IAGAIN, IAGAIN, AND IAGAIN. THE VALUES STORED IN THE RESPECTIVE VARIABLES SHOULD BE 1., 2., 3., 4., 5., 5, 3.14159, 5., 10., 9., 8., 7., 6. THERE SHOULD BE NO OTHER STORES MESSAGES.

/DEBUG/	SAMPLE	AT LINE	45-	THE NEW VALUE OF THE VARIABLE A1	IS	1.000000000
/DEBUG/		AT LINE	45-	THE NEW VALUE OF THE VARIABLE A1	IS	2.000000000
/DEBUG/		AT LINE	45-	THE NEW VALUE OF THE VARIABLE A1	IS	3.000000000
/DEBUG/		AT LINE	45-	THE NEW VALUE OF THE VARIABLE A1	IS	4.000000000
/DEBUG/		AT LINE	45-	THE NEW VALUE OF THE VARIABLE A1	IS	5.000000000
/DEBUG/		AT LINE	48-	THE NEW VALUE OF THE VARIABLE AGAIN	IS	3.141590000
/DEBUG/		AT LINE	5-	THE NEW VALUE OF THE VARIABLE A2	IS	5.000000000
/DEBUG/		AT LINE	51-	THE NEW VALUE OF THE VARIABLE IAGAIN	IS	10
/DEBUG/		AT LINE	51-	THE NEW VALUE OF THE VARIABLE IAGAIN	IS	9
/DEBUG/		AT LINE	51-	THE NEW VALUE OF THE VARIABLE IAGAIN	IS	8
/DEBUG/		AT LINE	51-	THE NEW VALUE OF THE VARIABLE IAGAIN	IS	7
/DEBUG/		AT LINE	51-	THE NEW VALUE OF THE VARIABLE IAGAIN	IS	6

MESSAGES SHOULD FOLLOW FOR TRANSFERS OF CONTROL FROM 502 TO 503, 503 TO 504, 504 TO 505, AND 505 TO 506. THERE SHOULD BE NO OTHER CONTROL TRANSFER MESSAGES.

/DEBUG/	SAMPLE	AT LINE	6A-	CONTROL WILL BE TRANSFERRED TO STATEMENT 503	AT LINE	70
/DEBUG/		AT LINE	7-	CONTROL WILL BE TRANSFERRED TO STATEMENT 504	AT LINE	72
/DEBUG/		AT LINE	72-	CONTROL WILL BE TRANSFERRED TO STATEMENT 505	AT LINE	74
/DEBUG/		AT LINE	74-	CONTROL WILL BE TRANSFERRED TO STATEMENT 506	AT LINE	75

WILL NOW ATTEMPT AN ASSIGNED GOTO. SHOULD ISSUE MESSAGE.

/DEBUG/	SAMPLE	AT LINE	9-	ASSIGNED GOTO INDEX CONTAINS THE ADDRESS 002450. NO MATCH FOUND IN STATEMENT LABEL ADDRESS LIST
---------	--------	---------	----	-------------------------------------------------------------------------------------------------

END OF SAMPLE DEBUG PROGRAM.

## 12.1 CONTROL CARD FORMAT

The control card for compilation of a FORTRAN source program consists of the characters FTN and an optional parameter list enclosed in parentheses. If parameters are omitted, FTN is followed by a period. Comments following the right parenthesis or period are transcribed to the dayfile in a normal installation. The first improperly formed parameter terminates the FTN control card scan.

FTN ( $p_1, p_2, \dots, p_n$ ) comments

or

FTN. comments

When an error is detected in a control card, a dayfile entry is made consisting of an asterisk (below the approximate column in which the compiler encountered the error) and the following message:

*POINTS TO FTN CONTROL CARD ERROR

Example of dayfile:

```
06.52.35.FTN(I=0/L=LIST)
06.52.36.      *
06.52.36. * POINTS TO FTN CONTROL CARD ERROR
```

The job will proceed with the options already processed or terminate and branch to an EXIT(S) card, depending upon an installation option. Default files or files specified in the control card must be in SCOPE 3 format.

## 12.2 SOURCE INPUT PARAMETER

If the source input parameter is omitted (default condition), the FORTRAN source input file is assumed to be INPUT. If it is on any other file, a parameter of the following form must be provided:

I=lfm (default I=INPUT)

lfm is the logical file name of the file containing the source input. Source input parameters of the forms I=INPUT and I are equivalent to omitting the parameter.

### 12.3

#### **BINARY (OBJECT)**

**OUTPUT PARAMETER** If the binary output parameter is omitted (default condition), a relocatable binary (object) file is written on a file named LGO. For any other output file, a parameter of the following form must be provided:

B=lfm (default B=LGO)

lfm is the name of the file on which binary output is to be written. Binary output parameters of the form B=LGO or B are equivalent to omitting the parameter.

To suppress production of an object output file, the binary output parameter must be of the form:

B=0

If the letter G appears in the binary output parameter, the object file will be loaded and executed at the end of compilation.

G=lfm BG=lfm GB=lfm G

### 12.4

#### **LIST PARAMETER**

If this parameter is omitted (default condition), a normal listing is provided on OUTPUT; it includes the source program and informative and fatal diagnostics. Other list options may be selected as follows:

y=lfm (default L=OUTPUT, R=1)

y may be one or more of the following:

- L Normal listing
- X Listing of diagnostics which indicate non-ANSI language usage
- R Source keyed cross reference map (implies R=2) see paragraph 12.16
- O Listing of generated object code
- N Suppress listing of informative diagnostics and list only diagnostics fatal to execution

lfn is the file name on which list output is to be written. If lfn is omitted, listing will be on OUTPUT. If L=0 fatal diagnostics with the statements that caused them will be listed; but all other listable output including intermixed COMPASS will be suppressed.

Any combination (with no comma) of the above parameters provides the features indicated. (Note: X and N cannot be used at the same time.)

LRON=lfn specifies all options are to be listed for the file named except non-ANSI diagnostics, and LO selects source and assembly listing on OUTPUT.

#### CROSS REFERENCE MAP

The FORTRAN Extended cross reference map can be obtained using the R option. This map is described in Appendix C.

### **12.5 ERROR TRACEBACK AND CALLING SEQUENCE PARAMETER**

The T mode of compilation is intended for use with programs in the debugging stages. This parameter is indicated by T†. When it is present, calls to library functions will be made (with the call-by-name sequence), and maximum error checking will be done. T mode forces inline functions to become external functions. Full error traceback will be done if an error is detected. T mode is selected automatically when D† or OPT=0 are specified.

When T is omitted, the compiler generates calls to library functions with the call-by-value sequence (e.g., cause X1 to contain the parameter, RJ function). Minimum error checking will be done and no traceback will be provided when errors are encountered. A significant saving in memory space and execution time is realized.

### **12.6 UPDATE PARAMETER †† (EDITING PARAMETERS)**

An E or E=lfn (default E = COMPS) as a parameter requests that the object code output from the compiler produce COMPASS subprogram line images for UPDATE input. This output facilitates hand optimization of the compiler generated object code.

*DECK, name (name = program unit name) is the first card image written on the object code output file, COMPS (assumed when lfn does not appear). An *END card image is written as the last card on the file. COMPASS is not called automatically. The output file lfn or COMPS is rewound and ready for UPDATE input. No binary file is produced.

The O and C options are illegal when E is used.

---

†See Debugging Mode Parameter section in this chapter.

††KRONOS 2.0 does not support UPDATE.

When the E parameter is specified, the COMPS file will contain:

```
*COMDECK,FTN=MAC
  Necessary FTNMAC text
*DECK prog etc.
  IDENT prog
*CALL,FTN=MAC
  assembly code
```

This is intended to be used as follows:

```
jobcard
FTN(E)
UPDATE(I=COMPS,N=OLDPL)
UPDATE(F)      (to perform any user modifications)
COMPASS(I)     (assembly code)
7/8/9
  FORTRAN source code
7/8/9
  UPDATE IDENTs to modify assembly code
7/8/9
  Data if necessary
6/7/8/9
```

## 12.7 OPTIMIZATION PARAMETER

The OPT parameter is of the form:

```
OPT=m
```

The level of optimization the compiler will perform is determined by the value of m as follows:

```
m=0    fast compile mode (implies T mode)
m=1    standard compile mode
m=2    fast object code mode
```

If this parameter is omitted, the installation default option is assumed. Debug mode D option on FTN card implies that OPT=0.

The OPT=2 level of optimization can offer significant execution speed increases for certain classes of loops. Two types of optimization are performed:

- Moving of invariant computations from frequently executed regions to less frequently executed regions.
- Assignment of variables and constants to registers over the body of a loop.



Both DO loops and IF loops can be optimized within these constraints.

- The loop must be the innermost loop (i.e., contain no loops).
- The loop must contain no branching statements (GO TO, IF or RETURN) except a branch back to the start of the loop for IF loops.
- The loop does not contain nonstandard input/output statements such as BUFFER IN/BUFFER OUT, ENCODE/DECODE. In case standard I/O statements occur (or any external calls), only invariant code removal will be attempted.
- Control must flow to the statement following the end of the loop when the loop completes.
- Entry into the loop must be through the sequence of statements preceding the start of the loop.

#### Invariant Computations

In many instances, either for clarity or by accident, calculations which do not change on successive iterations are made within a loop. When these computations are made outside the loop, the speed of the loop is improved without changing the results.

Example 1:

```
DO 100 I = 1,2000
100 A(I) = 3*I + J/K+5
```

A more efficient loop would be:

```
ITERM = J/K+5
DO 100 I = 1,2000
100 A(I) = 3*I + ITERM
```

For clarity, the programmer may not wish to write the code in this form. Using the OPT=2 level, the more efficient loop structure would be produced. A message will be issued stating:

```
n WORDS OF INVARIANT RLIST REMOVED FROM
THE LOOP STARTING AT LINE x
```

RLIST is the intermediate language of the compiler. The message serves two functions. It notifies the programmer that his loop has been modified, and it informs him of the magnitude of the change.

Example 2:

```
I = 1
200 J = K+L+4
    A(I) = M+I
    I = I+1
    IF(I. LE. 100) GO TO 200
```

Use of OPT=2 will produce code as if Example 2 had been written as shown below:

```
I = 1
    J = K+L+4
200 A(I) = M+I
    I = I+1
    IF(I. LE. 100) GO TO 200
```

Example 3:

```
DO 300 I=1,2000
    A(I) = SQRT(FLOAT(I))
    A(I) = A(I) + 3.5*R
300 CONTINUE
```

The computation of  $3.5 * R$  will be removed from the loop in spite of the external call. In general, this process will occur unless  $R$  is a parameter to the external routine or in COMMON. The use of a variable will not be recognized as invariant if it is a member of an equivalence class for which some member of the class is referenced inside the loop using nonstandard subscripts. For standard subscripts, optimization will occur, although the assumption is made that all subscripting is within the bounds of dimensional declarations.

### Register Assignment

For many loops, it is possible to keep commonly used variables and constants in the registers. Eliminating loads and stores from the body of the loop has two advantages:

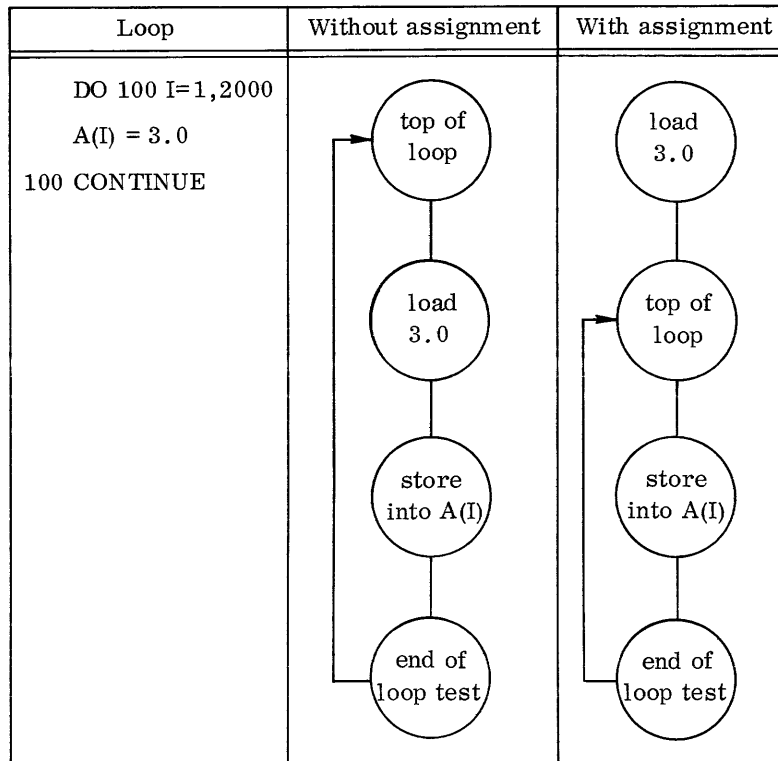
- Reducing the number of loads and stores increases the execution speed.
- The loop is shortened and may fit in the instruction stack of the 6600.

Presently up to four X registers may be assigned over a loop. The actual number assigned depends on the number of candidates available for selection and the complexity of the operations performed within the loop. When registers are assigned, an informative message is printed:

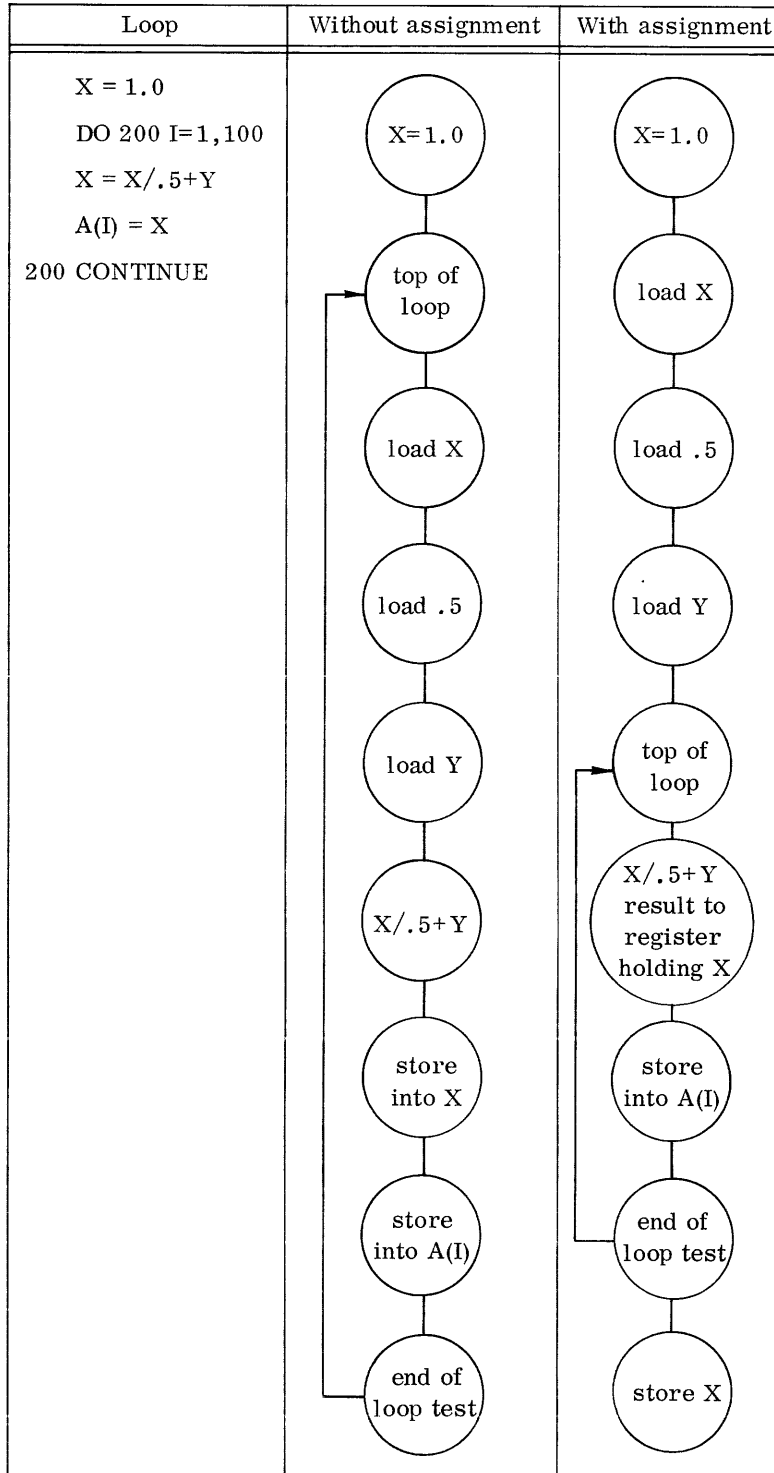
n REGISTERS ASSIGNED OVER THE LOOP BEGINNING AT LINE x

Register assignment will not be performed for loops containing external references.

Example 1:



Example 2:



## 12.8 ROUNDED ARITHMETIC PARAMETER

The compiler will produce rounded arithmetic instructions for any combination of arithmetic operators (+ - * /) if the parameter is specified in the form:

ROUND= operators (default=OFF)

If this parameter is omitted (default condition), rounded arithmetic processing does not take place.

## 12.9 DEBUGGING MODE PARAMETER

When this parameter is selected, the OPT=0 compilation and T error traceback modes are assumed. If the debugging mode parameter is omitted (default condition), this mode of compilation does not take place. (The debugging mode is described in Chapter 11.)

D or D=lfm (default = INPUT)

lfm specifies the file name of the debugging aid selection package.

## 12.10 EXIT PARAMETER

† When this parameter is specified, the run will terminate and branch to an EXIT(S) control card if fatal errors occur during compilation. The form is:

A (default off)

## 12.11 SYSTEM TEXT FILE PARAMETER

The S parameter specifies the systems text file to be used for intermixed COMPASS programs.

S = 0 or S = lfm (default lfm = SYSTEXT)

If S=0 when COMPASS is called to assemble any intermixed COMPASS programs, it will not read in a system text file. If this parameter is omitted (default condition), S=SYSTEXT is assumed.

---

†Not supported by 7600 SCOPE 1.1.

**12.12  
SYSTEM EDITING  
AND I/O REFERENCE  
PARAMETER**

This parameter is of the following form:

SYSEEDIT= FILES

SYSEEDIT=IDENT

SYSEEDIT (default, both FILES and IDENT)

The FILES specification assures that all INPUT/OUTPUT references will be accomplished indirectly through GETBA. In addition, the file names are not entry points in the main program, and subprograms do not produce external references to the file names. When IDENT is specified, a \$ is appended to the program name on both the IDENT and ENTRY cards if the program name is the same as that of any FORTRAN object library program.

**12.13  
ASSEMBLER  
PARAMETER**

The COMPASS assembler, rather than the FTN built-in assembler, can be used to assemble the code generated by FTN. The COMPASS assembler is specified with the following parameter:

C (default off)

**12.14  
CONTROL CARD  
EXAMPLES**

The control card FTN. is equivalent to:

FTN (I=INPUT,L=OUTPUT,B=LGO,S=SYSTEXT,OPT=1,R=1)

The control card:

FTN (A,LRN,G,S=0)

will select the following options:

- A Abort, branch to EXIT(S) card when errors occur in compilation
- LRN List on the file OUTPUT, which will include a source-keyed cross reference map, and suppress the informative diagnostics.
- G The relocatable binary file is placed on file LGO. If compilation is successful, it will be loaded and executed.
- S=0 When COMPASS is called to assemble intermixed COMPASS subprograms, it will not read in a system text file.

**12.15  
SMALL  
BUFFERS**

When this option is selected, the compiler is forced to use 513-word buffers for compiler intermediate files. Programs with a large number of declarations may be compiled with a smaller field length if this parameter is specified. Since less space is used in the buffers, compile time may increase. The form of the parameter is:

V                    (default = off)

**12.16  
REFERENCE  
MAP LEVEL**

The kind of reference map (Appendix C) produced is determined by the R option on the control card:

- R = 0    No map
- R = 1    Short map (symbols, addresses, properties)
- R = 2    Long map (short map, references by line number and a DO-loop map)
- R = 3    Long map and printout of common block members and equivalence classes
  
- R        Implies R = 2

If R is not specified the default option is R = 1 unless the L option equals 0; then R = 0.

## STANDARD SCOPE CHARACTER SETS

A

---

The character set selected when the system is installed should be compatible with the printers.

With an installation parameter, the installation keypunch format standard can be selected as 026 or 029; the installation parameter can also allow a user to override the standard; a user may select a keypunch mode for his input deck by punching 26 or 29 in columns 79 and 80 of his JOB card or any 7/8/9 end-of-record card. The mode remains set for the remainder of the job or until it is reset by a different mode selection on another 7/8/9 card.



## CDC 64-CHARACTER SET

Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD	Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD
00	: †	8-2	8-2	00*	40	5	5	5	05
01	A	12-1	12-1	61	41	6	6	6	06
02	B	12-2	12-2	62	42	7	7	7	07
03	C	12-3	12-3	63	43	8	8	8	10
04	D	12-4	12-4	64	44	9	9	9	11
05	E	12-5	12-5	65	45	+	12	12-8-6	60
06	F	12-6	12-6	66	46	-	11	11	40
07	G	12-7	12-7	67	47	*	11-8-4	11-8-4	54
10	H	12-8	12-8	70	50	/	0-1	0-1	21
11	I	12-9	12-9	71	51	(	0-8-4	12-8-5	34
12	J	11-1	11-1	41	52	)	12-8-4	11-8-5	74
13	K	11-2	11-2	42	53	\$	11-8-3	11-8-3	53
14	L	11-3	11-3	43	54	=	8-3	8-6	13
15	M	11-4	11-4	44	55	blank	no punch	no punch	20
16	N	11-5	11-5	45	56	, (comma)	0-8-3	0-8-3	33
17	O	11-6	11-6	46	57	. (period)	12-8-3	12-8-3	73
20	P	11-7	11-7	47	60	≡	0-8-6	8-3	36
21	Q	11-8	11-8	50	61	[	8-7	8-5	17
22	R	11-9	11-9	51	62	]	0-8-2	12-8-7	32
23	S	0-2	0-2	22	63	%	8-6	0-8-4	16
24	T	0-3	0-3	23	64	≠	8-4	8-7	14
25	U	0-4	0-4	24	65	→	0-8-5	0-8-5	35
26	V	0-5	0-5	25	66	v	11-0 or 11-8-2	11-0 or 11-8-2	52
27	W	0-6	0-6	26					
30	X	0-7	0-7	27	67	^	0-8-7	12	37
31	Y	0-8	0-8	30	70	†	11-8-5	8-4	55
32	Z	0-9	0-9	31	71	‡	11-8-6	0-8-7	56
33	0	0	0	12	72	<	12-0 or 12-8-2	12-0 or 12-8-2	72
34	1	1	1	01					
35	2	2	2	02					
36	3	3	3	03	73	>	11-8-7	0-8-6	57
37	4	4	4	04	74	∨	8-5	12-8-4	15
					75	∧	12-8-5	0-8-2	75
					76	∩	12-8-6	11-8-7	76
					77	;(semicolon)	12-8-7	11-8-6	77

†This character is lost on even parity magnetic tape.

*Since 00 cannot be represented on magnetic tape, it is converted to BCD 12. On input, it will be translated to display code 33 (number zero).

