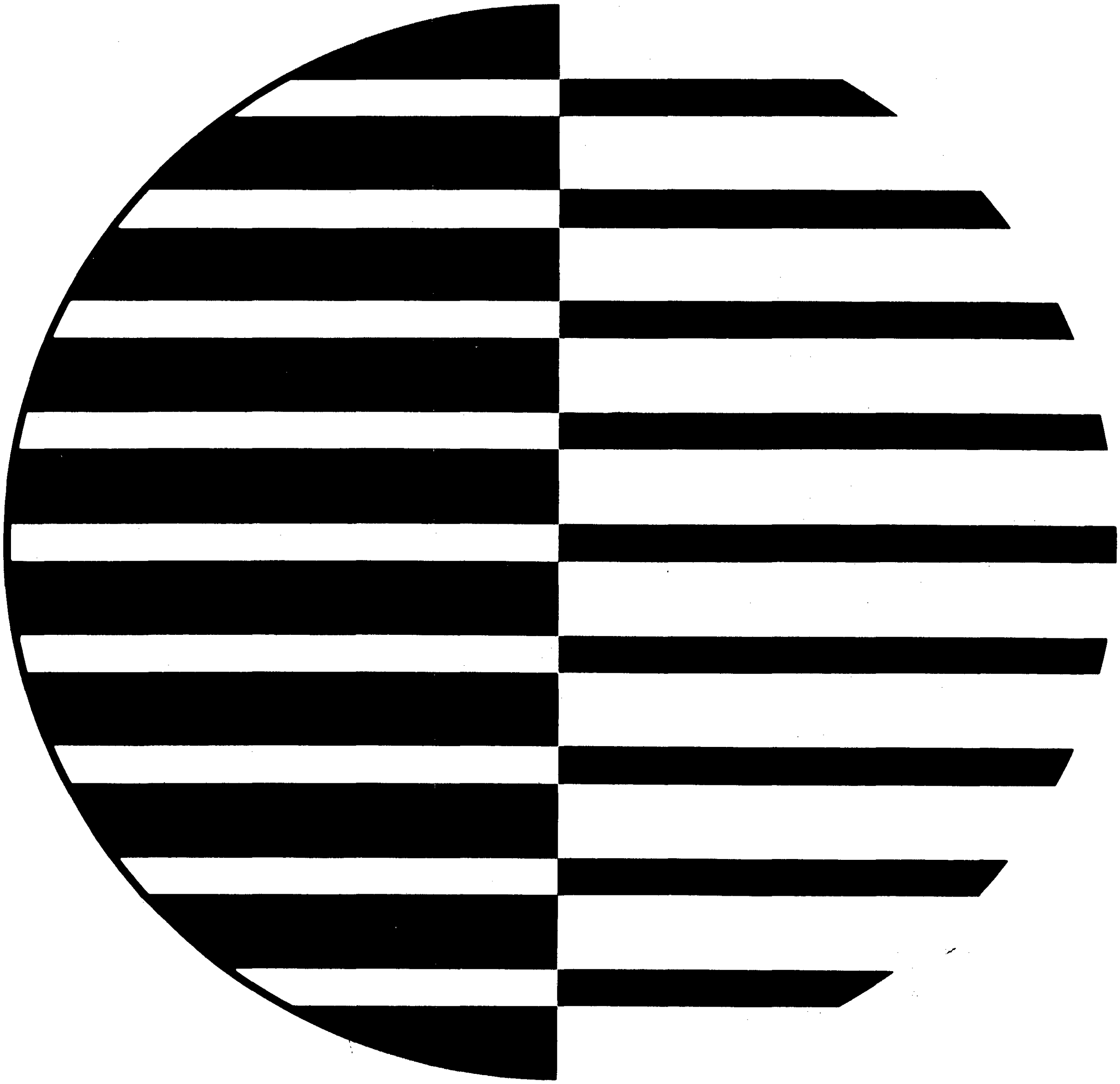
CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
FORTRAN Reference Manual



REVISION RECORD

REVISION	DESCRIPTION
C	Documentation changes to FORTRAN version 2.3 made to clarify concepts.
(2-20-69)	Additions include: Examples, Random Access Files (Mass Storage) and File
	Name Handling (Appendixes I and J), Library Subroutines READEC,WRITEC,
	READMS, WRITMS, OPENMS, STINDX, FTNBIN. This manual obsoletes all
	previous editions.
D	Project updating of system and corrections in response to user comments.
(11-21-69)	Additions include: Examples, Library Subroutines REMARK, and DISPLA,
	SEGMENT parameters, Execution Diagnostics, Library Routine Entry Points,
	File Structure (Appendix M), Print File Conventions (Appendix O). Affected
	pages: 1-1; 2-1, 2, 4; 5-7, 13; 6-10; 7-2 thru 7-4, 7-10 thru 7-12; 8-2, 3;
	9-3, 5, 6, 9, 12, 16, 22, 24, 25; 10-1, 2, 6, 10, 16, 17; B-1 thru B-4; C-1 thru
	C-3; F-1, 4; G-1; J-2; K-1 thru K-6; L-3; M-1, 2; N-1, 2; O-1, 2; Index;
	Comment Sheet.
E	This revision reflects standardization of the SCOPE 3.3 63-character set and
(4-23-71)	the 64-character set options. Pages affected are vii, viii, A-1 thru A-4.
Publication No. 60174900	

Additional copies of this manual may be
obtained from the nearest Control Data
Corporation sales office.