## ASCII 64-CHARACTER SUBSET*

Display Code	Character	Hollerith (026)	Hollerith (029)	ASCII Code	Display Code	Character	Hollerith (026)	Hollerith (029)	ASCII Code
00	: †	8-2	8-2	072	40	5	5	5	065
01	A	12-1	12-1	101	41	6	6	6	066
02	B	12-2	12-2	102	42	7	7	7	067
03	C	12-3	12-3	103	43	8	8	8	070
04	D	12-4	12-4	104	44	9	9	9	071
05	E	12-5	12-5	105	45	+	12	12-8-6	053
06	F	12-6	12-6	106	46	-	11	11	055
07	G	12-7	12-7	107	47	*	11-8-4	11-8-4	052
10	H	12-8	12-8	110	50	/	0-1	0-1	057
11	I	12-9	12-9	111	51	(	0-8-4	12-8-5	050
12	J	11-1	11-1	112	52	)	12-8-4	11-8-5	051
13	K	11-2	11-2	113	53	\$	11-8-3	11-8-3	044
14	L	11-3	11-3	114	54	=	8-3	8-6	075
15	M	11-4	11-4	115	55	blank	no punch	no punch	040
16	N	11-5	11-5	116	56	, (comma)	0-8-3	0-8-3	054
17	O	11-6	11-6	117	57	. (period)	12-8-3	12-8-3	056
20	P	11-7	11-7	120	60	#	0-8-6	8-3	043
21	Q	11-8	11-8	121	61	' (apostrophe)	8-7	8-5	047
22	R	11-9	11-9	122	62	!	0-8-2	12-8-7	041
23	S	0-2	0-2	123	63	%	8-6	0-8-4	045
24	T	0-3	0-3	124	64	" (quote)	8-4	8-7	042
25	U	0-4	0-4	125	65	_ (underline)	0-8-5	0-8-5	137
26	V	0-5	0-5	126	66	]	11-0 or 11-8-2	11-0 or 11-8-2	135
27	W	0-6	0-6	127					
30	X	0-7	0-7	130	67	&	0-8-7	12	046
31	Y	0-8	0-8	131	70	@	11-8-5	8-4	100
32	Z	0-9	0-9	132	71	?	11-8-6	0-8-7	077
33	0	0	0	060	72	[	12-0 or 12-8-2	12-0 or 12-8-2	133
34	1	1	1	061					
35	2	2	2	062	73	>	11-8-7	0-8-6	076
36	3	3	3	063	74	<	8-5	12-8-4	074
37	4	4	4	064	75	\	12-8-5	0-8-2	134
					76	^ (circumflex)	12-8-6	11-8-7	136
					77	;(semicolon)	12-8-7	11-8-6	073

†This character is lost on even parity magnetic tape.

*BCD representation is used when data is recorded on even parity magnetic tape. In this case, the octal BCD/display code correspondence is the same as for the CDC 64-character set.

## CDC 63-CHARACTER SET

Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD	Display Code	Character	Hollerith (026)	Hollerith (029)	External BCD
00	(none)†			16	40	5	5	5	05
01	A	12-1	12-1	61	41	6	6	6	06
02	B	12-2	12-2	62	42	7	7	7	07
03	C	12-3	12-3	63	43	8	8	8	10
04	D	12-4	12-4	64	44	9	9	9	11
05	E	12-5	12-5	65	45	+	12	12-8-6	60
06	F	12-6	12-6	66	46	-	11	11	40
07	G	12-7	12-7	67	47	*	11-8-4	11-8-4	54
10	H	12-8	12-8	70	50	/	0-1	0-1	21
11	I	12-9	12-9	71	51	(	0-8-4	12-8-5	34
12	J	11-1	11-1	41	52	)	12-8-4	11-8-5	74
13	K	11-2	11-2	42	53	\$	11-8-3	11-8-3	53
14	L	11-3	11-3	43	54	=	8-3	8-6	13
15	M	11-4	11-4	44	55	blank	no punch	no punch	20
16	N	11-5	11-5	45	56	, (comma)	0-8-3	0-8-3	33
17	O	11-6	11-6	46	57	. (period)	12-8-3	12-8-3	73
20	P	11-7	11-7	47	60	≡	0-8-6	8-3	36
21	Q	11-8	11-8	50	61	[	8-7	8-5	17
22	R	11-9	11-9	51	62	]	0-8-2	12-8-7	32
23	S	0-2	0-2	22	63	:(colon) †	8-2	8-2	00*
24	T	0-3	0-3	23	64	≠	8-4	8-7	14
25	U	0-4	0-4	24	65	→	0-8-5	0-8-5	35
26	V	0-5	0-5	25	66	v	11-0 or	11-0 or	52
27	W	0-6	0-6	26			11-8-2	11-8-2	
30	X	0-7	0-7	27	67	^	0-8-7	12	37
31	Y	0-8	0-8	30	70	↑	11-8-5	8-4	55
32	Z	0-9	0-9	31	71	↓	11-8-6	0-8-7	56
33	0	0	0	12	72	<	12-0 or	12-0 or	72
34	1	1	1	01			12-8-2	12-8-2	
35	2	2	2	02					
36	3	3	3	03	73	>	11-8-7	0-8-6	57
37	4	4	4	04	74	≤	8-5	12-8-4	15
					75	≥	12-8-5	0-8-2	75
					76	┘	12-8-6	11-8-7	76
					77	;(semicolon)	12-8-7	11-8-6	77

† When the 63-Character Set is used, the punch code 8-2 is associated with display code 63, the colon. Display code 00_g is not included in the 63-Character Set and is not associated with any card punch. The 8-6 card punch (026 keypunch) and the 0-8-4 card punch (029 key-punch) in the 63-Character Set are treated as blank on input.

*Since 00 cannot be represented on magnetic tape, it is converted to BCD 12. On input, it will be translated to display code 33 (number zero).

---

Diagnostic messages are produced by the FORTRAN processor to inform the user of errors in the program. Messages are produced during compilation and execution; compilation errors are discussed in this appendix, a detailed discussion of the execution errors is given in Appendix G.

Errors detected during compilation are noted on the source listing, immediately following the END card. Figure B-1 illustrates a listing and the format used by the processor in noting compilation errors.

```
100 WRITE (6,8)
      8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
101 I=5
5      8 A=I
102 A=SQRT(A)
103 J=A
104 DO 1 K=3,J,2
105 L=I/K
10  106 IF(L*K-I)1,2,4
      1 GO TO 108
107 WRITE (6,9)
      5 FORMAT (I20)
      2 I=I+2
15  108 IF(1000-I)7,4,3
      4 WRITE (6,7)
      9 FORMAT (14H PROGRAM ERROR)
      7 WRITE (6,6)
      6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
20  109 STOP
      END
```

CARD NO.	SEVERITY		DIAGNOSTIC
01	I	START.	ASSUMED PROGRAM NAME WHEN NO HEADER STATEMENT APPEARS
05	FE	8	DUPLICATE STATEMENT LABEL
	FE		A DO LOOP MAY NOT TERMINATE ON THIS TYPE OF STATEMENT
21	FE		UNDEFINED STATEMENT NUMBERS
	FE		UNDEFINED LABELS
		3	16

Figure B-1. Sample Source Listing

The source of the errors is identified by the card number. This number corresponds to the card number assigned by the processor indicated by the numbers on the extreme left side of the example. The severity of the error is indicated by the code accompanying the message: I means informative and has no effect on compilation or execution, FE indicates catastrophic to execution, FC means the error is catastrophic to compilation, and ANSI indicates that the particular usage does not conform to ANSI standards. ANSI diagnostics are not listed unless requested by an X parameter on the FTN control card.

ANSI A RELATIONAL HAS A COMPLEX OPERAND.  
ANSI ARRAY NAME OPERAND NOT SUBSCRIPTED, FIRST ELEMENT WILL BE USED  
ANSI ARRAY NAME REFERENCED WITH FEWER SUBSCRIPTS THAN DIMENSIONALITY OF ARRAY.  
ANSI DOLLAR SIGN STATEMENT SEPARATOR IS NON-ANSI USAGE  
ANSI END STATEMENT ACTING AS A RETURN IS NON-ANSI  
ANSI ENTRY STATEMENT IS NON-ANSI  
ANSI FLOATING PT DESCRIPTOR EXPECTED FOLLOWING SCALE FACTOR DESIGNATOR.  
ANSI GO TO STATEMENT CONTAINS NON-ANSI USAGES  
ANSI HOLLERITH CONSTANT APPEARS OTHER THAN IN AN ARGUMENT LIST OF A CALL STATEMENT OR IN A DATA STATEMENT.  
ANSI HOLLERITH STRING DELINEATED BY SYMBOLS IS NON ANSI.  
ANSI LOGICAL OPERATOR OR CONSTANT USAGE IS NON-ANSI  
ANSI MASKING EXPRESSION IS NON-ANSI.  
ANSI MULTIPLE REPLACEMENT STATEMENT IS NON-ANSI.  
ANSI NAMELIST STATEMENT IS NON-ANSI  
ANSI NON-ANSI FORM OF DATA STATEMENT  
ANSI NON-STANDARD SUBSCRIPT IS NON-ANSI.  
ANSI OCTAL CONSTANT OR R,L FORMS OF HOLLERITH CONSTANT IS NON-ANSI  
ANSI PLUS SIGN IS AN ILLEGAL CHARACTER.  
ANSI PRECEDING FIELD DESCRIPTOR IS NON-ANSI.  
ANSI RETURNS PARAMETERS IN CALL STATEMENT.  
ANSI TAB SETTING DESIGNATOR IS NON-ANSI.  
ANSI THE EXPRESSION IN AN IF STATEMENT IS TYPE COMPLEX.  
ANSI THE FORMAT OF THIS END LINE DOES NOT CONFORM TO ANSI SPECIFICATIONS.  
ANSI THE NON-STANDARD RETURN STATEMENT IS NON-ANSI  
ANSI THE TYPE COMBINATION OF THE OPERANDS OF AN EXPONENT OPERATOR IS NON-ANSI.  
ANSI THE TYPE COMBINATION OF THE OPERANDS OF AN EQUAL-SIGN OPERATOR IS NON-ANSI.  
ANSI THE TYPE COMBINATION OF THE OPERANDS OF A RELATIONAL OR ARITHMETIC OPERATOR (OTHER THAN **) IS NON-ANSI.

ANSI THIS FORM OF AN I/O STATEMENT DOES NOT CONFORM TO ANSI SPECIFICATIONS  
ANSI THIS FORMAT DECLARATION IS NON-ANSI  
ANSI TWO-BRANCH IF STATEMENT IS NON-ANSI.  
ANSI USE OF COMMENT CARD IN CONTINUED STATEMENT IS NON-ANSI  
ANSI 7 CHARACTER SYMBOLIC NAME IS NON-ANSI  
FC ERROR TABLE OVERFLOW  
FC MEMORY OVERFLOW DURING ASF EXPANSION  
FC MISSING OR OUT OF RANGE LABEL ON DO STATEMENT  
FC NOT ENOUGH ROOM IN WORKING STORAGE TO HOLD ALL OVERLAY CONTROL CARD INFORMATION  
FC SYMBOL TABLE OVERFLOW  
FC TABLE OVERFLOW, INCREASE FL  
FC TABLES OVERLAP, INCREASE FL  
FC THIS SUBPROGRAM HAS TOO MANY DO LOOPS  
FE .NOT. MAY NOT BE PRECEDED BY NAME, CONSTANT, OR RIGHT PARENS.  
FE + OR - SIGN MUST BE FOLLOWED BY A CONSTANT  
FE A COMMA, LEFT PAREN., =, .OR., OR .AND. MUST BE FOLLOWED BY A NAME, CONSTANT, LEFT PAREN., -, .NOT., OR +.  
FE A COMPLEX BASE MAY ONLY BE RAISED TO AN INTEGER POWER  
FE A CONSTANT ARITHMETIC OPERATION WILL GIVE AN INDEFINITE OR OUT-OF-RANGE RESULT.  
FE A CONSTANT CANNOT BE CONVERTED. CHECK CONSTANT FOR PROPER CONSTRUCT.  
FE A CONSTANT DO PARAMETER MUST BE BETWEEN 0 AND 131K  
FE A CONSTANT MAY NOT BE FOLLOWED BY AN EQUAL SIGN, NAME, OR ANOTHER CONSTANT.  
FE A CONSTANT OPERAND OF A REAL OPERATION IS OUT OF RANGE OR INDEFINITE.  
FE A DO LOOP MAY NOT TERMINATE ON A FORMAT STATEMENT  
FE A DO LOOP MAY NOT TERMINATE ON THIS TYPE OF STATEMENT  
FE A DO PARAMETER MUST BE A POSITIVE INTEGER CONSTANT OR AN INTEGER VARIABLE.  
FE A FUNCTION REFERENCE REQUIRES AN ARGUMENT LIST.  
FE A NAME MAY NOT BE FOLLOWED BY A CONSTANT.

FE A PREVIOUS STATEMENT MAKES AN ILLEGAL TRANSFER TO THIS LABEL  
FE A PREVIOUSLY MENTIONED ADJUSTABLE SUBSCRIPT IS NOT TYPE INTEGER.  
FE A REFERENCE TO THIS ARITHMETIC STATEMENT FUNCTION HAS UNBALANCED PARENTHESIS WITHIN THE PARAMETER LIST  
FE A REFERENCE TO THIS ASF HAS A PARAMETER MISSING  
FE A VARIABLE DIMENSION OR THE ARRAY NAME WITH A VARIABLE DIMENSION IS NOT A FORMAL PARAMETER  
FE ALL ECS VARIABLES MUST APPEAR IN AN ECS COMMON BLOCK.  
FE ALL ELEMENTS IN AN ECS COMMON BLOCK MUST BE TYPE ECS.  
FE AN ARRAY REFERENCE HAS TOO MANY SUBSCRIPTS.  
FE APPEARED WHERE A VARIABLE SHOULD HAVE  
FE ARGUMENT NOT FOLLOWED BY COMMA OR RIGHT PARENTHESIS.  
FE ARRAY OR COMMON VARIABLE MAY NOT BE DECLARED EXTERNAL  
FE ASF HAS MORE DUMMY PARAMETERS THAN ALLOWED  
FE BAD SUBSCRIPT IN EQUIV STMT  
FE BAD SYNTAX ENCOUNTERED.  
FE BASIC EXTERNAL OR INTRINSIC FUNCTION CALLED WITH WRONG TYPE ARGUMENT  
FE BASIC OR INTRINSIC FUNCTION WITH AN INCORRECT ARGUMENT COUNT  
FE COMMON BLOCK NAME NOT ENCLOSED IN SLASHES  
FE COMMON VARIABLE IS FORMAL PARAMETER OR PREVIOUSLY DECLARED IN COMMON OR ILLEGAL NAME.  
FE COMMON-EQUIVALENCE ERROR  
FE CONSTANT DATA ITEM MUST BE FOLLOWED BY A , / OR RIGHT PAREN  
FE CONSTANT SUBSCRIPT VALUE EXCEEDS ARRAY DIMENSIONS  
FE CONSTANT TABLE CONSTORS OVERFLOWED-STATEMENT TRUNCATED.ENLARGE TABLE OR SIMPLIFY STATEMENT  
FE DATA ITEM LISTS MAY ONLY BE NESTED 1 DEEP  
FE DATA VARIABLE LIST SYNTAX ERROR  
FE DEBUG EXECUTION OPTION SUPPRESSED DUE TO NATURE OF ABOVE FATAL ERRORS  
FE DECLARATIVE STATEMENT OUT OF SEQUENCE  
FE DEFECTIVE HOLLERITH CONSTANT. CHECK FOR CHARACTER COUNT ERROR OR LOST CONTINUATION CARD.



FE DIVISION BY CONSTANT ZERO.  
FE DO LIMIT OR REP FACTOR MUST BE AN INTEGER OR OCTAL CONSTANT BETWEEN 1 AND 131K  
FE DO LOOPS TERMINATING ON THIS LABEL ARE IMPROPERLY NESTED  
FE DOUBLY DEFINED FORMAL PARAMETER  
FE DUMMY PARAMETER IN ASF DEFINITION OCCURED TWICE  
FE DUPLICATE LOOP INDEX OR DOESNT MATCH ANY SUBSCRIPT VARIABLE  
FE DUPLICATE STATEMENT LABEL  
FE ECS COMMON BLOCK MUST BE LABELED.  
FE ECS REFERENCE MUST BE A STAND-ALONE ARGUMENT.  
FE ECS VARIABLE MAY NOT APPEAR IN AN EQUIV STMT  
FE ENTRY POINT NAMES MUST BE UNIQUE - THIS ONE HAS BEEN PREVIOUSLY USED IN THIS SUBPROGRAM  
FE ENTRY STATEMENT MAY NOT APPEAR IN A PROGRAM  
FE ENTRY STATEMENT MAY NOT BE LABELED  
FE ENTRY STATEMENTS MAY NOT OCCUR WITHIN THE RANGE OF A DO STATEMENT  
FE EQUATED FILENAME NOT PREVIOUSLY DEFINED  
FE EXPRESSION TRANSLATOR TABLE (ARLIST) OVERFLOWED. SIMPLIFY THE EXPRESSION.  
FE EXPRESSION TRANSLATOR TABLE (FRSTB ) OVERFLOWED. SIMPLIFY THE EXPRESSION.  
FE EXPRESSION TRANSLATOR TABLE (OPSTAK) OVERFLOWED. SIMPLIFY THE EXPRESSION.  
FE F.P. WITH VARIABLE DIMENSIONS NOT ALLOWED IN A NAMELIST STATEMENT  
FE FIELD WIDTH IS GREATER THAN 150 CHARACTERS. FORMAT ERROR SCAN WILL CONTINUE.  
FE FILENAME IS GREATER THAN 6 CHARACTERS  
FE FILENAME PREVIOUSLY DEFINED  
FE FOLLOWED BY AN ILLEGAL ITEM  
FE FORMAL PARAMETERS MAY NOT APPEAR IN COMMON OR EQUIV STMTS  
FE FORMAT REFERENCE MUST BE A LEGAL STATEMENT NUMBER OR AN ARRAY REFERENCE.  
FE FORMAT STATEMENT ENDS BEFORE END OF HOLLERITH STRING. ERROR SCANNING STOPS HERE.  
FE FORMAT STATEMENT ENDS BEFORE LAST HOLLERITH COUNT IS COMPLETE. ERROR SCAN FOR THIS FORMAT STOPS AT H .

FE FUNCTION NAME DOES NOT APPEAR AS A VARIABLE IN THIS SUBPROGRAM  
FE GO TO STATEMENT - SYNTAX ERROR  
FE GROUP NAME NOT SURROUNDED BY SLASHS  
FE GROUP NAME PREVIOUSLY REFERENCED IN ANOTHER CONTEXT  
FE HEADER CARD NOT FIRST STATEMENT  
FE HEADER CARD SYNTAX ERROR  
FE I/O LIST SYNTAX ERROR  
FE ILLEGAL BLOCK NAME  
FE ILLEGAL CALL FORMAT  
FE ILLEGAL CHARACTER FOLLOWS PRECEDING A,I,L,O,OR R DESCRIPTOR. ERROR SCAN FOR THIS FORMAT STOPS HERE.  
FE ILLEGAL CHARACTER FOLLOWS PRECEDING FLOATING PT DESCRIPTOR. ERROR SCAN FOR THIS FORMAT STOPS HERE.  
FE ILLEGAL CHARACTER FOLLOWS PRECEDING SIGN CHARACTER. ERROR SCANNING FOR THIS FORMAT STOPS HERE.  
FE ILLEGAL CHARACTER FOLLOWS TAB SETTING DESIGNATOR. ERROR SCAN FOR THIS FORMAT STOPS HERE.  
FE ILLEGAL CHARACTER. THE REMAINDER OF THIS STATEMENT WILL NOT BE COMPILED.  
FE ILLEGAL EXTENSION OF COMMON BLOCK ORIGIN  
FE ILLEGAL FORM INVOLVING THE USE OF A COMMA.  
FE ILLEGAL INPUT/OUTPUT ADDRESS.  
FE ILLEGAL LABEL FIELD IN THIS STATEMENT  
FE ILLEGAL LABELS IN IF STATEMENT.  
FE ILLEGAL NAMELIST VARIABLE  
FE ILLEGAL RETURNS PARAMETER.  
FE ILLEGAL SEPARATOR BETWEEN VARIABLES  
FE ILLEGAL SEPARATOR ENCOUNTERED.  
FE ILLEGAL SEPARATOR IN EXTERNAL STATEMENT  
FE ILLEGAL USE OF THE EQUAL SIGN.  
FE ILLEGAL VARIABLE NAME FIELD IN ASSIGN OR ASSIGNED GOTO

FE IMPROPER FORM OF ENTRY STATEMENT. ONLY ALLOWABLE FORM IS [ ENTRY NAME ]  
FE INTRINSIC FUNCTION REFERENCE MAY NOT USE A FUNCTION NAME AS AN ARGUMENT  
FE INVOLVED IN CONTRADICTIONARY EQUIVALENCING  
FE IS THE FIRST LINE NO. OF A LOOP THAT IS ENTERED FROM OUTSIDE ITS RANGE AND HAS NO EXITS  
FE LEFT SIDE OF REPLACEMENT STATEMENT IS ILLEGAL.  
FE LOGICAL AND NON-LOGICAL OPERANDS MAY NOT BE MIXED  
FE LOGICAL EXPRESSION IN 3-BRANCH IF STATEMENT.  
FE LOGICAL OPERAND USED WITH NON-LOGICAL OPERATORS.  
FE LOOPS ARE NESTED MORE THAN 50 DEEP  
FE MAXIMUM PARENTHESIS NESTING LEVEL EXCEEDED. ERROR SCAN FOR THIS FORMAT STOPS HERE.  
FE MAY NOT BE FUNCTION, EXTERNAL, F.P. OR IN BLANK COMMON  
FE MISSING OR SYNTAX ERROR IN LIST OF TRANSFER LABELS  
FE MORE THAN ONE RELATIONAL OPERATOR IN A RELATIONAL EXPRESSION.  
FE MORE THAN 50 FILES ON PROGRAM CARD OR 63 PARAMETERS ON A SUBROUTINE OR FUNCTION CARD  
FE MORE THAN 63 ARGUMENTS IN ARGUMENT LIST.  
FE NAMELIST STATEMENT SYNTAX ERROR  
FE NO MATCHING LEFT PARENTHESIS.  
FE NO MATCHING RIGHT PARENTHESIS IN ARGUMENT LIST.  
FE NO MATCHING RIGHT PARENTHESIS IN SUBSCRIPT.  
FE NO MATCHING RIGHT PARENTHESIS.  
FE NO TERMINATING RIGHT PARENTHESIS IN OVERLAY, SEGMENT, SEGZERO OR SECTION CARD  
FE NON DIMENSIONED NAME APPEARS FOLLOWED BY LEFT PAREN  
FE NON-STANDARD RETURN STATEMENT MAY NOT APPEAR IN A FUNCTION SUBPROGRAM  
FE NUMBER OF CHARACTERS IN AN ENCODE/DECODE STATEMENT MUST BE AN INTEGER CONSTANT OR VARIABLE  
FE NUMBER OF SUBSCRIPTS IS INCOMPATIBLE WITH THE NUMBER OF DIMENSIONS DURING EQUIVALENCING  
FE ONLY ONE ECS COMMON BLOCK MAY BE DECLARED  
FE ONLY ONE SYMBOLIC NAME IN EQUIVALENCE GROUP

FE PARAMETER ON NON-STANDARD RETURN STATEMENT IS NOT A RETURNS FORMAL PARAMETER  
FE PRECEDING CHARACTER ILLEGAL AT THIS POINT IN CHARACTER STRING. ERROR SCAN FOR THIS FORMAT STOPS HERE.  
FE PRECEDING CHARACTER ILLEGAL. SCALE FACTOR EXPECTED. ERROR SCANNING FOR THIS FORMAT STOPS HERE.  
FE PRECEDING HOLLERITH COUNT IS EQUAL TO ZERO. ERROR SCANNING FOR THIS FORMAT STOPS HERE.  
FE PRECEDING HOLLERITH INDICATOR IS NOT PRECEDED BY A COUNT. SCANNING STOPS HERE.  
FE PRESENT USE OF THIS LABEL CONFLICTS WITH PREVIOUS USES  
FE REFERENCED LABEL IS MORE THAN FIVE CHARACTERS  
FE RETURN STATEMENT APPEARS IN MAIN PROGRAM  
FE RETURNS LIST ERROR  
FE RETURNS OR EXTERNAL NAMES MAY NOT APPEAR IN DECLARATIVE STATEMENTS  
FE RIGHT PARENTHESIS FOLLOWED BY A NAME, CONSTANT, OR LEFT PARENTHESIS.  
FE SIMPLE VARIABLE OR CONSTANT FOLLOWED BY LEFT PARENTHESIS.  
FE STATEMENT TOO LONG  
FE SUBPROGRAM NAME MAY NOT BE REFERENCED IN A DECLARATIVE STATEMENT  
FE SUBROUTINE NAME REFERRED TO BY CALL IS USED ELSEWHERE AS A NON-SUBROUTINE NAME.  
FE SYMBOLIC NAME HAS TOO MANY CHARACTERS  
FE SYNTAX ERROR IN ASF DEFINITION  
FE SYNTAX ERROR IN DATA ITEM LIST  
FE SYNTAX ERROR IN DATA STATEMENT  
FE SYNTAX ERROR IN EQUIVALENCE STATEMENT  
FE SYNTAX ERROR IN IMPLIED DO NEST  
FE SYNTAX ERROR IN SUBSCRIPT LIST, MUST BE OF FORM  $CON1*IVAR+CON2$   
FE TAB SETTING MAY NOT EXCEED 150. FORMAT ERROR SCAN WILL CONTINUE.  
FE THE CONTROL VARIABLE OF A DO OR DO IMPLIED LOOP MUST BE A SIMPLE INTEGER VARIABLE  
FE THE EXPRESSION IN A LOGICAL IF IS NOT TYPE LOGICAL  
FE THE FIELD FOLLOWING STOP OR PAUSE MUST BE 5 OR LESS OCTAL DIGITS

FE THE OPERATOR INDICATED (.NOT. OR A RELATIONAL) MUST BE FOLLOWED BY A CONSTANT, NAME, LEFT PAREN., - OR +.

FE THE OPERATOR INDICATED (-,+,*,/, OR **) MUST BE FOLLOWED BY A CONSTANT, NAME, OR LEFT PARENTHESIS.

FE THE STATEMENT IN A LOGICAL IF MAY BE ANY EXECUTABLE STATEMENT OTHER THAN A DO OR ANOTHER LOGICAL IF

FE THE SYNTAX OF DO PARAMETERS MUST BE I=M1,M2,M3 OR I=M1,M2

FE THE TERMINAL STATEMENT OF THIS DO PRECEDES IT

FE THE TYPE OF THIS IDENTIFIER IS NOT LEGAL FOR ANY EXPRESSION

FE THE VALUE OF THE PARITY INDICATOR IN A BUFFER I/O STATEMENT MUST BE 0 OR 1

FE THIS ASSIGN STATEMENT HAS IMPROPER FORMAT, ONLY ALLOWABLE IS [ASSIGN LABEL TO VARIABLE ]

FE THIS NAME MAY NOT BE USED IN A DATA STMT

FE THIS STATEMENT MAKES AN ILLEGAL TRANSFER INTO A PREVIOUS DO LOOP

FE THIS STATEMENT TYPE IS ILLEGAL IN BLOCK DATA SUBPROGRAM

FE TOO MANY LABELED COMMON BLOCKS, ONLY 125 BLOCKS ARE ALLOWED.

FE TOO MANY SUBSCRIPTS IN ARRAY REFERENCE.

FE TOTAL RECORD LENGTH IS GREATER THAN 150 CHARACTERS. FORMAT ERROR SCAN WILL CONTINUE.

FE UNDEFINED STATEMENT NUMBERS, SEE BELOW

FE UNIT NUMBER MUST BE BETWEEN 1 AND 99 INCLUSIVE.

FE UNIT NUMBER OR PARITY INDICATOR MUST BE AN INTEGER CONSTANT OR VARIABLE

FE UNMATCHED PARAMETER COUNT IN A REFERENCE TO THIS STATEMENT FUNCTION

FE UNMATCHED PARENTHESIS

FE UNRECOGNIZED STATEMENT

FE USE OF THIS PROGRAM OR SUBROUTINE NAME IN AN EXPRESSION.

FE VALUE OF ARRAY SUBSCRIPT IS .LT. 1 OR .GT. DIMENSIONALITY IN IMPLIED DO NEST

FE VARIABLE HAS MORE THAN THREE SUBSCRIPTS

FE VARIABLE IN ASSIGN OR ASSIGNED GO TO IS ILLEGAL

FE VARIABLE SUBSCRIPTS MAY NOT APPEAR WITHOUT DO LOOPS

FE VARIABLE WITH ILLEGAL SUBSCRIPTS

FE WAS LAST CHARACTER SEEN AFTER TROUBLE, REMAINDER OF STATEMENT IGNORED

FE ZERO LEVEL RIGHT PARENTHESIS MISSING. SCANNING STOPS.  
FE ZERO STATEMENT LABELS ARE ILLEGAL  
I A HOLLERITH CONSTANT IS AN OPERAND OF AN ARITHMETIC OPERATOR.  
I ARRAY NAME OPERAND NOT SUBSCRIBED, FIRST ELEMENT WILL BE USED  
I ARRAY REFERENCE OUTSIDE DIMENSION BOUNDS  
I ASSUMED PROGRAM NAME WHEN NO HEADER STATEMENT APPEARS  
I COMMA MISSING BEFORE VARIABLE INDICATED.  
I CONSTANT LENGTH .GT. VARIABLE LENGTH, CONSTANT TRUNCATED  
I DATA ITEM LIST EXCEEDS VARIABLE LIST, EXCESS CONSTANTS IGNORED  
I DATA VARIABLE LIST EXCEEDS ITEM LIST, EXCESS VARIABLES NOT INITIALIZED  
I DIMENSIONAL RANGE IS EXTENDED FOR EQUIVALENCING PURPOSES  
I DUE TO THE NUMEROUS ERRORS NOTED, ONLY THOSE WHICH ARE FATAL TO EXECUTION WILL BE LISTED BEYOND THIS POINT  
I EW.D OR DW.D DESCRIPTOR BAD FOR OUTPUT, W SHOULD SATISFY W-7 .GE. D  
I FIELD WIDTH IS GREATER THAN 137 CHARACTERS. IT MAY EXCEED THE I/O DEVICE CAPACITY.  
I FILE LENGTH REQUESTED IS TOO LARGE. STANDARD LENGTH OF 20JOB SUBSTITUTED.  
I FLOATING POINT DESCRIPTOR EXPECTS DECIMAL POINT SPECIFIED. OUTPUT WILL INCLUDE NO FRACTIONAL PARTS.  
I FLOATING POINT SPECIFICATION EXPECTS DECIMAL DIGITS TO BE SPECIFIED. ZERO DECIMAL DIGITS ASSUMED.  
I MASK ARGUMENT OUT OF RANGE. A MASK OF 0 OR 63 WILL BE SUBSTITUTED FOR ARGUMENT  
I MAY NOT USED IN A DEBUG STATEMENT  
I MISSING I/O LIST OR SPURIOUS COMMA  
I MORE STORAGE REQUIRED BY DO STATEMENT PROCESSOR FOR OPTIMIZATION  
I NO END CARD, END LINE ASSUMED  
I NON-BLANK CHARACTERS FOLLOW ZERO-LEVEL RIGHT PARENTHESIS. THESE CHARACTERS WILL BE IGNORED.  
I NUMBER OF DIGITS IN CONSTANT EXCEED POSSIBLE SIGNIFICANCE. HIGH ORDER DIGITS RETAINED IF POSSIBLE.  
I NUMERIC FIELD FOLLOWING TAB SETTING DESIGNATOR IS EQUAL TO ZERO, COLUMN ONE IS ASSUMED.  
I NUMERIC FIELD OMITTED IN PRECEDING SCALE FACTOR. ZERO SCALE ASSUMED.  
I PRECEDING FIELD WIDTH IS ZERO.

I PRECEDING FIELD WIDTH SHOULD BE 7 OR MORE.  
I PRECEDING SCALE FACTOR IS OUTSIDE LIMITS OF REPRESENTATION WITHIN THE MACHINE.  
I PRESENT USE IN CONTEXT OF THIS NAME DOES NOT MATCH PREVIOUS OCCURANCES IN DEBUG STMTS  
I PREVIOUSLY DIMENSIONED VARIABLE, FIRST DIMENSIONS WILL BE RETAINED  
I PREVIOUSLY TYPED VARIABLE, FIRST ENCOUNTERED TYPE IS RETAINED  
I REPEAT COUNT FOR PRECEDING FIELD DESCRIPTOR IS ZERO.  
I SEPARATOR MISSING. SEPARATOR ASSUMED HERE.  
I SINGLE WORD CONSTANT MATCHED WITH DOUBLE OR COMPLEX VARIABLE. PRECISION LOST.  
I SUPERFLUOUS SCALE FACTOR ENCOUNTERED PRECEDING CURRENT SCALE FACTOR.  
I TAB SETTING MAY EXCEED RECORD SIZE, DEPENDING ON USE.  
I THE CONSTANT LOWER LIMIT IS GREATER THAN THE CONSTANT UPPER LIMIT OF A DO  
I THE NUMBER OF ARGUMENTS IN A SUBROUTINE ARGUMENT LIST IS INCONSISTENT.  
I THE NUMBER OF ARGUMENTS IN THE ARGUMENT LIST OF A NON-BASIC EXTERNAL FUNCTION IS INCONSISTENT.  
I THE VARIABLE UPPER LIMIT AND THE CONTROL VARIABLE OF THIS DO ARE THE SAME PRODUCING A NON-TERMINATING LOOP  
I THERE IS NO PATH TO THIS STATEMENT  
I THIS IF DEGENERATES INTO A SIMPLE TRANSFER TO THE LABEL INDICATED.  
I THIS STATEMENT BRANCHES TO ITSELF.  
I THIS STATEMENT REDEFINES A CURRENT LOOP CONTROL VARIABLE OR PARAMETER  
I TOTAL RECORD LENGTH IS GREATER THAN 137 CHARACTERS. IT MAY EXCEED THE I/O DEVICE CAPACITY.  
I X-FIELD PRECEDED BY A BLANK. 1X ASSUMED  
I X-FIELD PRECEDED BY A ZERO. NO SPACING OCCURS

## CROSS REFERENCE MAP

C

---

The cross reference map is a dictionary of all programmer created symbols appearing in a program unit, with the properties of each symbol and references to each symbol listed by source line number. The symbol names are grouped by class and listed alphabetically within the groups. The reference map begins on a separate page following the source listing of the program and the error dictionary.

The kind of reference map produced is determined by the R option on the control card:

- R = 0 No map
- R = 1 Short map (symbols, addresses, properties)
- R = 2 Long map (short map, references by line number and a DO-loop map)
- R = 3 Long map and printout of common block members and equivalence classes
- R Implies R = 2

If R is not specified the default option is R = 1 unless the L option equals 0; then R = 0.

Fatal errors in the source program will cause certain parts of the map to be suppressed, incomplete, or inaccurate. Fatal to execution (FE) and fatal to compilation (FC) errors will cause the DO-loop map to be suppressed, and assigned addresses will be different; symbol references may not be accumulated for statements containing syntax errors.

For the long map, it may be necessary to increase field length by 1000(octal).

The number of references that can be accumulated and sorted for mapping is: field length minus 20000 (octal) minus 4 times the number of symbols. For a source program containing 1000 (decimal) symbols, approximately 8000 (decimal) references can be accumulated with a field length of 50000 octal.

Examples from the cross-reference map produced by the program which follows are interspersed with the general format discussions.

The source program and the reference maps produced for both R = 1 and R = 3 follow. A complete set of maps for R = 2 is not included, but samples are shown with the discussion.



## SOURCE PROGRAM

### Main Program

PROGRAM	MAPS		
		PROGRAM MAPS	
		1(INPUT,OUTPUT,TAPES=INPUT,TAPE6=OUTPUT)	
		INTEGER SIZE1, S1, SIZE2, S2	
		EQUIVALENCE(SIZE1,S1),(SIZE2,S2)	
5		NAMelist/PARAMS/SIZE1,SIZE2	
		DATA S1,S2/12,12/	
		READ(5,PARAMS)	
		WRITE(6,PARAMS)	
		PRINT1	MAPS 01
10	1	FORMAT(#0SAMPLE PROGRAM TO ILLUSTRATE THE VARIOUS COMPILER MAPS.#)	MAPS 02
		CALL PASCAL(S1)	MAFS 03
		PRINT2	MAFS 04
	2	FORMAT(#0THE FOLLOWING WILL HAVE NO HEADINGS.#)	MAFS 05
		CALL NOHEAD(S2)	MAFS 06
15		STOP	MAFS 07
		END	MAFS 08

### Block Data Subprogram

BLOCK DATA			
		BLOCK DATA	MAFS 09
		COMMON/ARRAY/X(22)	MAFS 10
		INTEGER X	MAFS 11
		DATA X(22)/1/	MAFS 12
5		END	MAFS 13

### Subprogram with second entry

SUBROUTINE PASCAL			
		SUBROUTINE PASCAL(SIZE)	MAFS 14
		INTEGER L(22),SIZE	MAFS 15
		COMMON/ARRAY/L	MAFS 16
		PRINT4,(I,I=1,SIZE)	MAFS 17
5	4	FORMAT(44H0COMBINATIONS OF M THINGS TAKEN N AT A TIME.//20X,3H-N-/	MAFS 18
		\$22I6)	MAFS 19
		ENTRY NOHEAD	MAFS 20
		M=MIN0(21,MAX0(2,SIZE-1))	MAFS 21
		DO2I=1,M	MAFS 22
10		K=22-I	MAFS 23
		L(K)=1	MAFS 24
		DO1J=K,21	MAFS 25
	1	L(J)=L(J)+L(J+1)	MAFS 26
	2	PRINT3,(L(J),J=K,22)	MAFS 27
15	3	FORMAT(22I6)	MAFS 28
		RETURN	MAFS 29
		END	MAFS 30

7/8/9 in column 1.

### Namelist data

```
$PARAMS
    SIZE2 = 7,
$END
6/7/8/9 in column 1.
```

# R=1 MAPS

```

PROGRAM      MAPS
SYMBOLIC REFERENCE MAP

ENTRY POINTS
+052 MAPS

VARIABLES   SN  TYPE      RELOCATION
+125 SIZE1  INTEGER
+125 S1     INTEGER      4126 SIZE2  INTEGER
                               4126 S2     INTEGER

FILE NAMES   MODE
0 INPUT      2022 OUTPUT  FMT          0 TAPES  FMT      2022 TAPE6  FMT

EXTERNALS    TYPE  ARGS
NOHEAD      1          PASCAL          1

NAMELISTS
PARAMS

STATEMENT LABELS
4110 1      FMT          +117 2      FMT

STATISTICS
PROGRAM LENGTH 55B      53
BUFFER LENGTH  4044B   2044
  
```

```

BLOCK DATA
SYMBOLIC REFERENCE MAP

VARIABLES   SN  TYPE      RELOCATION
0 X         INTEGER  APRAY  ARRAY

COMMON BLOCKS LENGTH
ARRAY      22

STATISTICS
PROGRAM LENGTH 0B      0
COMMON LENGTH  26B    22
  
```

```

SUBROUTINE PASCAL
SYMBOLIC REFERENCE MAP

ENTRY POINTS
27 NOHEAD      2 PASCAL

VARIABLES   SN  TYPE      RELOCATION
100 I       INTEGER
102 K       INTEGER      103 J       INTEGER
101 M       INTEGER      0 L        INTEGER  ARRAY  ARRAY
                               0 SIZE     INTEGER  F.P.

FILE NAMES   MODE
OUTPUT      FMT

INLINE FUNCTIONS TYPE  ARGS
MAX0       INTEGER  0 INTRIN      MIN0     INTEGER  0 INTRIN

STATEMENT LABELS
0 1        FMT          0 2
65 4      FMT          76 3      FMT

COMMON BLOCKS LENGTH
ARRAY      22

STATISTICS
PROGRAM LENGTH 106B    70
COMMON LENGTH  26B    22
  
```

# R=3 MAPS

```

PROGRAM      MAPS
SYMBOLIC REFERENCE MAP
ENTRY POINTS  DEF LINE  REFERENCES
4052  MAPS      1

VARIABLES    SN  TYPE      RELOCATION      REFS      3      4      5
4125  SIZE1    INTEGER
4126  SIZE2    INTEGER      REFS      3      4      5
4125  S1       INTEGER      REFS      3      4      11  DEFINED  6
4126  S2       INTEGER      REFS      3      4      14  DEFINED  6

FILE NAMES   MODE
0  INPUT
2022 OUTPUT  FMT      WRITES      9      12
0  TAPES     FMT      READS       7
2022 TAPE6   FMT      WRITES      8

EXTERNALS    TYPE  ARGS      REFERENCES
NOHEAD
PASCAL      1      11

NAMELISTS    DEF LINE  REFERENCES
PARAMS      5      7      8

STATEMENT LABELS  DEF LINE  REFERENCES
4110 1      FMT      10      9
4117 2      FMT      13      12

EQUI CLASSES  LENGTH  MEMBERS -- BIAS NAME(LENGTH)
SIZE1        1      0 S1      (1)
SIZE2        1      0 S2      (1)

```

← missing for R=2 map

```

STATISTICS
PROGRAM LENGTH  65B  53
BUFFER LENGTH  40448  2084

```

```

BLOCK DATA
SYMBOLIC REFERENCE MAP
VARIABLES    SN  TYPE      RELOCATION      REFS      2      3  DEFINED
0  X         INTEGER  ARRAY  ARPAY

COMMON BLOCKS  LENGTH  MEMBERS -- BIAS NAME(LENGTH)
ARRAY          22      0 X      (22)

```

← missing for R=2 map

```

STATISTICS
PROGRAM LENGTH  0B  0
COMMON LENGTH  26B  22

```

```

SUBROUTINE  PASCAL
SYMBOLIC REFERENCE MAP
ENTRY POINTS  DEF LINE  REFERENCES
27  NOHEAD    7      16
2  PASCAL     1

VARIABLES    SN  TYPE      RELOCATION      REFS      4      10  DEFINED  4      9
100 I        INTEGER
103 J        INTEGER      REFS      3*13    14  DEFINED  12      14
102 K        INTEGER      REFS      11      12      14  DEFINED  10
0  L         INTEGER  ARRAY  ARRAY      REFS      2      3      2*13    14  DEFINED  11      13
101 M        INTEGER      REFS      9      DEFINED  8
0  SIZE      INTEGER      F.P.     REFS      2      4      8  DEFINED  1

FILE NAMES   MODE
OUTPUT      FMT      WRITES      4      14

INLINE FUNCTIONS  TYPE  ARGS      DEF LINE  REFERENCES
MAX0            INTEGER  0  INTRIN  8
MIN0            INTEGER  0  INTRIN  8

STATEMENT LABELS  DEF LINE  REFERENCES
0  1              13      12
0  2              14      9
76 3             FMT      15      14
65 4             FMT      5       4

LOOPS LABEL  INDEX  FROM-TO  LENGTH  PROPERTIES
20          * I    4         5B     EXT REFS
44          * I    9 14       20B    EXT REFS NOT INNER
51          * J    12 13      2B     INSTACK

COMMON BLOCKS  LENGTH  MEMBERS -- BIAS NAME(LENGTH)
ARRAY          22      0 X      (22)

```

← missing for R=2 map

```

STATISTICS
PROGRAM LENGTH  106B  70
COMMON LENGTH  26B  22

```

# OUTPUT

\$PARANS

SIZE1 = 12

SIZE2 = 7

\$END

SAMPLE PROGRAM TO ILLUSTRATE THE VARIOUS COMPILER MAPS.

COMBINATIONS OF M THINGS TAKEN N AT A TIME.

	-N-											
1	2	3	4	5	6	7	8	9	10	11	12	
2	1											
3	3	1										
4	6	4	1									
5	10	10	5	1								
6	15	20	15	6	1							
7	21	35	35	21	7	1						
8	28	56	70	56	28	8	1					
9	36	84	126	126	84	36	9	1				
10	45	120	210	252	210	120	45	10	1			
11	55	165	330	462	462	330	165	55	11	1		
12	66	220	495	792	924	792	495	220	66	12	1	

THE FOLLOWING WILL HAVE NO HEADINGS.

2	1										
3	3	1									
4	6	4	1								
5	10	10	5	1							
6	15	20	15	6	1						
7	21	35	35	21	7	1					

## General Format

Each class of symbol is preceded by a subtitle line that specifies the class and the properties listed.

Formats for each symbol class are different, but printouts contain the following information:

The octal address associated with each symbol relative to the origin of the program unit.

Properties associated with the symbol

List of references to the symbol for R=2 and R=3

All line numbers in the reference list refer to the line of the statement in which the reference occurs. Multiple references in a statement are printed as n*1 where n is the number of references on line 1.

All numbers to the right of the name are decimal integers unless they are suffixed with B to indicate octal.

Names of symbols generated by the compiler (such as system library routines called for input/output) do not appear in the reference map.

## ENTRY POINTS

Entry point names include program and subprogram names and names appearing in ENTRY statements. The format of this map is:

	ENTRY POINTS	DEFINITION	REFERENCES
	<b>addr</b> <b>name</b>	<b>def</b>	<b>ref</b>
<b>addr</b>	Relative address assigned to the entry point.		
<b>name</b>	Entry point name as defined in FORTRAN source.		
<b>def</b>	Line number on which entry point <b>name</b> is defined (PROGRAM statement, SUBROUTINE statement, ENTRY statement, etc.). (Not on R=1 maps.)		
<b>ref</b>	In subprograms only, line number of RETURN statements. (Not on R=1 maps.)		

R=1:

```
ENTRY POINTS
 27 NOHEAD      2 PASCAL
```

R=2 and R=3:

```
ENTRY POINTS   DEF LINE   REFERENCES
 27 NOHEAD      7         16
 2 PASCAL       1
```

## VARIABLES

Variable names include local and COMMON variables and arrays, formal parameters, RETURNS names, and for FUNCTION subprograms, the defined function name when used as a variable. The format of this map is:

	VARIABLES	SN	TYPE	RELOCATION
	addr	name	* type	prop block refs
<b>addr</b>	Relative address assigned to variable <b>name</b> . If <b>name</b> is a member of a COMMON block, <b>addr</b> is relative to the start of <b>block</b> .			
<b>name</b>	Variable name as it appears in FORTRAN source listing. Variables are listed in alphabetical order.			
<b>*</b>	SN = stray name flag. (No entry appears under SN when R=1 is specified.) Variable names which appear only once in a subprogram are indicated by * under the SN headline. Such variable names are likely keypunch errors, misspellings, etc. In the long map, DO loops where the index variable is not referenced will cause the index variable to be flagged as a (legal) stray name.			
<b>type</b>	LOGICAL, INTEGER, REAL, COMPLEX, DOUBLE, or ECS. Gives the arithmetic mode associated with the variable <b>name</b> . RETURNS appears if <b>name</b> is a RETURNS formal parameter.			
<b>prop</b>	Properties associated with variable <b>name</b> are printed by keywords in this column: *UNDEF Variable <b>name</b> has not been defined. A variable is defined if any of the following conditions hold: <b>name</b> appears in a COMMON or DATA statement. is EQUIVALENCED to a variable that is defined. appears on the left side of an assignment statement at the outermost parenthesis level. is the index variable in a DO loop. appears as a stand alone actual parameter in a subroutine or function call. appears in an input list (READ, BUFFERIN, etc.). Otherwise, the variable is considered undefined. However, variables which are used (in arithmetic expressions, etc.) before they are defined (by an assignment statement or subprogram call) are not flagged. ARRAY Variable <b>name</b> is dimensioned. *UNUSED <b>name</b> is an unused formal parameter.			
<b>block</b>	Name of COMMON block in which variable <b>name</b> appears. If blank, <b>name</b> is a local variable. // indicates <b>name</b> is in blank COMMON. F.P. indicates <b>name</b> is a formal parameter.			

**refs**

(Does not appear in short map, R=1.)

References and definitions associated with variable **name** are listed by line number, beginning with the following in-line subheadings:

**REFS** All appearances of **name** in declarative statements or statements where the value of **name** is used.

**DEFINED** All appearances of **name** where its value may be altered such as in **DATA**, **ASSIGN**, **READ**, **ENCODE**, or **DECODE**, **BUFFER IN**, assignment statements, or as a **DO** loop index.

**IO REFS** All appearances of **name** in use as a variable file name in I/O statements.

R=1: This map form uses a double column format to conserve space. Headings appear only on the first columns.

VARIABLES	SN	TYPE	RELOCATION				
100	I	INTEGER		103	J	INTEGER	
102	K	INTEGER		0	L	INTEGER	ARRAY
101	M	INTEGER		0	SIZE	INTEGER	ARRAY F.P.

R=2 and R=3:

VARIABLES	SN	TYPE	RELOCATION	REFS		DEFINED				
100	I	INTEGER		REFS	4	10	DEFINED	4		9
103	J	INTEGER		REFS	3*13	14	DEFINED	12		14
102	K	INTEGER		REFS	11	12	14	DEFINED		10
0	L	INTEGER	ARRAY	REFS	2	3	2*13	14	DEFINED	
101	M	INTEGER		REFS	9	DEFINED	8			11
0	SIZE	INTEGER	F.P.	REFS	2	4	8	DEFINED		1

## FILE NAMES

File names include those explicitly defined in the PROGRAM header card as well as those implicitly defined (in subprograms) through usage in I/O statements. The format of this map is:

	FILE NAMES	MODE
	<b>addr name</b>	<b>mode refs</b>
<b>addr</b>	Relative address of the file environment table (FET) associated with the file <b>name</b> . The file's buffer starts at <b>addr+21B</b> . This column appears only in main programs (where the file is actually defined). In subprograms, this column is blank.	
<b>name</b>	Name of the file as defined in PROGRAM statement or implied from usage in I/O statements. For example, in a subprogram, WRITE(2) implies a reference to file TAPE2.	
<b>mode</b>	Indicates the mode of the file, as implied from its usage. One of the following will be printed:	
	FMT	Formatted I/O e.g. READ(2,901)
	UNFMT	Unformatted I/O READ(2)
	BUF	Buffer I/O BUFFER IN(2,0)
	MIXED	Some combination of the above.
	blank	Mode cannot be determined.
<b>refs</b>	(Does not appear in short map, R=1.) References are divided into three categories by in-line subheadings:	
	READS	followed by list of line numbers referencing file <b>name</b> in input operations.
	WRITES	line numbers of output operations on file <b>name</b> .
	MOTION	line numbers of positioning operations (REWIND, BACKSPACE, ENDFILE) on file <b>name</b> .