Address comments concerning
this manual to:

CONTROL DATA CORPORATION
Documentation Department
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the
back of this manual

PREFACE

FORTRAN Version 2.3 for the Control Data 6000 Series Computer Systems includes the extensions allowed in FORTRAN Version 2.0 and some improvement in generated object code. The compiler and execution time routines operate under 6400/6500/6600 SCOPE Version 3. Subprograms are compiled independently and a file consisting of relocatable binary subprograms is produced. Upon option, it will also produce a source listing, an object code listing, a cross reference listing, and a copy of the relocatable file on the PUNCHB file which allows the operating system to produce a relocatable binary deck. Extensions and restrictions provided by Version 2.3 are for USASI compatibility or additional features and requirements of SCOPE 3. The statement language is compatible with FORTRAN II and FORTRAN IV; but new programs should be written in FORTRAN IV.

The compiler can operate as load-and-go and produce 6000 Series machine language output. It operates as an independent program under control of the operating system and can be called to use only the storage required for compilation of a particular program. Several compilations may be processed simultaneously using the multi-programming features.

FORTRAN accepts main programs and subprograms written either in FORTRAN source language or 6000 Series assembly language. These features permit a flexible program arrangement for each particular job.

This document assumes a knowledge of the FORTRAN language and the CONTROL DATA® 6000 Series Computer System.

CONTENTS

APPENDIX SECTION

CODING PROCEDURES 1

1.1 CODING LINE

A FORTRAN coding line contains 80 columns in which FORTRAN characters are written one per column. The five types of coding lines are listed below:

	Column		Content	
Statement	1-5	or	statement number or blank	
	1		D, I, B, F	FORTRAN II
	6		blank or zero	
	7-72		FORTRAN statement	
	73-80		identification field	
Continuation	1-5		blank	
	6		FORTRAN character other than blank or zero	
	7-72		continued FORTRAN statement	
	73-80		identification field	
Comment	1		C $ or *	
	2-80		comments	
Data	1-80		data	
Page eject	1		period	

1.1.1 STATEMENT

Statement information is written in columns 7 through 72. Statements longer than 66 columns may be continued to the next line. Blanks are ignored by the FORTRAN compiler except in H fields. The character $ may be used to separate statements when more than one is written on a coding line, however, it may not be used with FORMAT or DATA statements. A blank card may be used to separate the statements.

These statements are equivalent:

```
    I = 10              I = 10 $ JLIM = 1 $ K = K+1 $ GO TO 10
    JLIM = 1
    K = K+1
    GO TO 10
```

Also:

```
    DO 1 I=1, 10        DO 1 I=1, 10 $ A(I)=B(I)+C(I)
    A(I)=B(I)+C(I)    1 CONTINUE $ I=3
  1 CONTINUE
    I=3
```

1.1.2 CONTINUATION

The first line of every statement must have a blank or zero in column 6. If statements occupy more than one line, all subsequent lines must have a FORTRAN character other than blank or zero in column 6. Continuation cards may be separated by cards whose first 72 columns are blank. A statement may have up to 19 continuation lines.

1.1.3 STATEMENT NUMBER

Any statement except END may have an identifier, statement number, but only statements referred to elsewhere in the program require identifiers. A statement number is a string of 1 to 5 digits occupying any column positions 1 through 5.

1.1.4 IDENTIFICATION FIELD

Columns 73 through 80 are always ignored in the compilation process. They may be used for identification when the program is to be punched on cards. Usually these columns contain sequencing information provided by the programmer.

1.1.5 COMMENTS

Each line of comment information is designated by a C, *, or $ in column 1. Comment information appears in the source program and the source program listing, but it is not translated into object code. The continuation character in column 6 is not applicable to comments cards.

1.1.6 PAGE EJECT

A card with a period punched in its first column will eject the page. The remainder of the card is neither listed nor compiled. The listing of subsequent material in the source program will begin on a new page. For the method of ejecting a page during printing see Output Statements in chapter 10.

1.2 PUNCHED CARDS

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

When cards are being used for data input, all 80 columns may be used.

ELEMENTS OF FORTRAN 2

2.1 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)	blank
	(	left parenthesis		

All characters appear internally in 6000 series display code (appendix A). A blank is ignored by the compiler except in Hollerith fields; otherwise it may be used freely to improve program readability. Appendix A includes a list of additional characters which may appear in Hollerith literals and, with the exception of semi-colon, in DATA statements.

2.2 IDENTIFIERS

2.2.1 ALPHANUMERIC IDENTIFIER

An alphanumeric identifier can be any combination of 1-7 characters beginning with a letter, with certain exceptions. The combination of the letter O and 6 digits is recognized as an octal constant. Embedded blanks within an identifier are ignored. Attempts to use the FORTRAN library routine entry point names (Appendix L) or the four characters CALL as an identifier or variable name will result in a compilation diagnostic (VC or CL).