R=1:

```

FILE NAMES      MODE
 0  INPUT          2022  OUTPUT      FMT          0  TAPE5      FMT          2022  TAPE6      FMT

```

R=2 and R=3:

```

FILE NAMES      MODE
 0  INPUT          2022  OUTPUT      FMT          WRITES      9          12
 0  TAPE5          02022  TAPE6      FMT          READS       7
 02022  TAPE6      FMT          WRITES      8

```

When a variable is used as a unit number in an I/O statement the following message is printed:

```

..... VARIABLE USED AS FILE NAMES, SEE ABOVE .....

```



## EXTERNAL REFERENCES

External references include names of functions or subroutines called explicitly from a program or subprogram, as well as names declared in an EXTERNAL statement. Implicit external references, such as those called by certain FORTRAN source statements (READ, ENCODE, etc.) are not listed. The format of this map is:

	EXTERNALS	TYPE	ARGS		REFERENCES
	name	type	args	prop	refs
<b>name</b>	Name defined EXTERNAL as it appears in source listing.				
<b>type</b>	Applies to externals used as functions. Possible keywords are: REAL, INTEGER, COMPLEX, DOUBLE, LOGICAL Gives the arithmetic mode of external function. NO TYPE    No specific arithmetic mode defined. Applies to certain library functions listed as externals in T mode. (T mode is implied when OPT=0 or D mode is selected.) This column will be blank for all externals used as subroutines in CALL statements.				
<b>args</b>	Number of arguments in call to external <b>name</b> .				
<b>prop</b>	Special properties associated with external <b>name</b> : F.P <b>name</b> is a formal parameter (applies only for references within a program). LIBRARY <b>name</b> is a library function called by value. In T compile modes, no LIBRARY entries appear since all references to library functions (SIN, COS, etc.) will be by name. (OPT=0 or D mode automatically implies T mode.)				
<b>refs</b>	Line number on which <b>name</b> is referenced. (Does not appear in short map, R=1.)				

R=1:

EXTERNALS	TYPE	ARGS		REFERENCES
NOHEAD		1	PASCAL	1

R=2 and R=3:

EXTERNALS	TYPE	ARGS	REFERENCES
NOHEAD		1	14
PASCAL		1	11

## INLINE FUNCTIONS

Inline functions include names of intrinsic and statement functions appearing in the subprogram. The subtitle line is:

	INLINE FUNCTIONS	TYPE	ARGS	DEF	LINE	REFERENCES
	name	mode	args	ftype	def	refs
<b>name</b>	Symbol name as it appears in the listing.					
<b>mode</b>	Arithmetic mode, NO TYPE means no conversion in mixed mode expressions.					
<b>args</b>	Number of arguments with which the function is referenced.					
<b>ftype</b>	INTRIN      Intrinsic function. SF            Statement function.					
<b>def</b>	Blank for intrinsic functions; the definition line for statement functions.					
<b>refs</b>	Lines on which function is referenced.					

R=1:

```

  INLINE FUNCTIONS  TYPE  ARGS
  MAX0             INTEGER 0 INTRIN
  MIN0             INTEGER 0 INTRIN
  
```

R=2 and R=3:

```

  INLINE FUNCTIONS  TYPE  ARGS  DEF LINE  REFERENCES
  MAX0             INTEGER 0 INTRIN      8
  MIN0             INTEGER 0 INTRIN      8
  
```

## NAMELISTS

	NAMELISTS	DEF LINE	REFERENCES
	name	def	refs
<b>name</b>	Namelist group name as defined in FORTRAN source.		
<b>def</b>	Line on which namelist is defined.		
<b>refs</b>	Line numbers of references to <b>name</b> .		

} (Does not appear in short map.)

R=1:

```

  NAMELISTS
  PARAMS
  
```

R=2 and R=3:

```

  NAMELISTS  DEF LINE  REFERENCES
  PARAMS    5         7         8
  
```

## STATEMENT LABELS

The statement label map includes all statement labels defined in the program or subprogram. The format of this map is:

	STATEMENT LABELS	DEF LINE	REFERENCE
	<b>addr</b> <b>label</b> <b>type</b>	<b>act</b> <b>def</b>	<b>refs</b>
<b>addr</b>	Relative address assigned to statement <b>label</b> . Inactive labels will have <b>addr</b> zero.		
<b>label</b>	Statement label from FORTRAN source. Statement labels are listed in numerical order.		
<b>type</b>	One of the following keywords:		
	FMT	Statement <b>label</b> is a FORMAT statement.	
	UNDEF	Statement <b>label</b> is undefined. <b>refs</b> will list all references to this undefined label.	
	blank	Statement <b>label</b> appears on a valid executable statement.	
<b>act</b>	One of the following keywords:		
	INACTIVE	<b>label</b> is considered inactive. It may have been deleted by optimization. Terminal statements of a DO loop are inactive unless referenced as the object of a transfer of control. Inactive labels will have <b>addr</b> zero.	
	NO REFS	<b>label</b> is not referenced by any statements. This label may be removed safely from the FORTRAN source.	
	blank	<b>label</b> is active or referenced.	
<b>def</b>	Line number on which <b>label</b> was defined. (Does not appear in short map.)		
<b>refs</b>	Line numbers on which <b>label</b> was referenced. (Does not appear in short map.)		

R=1:

STATEMENT LABELS			
0	1	0	2
65	4	FMT	76 3 FMT

R=2 and R=3:

STATEMENT LABELS		DEF LINE	REFERENCES
0	1	13	12
0	2	14	9
76	3	15	14
65	4	5	4
		FMT	
		FMT	

## DO-LOOPS

The DO-loop map includes all DO loops as well as implied DO loops not in DATA statements that appear in the program and lists their properties. This map is generated only in the long map (R=2 and R=3). Loops are listed in order of appearance in the program. The format of this map is:

	LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES	
	<b>fwa</b>	<b>term</b>	<b>mf</b>	<b>index</b>	<b>first-last</b>	<b>len</b>	<b>prop</b>
<b>fwa</b>							Relative address assigned to the start of loop body.
<b>term</b>							Statement label defined as end of loop, or blank for implied DO-loops in I/O statements.
<b>mf</b>			*				Indicates <b>index</b> is materialized (value of <b>index</b> in memory is the current value of loop count).
			blank				Indicates <b>index</b> is not materialized ( <b>index</b> is not used directly and is updated in a register only; value in memory will not correspond to current loop count).
<b>index</b>							Variable name used as control index for loop, as defined by DO statement.
<b>first-last</b>							Line numbers of the <b>first</b> and <b>last</b> statements of the loop.
<b>len</b>							Number of computer words generated for the body of the loop (octal).
<b>prop</b>							Various keyword prints are possible, describing optimization properties of the loop:
						OPT	Loop has been optimized.
						INSTACK	Loop fits into instruction stack (7 words or less, 6600 only [†] ).
						EXT REFS	Loop not optimized because it contains references to an external subprogram, or it is the implied loop of an I/O statement.
						ENTRIES	Loop not optimized because it contains entries from outside its range.
						NOT INNER	Loop not optimized because it is not the innermost loop in a nest.
						EXITS	Loop not optimized because it contains references to statement labels outside its range.

R=2 and R=3:

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES
20		* I	4	5B	EXT REFS
44	2	* I	9 14	20B	EXT REFS NOT INNER
51	1	J	12 13	2B	INSTACK

[†]Loops that fit in the 6600 instruction stack have a maximum length of 7 words and usually run two to three times as fast as a comparable loop that does not fit into the stack.

## COMMON BLOCKS

The common block map lists common blocks and their members as defined in the source program. The format of this map is:

	COMMON BLOCKS	LENGTH	MEMBER – BIAS NAME(LENGTH)
	<b>block</b>	<b>blen</b>	<b>bias member (size)</b>
<b>block</b>	Common block name as defined in COMMON statement. // represents blank common.		
<b>blen</b>	Total length of <b>block</b> in decimal.		

If the long map is specified (R=3) the following details are printed for each member of each block:

<b>bias</b>	Relative position of <b>member</b> in <b>block</b> ; in decimal, gives the distance from the block origin.
<b>member</b>	Variable name defined as a member of <b>block</b> .
<b>size</b>	Number of words allocated for <b>member</b> .

Only variables defined as members of a common block explicitly by a COMMON statement are listed in this map. Variables which become implicit members of a common block by EQUIVALENCE statements are listed in the EQUIV CLASS map and the variable map.

R=1 and R=2:

```
COMMON BLOCKS  LENGTH
  ARRAY        22
```

R=3:

```
COMMON BLOCKS  LENGTH  MEMBERS - BIAS NAME(LENGTH)
  ARRAY        22      0 L      (22)
```

## EQUIVALENCE CLASSES

This map appears only when R=3 is selected. All members of an equivalence class of variables explicitly equated in EQUIVALENCE statements are listed. Variables added through linkage to common blocks are not included. The format of the map is:

	EQUIV CLASSES	LENGTH	MEMBERS – BIAS NAME (LENGTH)
	<b>cbase</b> <b>base</b>	<b>clen</b>	<b>bias member (size)</b>
<b>cbase</b>	Common base. A variable name appears here if the equivalence class is in a common block. In such a case, <b>cbase</b> is the variable name of the first member in that common block. *ERROR* Indicates this class is in error because more than one member is in common or the origin of the block is extended by equivalence.		
<b>base</b>	If the class is local (not in a common block), <b>base</b> is the name of the variable with the lowest address. If the class is in a common block, <b>base</b> is the name of the variable in that common block to which other variables were linked through an EQUIVALENCE statement.		
<b>clen</b>	Number of words allocated for <b>base</b> , (considered the class length).		
<b>bias</b>	Position of <b>member</b> relative to <b>base</b> ; <b>bias</b> is in decimal.		
<b>member</b>	Variable name defined as a member of an equivalence class. (Members having the same <b>bias</b> which are associated with the same <b>base</b> and thus occupy the same locations.)		
<b>size</b>	Size of <b>member</b> as defined by DIMENSION, etc.		

R=3 only:

EQUIV CLASSES	LENGTH	MEMBERS – BIAS NAME(LENGTH)
SIZE1	1	0 S1 (1)
SIZE2	1	0 S2 (1)

## PROGRAM STATISTICS

At the end of the reference map, the statistics are printed in octal and decimal. The format is:

### STATISTICS

<b>PROGRAM LENGTH</b>	Length of program including code, storage for local variables, arrays, constants, temporaries, etc., but excluding buffers and common blocks.
<b>BUFFER LENGTH</b>	Total space occupied by I/O buffers and FETs.
<b>COMMON LENGTH</b>	Total length of common, excluding blank common.
<b>BLANK COMMON</b>	Length of blank common.

R=1, R=2, and R=3:

<b>STATISTICS</b>		
<b>PROGRAM LENGTH</b>	<b>1068</b>	<b>70</b>
<b>COMMON LENGTH</b>	<b>268</b>	<b>22</b>

## ERROR MESSAGES

The following error messages are printed if sufficient storage is not available:

**CANT SORT THE SYMBOL TABLE      INCREASE FL BY NNNB**

or

**REFERENCES AFTER LINE NNN LOST INCREASE FL BY NNNB**

## DEBUGGING (Using the Reference Map)

New Program:

The reference map can be used to find names that have been punched incorrectly as well as other items that will not show up as compilation errors. The basic technique consists of using the compiler as a verifier and correcting the FE errors until the program compiles.

Using the listing, the R=3 reference map, and the original flowcharts, the following information should be checked by the programmer:

Names incorrectly punched

Stray name flag in the variable map

Functions that should be arrays

Functions that should be inline instead of external

Variables or functions with incorrect type

Unreferenced format statements

Unused formal parameters

Ordering of members in common blocks

Equivalence classes

**Existing Program:**

The reference map can be used to understand the structure of an existing program. Questions concerning the loop structure, external references, common blocks, arrays, equivalence classes, input/output operations, and so forth, can be answered by checking the reference map.



# LIBRARY SUBPROGRAMS

D

Intrinsic Function & No. of Arguments	Definition	Example	Symbolic Name	Type of	
				Argument	Function
Absolute value (1)	a	Y=ABS(X)	ABS	Real	Real
		J=IABS(I)	IABS	Integer	Integer
		DOUBLE A, B B=DABS(A)	DABS	Double	Double
Truncation (1)	sign of a times largest integer $\leq  a $	Y=AINT(X)	AINT	Real	Real
		I=INT(X)	INT	Real	Integer
		DOUBLE Z J=IDINT(Z)	IDINT	Double	Integer
Modulo	MOD or AMOD ( $a_1, a_2$ ) is defined to be $a_1 - \text{trunc}(a_1/a_2) * a_2$	B=AMOD(A1, A2)	AMOD	Real	Real
		J=MOD(I1, I2)	MOD	Integer	Integer
		DM=DMOD(D1, D2)	DMOD	Double	Double
Choosing largest value ( $\geq 2$ )	Max ( $a_1, a_2, \dots$ )	X=AMAX0(I, J, K)	AMAX0	Integer	Real
		A=AMAX1(X, Y, Z)	AMAX1	Real	Real
		L=MAX0(I, J, K, N)	MAX0	Integer	Integer
		I=MAX1(A, B)	MAX1	Real	Integer
		DOUBLE W, X, Y, Z W=DMAX1(X, Y, Z)	DMAX1	Double	Double
Choosing smallest value ( $\geq 2$ )	Min ( $a_1, a_2, \dots$ )	Y=AMIN0(I, J)	AMIN0	Integer	Real
		Z=AMIN1(X, Y)	AMIN1	Real	Real
		L=MIN0(I, J, K)	MIN0	Integer	Integer
		J=MIN1(X, Y)	MIN1	Real	Integer
		DOUBLE A, B, C C=DMIN1(A, B)	DMIN1	Double	Double
Float (1)	Conversion from integer to real	XI=FLOAT(I)	FLOAT	Integer	Real
Fix (1)	Conversion from real to integer - same as INT	IY=IFIX(Y)	IFIX	Real	Integer

<u>Intrinsic Function &amp; No. of Arguments</u>	<u>Definition</u>	<u>Example</u>	<u>Symbolic Name</u>	<u>Type of Argument Function</u>	
Transfer of sign (2)	Sign of $a_2$ times $ a_1 $	$Z=\text{SIGN}(X, Y)$	SIGN	Real	Real
		$J=\text{ISIGN}(I1, I2)$	ISIGN	Integer	Integer
			DSIGN	Double	Double
Positive difference (2)	$a_1 - \text{Min}(a_1, a_2)$	$Z=\text{DIM}(X, Y)$	DIM	Real	Real
		$J=\text{IDIM}(I1, I2)$	IDIM	Integer	Integer
Truncate to obtain most significant part of double precision argument (1)		$\text{DOUBLE } Y$ $X=\text{SNGL}(Y)$	SNGL	Double	Real
Obtain real part of complex argument (1)		$\text{COMPLEX } A$ $B=\text{REAL}(A)$	REAL	Complex	Real
Obtain imaginary part of complex argument (1)		$D=\text{AIMAG}(A)$	AIMAG	Complex	Real
Express single precision argument in double precision form (1)		$\text{DOUBLE } Y$ $Y=\text{DBLE}(X)$	DBLE	Real	Double
Express two real arguments in complex form (2)	$a_1 + a_2 \sqrt{-1}$	$\text{COMPLEX } C$ $C=\text{CMPLX}(A1, A2)$	CMPLX	Real	Complex
Obtain conjugate of a complex argument (1)		$\text{COMPLEX } X, Y$ $Y=\text{CONJG}(X)$	CONJG	Complex	Complex
Shift (2)	Shift $a_1$ by $a_2$ bit positions: left circular if $a_2$ is positive; right with sign extension if $a_2$ is negative	$B=\text{SHIFT}(A, I)$	SHIFT	$a_1$ : Single word $a_2$ : Integer	Octal
Logical product (2)	$a_1 \wedge a_2$	$C=\text{AND}(A1, A2)$	AND	Single word	Octal
Logical sum (2)	$a_1 \vee a_2$	$D=\text{OR}(A1, A2)$	OR	Single word	Octal
Complement (1)	$\neg a$	$B=\text{COMPL}(A)$	COMPL	Single word	Octal
Masking (1)	Generate a left justified bit mask of $a_1$ one bits	$J=\text{MASK}(I)$	MASK	Integer	Octal

External Function & No. of Arguments	Definition	Example	Symbolic Name	Type of		
				Argument	Function	
Exponential (1)	$e^a$	Z=EXP(Y)	EXP	Real	Real	
		DOUBLE X, Y Y=DEXP(X)	DEXP	Double	Double	
		COMPLEX A, B B=CEXP(A)	CEXP	Complex	Complex	
Natural logarithm (1)	$\log_e(a)$	Z=ALOG(Y)	ALOG	Real	Real	
		Y=DLOG(X)	DLOG	Double	Double	
		B=CLOG(A)	CLOG	Complex	Complex	
Common Logarithm (1)	$\log_{10}(a)$	B=ALOG10(A)	ALOG10	Real	Real	
		DD=DLOG10(D)	DLOG10	Double	Double	
Trigonometric sine (1)	$\sin(a)$	Y=SIN(X)	SIN	Real	Real	
		DS=DSIN(D)	DSIN	Double	Double	
		CS=CSIN(C)	CSIN	Complex	Complex	
Trigonometric cosine (1)	$\cos(a)$	X=COS(Y)	COS	Real	Real	
		DC=DCOS(D)	DCOS	Double	Double	
		CC=CCOS(C)	CCOS	Complex	Complex	
Hyperbolic tangent (1)	$\tanh(a)$	B=TANH(A)	TANH	Real	Real	
Square root (1)	$(a)^{1/2}$	Y=SQRT(X)	SQRT	Real	Real	
		DY=DSQRT(DX)	DSQRT	Double	Double	
		CY=CSQRT(CX)	CSQRT	Complex	Complex	
Arctangent (1)	arctan(a)	Y=ATAN(X)	ATAN	Real	Real	
		DY=DATAN(DX)	DATAN	Double	Double	
	(2)	arctan(a ₁ /a ₂ )	B=ATAN2(A1, A2)	ATAN2	Real	Real
			D=DATAN2(D1, D2)	DATAN2	Double	Double
Modulus (1)	$\sqrt{\text{AIMAG}^2(a)+\text{REAL}^2(a)}$	CM=CABS(CX)	CABS	Complex	Real	
Arccosine (1)	arccos(a)	X=ACOS(Y)	ACOS	Real	Real	

External Functions & No. of Arguments	Definition	Example	Symbolic Name	Type of	
				Argument	Function
Arcsine (1)	arcsin (a)	X=ASIN(Y)	ASIN	Real	Real
Trigonometric tangent (1)	tan (a)	Y=TAN(X)	TAN	Real	Real
Random number generator (1)	ranf (a) returns values uniformly distributed over the range [0,1)	X=RANF(DUM)	RANF	Dummy	Real
Address of argument a (1)	loc (a)	P=LOCF(X)	LOCF	Symbolic	Integer
I/O status on buffer unit (1)	= -1 unit ready; no error = 0 EOF on last operation = +1 parity error	IO=UNIT(6)	UNIT	Integer	Real
I/O status on non- buffer unit (1)	= 0 no EOF in previous read	IFL=EOF(4)	EOF	Integer	Real
Length (1)	Number of central memory words read on previous buffered input request	L=LENGTH(J)	LENGTH	Integer	Integer
Variable character- istic (1)	-1 = indefinite +1 = out of range 0 = Normal	LEN=LEGVAR(V)	LEGVAR †	Real	Integer
Parity status on non-buffer unit (1)	0 = no parity error on previous read	IP=IOCHEC(5)	IOCHEC	Integer	Integer
Date as returned by 6000 SCOPE is bMM.DD.YY. (1) 7000 SCOPE is bYY.MM.DD. (1)	date(a)	WHEN=DATE(D)	DATE ††	Hollerith	Hollerith
Current reading of system clock as returned by SCOPE is ΔHH.MM.SS. (1)	time(a)	CLTIM=TIME(A)	TIME ††	Hollerith	Hollerith
Time in seconds (1)	second(a) (accumu- lated CP time)	CLTM=SECOND(A)	SECOND ††	Real	Real

† To use this routine with 7600, the program must be run with a MODE(0) card.

†† May be used as functions or subroutines; value is returned via argument and normal function.

<u>Subroutine &amp; No. of Arguments</u>	<u>Definition</u>	<u>Example</u>	<u>Symbolic Name</u>	<u>Type of Argument</u>
Set Sense Light (1)	$1 \leq i \leq 6$ turn sense light $i$ on. $i = 0$ turn off all sense lights.	CALL SLITE(I)	SLITE	Integer
Test Sense Light (2)	If sense light $i$ is on $j = 1$ . If off $j = 2$ Always turn sense light $i$ off	CALL SLITET(I,J)	SLITET	Integer
Test Sense Switch (2)	If sense switch $i$ is on (down) $j = 1$ . If sense switch $i$ is off (up) $j = 2$ .	CALL SSWTCH(I,J)	SSWTCH	Integer
Terminate (0)	Terminate program execution and return control to the monitor	CALL EXIT	EXIT	
Console Comment (1)	Place a message of up to 80 characters on dayfile†	CALL REMARK(2HHI)	REMARK	Hollerith
Console Value (2)	Display up to a 10 character message and value in the dayfile†	CALL DISPLA (2HX=,20.2)	DISPLA	$a_1$ =Hollerith $a_2$ =real or integer
Obtain current generative value of RANF between 0 and 1 (1)	ranget (a)	CALL RANGET(X)	RANGET	Symbolic
Initialize generative value of RANF (1)	ranset (a), the generic value is set to the nearest odd number $\geq a$	CALL RANSET(X)	RANSET	Real
Dump memory (3-60)	dump(a,b,f) dump A to B according to f	CALL DUMP(A,B,1) CALL PDUMP(X,Y,0)	DUMP PDUMP	Logical Integer Real Double Complex
Input checking (2)	ERRSET (a;b), set maximum number of errors, b, allowed in input data before fatal termination. Error count kept in a.	CALL ERRSET(A,B)	ERRSET	Symbolic Integer

†Characters with a display code value above 57_g are not allowed. The message must be terminated with binary zeros, even if an entire word is necessary. (Use of a Hollerith constant of any form will provide such a termination automatically.)

---

Subprograms written in COMPASS may be intermixed with FORTRAN coded subprograms in the source deck. COMPASS subprograms must begin with a card containing the word IDENT in columns 11-15, and terminate with card containing the word END in columns 11-13. Columns 1-10 of the IDENT and END cards must be blank; column 14 of the END card must be blank.

## Calling Sequence

When the FORTRAN compiler encounters a reference to an external subprogram, subroutine, or function the following calling sequence is generated:

SA1     Argument list (if parameters appear)  
RJ     Subprogram name

where the argument list consists of consecutive words of the form:

VFD 60/argument₁

followed by a zero word.

## Control Return

The COMPASS subprogram must restore the initial contents of A0 in A0 upon returning control to the calling subprogram. When the COMPASS subprogram is entered via a function reference, the result of that function must be in X6 or X6 and X7 with the least significant or imaginary part of the double precision or complex result appearing in X7.

## Example

The following page contains an example of a simple COMPASS Function and the calling FORTRAN main program. Since the function PF returns an integer value, it must be declared integer in the calling program. The argument to PF may be either real or integer.

Job Card

MAP(OFF)  
FTN(S=0) SYSTEXT IS NOT USED  
LGO.

7/8/9 in column 1.

```
PROGRAM PSAMP (OUTPUT)
INTEGER PF
X=SQRT(2.)
I=PF(X)
PRINT1,I
1 FORMAT(25H1THE PARITY OF SQRT(2) IS,I2)
STOP
END

IDENT PF
ENTRY PF
PF TITLE PF - COMPUTE PARITY OF WORD.
COMMENT COMPUTE PARITY OF WORD.
PF SPACE 4,11
*** PF - COMPUTE PARITY OF WORD.
*
* FORTRAN SOURCE CALL --
*
* PARITY = PF (ARG)
*
* RESULT = 1, IFF ARG HAS ODD NUMBER OF BITS SET.
* = 0, OTHERWISE.
**
** ENTRY (X1) = ADDRESS OF ARGUMENT.
* EXIT (X6) = RESULT.

PF EQ **1S17 ENTRY/EXIT...
SA2 X1
CX3 X2
MX0 -1
BX6 -X0*X3 ISOLATE LOWEST BIT
EQ PF EXIT..

END
```

6/7/8/9 in column 1.

# STATEMENT FORMS

F

<u>Statements</u>	<u>Classification</u>	<u>Page</u>
<u>Entry Points</u>		
PROGRAM s	N [†]	9-1
PROGRAM s ( $f_1, f_2, \dots, f_n$ )	N	9-1
SUBROUTINE s	N	9-2
SUBROUTINE s ( $a_1, a_2, \dots, a_n$ )	N	9-2
SUBROUTINE s, RETURNS ( $b_1, b_2, \dots, b_m$ )	N	9-2
SUBROUTINE s ( $a_1, a_2, \dots, a_n$ ), RETURNS ( $b_1, b_2, \dots, b_m$ )	N	9-2
FUNCTION f ( $a_1, a_2, \dots, a_n$ )	N	9-8
REAL FUNCTION f ( $a_1, a_2, \dots, a_n$ )	N	9-8
DOUBLE FUNCTION f ( $a_1, a_2, \dots, a_n$ )	N	9-8
COMPLEX FUNCTION f ( $a_1, a_2, \dots, a_n$ )	N	9-8
INTEGER FUNCTION f ( $a_1, a_2, \dots, a_n$ )	N	9-8
LOGICAL FUNCTION f ( $a_1, a_2, \dots, a_n$ )	N	9-8
DOUBLE PRECISION FUNCTION f ( $a_1, a_2, \dots, a_n$ )	N	9-8
ENTRY s	N	9-5
 <u>Specification Program Declaration</u>		
BLOCK DATA	N	9-10
BLOCK DATA d	N	9-10
 <u>Inter-subroutine</u>		
EXTERNAL $v_1, v_2, \dots, v_n$	NS	8-7
 <u>Inter-subroutine Transfer Statements</u>		
CALL s	E	4-12
CALL s ( $a_1, a_2, \dots, a_n$ )	E	4-12
CALL s, RETURNS ( $b_1, b_2, \dots, b_m$ )	E	4-12

[†] N=Non-executable, S=Specification, E=Executable.



<u>Statements (Cont'd)</u>	<u>Classification</u>	<u>Page</u>
CALL s ( $a_1, a_2, \dots, a_n$ ), RETURNS ( $b_1, b_2, \dots, b_m$ )	E	4-12
RETURN	E	4-14
RETURN a	E	4-14

### Data Declaration and Storage Allocation

#### Type Declaration

REAL $v_1, v_2, \dots, v_n$	NS	8-7
DOUBLE $v_1, v_2, \dots, v_n$	NS	8-7
COMPLEX $v_1, v_2, \dots, v_n$	NS	8-7
INTEGER $v_1, v_2, \dots, v_n$	NS	8-7
LOGICAL $v_1, v_2, \dots, v_n$	NS	8-7
DOUBLE PRECISION $v_1, v_2, \dots, v_n$	NS	8-7
ECS $v_1, v_2, \dots, v_n$	NS	8-7
TYPE REAL $v_1, v_2, \dots, v_n$	NS	8-7
TYPE DOUBLE $v_1, v_2, \dots, v_n$	NS	8-7
TYPE COMPLEX $v_1, v_2, \dots, v_n$	NS	8-7
TYPE INTEGER $v_1, v_2, \dots, v_n$	NS	8-7
TYPE LOGICAL $v_1, v_2, \dots, v_n$	NS	8-7
TYPE DOUBLE PRECISION $v_1, v_2, \dots, v_n$	NS	8-7
TYPE ECS $v_1, v_2, \dots, v_n$	NS	8-7

#### Storage Allocation

DIMENSION $v_1(i_1), v_2(i_2), \dots, v_n(i_n)$	NS	8-1
COMMON $/x_1/a_1/ \dots /x_n/a_n$	NS	8-3
EQUIVALENCE ( $k_1$ ), ( $k_2$ ), $\dots$ , ( $k_n$ )	NS	8-6
DATA $k_1/d_1/, k_2/d_2/, \dots, k_n/d_n/$	N	8-8
DATA ( $r_1=d_1$ ), ( $r_2=d_2$ ), $\dots$ , ( $r_n=d_n$ )	N	8-10

#### Statement Function

$f(a_1, a_2, \dots, a_n) = e$	E	9-7
-------------------------------	---	-----

<u>Symbol Manipulation and Control</u>	<u>Classification</u>	<u>Page</u>
Replacement Statements		
v=e { Arithmetic	E	3-1
Logical	E	3-3
Masking	E	3-3
Intra-program Transfers		
GO TO k	E	4-1
GO TO i, (k ₁ ,k ₂ ,...,k _n )	E	4-1
GO TO (k ₁ ,k ₂ ,...,k _n ), e	E	4-2
IF (e) k ₁ ,k ₂ ,k ₃	E	4-3
IF (e) k ₁ ,k ₂	E	4-4
IF (e) s	E	4-4
Loop Control		
DO n i = m ₁ ,m ₂ ,m ₃	E	4-5
<u>Miscellaneous Program Controls</u>		
ASSIGN k TO i	E	4-1
CONTINUE	E	4-12
PAUSE	E	4-15
PAUSE n	E	4-15
STOP	E	4-14
STOP n	E	4-14
<u>Input/Output</u>		
I/O Format		
FORMAT (q ₁ t ₁ z ₁ t ₂ z ₂ ...t _n z _n q ₂ )	N	6-1

<u>I/O Control Statements</u>	<u>Classification</u>	<u>Page</u>
READ f,k	E	5-3
READ (u) k	E	5-4
READ (u)	E	5-4
READ (u,f) k	E	5-2
READ (u,f)	E	5-2
WRITE (u) k	E	5-4
WRITE (u,f)	E	5-3
WRITE (u,f) k	E	5-3
PRINT f,k	E	5-4
PUNCH f,k	E	5-4
BUFFER IN (u,p) (A, B)	E	7-2, I-2
BUFFER OUT (u,p) (A, B)	E	7-2, I-2
 <u>Internal Manipulation</u>		
ENCODE (n,f,A) k	E	I-6, 7-3
DECODE (n,f,A) k	E	I-6, 7-3
 <u>Tape Handling</u>		
ENDFILE u	E	5-10
REWIND u	E	5-9, I-5
BACKSPACE u	E	5-9, I-5
 <u>Miscellaneous</u>		
NAMELIST /y ₁ /a ₁ /y ₂ /a ₂ /.../y _n /a _n	N	5-6
 <u>Program Termination</u>		
END	N	4-15

The SYSTEM routine handles error tracing, diagnostic printing, termination of output buffers, and transfer to specified non-standard error procedures. All the FORTRAN mathematical routines rely on SYSTEM to complete these tasks. Also a FORTRAN coded routine may call SYSTEM. Any of the parameters used by SYSTEM relating to a specific error may be changed by a user routine during execution. The END processor also makes use of SYSTEM to dump the output buffers and print an error summary. Since the initialization routine (Q8NTRY.), the end processors (END., STOP., and EXIT.), and SYSTEM must always be available, these routines are combined into one subprogram with multiple entry points.

The calling sequence to SYSTEM passes the error number as the first parameter and an error message as the second parameter. Several different messages may be associated with one error number. The error summary given at program termination lists the total number of times each error number was encountered.

The error number of zero is accepted as a special call to end the output buffers and return. If no OUTPUT file is defined before SYSTEM is called, no errors are printed and a message to this effect appears in the dayfile. Each printed line is subjected to the line limit of the OUTPUT buffer; when the limit is exceeded, the job is terminated.

The error table is ordered serially (the first error corresponds to error number 1) and it is expandable at assembly time. The last entry in the table is a catch-all for any error number which exceeds the table length. An entry in the error table appears as follows:

Print Frequency	Print Frequency Increment	Print Limit	Error Detection Total	F/ NF	A/ NA	Non-Standard Recovery Address
8	8	12	12	1	1	18

Print Frequency = PF

Print Frequency Increment = PFI

PF = 0 and PFI = 0, the diagnostic and traceback information are not listed.

PF = 0 and PFI = 1, the diagnostic and traceback information are listed until the print limit is reached.

PF = 0 and PFI = n, the diagnostic and traceback information are listed only the first n times unless the print limit is reached first.

PF = n, the diagnostic and traceback information are listed every nth time until the print limit is reached.

### Fatal (F)/ Non-Fatal (NF)

If the error is non-fatal and a non-standard recovery address is not specified, error messages are printed according to PRINT FREQUENCY and control is returned to the calling routine.

If the error is fatal and no non-standard recovery address is specified, error messages are printed according to PRINT FREQUENCY, an error summary is listed, all output buffers are terminated, and the job is terminated.

If a non-standard recovery address is specified, see Non-Standard Recovery.

### Non-Standard Recovery

SYSTEM supplies the non-standard recovery routine with the following information:

- A1 Address of parameter list passed to the routine which detected the error
- X1 Address of the first parameter
- A0 Address of parameter list of the routine that called the routine which detected the error
- B1 Address of a secondary parameter list, which contains, in successive words:
  - Error number passed in SYSTEM
  - Address of diagnostic word available to SYSTEM
  - Address within auxiliary table if A/NA bit is set, otherwise zero
  - Instruction consisting of RJ to SYSTEM in upper 30 bits and trace back information in lower 30 bits for the routine that called SYSTEM
- A2 Address of error table entry in SYSTEM
- X2 Contents of error table entry

Information in the secondary parameter list is not available to FORTRAN-coded routines.

### Non-Fatal Error

The routine which detected the error and SYSTEM are delinked from the calling chain and the non-standard recovery routine is entered. When this routine exits in the normal routine, control returns to the routine which called the routine which detected the error.

Thus, any faulty arguments may be corrected, and the recovery routine is allowed to call the routine which detected the error, providing corrected arguments. By not correcting the faulty arguments in the recovery routine, a three routine loop can develop between the routine which detects the error, SYSTEM, and the recovery routine. No checking is done for this case.

### Fatal Error

SYSTEM calls the non-standard recovery routine in the normal fashion, with the registers set as indicated above. If the non-standard recovery routine exits in the normal fashion returning control to SYSTEM, an error summary is listed, all output buffers are terminated, and the job is terminated.

### Use of the A/NA Bit

The A/NA bit is used only when a non-standard recovery address is specified.

If this bit is set, the address within an auxiliary table is passed in the third word of the secondary parameter list to the recovery routine. This bit allows more information than is normally supplied by SYSTEM to be passed to the recovery routine. The bit may be set only during assembly of SYSTEM, as an entry must also be made into the auxiliary table. Each word in the auxiliary table must have the error number in its upper 10 bits so that the address of the first error number match is passed to the recovery routine. An entry in the auxiliary table for an error is not limited to any specific number of words.

The traceback information is terminated as soon as one of the following conditions is detected:

- The calling routine is a program.
- The maximum traceback limit is reached.
- No traceback information is supplied.

To change an error table during execution, a FORTRAN type call is made to SYSTEMC with the following parameters:

#### Error number

List containing the consecutive locations:

- Word 1 Fatal/non-fatal (fatal = 1, non-fatal = 0)
- Word 2 Print frequency
- Word 3 Print frequency increment (only significant if word 2 = 0) special values:
  - word 2 = 0, word 3 = 0 never list error
  - word 2 = 0, word 3 = 1 always list error
  - word 2 = 0, word 3 = X list error only the first X times
- Word 4 Print limit
- Word 5 Non-standard recovery address
- Word 6 Maximum traceback limit

If any word within the parameter list is negative, the value already in table entry is not to be altered.

(Since auxiliary table bit may be set only during assembly of SYSTEM, only then can an auxiliary table entry be made.)

### Error Listing

Message supplied by calling routine:

ERROR NUMBER xxxx DETECTED BY zzzzzzz AT yyyyyy      zzzzzzz and cccccc are routine names, yyyyyy and wwwwww are relocatable addresses  
CALLED FROM cccccc AT ADDRESS wwwwww

or

CALLED FROM cccccc AT LINE dddd

(dddd is FORTRAN source line count)

#### ERROR SUMMARY

ERROR	TIMES
xxxxx	yyyy
:	:

(all numbers are decimal)

NO OUTPUT FILE FOUND

### Functions of Entry Points

Q8NTRY.	Initialize I/O buffer parameters
STOP.	Enter STOP in dayfile and begin END processing
EXIT.	Enter EXIT in dayfile and begin END processing
END.	Terminate all output buffers, print an error summary, transfer control to the main overlay if within an overlay; in any other case exit to monitor.
SYSTEM	Handles error tracing, diagnostic printing, termination of output buffers and either transfers to specified non-standard error recovery address, terminates the job or returns to calling routine depending on type of error.
SYSTEMC	Changes entry to SYSTEM's error table according to arguments passed.

Execution Diagnostics

<u>Routine</u>	<u>Message</u>	<u>Error No.</u>
ACGOER\$	ERROR IN COMPUTED GO TO STATEMENT: INDEX VALUE INVALID	1
ACOS\$	ABS(R).GT. 1.0 INFINITE ARGUMENT INDEF ARGUMENT	2
ALOG\$	ZERO ARGUMENT NEGATIVE ARGUMENT INFINITE ARGUMENT INDEF ARGUMENT	3
ALOG10\$	ZERO ARGUMENT NEGATIVE ARGUMENT INFINITE ARGUMENT INDEF ARGUMENT	4
ASIN\$	ABS(R).GT. 1.0 INFINITE ARGUMENT INDEF ARGUMENT	5
ATAN\$	INFINITE ARGUMENT INDEF ARGUMENT	6
ATAN2\$	X=Y=0.0 INFINITE OR INDEF ARGUMENT	7
CABS\$	FLOATING OVERFLOW INFINITE OR INDEF ARGUMENT	8
ZTOI\$	ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER INFINITE OR INDEF ARGUMENT	9
CCOS\$	INFINITE OR INDEF ARGUMENT ABS (REAL PART) TOO LARGE ABS (IMAG PART) TOO LARGE	10
CEXP\$	INFINITE OR INDEF ARGUMENT ABS (REAL PART) TOO LARGE ABS (IMAG PART) TOO LARGE	11
CLOG\$	ZERO ARGUMENT INFINITE OR INDEF ARGUMENT	12
COS\$	ARG TOO LARGE, ACCURACY LOST INFINITE ARGUMENT INDEF ARGUMENT	13



<u>Routine</u>	<u>Message</u>	<u>Error No.</u>
CSIN\$	INFINITE OR INDEF ARGUMENT ABS (REAL PART) TOO LARGE ABS (IMAG PART) TOO LARGE	14
CSQRT\$	INFINITE OR INDEF ARGUMENT	15
DABS\$	INFINITE ARGUMENT INDEF ARGUMENT	16
DATAN\$	INFINITE ARGUMENT INDEF ARGUMENT	17
DATAN2\$	X=Y=0.0 INFINITE OR INDEF ARGUMENT	18
DTOD\$	FLOATING OVERFLOW ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER NEGATIVE TO THE DBLE POWER INFINITE OR INDEF ARGUMENT	19
DTOI\$	ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER INFINITE OR INDEF ARGUMENT	20
DTOZ\$	FLOATING OVERFLOW IN D**REAL(Z) ZERO TO THE ZERO OR NEGATIVE POWER NEGATIVE TO THE COMPLEX POWER IMAG(Z)*LOG(D) TOO LARGE INFINITE OR INDEF ARGUMENT	21
DTOX\$	FLOATING OVERFLOW ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER NEGATIVE TO THE DBLE POWER INFINITE OR INDEF ARGUMENT	21
DCOS\$	ARG TOO LARGE, ACCURACY LOST INFINITE ARGUMENT INDEF ARGUMENT	22
DEXP\$	ARGUMENT TOO LARGE, FLOATING OVERFLOW INFINITE ARGUMENT INDEF ARGUMENT	23

<u>Routine</u>	<u>Message</u>	<u>Error No.</u>
DLOG\$	ZERO ARGUMENT NEGATIVE ARGUMENT INFINITE ARGUMENT INDEF ARGUMENT	24
DLOG10\$	ZERO ARGUMENT NEGATIVE ARGUMENT INFINITE ARGUMENT INDEF ARGUMENT	25
DMOD\$	DP INTEGER EXCEEDS 96 BITS 2ND ARGUMENT ZERO INFINITE OR INDEF ARGUMENT	26
DSIGN\$	INFINITE ARGUMENT INDEF ARGUMENT	27
DSIN\$	ARG TOO LARGE, ACCURACY LOST INFINITE ARGUMENT INDEF ARGUMENT	28
DSQRT\$	NEGATIVE ARGUMENT INFINITE ARGUMENT INDEF ARGUMENT	29
EXP\$	ARGUMENT TOO LARGE, FLOATING OVERFLOW INFINITE ARGUMENT INDEF ARGUMENT	30
IT0J\$	INTEGER OVERFLOW ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER	31
IDINT\$	INTEGER OVERFLOW INFINITE OR INDEF ARGUMENT	32
XTOD\$	FLOATING OVERFLOW ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER NEGATIVE TO THE DBLE POWER INFINITE OR INDEF ARGUMENT	33
XTOI\$	ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER INFINITE OR INDEF ARGUMENT	34

<u>Routine</u>	<u>Message</u>	<u>Error No.</u>
XTOY\$	FLOATING OVERFLOW ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER NEGATIVE TO THE REAL POWER INFINITE OR INDEF ARGUMENT	35
SIN\$	ARG TOO LARGE, ACCURACY LOST INFINITE ARGUMENT INDEF ARGUMENT	36
SLITE\$	ILLEGAL SENSE LITE NUMBER	37
SLITET\$	ILLEGAL SENSE LITE NUMBER	38
SQRT\$	NEGATIVE ARGUMENT INFINITE ARGUMENT INDEF ARGUMENT	39
SSWTC\$	ILLEGAL SENSE SWITCH NUMBER	40
TAN\$	ARG TOO LARGE, ACCURACY LOST INFINITE ARGUMENT INDEF ARGUMENT	41
TANH\$	INFINITE ARGUMENT INDEF ARGUMENT	42
ITOD\$	FLOATING OVERFLOW ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER NEGATIVE TO THE DBLE POWER INFINITE OR INDEF ARGUMENT	44
ITOX\$	FLOATING OVERFLOW ZERO TO THE ZERO POWER ZERO TO THE NEGATIVE POWER NEGATIVE TO THE REAL POWER INFINITE OR INDEF ARGUMENT	45
ITOZ\$	FLOATING OVERFLOW IN I**REAL(Z) ZERO TO THE ZERO OR NEGATIVE POWER NEGATIVE TO THE COMPLEX POWER IMAG(Z)*LOG(I) TOO LARGE INFINITE OR INDEF ARGUMENT	46

<u>Routine</u>	<u>Message</u>	<u>Error No.</u>
XTOZ\$	FLOATING OVERFLOW IN X**REAL(Z) ZERO TO THE ZERO OR NEGATIVE POWER NEGATIVE TO THE COMPLEX POWER IMAG(Z)*LOG(X) TOO LARGE INFINITE OR INDEF ARGUMENT	47
FTNERR\$	COMPILATION ERROR ENCOUNTERED DURING PROGRAM EXECUTION	48
INPUTN\$	TOO FEW CONSTANTS FOR UNSUBSCRIPTED ARRAY	49
OVERLA\$	FATAL ERROR IN LOADER	50
SEGMEN\$	FATAL ERROR IN LOADER	51
	NON-FATAL ERROR IN LOADER	52
BACKSP\$	UNASSIGNED MEDIUM, FILE NAME: xxxxxxxx	53
BUFFEI\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	54
	END-OF-FILE ENCOUNTERED, FILENAME: xxxxxxxx	55
	WRITE FOLLOWED BY READ ON FILE: xxxxxxxx	56
	BUFFER DESIGNATION BAD--FWA.GT.LWA	57
BUFFEO\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	58
	BUFFER SPECIFICATION BAD--FWA.GT.LWA	59
ENDFIL\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	60
IFENDF\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	61
INPUTB\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	62
	END-OF-FILE ENCOUNTERED, FILENAME xxxxxxxx	63
INPUTO\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	64
OUTPTN\$		
INPUTC\$	END-OF-FILE ENCOUNTERED, FILENAME: xxxxxxxx	65
INPUTN\$	PRECISION LOST IN FLOATING INTEGER CONSTANT NAMELIST DATA TERMINATED BY EOF, NOT \$ NAMELIST NAME NOT FOUND NO I/O MEDIUM ASSIGNED WRONG TYPE CONSTANT INCORRECT SUBSCRIPT TOO MANY CONSTANTS (, \$, OR = EXPECTED, MISSING VARIABLE NAME NOT FOUND BAD NUMERIC CONSTANT MISSING CONSTANT AFTER * UNCLEARED EOF ON A READ READ PARITY ERROR	66

<u>Routine</u>	<u>Message</u>	<u>Error No.</u>
INPUTS\$	*DECODE*CHAR/REC.GT. 150*	66
IOCHECK\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	67
KODER\$	*ILLEGAL FUNCTIONAL LETTER	68
	*IMPROPER PARENTHESIS NESTING	69
	*EXCEEDED RECORD SIZE	70
	*SPECIFIED FIELD WIDTH ZERO	71
	*FIELD WTH . LE. DECIMAL WTH	72
	*HOLLERITH FORMAT WITH LIST	73
KRAKER\$	*ILLEGAL FUNCTIONAL LETTER	74
	*IMPROPER PARENTHESIS NESTING	75
	*SPECIFIED FIELD WIDTH ZERO	76
	*EXCEEDED RECORD SIZE	77
	*ILLEGAL DATA IN FIELD * ↓ *	78
	*DATA OVERFLOW *>*	79
	*HOLLERITH FORMAT WITH LIST	80
LENGTH\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	81
FTNBIN\$ OUTPTB\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	82
OUTPTC\$ CONNEC\$	UNASSIGNED MEDIUM, FILENAME: xxxx	83
OUTPTN\$	OUTPUT FILE LINE LIMIT EXCEEDED	84
OUTPTS\$	ENCODE*CHAR/REC . GT. 150*	85
REWIM\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	86
KODER\$	*LIST/FMT CONFLICT, SNGL/DBLE	87
INPUTB\$	WRITE FOLLOWED BY READ ON FILE: xxxxxxxx	88
	LIST EXCEEDS DATA, FILENAME: xxxxxxxx	89
	PARITY ERROR READING (BINARY) FILE: xxxxxxxx	90
INPUTC\$	WRITE FOLLOWED BY READ ON FILE: xxxxxxxx	91
	PARITY ERROR READING (CODED) FILE: xxxxxxxx	92
OUTPTB\$	PARITY ERROR ON LAST READ ON FILE: xxxxxxxx	93
OUTPTC\$	PARITY ERROR ON LAST READ ON FILE: xxxxxxxx	94

<u>Routine</u>	<u>Message</u>	<u>Error No.</u>
IOCHEC\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	95
	*STATUS OF BUFFER I/O MUST BE CHECKED BY THE UNIT FUNCTION * FILENAME: xxxxxxxx	96
INITMS\$ READMS\$ WRITMS\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	97
INITMS\$	FILE DOES NOT RESIDE ON A RANDOM ACCESS DEVICE	98
READMS\$	FILE WAS NOT OPENED BY A CALL TO SUBROUTINE OPENMS	99
WRITEMS\$		
READMS\$	RECORD NAME REFERRED TO IN CALL IS NOT IN THE FILE INDEX	100
INITMS\$ WRITMS\$	INDEX BUFFER IS OF INSUFFICIENT LENGTH	101
LABEL\$	UNASSIGNED MEDIUM, FILENAME: xxxxxxxx	102
READMS\$	*READ PARITY ERROR*	102
READMS\$	SPECIFIED INDEX IN THIS MASS STORAGE CALL .GT. MASTER INDEX OR IS ZERO	110
WRITEC\$	ECS UNIT HAS LOST POWER OR IS IN MAINTENANCE MODE	112
READEC\$	ECS READ PARITY ERROR	113

The following table shows the general form of a FORTRAN program unit. Groups, when present must be ordered as shown. Statements within a group can appear in any order.

STATEMENTS			
OVERLAY			
PROGRAM*			
FUNCTION*			
SUBROUTINE*			
BLOCK DATA			
type			* F O R M A T
COMMON			
DIMENSION			
EQUIVALENCE			
EXTERNAL*			
Statement function*	N†	A	D
definitions			
ENTRY*	L	I	S
Executable statements*			
END			

* Not allowed in BLOCK DATA Subprograms

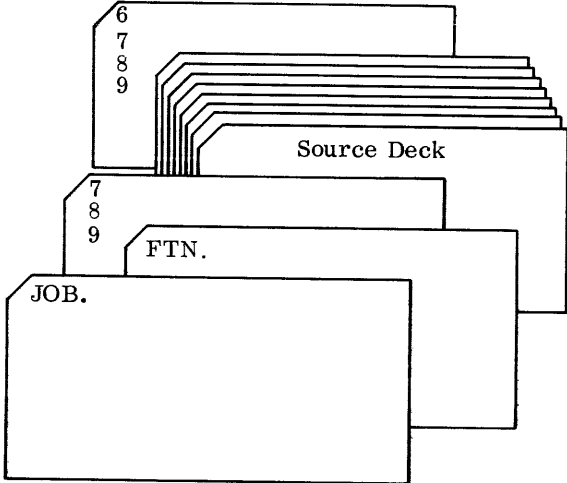
† Namelist group name must be defined before it is used

### Source Decks

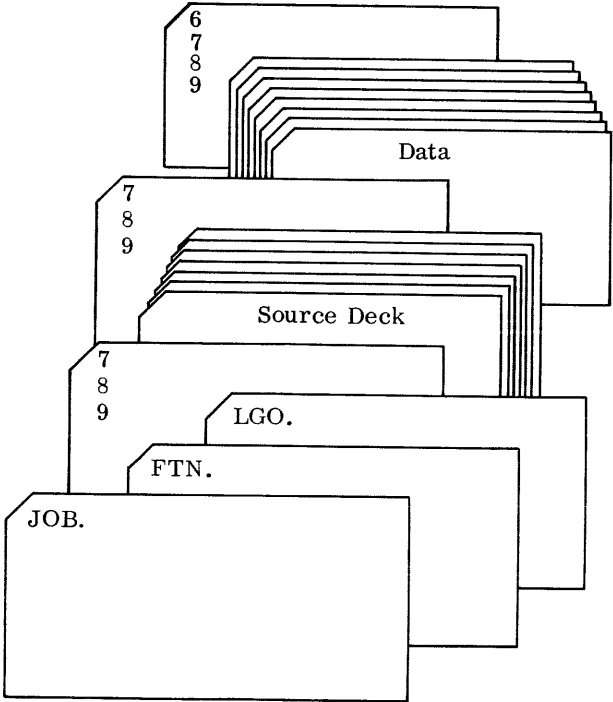
Source decks are comprised of complete FORTRAN program unit source decks and/or COMPASS source decks. Each COMPASS source deck must begin with an IDENT card (columns 11-15) and terminate with an END card (columns 11-13); in both cases columns 1-10 must be blank. FORTRAN and COMPASS program unit source decks may be in any order.

SAMPLE DECK STRUCTURE

1. Compilation only

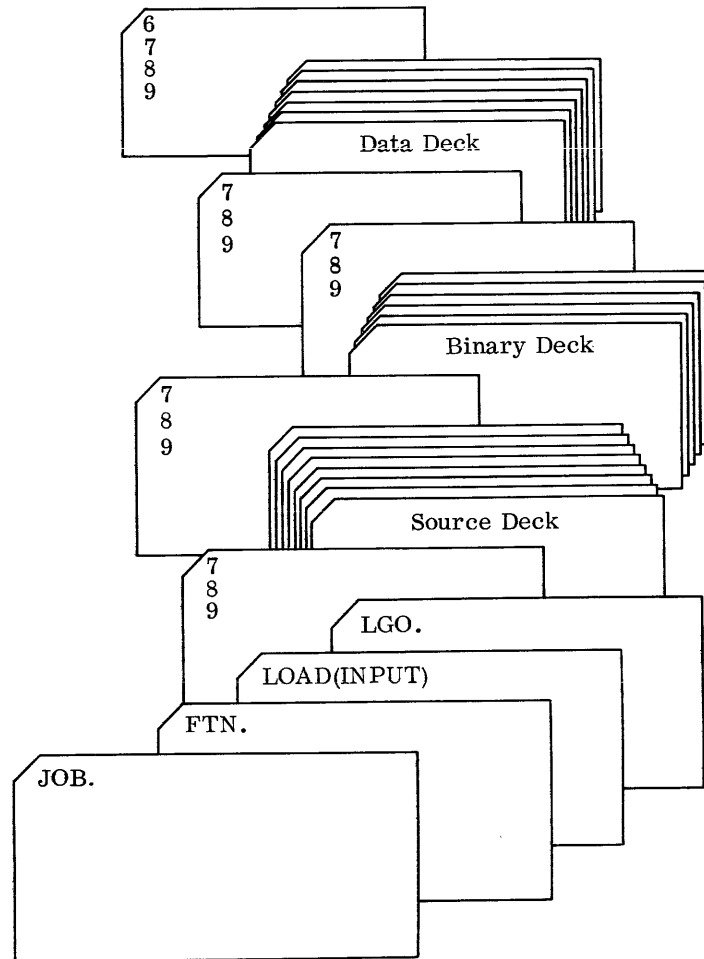


2. Compilation and Execution

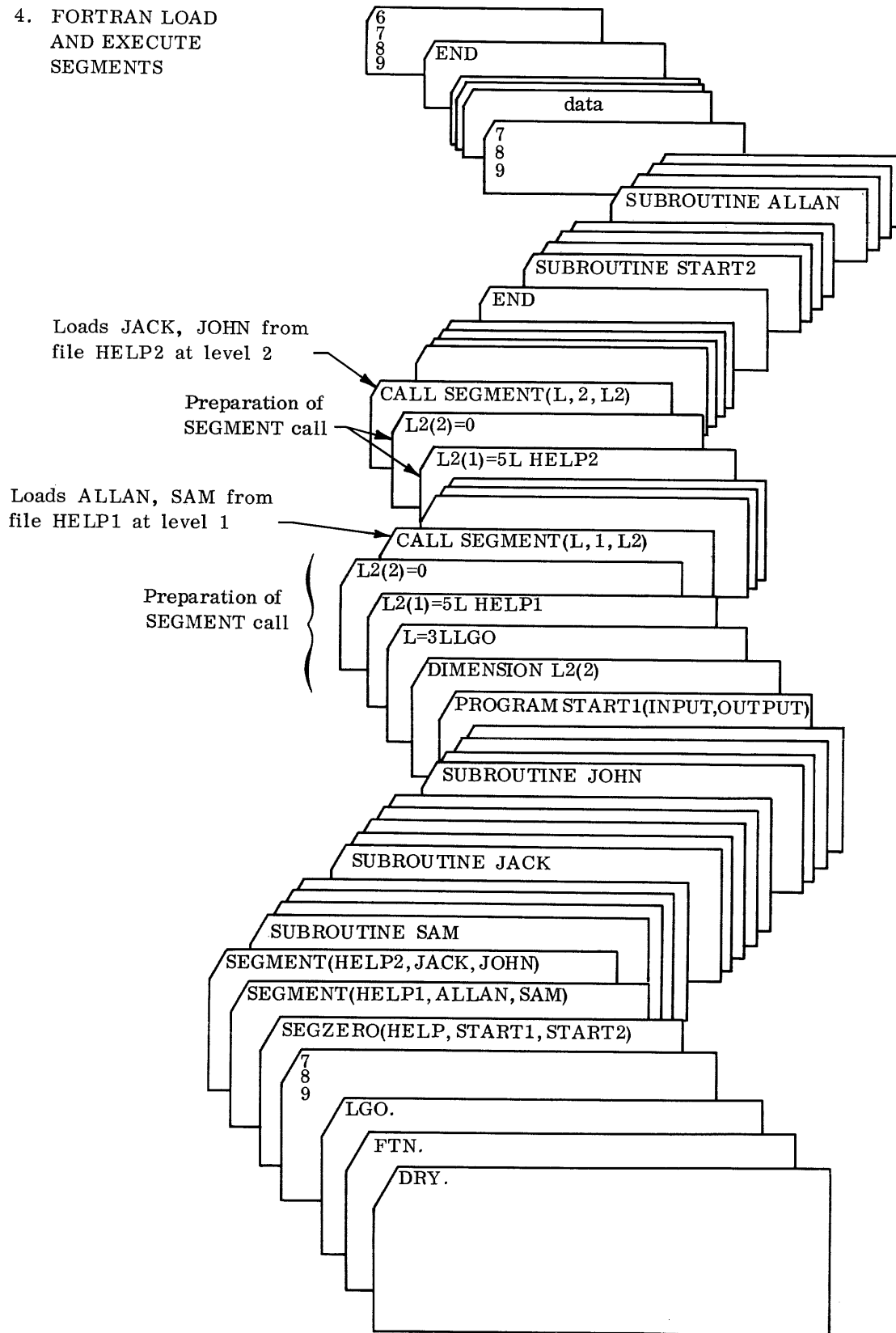




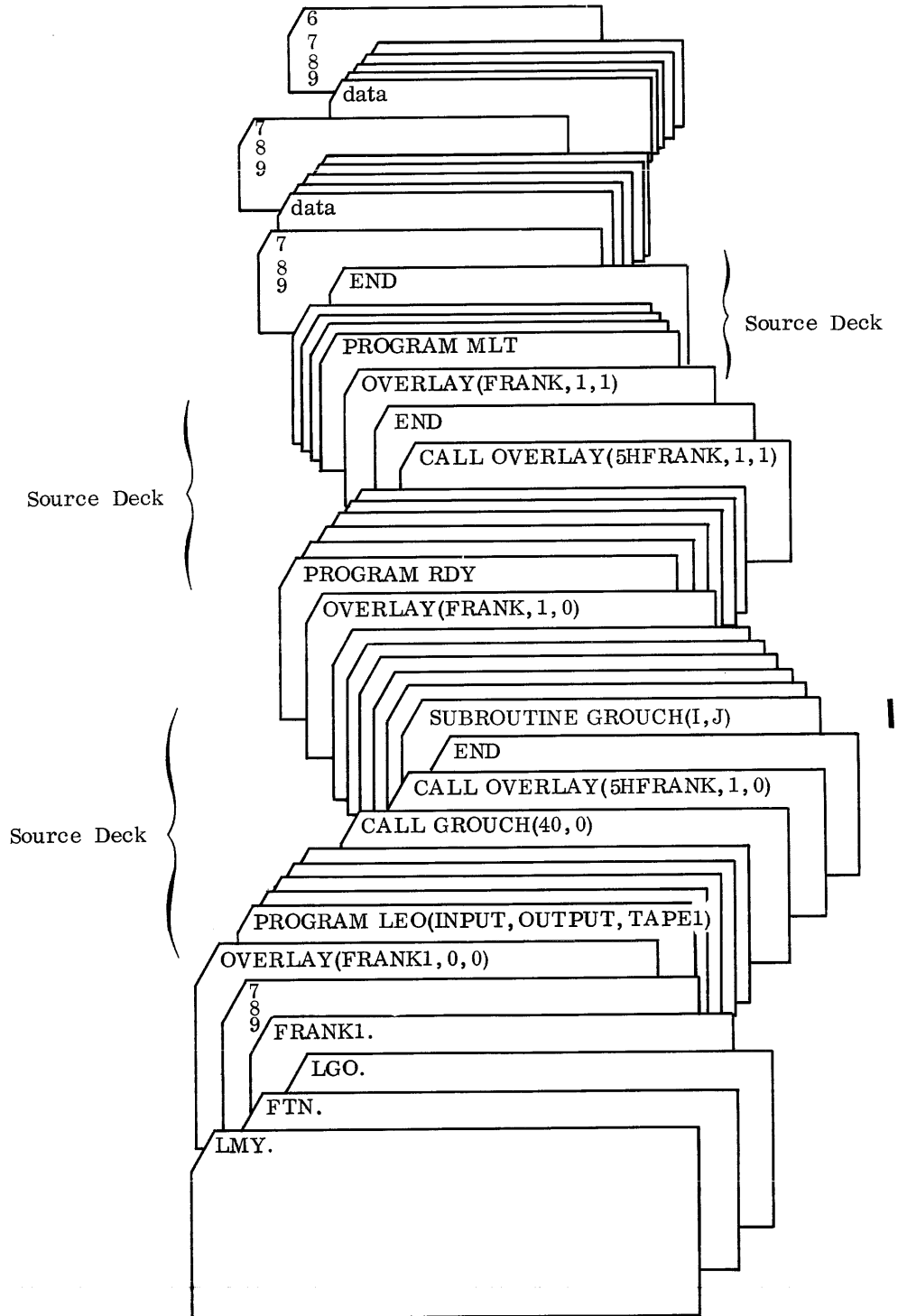
3. Compilation and Execution with Binary Subroutine



4. FORTRAN LOAD  
AND EXECUTE  
SEGMENTS



5. OVERLAY PREPARATION OF  
0,0; 1,0; and 1,1



# EXECUTION TIME I/O

---

## STRUCTURE OF FILES

A file is an ordered sequence of logical records. Logical records are defined for each type of input/output. For 7000 SCOPE version 1 FORTRAN Extended, all files are 7600 Z format, chain files in which records are composed of a maximum of 511 words of data headed by a boundary control word. All format conversion from or to X-mode tapes is done by the input/output stations supporting 7000 SCOPE version 1.

Nine track tape is supported for BUFFER IN and BUFFER OUT statements only.

## FORMATTED I/O

READ(u,f)k WRITE(u,f)k READ f,k PRINT f,k PUNCH f,k

f can be: a FORMAT statement number; the name of an array which contains the FORMAT information; or, a NAMELIST name.

k is an I/O list. k is optional, when k is omitted the preceding comma also must be omitted.

u is a logical unit number.

The length of a logical record is determined by the I/O list and the FORMAT statement.

Formatted logical records can be a maximum of 150₁₀ characters for input, 137₁₀ characters for output. The maximum length formatted logical record for cards is 80 characters. A logical record corresponds to a tape block on S and L tapes†; on X tapes it is always 136₁₀ characters.

## UNFORMATTED I/O

READ(u)k WRITE(u)k

k and u are the same as defined above, however, u should not be assigned to the file OUTPUT.

The length of an unformatted logical record is determined by the length of the I/O list, and can be any size. With unformatted I/O it is possible for a single logical record to be on several cards.

When I/O is unformatted, the logical record is the same as a SCOPE logical record on internal files or X†† magnetic tape files. On an S and L magnetic tape, the physical representation of logical records is the same as that on a SCOPE internal tape; although SCOPE logical records are not defined (on S- and L-style tapes, each tape block consists of a maximum of 5120₁₀ characters with a logical record terminated by a tape block shorter than 5120₁₀ characters).

Since the physical representation of FORTRAN unformatted logical records is the same on S and L tapes as that on SCOPE internal tapes, the files may be used interchangeably; a tape created as a SCOPE internal tape may be read as an S or L tape. Likewise, a tape created by a FORTRAN job as an S or L tape may be read as a SCOPE internal tape. In general, binary S and L tapes which were not written by CDC FORTRAN (RUN or FTN) or as SCOPE tapes, cannot be read by FORTRAN Extended unformatted reads.

†Stranger tape and long record tape.

††External tape in SCOPE 2 format.

Jobs using small logical records will be more efficient if S magnetic tapes are used instead of SCOPE internal or L tapes. Often, non-stop tape motion can be achieved when the buffer size is in excess of 2048₁₀ words (four physical record units on magnetic tape).

#### BUFFER I/O

BUFFER IN (u,k) (A,B)                      BUFFER OUT (u,k) (A,B)

On SCOPE internal files (including tape files) and binary S magnetic tapes, the user logical record is represented as a SCOPE logical record. On a coded X tape, the user logical record will always consist of 14 words (136 characters on tape), and any attempt to write a record longer will result in a fatal diagnostic. On S and L magnetic tapes, the user logical record is defined to be one tape block, the information between two record gaps or between the load point and a record gap. On S magnetic tapes, 512 words is the maximum record length.

#### BUFFER I/O

##### BUF FEI (BUFFER IN)

Only one logical record is read each time BUF FEI is called. If the block length specified by the call is longer than the logical record, excess block locations will not be changed by the read. If the logical record is longer than the block, excess words in the logical record are passed over. The number of CM words transmitted to the program block may be obtained by referencing LENGTH.

After using a BUFFER IN (or BUFFER OUT) statement on unit i, and prior to a subsequent reference to unit i, or to the information, the status of the BUFFER operation must be checked by a reference to the UNIT function. This check insures that requested data has been transferred, and the buffer parameters for the file have been properly restored. If an attempt is made to BUFFER IN past an end-of-file without referencing the UNIT function, BUF FEI will abort the program with the diagnostic: *BUF IN**ENDFILE file name

If a read is attempted, when the last operation on the file was a write, BUF FEI will abort the program with the diagnostic: *BUF IN**LAST OP WRITE, file name

If the starting address for the block is greater than the terminal address, BUF FEI will abort the program with the diagnostic: *BUF IN***FWA.GT.LWA, file name

If an attempt is made to BUFFER IN from an undefined file (file not declared on the PROGRAM card), BUF FEI will abort the job with the diagnostic: *BUF IN**UNASSIGNED MEDIUM, file name

##### BUF FE0 (BUFFER OUT)

One logical record is written each time the routine is called; record length is LWA-FWA+1.

A BUFFER OUT operation must be followed by a reference to the UNIT function. Since BUF FE0 changes the buffer arguments for the file to point to the CM block specified in the call, calls to other routines involving the same file may not follow any buffer operation until the pointers have been restored by the UNIT function. If LWA is less than FWA, the program will be aborted and the following diagnostic will appear in the dayfile: *BUF OUT**FWA.GT.LWA, file name

The UNASSIGNED MEDIUM diagnostic is similar to that issued by BUF FEI.

### Random Access Files (Mass Storage)†

There are two ways to use the mass storage subroutines. They may be used in the normal fashion with one master index, or they may have a master index and many sub-indexes. A file has a name or number index, and it is referenced by the following statements:

```
CALL OPENMS (u, ix, l, p)      CALL WRITMS (u, fwa, n, i)
CALL READMS (u, fwa, n, i)    CALL STINDX (u, ix, l)
```

u is a logical unit number; ix is the first word address of the index in central memory; l is the index length; p indicates how the file is referenced; fwa is central memory address of first word of record; n is number of central memory words to be transferred; i is record number or cell address of the record name or number. (See Chapter 5, Mass Storage I/O.)

In all cases, it is necessary to open (CALL OPENMS) the mass storage file before calling READMS, WRITMS, or STINDX. If the file exists, OPENMS reads the master index into the central memory area specified in the call (the ix parameter).

The STINDX subroutine does not transfer data, it merely changes the file index in the FET to the base specified in the call. After STINDX is called, READMS or WRITMS must be called to read in or create the new index. If the next operation is to be a random access write (WRITMS) and the file is referenced through a name index, the programmer (prior to calling WRITMS) must set to zero the area reserved for the new index buffer (whose first word address is specified by the ix parameter in the call to STINDX). The master index must be reset before job termination so that the correct index will be written on the file.

Upon job termination, the mass storage file is closed automatically by FORTRAN. At this time the index, as specified in the FET, is written as a record on the file.

Examples:

```
1. PROGRAM MS (TAPE5)
   DIMENSION I(10), B(20), C(30)
   CALL OPENMS(5, I, 10, 0)
C READ MASTER INDEX INTO I
   :
   :
   CALL READMS (5, B, 20, 4)
C READ RECORD 4 INTO B (ASSUME THIS RECORD IS A SUB-INDEX)
   CALL STINDX (5, B, 20)
C ALL SUBSEQUENT OPERATIONS ON UNIT 5 WILL USE
C B AS THE INDEX FOR THE FILE
   :
   :
   CALL STINDX (5, I, 10)
C RESTORE MASTER INDEX
END
```

† Random file processing employs a 30-bit index in 7000 SCOPE version 1 rather than a 24-bit index as is used in 6000 SCOPE.

```

2. PROGRAM MS (TAPE5)
C PROGRAM FOR CREATING RANDOM FILE
  DIMENSION J(10),B(7),XYZ(20),ZXY(10),YXZ(50)
  DATA JOE,SAM,PETE,SUB1/3LJOE,3LSAM,4LPETE,4LSUB1/,B/7*0/
  CALL OPENMS(5,J,10,1)
  CALL STINDX(5,B,7)
  DO 10 I=1,7
10 B(I)=0.
C USE INDEX B
  CALL WRITMS(5,XYZ,20,JOE)
  CALL WRITMS(5,ZXY,10,SAM)
  CALL WRITMS(5,YXZ,50,PETE)
  CALL STINDX(5,J,10)
  CALL WRITMS(5,B,7,SUB1)
C WRITE OUT THE SUB-INDEX
  END

3. PROGRAM MS (TAPE5)
C THIS MS FILE HAS NO SUB-INDEXES
  DIMENSION I(10)
  CALL OPENMS(5,I,10,0)
C READ MASTER INDEX INTO I
  :
  :
C ANY READ OR WRITE ON THIS FILE WILL USE THE INDEX IN
C ARRAY I
  :
  :
  END

```

The execution-time routine END will close the file, causing the index at I to be rewritten on the file.

### Status Checking

#### UNIT Function

The UNIT (i) function checks the status of a buffered operation (BUFFER IN or BUFFER OUT only) on logical unit i. The function returns values as follows:

- 1 unit ready, no previous error
- +0 previous read encountered an end-of-file
- +1 parity error on previous buffer operation

#### Example:

```
IF(UNIT(i)) 12,14,16
```

Upon return from the UNIT function, control is transferred to the statement labeled 12, 14 or 16 if the value returned was -1, 0, or +1 respectively.

If the value returned is 0 or +1 the condition indicator is cleared before returning to program control.

Note: If the UNIT function references a non-buffered unit (a unit referenced by I/O statements other than BUFFER IN and BUFFER OUT), the status returned will always indicate unit ready and no previous error (-1).

### EOF Function

The EOF (i) function tests for end-of-file read (non-buffered) on unit i; then clears the end-of-file indicator. The value zero is returned if no end-of-file was encountered on the previous read, or non-zero if end-of-file was encountered on unit i.

When EOF function is used to turn off the end-of-file indicator, several data files can be read from the same logical unit. On the file INPUT, either a 6/7/8/9 card (end-of-file) or a 7/8/9 card (end-of-record) sets the end-of-file indicator; however, the EOF function will only clear the indicator set by the 7/8/9 card. On files other than INPUT, an end-of-record does not set the end-of-file indicator; only an end-of-file sets the end-of-file indicator.

Example:

```
IF (EOF(i)) 10, 20
```

If i designates the file named INPUT, control will return to statement 10 if the previous read encountered an end-of-file, or any 7/8/9 end-of-record card. Otherwise, control will go to statement 20.

To insure against possible input/output errors, the user should make the EOF check after each READ operation. If a READ on unit i is attempted after an EOF occurs on the previous READ and the EOF bit was not cleared by an EOF test, execution is terminated. Following a WRITE operation on unit i, EOF always will return a zero value.

This function has no meaning for random access files. If it is called in reference to a random access file, a zero value always is returned.

### IOCHEC Function

The IOCHEC (i) function tests for parity errors on non-buffered reads on unit i. The value zero is returned if no error occurs.

Example:

```
J = IOCHEC(i)  
IF (J) 15, 25
```

A value of zero is returned to J if no parity error occurs, and non-zero is returned otherwise. Control would then transfer to the statement labeled 25 or 15 respectively. If a parity error occurs, IOCHEC will clear the parity indicator before returning.

Parity errors are handled in the above fashion regardless of the type of the external device.

Only READ parity errors are detected by the status checking functions. WRITE parity errors are detected and a message is written in the dayfile by the SCOPE system.



A parity error indication reveals parity error somewhere within the current logical record. For nonbuffered coded files, this does not necessarily mean the error occurred within the last record requested by the program because the I/O routines read a logical record ahead whenever possible.

#### Backspace/Rewind

If a BACKSPACE is requested on a coded file (except files created by the BUFFER OUT statement) the file is logically moved back one unit record. The backspace is attempted within the I/O buffer; if this is not possible, the external I/O device is repositioned.

Backspace on binary files and files created by BUFFER I/O statements reposition the external device so that the last logical record becomes the next logical record.

When a BACKSPACE (or REWIND) request follows a write operation on a file, an end-of-file is written followed by two backspaces (or by a rewind). Note that SCOPE may write trailer label information immediately following the end-of-file written by FORTRAN.

#### FORMAT Field Separators

Field descriptors are normally delimited by field separators; however, some exceptions are allowed. For example, the statement

```
10 FORMAT(F25.22F10.3)
```

would be interpreted as two descriptors, F25.22 and F10.3. Field separators should be used whenever ambiguity could result.

#### ENCODE/DECODE

Under SCOPE, a binary zero byte is used to terminate a unit record. When the DECODE processor encounters a zero character (6 bits of binary zeros), that character is interpreted as a blank. Conversion continues through n characters per record.

Whenever a record terminator (a slash or the right parenthesis if the list is not exhausted) is encountered in a FORMAT statement, the rest of the record is filled out with blanks (for ENCODE) or ignored (for DECODE), and conversion continues beginning with the next record. (The length of the record is specified by n in a DECODE (n,f,A)k or ENCODE (n,f,A)k statement.) The record is restricted to a maximum length of 150 characters.

Example:

```
10 FORMAT (16(F10.4)) is illegal (the diagnostic EXCEEDED RECORD SIZE is issued)
10 FORMAT (10F10.4,/,6F10.4) is allowed
```

## Labeled Files

Labeled files are not supported by 7000 SCOPE version 1. The LABEL subroutine is maintained for 6000 compatibility. If it is used with 7000 SCOPE version 1, a warning sent to the dayfile indicates that no label checking or processing was performed.

When the PROGRAM statement is compiled, FET's (file environment tables) are set up for each file declared. All fields in the FET label information for a given file are set to zero, except the reel number which is set to 1. If the file has been declared as labeled on a REQUEST control card, SCOPE compares the label with the information in the FET when the file is opened. The information will not compare, and if the initial use of the file is for input, SCOPE will allow the job to continue only after instructions are entered from the display console. If the initial use of the file is for output, SCOPE will write a default label on the tape, and the job will continue.

Only files recorded on 1/2 inch magnetic tape may be labeled. For the FORTRAN programmer to compare label information or to create a standard label containing given information, an execution subroutine (LABEL) is provided to set the desired information into the FORTRAN prepared FET.

If the label information is properly set up, and subroutine LABEL is referenced prior to any other reference to the file, then when the file is opened, the label and the information are compared for an input tape or the information is written on an output tape.

The form of the call is:

```
CALL LABEL (u,fwa)
```

u is the unit number

fwa is the address of the first of four consecutive words containing the desired label information to be placed into the FET. The information must be in the mode and format discussed in the SCOPE Reference Manual.

The four words beginning at fwa are transferred directly to words 10 through 13 of the FET for the file designated by u.

## Carriage Control Characters

<u>Character</u>	<u>Action Before Printing</u>	<u>Action After Printing</u>
A†	Space 1	Eject to top of next page††
B†	Space 1	Skip to last line of page††
1	Eject to top of next page††	No space
2†	Skip to last line on page††	No space
+	No space	No space
0 (zero)	Space 2	No space
- (minus)†	Space 3	No space
blank	Space 1	

†For 7600 FORTRAN Extended, vertical spacing for these and any other characters is determined by the input/output station servicing 7000 SCOPE, version 1.

††The top of a page is indicated by a punch in channel 8 of the carriage control tape for the 501 printer and channel 1 for the 512 printer. The bottom of page is channel 7 in the 501 and 12 in the 512.

When the following characters are used for carriage control, no printing takes place. The remainder of the line will not be printed.

Q	Clear auto page eject
R	Select auto page eject
S	Clear 8 vertical lines per inch (512 printer)
T	Select 8 vertical lines per inch (512 printer)
PM† (col 1-2)	Output remainder of line (up to 30 characters) on the B display and the dayfile and wait for the JANUS typein /OKuu. For files assigned to a printer, n.GO. must be typed to allow the operator to change form or carriage control tapes.
any other	See SCOPE Reference Manual.

Any pre-print skip operation of 1, 2 or 3 lines that follows a post skip operation will be reduced to 0, 1 or 2 lines.

The functions Q through T should be given at the top of a page. S and T can cause spacing to be different from the stated spacing if given in other positions on a page. Q and R will cause a page eject before the next line is printed.

Meaningful results are not guaranteed in the following circumstances:

1. Mixed mode files within a logical file.
2. Mixing buffer I/O statements and standard Read/Write statements on the same file (without a REWIND in between).
3. Requesting a LENGTH function on a buffer unit before requesting a UNIT function.
4. Two consecutive buffer I/O statements on the same file without the intervening execution of a UNIT function call.
5. Making an ENDFILE call on a random file and then calling any of the random-file subroutines, such as READMS, again.

A FORTRAN formatted WRITE will produce X's, R's, or I's in an output field under the following conditions:

1. Fixed point format will produce X's in the output field if the internal data is out of range (greater than or equal to  $2^{*48}$ ).
2. Floating point format will produce R's in the output field if the internal data is out of range or I's if it is indefinite (as defined for 6400/6600 hardware).

Disposition of files at run termination:

1. All indexed files (randomly accessible files) are closed through SCOPE.
2. Output files are demarcated by FORTRAN with an end-of-file and are not rewound. No action is taken on input files.

†Not supported by 7600 SCOPE 1.1.

### Structure of 7000 SCOPE version 1 Input/Output Files

All files are in 7600 Z format chain files which contain physical records of not more than 511 words, headed by a boundary control word. The boundary word contains a pointer to the next control word in the physical record.

Control words embedded in the file indicate one of three conditions:

End-of-record control word denotes the end of a logical record

End-of-file control word denotes the end of a logical file

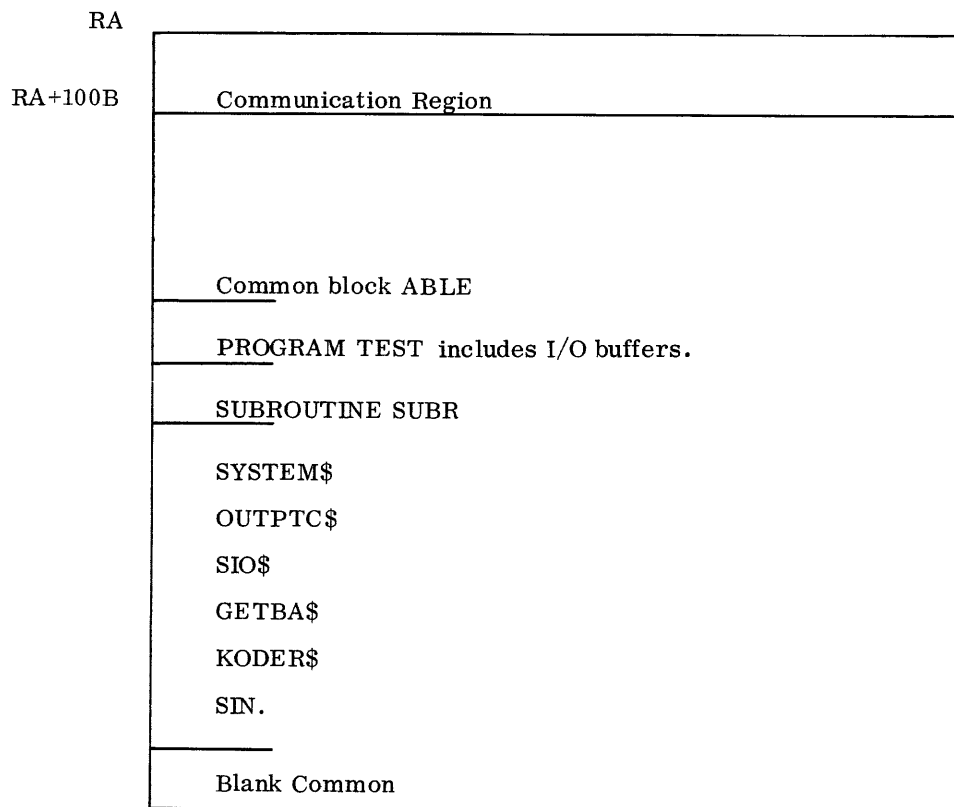
End-of-information control word denotes the end of the physical file.

All X-mode tapes are converted by the input/output stations supporting 7000 SCOPE version 1, and tapes are not processed on line by FORTRAN Extended.

## Memory Structure

Subprograms are loaded as encountered in the input file from RA+100B toward FL. Labeled common blocks are loaded prior to the subprogram in which they first occur. Library routines are loaded immediately after the last encountered subprogram and these are followed by blank common.

The following is a typical memory layout.



---

When a program is entered at an INTERCOM control point, INTERCOM associates INPUT and OUTPUT files of the program with the user's remote terminal device, and all references to these files are directed to the terminal. With calls to the CONDIS library subprogram, the user may specify other files to be associated with the terminal.

The user can associate any logical file in his program with a remote device, with the statement:

```
CALL CONNEC (lfn)
```

If a file is already connected, the request will be ignored. If the file has been used already, but not connected, this request will clear the file's buffer, write an end-of-file, and backspace over it before the connection is performed.

A file is disconnected by:

```
CALL DISCON (lfn)
```

This request will be ignored if the file is not connected. After a disconnect, the file is reassociated with its former device.

lfn File name parameter of the form:

tape logical unit number, 1 to 99

Hollerith constant in the format hLfilename

integer variable containing either of the above

Examples:

```
CALL CONNEC (3LEWT)
CALL DISCON (6)
K=5LINPUT
CALL DISCON (K)
J=12
CALL CONNEC (J)
```

Any files listed on the PROGRAM card may be connected or disconnected during program execution. An attempt to connect or disconnect an undefined file will result in a fatal execution time error, and the job will be terminated.

CONNEC and DISCON calls are ignored when programs are not executed through an INTERCOM control point.

Interactive input/output is supported only for formatted and NAMELIST reads and writes.

---

† KRONOS 2.0 does not support INTERCOM.

# INDEX

---

- A
  - Input / Output of Alphanumeric Data with A Specification 6-11
- ABS
  - Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1
- ACOS
  - Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3
- Actual
  - Actual Arguments or Parameters in a CALL Statement 4-12
  - Actual Arguments and Dummy or Formal Arguments to a Subroutine Subprogram 9-5
- AIMAG
  - Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2
- AINT
  - Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1
- Allocation
  - Ordering of Storage Allocation for an Array 1-8, 1-11
  - Allocation of Storage for an Array via DIMENSION Statement 8-1
- ALOG
  - Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3
- ALOG10
  - Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3
- Alphanumeric
  - Alphanumeric or Hollerith Constants 1-6
  - Input / Output of Alphanumeric Data with A Specification 6-11
  - Input / Output of Alphanumeric Data with R Specification 6-12
  - H Specification for Input / Output of Hollerith or Alphanumeric Data 6-15
- AMAX0
  - Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1
- AMAX1
  - Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1
- AMIN0
  - Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1
- AMIN1
  - Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1
- AMOD
  - Zero as Argument to SIGN, ISIGN, DSIGN, AMOD, and MOD 9-9
  - Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1
- AND
  - Logical Operators OR, AND, NOT 2-5
  - Masking Operators NOT, AND, OR 2-7
  - Library Functions DBLE, CMLPX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2
- AREA
  - AREA Debugging Statement 11-15
- Arguments
  - Actual Arguments or Parameters in a CALL Statement 4-12
  - Actual Arguments and Dummy or Formal Arguments to a Subroutine Subprogram 9-5

- Arithmetic
  - Arithmetic Expressions and Operators 2-1
  - Arithmetic Assignment Statement 3-1
  - Three-Branch Arithmetic IF 4-4
  - Two-Branch Arithmetic IF 4-5
- Array
  - Definition of Array 1-8
  - Array Name without Subscript 1-8
  - Array with More than One Dimension 1-8, 1-9
  - Subscript of an Array Element 1-9
  - Array Name in NAMELIST Statement 5-8
  - Maximum Bounds of an Array Subscript 8-1
  - Variable Dimension of an Array in a Subprogram 8-2
  - Ordering of Storage Allocation for an Array 1-8, 1-11
  - Allocation of Storage for an Array via DIMENSION Statement 8-1
- ARRAYS
  - ARRAYS Debugging Statement 11-2
- Array Element
  - Subscript of an Array Element 1-9
- ASCII
  - ASCII 64 Character Subset A-3
- ASIN
  - Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4
- Assembler
  - Specification of COMPASS Assembler via C Parameter on FTN Control Card 12-11
- ASSIGN
  - ASSIGN Statement 4-1
- Assigned
  - Assigned GO TO Statement 4-2
- Assignment
  - Arithmetic Assignment Statement 3-1
  - Multiple Assignment Statement 3-1
  - Logical Assignment Statement 3-3
  - Masking Assignment Statement 3-3
- ATAN
  - Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3
- ATAN2
  - Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3
  
- B
  - Octal Constants Designated by B 1-6
- BACKSPACE
  - Syntax of REWIND and BACKSPACE Statements 5-9
  - Execution of BACKSPACE and REWIND Statements I-6
- BCD
  - External BCD Codes A-1 through A-4
- Binary
  - Binary Output Parameter on FTN Card 12-2
- Blank
  - Use of Blank in Source Program 1-2
  - Unlabeled Common or Blank Common Blocks 8-4
  - Blank Card in Source Program 1-2
- BLOCK DATA
  - Initialization of Labeled Common Blocks via BLOCK DATA Subprogram 9-11



Bounds

Maximum Bounds of an Array Subscript 8-1

Buffer

Small Buffer Specification via V on FTN Control Card 12-11

Implementation of Buffer Input / Output I-2

Buffer Size Assignment 9-2

Buffered

Characteristics of Buffered Input and Output 7-1

BUFFER IN

Syntax of BUFFER IN and BUFFER OUT Statements 7-2

Execution of BUFFER IN and BUFFER OUT Statements I-2

BUFFER OUT

Syntax of BUFFER IN and BUFFER OUT Statements 7-2

Execution of BUFFER IN and BUFFER OUT Statements I-2

BUGARR

Debug Printing Routines BUGARR, BUGCLL, BUGFUN, BUGGTA 11-17

BUGCLL

Debug Printing Routines BUGARR, BUGCLL, BUGFUN, BUGGTA 11-17

BUGFUN

Debug Printing Routines BUGARR, BUGCLL, BUGFUN, BUGGTA 11-17

BUGGTA

Debug Printing Routines BUGARR, BUGCLL, BUGFUN, BUGGTA 11-17

BUGSTO

Debug Printing Routines BUGSTO, BUGTRC, BUGTRT 11-17

BUGTRC

Debug Printing Routines BUGSTO, BUGTRC, BUGTRT 11-17

BUGTRT

Debug Printing Routines BUGSTO, BUGTRC, BUGTRT 11-17

Built-in Functions

Intrinsic or Built-in Functions 9-9

C

Specification of COMPASS Assembler via C Parameter on FTN Control Card 12-11

CABS

Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3

CALL

CALL ... RETURNS Statement 4-12

Syntax of CALL Statement for Subroutine Subprogram 4-12

CALL Statement and Actual Arguments for a Subroutine Subprogram 9-5

Calling Sequence

Calling Sequence and Control Return for COMPASS Subprogram E-1

CALLS

CALLS Debugging Statement 11-3

Card

Blank Card in Source Program 1-2

Carriage Control

Printer Carriage Control Requirement 5-4

Printer Carriage Control Characters I-7

CCOS

Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

CEXP

Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3

Character

- ASCII 64 Character Subset A-3
- FORTTRAN Character Set 1-1
- Standard SCOPE Character Sets A-1
- CDC 64 Character Set A-2
- CDC 63 Character Set A-4

CLOG

- Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3

CMPLX

- Library Functions DBLE, CMPLX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2

Column

- Tabular Column Selection Control Character T 6-17

Columns

- Card Columns Used in FORTRAN Statements 1-1

Comma

- Slash and Comma as FORMAT Field Separators 6-2

Comment

- Comment Information 1-2

COMMON

- Interaction of EQUIVALENCE and COMMON Statements 8-6
- EQUIVALENCE, COMMON, and DATA Statements in a Function Subprogram 9-9

Common Blocks

- Arrangement of Common Blocks 8-4
- Unlabeled Common or Blank Common Blocks 8-4
- Load of Common Blocks with Segments 10-4
- List of Common Blocks in Cross Reference Map C-14
- Transferring Program and Subprogram Values through Common Blocks 8-3
- Labeled Common Blocks 8-3
- Initialization of Labeled Common Blocks via BLOCK DATA Subprogram 9-11

COMPASS

- Specification of COMPASS Assembler via C Parameter on FTN Control Card 12-11
- Calling Sequence and Control Return for COMPASS Subprogram E-1
- Example of a COMPASS Subprogram E-2
- COMPASS Subprogram Intermixed in FORTRAN Source Deck E-1

Compilation

- T Mode of Compilation Specified on FTN Control Card 12-3

COMPL

- Library Functions DBLE, CMPLX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2

Complex

- Complex Constants 1-5
- Complex Elements in Relational Expressions 2-3
- Input / Output of Complex Data 6-12
- Double Precision and Complex Variables in Common Blocks 8-5

Computed

- Computed GO TO Statement 4-3

CONJG

- Library Functions DBLE, CMPLX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2

Constants

- Definition of Constants 1-3
- Integer Constants 1-3
- Real Constants 1-4
- Complex Constants 1-5
- Alphanumeric or Hollerith Constants 1-6
- Octal Constants Designated by B 1-6
- D in Double Precision Constants 1-4
- Logical Constants TRUE and FALSE 1-5

Continuation  
 Continuation Lines 1-2

Continue  
 Continue Statement 4-12

Control  
 Calling Sequence and Control Return for COMPASS Subprogram E-1

Control Card  
 SECTION Control Card 10-4  
 SEGZERO Control Card 10-5  
 SEGMENT Control Card 10-5  
 FTN ( FORTRAN Extended ) Control Card Format 12-1  
 Examples of Usage of Parameters on FTN Control Card 12-11  
 File Names as LGO Control Card Parameters 9-3  
 Format of OVERLAY Control Card 10-1, 10-2

Control Variables  
 Control Variables and Index Parameters in DO Statements 4-7

Conversion  
 Type Conversion in Mixed-Mode Arithmetic Expressions 2-2  
 Type Conversion in Mixed-Mode Relational Expressions 2-4  
 Type Conversion of Mixed-Mode in Assignment Statements 3-2, Table 3-4  
 Conversion Specifications for Formatted Input / Output 6-2

COS  
 Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

Cross  
 Cross Reference Map C-1

CSIN  
 Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

CSQRT  
 Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

C\$  
 Characters C\$ in Debugging Statement 11-2

D  
 D in Double Precision Constants 1-4  
 Input / Output of Double Precision Data with D Specification 6-10

DABS  
 Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1

Data  
 Transmission of Data Record According to FORMAT Specification 5-2  
 EQUIVALENCE, COMMON, and DATA Statements in a Function Subprogram 9-9  
 Data Initialization with DATA Statement 8-8

DATAN  
 Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3

DATAN2  
 Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3

Data Types  
 List of Data Types 1-3  
 Means of Specifying Data Types of Variables 1-7

DATE  
 Library Functions LEGVAR, IOCHEC, DATE, TIME, SECOND D-4

DBLE  
 Library Functions DBLE, CMLPX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2

DCOS  
 Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

- DEBUG
  - DEBUG Statement as Beginning of Debugging Deck 11-14
  - DEBUG File 11-17
- Debugging
  - Debugging Mode of Compilation 11-1
  - Format of Debugging Statements 11-2
  - Debugging Deck Structure 11-9
  - Debugging Mode Parameter on FTN Control Card 12-9
  - Examples of Debugging Facility Use 11-19 through 11-25
- Deck
  - Source Deck Structure for a FORTRAN Program with Examples H-5
- Declarations
  - Specification Statements or Declarations 8-1
- DECODE
  - Syntax of DECODE Statement 7-2, 7-4, 7-5
  - Execution of ENCODE and DECODE Statements I-6
- Delinking
  - Delinking of a Segment 10-4
- DEXP
  - Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3
- Diagnostics
  - Diagnostics and Traceback Information under Control of System Routine G-1
  - Execution Error Diagnostics by Controlling Routine G-5 through G-11
  - FORTRAN Diagnostics or Error Messages B-1
- DIM
  - Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2
- Dimension
  - Array with More than One Dimension 1-8, 1-9
  - Variable Dimension of an Array in a Subprogram 8-2
  - Allocation of Storage for an Array via DIMENSION Statement 8-1
- Disk
  - Input / Output for Random Access Devices ( Disk ) 5-10
- DISPLA
  - System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5
- Display Codes
  - Display Codes A-1 through A-4
- DLOG
  - Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3
- DLOG10
  - Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3
- DMAX1
  - Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1
- DMIN1
  - Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1
- DMOD
  - Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1
- DO
  - Syntax of DO Statements and DO Loops 4-6
  - Nested DO Loops 4-7
  - Control Variables and Index Parameters in DO Statements 4-7
  - Execution of DO Loops 4-8
  - Transfer from or into DO Loops 4-9
  - Implied DO Loops 5-1
  - Implied DO Loops in DATA Statement 8-8
  - List of DO Loops in Cross Reference Map C-13
  - DO Loop Optimization via OPT Parameter on FTN Control Card 12-4

- Double Precision
  - Double Precision and Complex Variables in Common Blocks 8-5
  - D in Double Precision Constants 1-4
  - Input / Output of Double Precision Data with D Specification 6-10
- DO Implied
  - DO Implied List 5-2
- DSIGN
  - Zero as Argument to SIGN, ISIGN, DSIGN, AMOD, and MOD 9-9
  - Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2
- DSIN
  - Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3
- DSQRT
  - Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3
- Dummy
  - Actual Arguments and Dummy or Formal Arguments to a Subroutine Subprogram 9-5
- DUMP
  - System Subroutines RANGET, RANSET, DUMP, PDUMP, ERRSET 9-7, 9-8, D-5
  
- E
  - E in Exponent Form of Real Constant 1-4
  - Input of Real Number with Exponent E Specification 6-4
  - Output of Real Number with Exponent E Specification 6-7
- ECS
  - ECS ( Extended Core Storage ) Variables 1-11
  - ECS Input / Output Routines READEC and WRITEC 5-10
- Editing
  - UPDATE or Editing Parameters on FTN Control Card 12-3
- ENCODE
  - Syntax of ENCODE Statement 7-2, 7-3
  - Execution of ENCODE and DECODE Statements I-6
- END
  - END Statement 4-15
- ENDFILE
  - Syntax of ENDFILE Statement 5-10
- ENTRY
  - ENTRY Statement in Subprogram 9-6
- Entry Point
  - Definition of Entry Point 10-3
  - List of Entry Points in Cross Reference Map C-6
- EOF
  - Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4
  - End-of-File Test via EOF Function I-5
- EQ
  - Relational Operators EQ, NE, GT, GE, LT, LE 2-4
- EQUIVALENCE
  - Sharing of Storage via EQUIVALENCE Statement 8-5
  - Interaction of EQUIVALENCE and COMMON Statements 8-6
  - EQUIVALENCE, COMMON, and DATA Statements in a Function Subprogram 9-9
  - List of Equivalence Classes in Cross Reference Map C-15
- Error
  - Error Listing by SYSTEM Routine G-4
  - Error Messages in Cross Reference Map C-16
  - Execution Error Diagnostics by Controlling Routine G-5 through G-11
  - Non-Standard Error Recovery G-2

## Error Messages

FORTRAN Diagnostics or Error Messages B-1

## ERRSET

System Subroutines RANGET, RANSET, DUMP, PDUMP, ERRSET 9-7, 9-8, D-5

## Evaluation

Evaluation of Expressions and Precedence or Hierarchy of Operators 2-8

## Exit

Exit Parameter on FTN Control Card 12-9

System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5

## EXP

Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3

## Exponent

E in Exponent Form of Real Constant 1-4

Input of Real Number with Exponent E Specification 6-4

Output of Real Number with Exponent E Specification 6-7

## Exponentiation

Type Conversion in Mixed Mode Exponentiation 2-3

## Expressions

Definition of Expressions 2-1

Arithmetic Expressions and Operators 2-1

Relational Expressions and Operators 2-3

Logical Expressions 2-5

Masking Expressions 2-6

Evaluation of Expressions and Precedence or Hierarchy of Operators 2-8

## Extended Core Storage

ECS ( Extended Core Storage ) Variables 1-11

## EXTERNAL

EXTERNAL Statement for Definition of External Procedure Names 8-7

Definition of an External Function or Function Subprogram 9-9

Reference to External Function within Calling Program 9-10

Definition of Unsatisfied External 10-3

External BCD Codes A-1 through A-4

List of External References in Cross Reference Map C-10

Definition of External Reference 10-3

## F

F Input Specification for Real Numbers 6-7

F Output Specification for Real Numbers 6-8

## FALSE

Logical Constants TRUE and FALSE 1-5

## Field

Slash and Comma as FORMAT Field Separators 6-2

Omission of FORMAT Field Separators I-6

Field Descriptors for FORMAT Statement 6-1

## File

SCOPE INPUT File 5-3

Specification and Equivalencing of a File in PROGRAM Statement 9-1, 9-2

List of File Names in Cross Reference Map C-9

File and Logical Record Structure I-1

File Label Information I-7

Input / Output File Structure for 7600 I-9

File Names as LGO Control Card Parameters 9-3

## FLOAT

Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1

## Floating Point

Floating Point. See Real.

## Formal

Actual Arguments and Dummy or Formal Arguments to a Subroutine Subprogram 9-5

## FORMAT

Transmission of Data Record According to FORMAT Specification 5-2

Slash and Comma as FORMAT Field Separators 6-2

Repeated Format Specifications 6-18

Variable Format Specifications 6-19

Omission of FORMAT Field Separators I-6

Field Descriptors for FORMAT Statement 6-1

## Formatted

Distinction between Formatted and Unformatted Input / Output 5-1

Input of Formatted Data via READ Statement 5-2

Output of Formatted Data via WRITE Statement 5-3

Conversion Specifications for Formatted Input / Output 6-2

Execution of Formatted and Unformatted Input / Output I-1

## FORTRAN

Statement Forms for FORTRAN Statements F-4

FORTRAN Interface with INTERCOM K-1

Source Deck Structure for a FORTRAN Program with Examples H-5

## FORTRAN Extended

Structure of FORTRAN Extended Program 9-1

FTN ( FORTRAN Extended ) Control Card Format 12-1

## FTN

FTN ( FORTRAN Extended ) Control Card Format 12-1

Examples of Usage of Parameters on FTN Control Card 12-11

## FUNCS

FUNCS Debugging Statement 11-5

## Function

Evaluation of a Function or Subscript within an Expression 2-8

Definition of an External Function or Function Subprogram 9-9

Reference to External Function within Calling Program 9-10

Arrangement of Code and Data for Subroutine and Function J-1, J-2

## Functions

Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1

Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3

Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4

Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1

Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1

Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2

Library Functions DBLE, CMLPX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2

Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3

Library Functions LEGVAR, IOCHEC, DATE, TIME, SECOND D-4

## G

G Input / Output Specification for Real Numbers 6-9

## GE

Relational Operators EQ, NE, GT, GE, LT, LE 2-4

## GOTOS

GOTOS Debugging Statement 11-7

## GO TO

Computed GO TO Statement 4-3

Unconditional GO TO Statement 4-1

- Assigned GO TO Statement 4-2
- GT
  - Relational Operators EQ, NE, GT, GE, LT, LE 2-4
- H
  - H, R, and L in Hollerith Constants 1-6
  - H Specification for Input / Output of Hollerith or Alphanumeric Data 6-15
- Hierarchy
  - Evaluation of Expressions and Precedence or Hierarchy of Operators 2-8
- Hollerith
  - Alphanumeric or Hollerith Constants 1-6
  - H Specification for Input / Output of Hollerith or Alphanumeric Data 6-15
  - *...* and /.../ as Hollerith String Delimiters 6-17
  - Hollerith 026 and 029 Punch Codes A-1 through A-4
  - Hollerith Constants and Octal Constants as Subroutine Arguments 9-5
- I
  - Integer Input / Output with I Specification 6-3
- IABS
  - Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1
- Identification
  - Identification Field in Input Lines 1-2
- IDIM
  - Library Functions .SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2
- IDINT
  - Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1
- IF
  - Three-Branch Arithmetic IF 4-4
  - Two-Branch Arithmetic IF 4-5
  - One-Branch Logical IF 4-5
  - Two-Branch Logical IF 4-6
- IFIX
  - Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1
- Implied
  - Implied DO Loops 5-1
  - Implied DO Loops in DATA Statement 8-8
- INDEF
  - Checking Operators for STORES Debugging Statement RANGE, INDEF, VALID 11-6
- Index
  - Permissible Range of Subscript or Index 1-4
  - Control Variables and Index Parameters in DO Statements 4-7
- Initialization
  - Data Initialization with DATA Statement 8-8
  - Initialization of Labeled Common Blocks via BLOCK DATA Subprogram 9-11
- Inline Functions
  - List of Inline Functions in Cross Reference Map C-11
- Input
  - Distinction between Formatted and Unformatted Input / Output 5-1
  - Input of Formatted Data via READ Statement 5-2
  - SCOPE INPUT File 5-3
  - Unformatted Input via READ Statement 5-5
  - Input of Data in NAMELIST Form via READ 5-7
  - ECS Input / Output Routines READEC and WRITEC 5-10
  - Characteristics of Buffered Input and Output 7-1



File Names INPUT, OUTPUT, PUNCH, and TAPE i in PROGRAM Statement 9-1  
 Implementation of Buffer Input / Output I-2  
 Execution of Formatted and Unformatted Input / Output I-1  
 Input / Output File Structure for 7600 I-9  
 Input / Output List 5-1  
 Mass Storage Input / Output via OPENMS, READMS, WRITMS, and STINDX 5-10, I-3

**INT**  
 Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1

**Integer**  
 Integer Constants 1-3  
 Integer Input / Output with I Specification 6-3

**INTERCOM**  
 FORTRAN Interface with INTERCOM K-1

**Intrinsic**  
 Intrinsic or Built-in Functions 9-9  
 Intrinsic Function Names in EXTERNAL Statement 8-7

**IOCHEC**  
 Parity Error Detection via IOCHEC Function I-5  
 Library Functions LEGVAR, IOCHEC, DATE, TIME, SECOND D-4

**ISIGN**  
 Zero as Argument to SIGN, ISIGN, DSIGN, AMOD, and MOD 9-9  
 Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2

**L**  
 H, R, and L in Hollerith Constants 1-6  
 Input / Output of Logical Data with L Specification 6-12

**Label**  
 File Label Information I-7

**Labeled**  
 Labeled Common Blocks 8-3  
 Initialization of Labeled Common Blocks via BLOCK DATA Subprogram 9-11

**Labels**  
 List of Statement Labels in Cross Reference Map C-12  
 Statement Labels or Numbers 1-2  
 Statement Labels in Computed GO TO 4-3  
 Statement Labels in an Assigned GO TO 4-1, 4-2

**LCM**  
 Transmission of Data to and from LCM ( 7600 only ) 5-10  
 LCM and SCM Buffer Size for 7600 9-2

**LE**  
 Relational Operators EQ, NE, GT, GE, LT, LE 2-4

**LEGVAR**  
 Library Functions LEGVAR, IOCHEC, DATE, TIME, SECOND D-4

**LENGTH**  
 Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4

**LGO**  
 File Names as LGO Control Card Parameters 9-3

**Library**  
 Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1  
 Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3  
 Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3  
 Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4  
 Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1  
 Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1  
 Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2

- Library Functions DBLE, CMPLX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2
- Library Functions ATAN, DATAN, ATAN2, DATAN2, CABS, ACOS D-3
- Library Functions LEGVAR, IOCHEC, DATE, TIME, SECOND D-4
- Link
  - Definition of Link 10-3
- List
  - List Parameter on FTN Card 12-2
  - Input / Output List 5-1
- Load
  - Load of Common Blocks with Segments 10-4
- LOCF
  - Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4
- Logical
  - Logical Expressions 2-5
  - Logical Operators OR, AND, NOT 2-5
  - Logical Assignment Statement 3-3
  - One-Branch Logical IF 4-5
  - Two-Branch Logical IF 4-6
  - Input / Output of Logical Data with L Specification 6-12
  - Logical Constants TRUE and FALSE 1-5
- Loops
  - Syntax of DO Statements and DO Loops 4-6
  - Nested DO Loops 4-7
  - Execution of DO Loops 4-8
  - Transfer from or into DO Loops 4-9
  - Implied DO Loops 5-1
  - Implied DO Loops in DATA Statement 8-8
  - List of DO Loops in Cross Reference Map C-13
  - DO Loop Optimization via OPT Parameter on FTN Control Card 12-4
- LT
  - Relational Operators EQ, NE, GT, GE, LT, LE 2-4
- Main Program
  - PROGRAM Statement in Main Program 9-1
- MASK
  - Library Functions DBLE, CMPLX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2
- Masking
  - Masking Expressions 2-6
  - Masking Operators NOT, AND, OR 2-7
  - Masking Assignment Statement 3-3
- Mass Storage
  - Mass Storage Input / Output via OPENMS, READMS, WRITMS, and STINDX 5-10, I-3
- MAX0
  - Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1
- MAX1
  - Library Functions DMOD, AMAX0, AMAX1, MAX0, MAX1, DMAX1 D-1
- MIN0
  - Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1
- MIN1
  - Library Functions AMIN0, AMIN1, MIN0, MIN1, DMIN1, FLOAT, IFIX D-1
- Mixed-Mode
  - Type Conversion in Mixed-Mode Arithmetic Expressions 2-2
  - Type Conversion in Mixed-Mode Relational Expressions 2-4
  - Type Conversion of Mixed-Mode in Assignment Statements 3-2, Table 3-4

MOD  
     Zero as Argument to SIGN, ISIGN, DSIGN, AMOD, and MOD 9-9  
     Library Functions ABS, IABS, DABS, AINT, INT, IDINT, AMOD, MOD D-1  
 Mode  
     Mode. See Type.  
 Multiple  
     Multiple Assignment Statement 3-1

NAMELIST  
     NAMELIST Statement 5-6  
     Input of Data in NAMELIST Form via READ 5-7  
     Array Name in NAMELIST Statement 5-8  
     Output of Data in NAMELIST Form via WRITE 5-9  
     List of NAMELIST Groups in Cross Reference Map C-11  
 Names  
     Formation and Use of Symbolic Names 1-3  
     Formation of Variable Names 1-7  
 NE  
     Relational Operators EQ, NE, GT, GE, LT, LE 2-4  
 Nested  
     Nested DO Loops 4-7  
 NOGO  
     NOGO Debugging Statement 11-8  
 Non-Standard  
     Non-Standard Error Recovery G-2  
 NOT  
     Logical Operators OR, AND, NOT 2-5  
     Masking Operators NOT, AND, OR 2-7  
 Numbers  
     Statement Labels or Numbers 1-2

O  
     Input / Output of Octal Data with O Specification 6-10  
 Octal  
     Octal Constants Designated by B 1-6  
     Input / Output of Octal Data with O Specification 6-10  
     Hollerith Constants and Octal Constants as Subroutine Arguments 9-5  
 OFF  
     OFF Statement in Debugging Mode 11-16  
 OPENMS  
     Mass Storage Input / Output via OPENMS, READMS, WRITMS, and STINDX 5-10, I-3  
 Operators  
     Arithmetic Expressions and Operators 2-1  
     Relational Expressions and Operators 2-3  
     Logical Operators OR, AND, NOT 2-5  
     Masking Operators NOT, AND, OR 2-7  
     Evaluation of Expressions and Precedence or Hierarchy of Operators 2-8  
 OPT  
     Optimization Parameter OPT on FTN Control Card 12-4  
 Optimization  
     Assignment to Registers for Optimization in Fast Object Code Mode 12-6  
     Optimization Parameter OPT on FTN Control Card 12-4  
 OR  
     Logical Operators OR, AND, NOT 2-5

- Masking Operators NOT, AND, OR 2-7
- Library Functions DBLE, CMPLX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2
- Output
  - Distinction between Formatted and Unformatted Input / Output 5-1
  - Output of Formatted Data via WRITE Statement 5-3
  - Formatted Output via PRINT and PUNCH Statements 5-4
  - Unformatted Output via WRITE Statement 5-5
  - Output of Data in NAMELIST Form via WRITE 5-9
  - ECS Input / Output Routines READEC and WRITEC 5-10
  - Characteristics of Buffered Input and Output 7-1
  - File Names INPUT, OUTPUT, PUNCH, and TAPE i in PROGRAM Statement 9-1
  - Implementation of Buffer Input / Output I-2
  - Execution of Formatted and Unformatted Input / Output I-1
  - Input / Output List 5-1
  - Mass Storage Input / Output via OPENMS, READMS, WRITMS, and STINDX 5-10, I-3
- Overlay
  - Definition of Overlay and Segment 10-1
  - Parameters for CALL OVERLAY Statement 10-3
  - Format of OVERLAY Control Card 10-1, 10-2
  
- P
  - P Specification for Scale Factor in Input / Output 6-13
- Parameters
  - Actual Arguments or Parameters in a CALL Statement 4-12
- Parity
  - Specification of Parity in a Buffer Control Statement 7-1
- PAUSE
  - PAUSE Statement 4-15
- PDUMP
  - System Subroutines RANGET, RANSET, DUMP, PDUMP, ERRSET 9-7, 9-8, D-5
- Precedence
  - Evaluation of Expressions and Precedence or Hierarchy of Operators 2-8
- PRINT
  - Formatted Output via PRINT and PUNCH Statements 5-4
- Printer
  - Printer Carriage Control Requirement 5-4
  - Printer Carriage Control Characters I-7
- Procedure
  - EXTERNAL Statement for Definition of External Procedure Names 8-7
- Program
  - Structure of FORTRAN Extended Program 9-1
  - PROGRAM Statement in Main Program 9-1
  - Specification and Equivalencing of a File in PROGRAM Statement 9-1, 9-2
  - Source Deck Structure for a FORTRAN Program with Examples H-5
- PUNCH
  - Formatted Output via PRINT and PUNCH Statements 5-4
  - File Names INPUT, OUTPUT, PUNCH, and TAPE i in PROGRAM Statement 9-1
  
- R
  - H, R, and L in Hollerith Constants 1-6
  - Input / Output of Alphanumeric Data with R Specification 6-12
- Random Access
  - Input / Output for Random Access Devices ( Disk ) 5-10

RANF  
 Obtaining or Initializing Generative Value of RANF with RANGET and RANSET 9-7  
 Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4

RANGE  
 Checking Operators for STORES Debugging Statement RANGE, INDEF, VALID 11-6

RANGET  
 System Subroutines RANGET, RANSET, DUMP, PDUMP, ERRSET 9-7, 9-8, D-5

RANSET  
 System Subroutines RANGET, RANSET, DUMP, PDUMP, ERRSET 9-7, 9-8, D-5

READ  
 READ and WRITE Statement Parameters 5-2  
 Input of Formatted Data via READ Statement 5-2  
 Unformatted Input via READ Statement 5-5  
 Input of Data in NAMELIST Form via READ 5-7

READEC  
 ECS Input / Output Routines READEC and WRITEC 5-10

READMS  
 Mass Storage Input / Output via OPENMS, READMS, WRITMS, and STINDX 5-10, I-3

Real  
 Real Constants 1-4  
 Input of Real Number with Exponent E Specification 6-4  
 Output of Real Number with Exponent E Specification 6-7  
 F Input Specification for Real Numbers 6-7  
 F Output Specification for Real Numbers 6-8  
 G Input / Output Specification for Real Numbers 6-9  
 Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2

Record  
 Transmission of Data Record According to FORMAT Specification 5-2  
 Slash as Record Terminator 6-16  
 File and Logical Record Structure I-1

Reference  
 Reference to External Function within Calling Program 9-10  
 Obtaining Cross Reference Map via List Parameter on FTN Card 12-3  
 Reference Map Level Parameter on FTN Control Card 12-11  
 Cross Reference Map C-1  
 Example Program with Cross Reference Map C-5

Registers  
 Assignment to Registers for Optimization in Fast Object Code Mode 12-6

Relational  
 Relational Expressions and Operators 2-3  
 Relational Operators EQ, NE, GT, GE, LT, LE 2-4  
 Evaluation of Relational Expressions 2-4  
 Type Conversion in Mixed-Mode Relational Expressions 2-4

REMARK  
 System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5

RETURN  
 RETURN Statement in a Subroutine Subprogram 9-4  
 RETURN from a Function Subprogram 9-9  
 Calling Sequence and Control Return for COMPASS Subprogram E-1  
 RETURN Statement in a Subprogram 4-14

RETURNS  
 CALL ... RETURNS Statement 4-12  
 RETURNS Option in a Subroutine 9-4

REWIND  
 Syntax of REWIND and BACKSPACE Statements 5-9  
 Execution of BACKSPACE and REWIND Statements I-6

ROUND  
     Rounded Arithmetic via ROUND Parameter on FTN Control Card 12-9

Scale Factor  
     P Specification for Scale Factor in Input / Output 6-13

SCM  
     LCM and SCM Buffer Size for 7600 9-2

SCOPE  
     SCOPE INPUT File 5-3  
     Standard SCOPE Character Sets A-1

SECOND  
     Library Functions LEGVAR, IOCHEC, DATE, TIME, SECOND D-4

SECTION  
     SECTION Control Card 10-4

Segment  
     Definition of Overlay and Segment 10-1  
     Terms Applicable to a Segment 10-3  
     SEGMENT Control Card 10-5  
     CALL SEGMENT Statement 10-5  
     Delinking of a Segment 10-4  
     BLOCK DATA Subprogram within a SEGMENT 9-11

SEGZERO  
     SEGZERO Control Card 10-5

SHIFT  
     Library Functions DBLE, CMLPX, CONJG, SHIFT, AND, OR, COMPL, MASK D-2

SIGN  
     Zero as Argument to SIGN, ISIGN, DSIGN, AMOD, and MOD 9-9  
     Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2

SIN  
     Library Functions EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN D-3

Slash  
     Slash and Comma as FORMAT Field Separators 6-2  
     Slash as Record Terminator 6-16

SLITE  
     System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5

SLITET  
     System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5

SNGL  
     Library Functions SIGN, ISIGN, DSIGN, DIM, IDIM, SNGL, REAL, AIMAG D-2

Source  
     Source Deck Structure for a FORTRAN Program with Examples H-5

Source Input  
     Source Input Parameter on FTN Card 12-1

Spacing  
     X Specification for Spacing in Input / Output 6-14

Specification  
     Specification Statements or Declarations 8-1

SQRT  
     Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

SSWTCH  
     System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5

Statement  
     \$ as Statement Separator 1-2  
     Statement Labels in Computed GO TO 4-3  
     Statement Forms for FORTRAN Statements F-4

Statement Labels in an Assigned GO TO 4-1, 4-2  
 Statement Functions 9-8  
 STINDEX  
   Mass Storage Input / Output via OPENMS, READMS, WRITMS, and STINDEX 5-10, I-3  
 STOP  
   STOP Statement 4-14  
 Storage  
   Sharing of Storage via EQUIVALENCE Statement 8-5  
   Ordering of Storage Allocation for an Array 1-8, 1-11  
   Allocation of Storage for an Array via DIMENSION Statement 8-1  
 STORES  
   STORES Debugging Statement 11-6  
 STRACE  
   STRACE Debugging Aid 11-18  
 Subprogram  
   Variable Dimension of an Array in a Subprogram 8-2  
   Subroutine Subprogram 9-3  
   ENTRY Statement in Subprogram 9-6  
   Definition of an External Function or Function Subprogram 9-9  
   Example of a COMPASS Subprogram E-2  
   Transferring Program and Subprogram Values through Common Blocks 8-3  
   COMPASS Subprogram Intermixed in FORTRAN Source Deck E-1  
   Syntax of CALL Statement for Subroutine Subprogram 4-12  
   RETURN Statement in a Subprogram 4-14  
   Actual Arguments and Dummy or Formal Arguments to a Subroutine Subprogram 9-5  
 Subroutine  
   Subroutine Subprogram 9-3  
   System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5  
   Arrangement of Code and Data for Subroutine and Function J-1, J-2  
   System Subroutines RANGET, RANSET, DUMP, PDUMP, ERRSET 9-7, 9-8, D-5  
   Syntax of CALL Statement for Subroutine Subprogram 4-12  
   Actual Arguments and Dummy or Formal Arguments to a Subroutine Subprogram 9-5  
 Subscript  
   Permissible Range of Subscript or Index 1-4  
   Array Name without Subscript 1-8  
   Subscript of an Array Element 1-9  
   Evaluation of a Function or Subscript within an Expression 2-8  
   Maximum Bounds of an Array Subscript 8-1  
 Symbolic  
   Formation and Use of Symbolic Names 1-3  
 SYSEEDIT  
   SYSEEDIT Parameter on FTN Control Card 12-10  
 SYSTEM  
   SYSTEM Routine Specifications G-1  
   Entry Points of SYSTEM Routine G-4  
   System Subroutines SLITE, SLITET, SSWTCH, EXIT, REMARK, DISPLA 9-7, D-5  
   System Subroutines RANGET, RANSET, DUMP, PDUMP, ERRSET 9-7, 9-8, D-5  
 Systems Text File  
   Specification of Systems Text File via S Parameter on FTN Control Card 12-9  
  
 T  
   Tabular Column Selection Control Character T 6-17  
   T Mode of Compilation Specified on FTN Control Card 12-3  
 Tabular  
   Tabular Column Selection Control Character T 6-17

TAN  
Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4

TANH  
Library Functions DSIN, CSIN, COS, DCOS, CCOS, TANH, SQRT, DSQRT, CSQRT D-3

TAPE  
File Names INPUT, OUTPUT, PUNCH, and TAPE i in PROGRAM Statement 9-1

TIME  
Library Functions LEGVAR, IOCHEC, DATE, TIME, SECOND D-4

TRACE  
TRACE Debugging Statement 11-7

Traceback  
Diagnostics and Traceback Information under Control of System Routine G-1

TRUE  
Logical Constants TRUE and FALSE 1-5

Type  
Type of Data 1-3  
Type Conversion in Mixed-Mode Arithmetic Expressions 2-2  
Type Conversion in Mixed-Mode Relational Expressions 2-4  
Type Conversion of Mixed-Mode in Assignment Statements 3-2, Table 3-4

Unconditional  
Unconditional GO TO Statement 4-1

Unformatted  
Distinction between Formatted and Unformatted Input / Output 5-1  
Unformatted Input via READ Statement 5-5  
Unformatted Output via WRITE Statement 5-5  
Execution of Formatted and Unformatted Input / Output I-1

UNIT  
Library Functions ASIN, TAN, RANF, LOCF, UNIT, EOF, LENGTH D-4  
Status Checking via UNIT Function I-4

Unlabeled  
Unlabeled Common or Blank Common Blocks 8-4

Unsatisfied  
Definition of Unsatisfied External 10-3

UPDATE  
UPDATE or Editing Parameters on FTN Control Card 12-3

V  
Small Buffer Specification via V on FTN Control Card 12-11

VALID  
Checking Operators for STORES Debugging Statement RANGE, INDEF, VALID 11-6

Variable  
Formation of Variable Names 1-7  
Variable Dimension of an Array in a Subprogram 8-2

Variables  
List of Variables in Cross Reference Map C-7

WRITE  
READ and WRITE Statement Parameters 5-2  
Output of Formatted Data via WRITE Statement 5-3  
Unformatted Output via WRITE Statement 5-5  
Output of Data in NAMELIST Form via WRITE 5-9



WRITEC  
 ECS Input / Output Routines READEC and WRITEC 5-10

WRITMS  
 Mass Storage Input / Output via OPENMS, READMS, WRITMS, and STINDX 5-10, I-3

X  
 X Specification for Spacing in Input / Output 6-14

Zero  
 Zero as Argument to SIGN, ISIGN, DSIGN, AMOD, and MOD 9-9

026  
 Hollerith 026 and 029 Punch Codes A-1 through A-4

029  
 Hollerith 026 and 029 Punch Codes A-1 through A-4

63  
 CDC 63 Character Set A-4

64  
 CDC 64 Character Set A-2  
 ASCII 64 Character Subset A-3

7600  
 Transmission of Data to and from LCM ( 7600 only ) 5-10  
 LCM and SCM Buffer Size for 7600 9-2  
 Input / Output File Structure for 7600 I-9

\$  
 \$ as Statement Separator 1-2

*...*  
 *...* and /.../ as Hollerith String Delimiters 6-17

/.../  
 *...* and /.../ as Hollerith String Delimiters 6-17

COMMENT SHEET



TITLE: FORTRAN Extended Reference Manual (6000 Version 3/7000 Version 1)

PUBLICATION NO. 60329100

REVISION D

This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

CUT ON THIS LINE

General comments:

FROM NAME: _____ POSITION: _____

COMPANY NAME: _____

ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.  
FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS  
 PERMIT NO. 8241  
 MINNEAPOLIS, MINN.

**BUSINESS REPLY MAIL**  
 NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

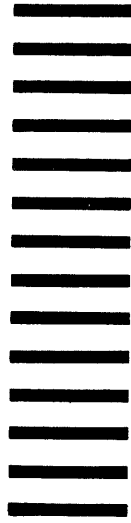
POSTAGE WILL BE PAID BY

**CONTROL DATA CORPORATION**

*Documentation Department*

**215 Moffett Park Drive**

**Sunnyvale, California 94086**



CUT ON THIS LINE

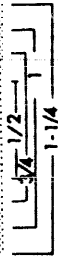
FOLD

FOLD

STAPLE

STAPLE

**CONTROL DATA**



▶ ▶ CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

**CONTROL DATA**  
CORPORATION

**CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440**  
**SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD**

Litho in U.S.A.