Examples:

O123456	Illegal (as an identifier)	SAM	Legal
O12KK3	Legal	LEN32	Legal
O123	Legal	CALL	Illegal
A	Legal		

Alphanumeric Identifiers are used for:

Formal parameters

Variables

Subprograms

Main programs

Input/output units

Labeled common blocks

2.2.2 STATEMENT IDENTIFIER

Statements are identified by unsigned numbers, 1-5 digits, which can be referred to from other sections of the program. A statement identifier (from 1-99999) may be placed anywhere in columns 1-5 of the initial line of a statement. Leading zeros and trailing blanks are ignored. In any given program or subprogram, each statement identifier must be unique.

2.3 CONSTANTS

Seven types of constants are used in FORTRAN: integer, octal, real, double precision, complex, Hollerith, and logical. The type of a constant is determined by its form. The computer word structure for each type is listed in Appendix E.

2.3.1 INTEGER CONSTANTS

An integer constant, N, is a string of up to 18 decimal digits in the range $-(2^{59}-1) \leq N \leq (2^{59}-1)$. The maximum value of the result of integer addition or subtraction must not exceed $2^{59}-1$. Subscripts and DO-index are limited to $2^{17}-2$.

Examples:

63	3647631	314159265	574396517802457165
247	464646464		

During execution, the maximum allowable value is $2^{48}-1$ when an integer constant is converted to real. If the result is greater than $2^{48}-1$, bits 48-58 will be ignored and errors may result. The maximum value of the operands and the result of integer multiplication or division must be less than $2^{48}-1$. High order bits will be lost if the value is larger, but no diagnostic is provided.

2.3.2 OCTAL CONSTANTS

An octal constant consists of 6 to 20 octal digits preceded by the letter O or 1 to 20 octal digits suffixed with a B. The form is:

$On_1 \ldots n_i$

$n_1 \ldots n_iB$

Both forms of a constant are assigned logical mode; the second form may be used only in arithmetic or DATA statements. The constants are right justified with zero fill. If the constant exceeds 20 digits or if a non-octal digit appears, a compiler diagnostic is provided.

Examples:

O000077777777700000000

O7777700077777

O2323232323232323

O000077

O7777777777777700

2374216B

777776B

777000777000777B

2.3.3 REAL CONSTANTS

A real constant is represented by a string of 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

The base is n; s is the exponent to the base 10; the plus sign may be omitted if s is positive. The range of a non-zero constant is approximately 10^{-294} to 10^{+322}. If the range is exceeded, a compiler diagnostic is provided.

All real numbers are carried in normalized form.

Examples:

3.E1 (means 3.0 x 10^1;i.e., 30.)

3.1415768

314.0749162

-3.141592E+279

31.41592E-01

.31415E01

.31415E+01

2.3.4 DOUBLE PRECISION CONSTANTS

A double precision constant is a string of digits represented internally by two words. The forms are similar to real constants, the base is n; s is the exponent to the base 10.

.nD±s   n.nD±s   n.D±s   nD±s

The D must always appear. The plus sign may be omitted for positive s. The range of non-zero constant is, approximately, from 10^{-294} to 10^{+322}; if the range is exceeded, a compiler diagnostic is provided.

Examples:

```
3.1415927D          3141.593D3
3.1416D0            31416.D-04
3141.593D-03
```

2.3.5 COMPLEX CONSTANTS

A complex constant is represented by a pair of real constants separated by a comma and enclosed in parentheses (r_1, r_2); r_1 represents the real part of the complex number, r_2 the imaginary part. Either constant may be preceded by a minus sign.

If the range of the real numbers comprising the constant is exceeded, a compiler diagnostic is provided. Diagnostics also occur when the pair contains integer constants, including (0,0).

Examples:

FORTRAN Representation	Complex Number
(1., 6.55)	1. + 6.55i
(15., 16.7)	15. + 16.7i
(-14.09, 1.654E-04)	-14.09 + .0001654i
(0., -1.)	0. - 1.0i

2.3.6 HOLLERITH CONSTANTS

A Hollerith constant is a string of FORTRAN characters of the form hHf; h is an unsigned decimal integer between 1 and n representing the length of the field f. The maximum number of characters allowed in a Hollerith constant of H form depends upon its usage; n is limited to 10 characters when used in an expression. In a properly formed DATA statement it is limited only by the number of characters that can be contained in up to 19 continuation lines. Spaces are significant in the field f. When h is not a multiple of 10, the last computer word is left justified with blank fill. Alternate forms are nLf (left justified) and nRf (right justified) Hollerith constants with zero fill for incomplete words. A maximum of 10 characters is allowed in expressions for these forms. If more than 10 characters are used in a DATA statement for such a constant, only the last word has the zero fill. They may be used in an arithmetic statement. Hollerith constants are stored internally in 6000 Series console display code.

Examples:

6HCOGITO

4HERGO

3HSUM

5RSUMbb = 00000SUMbb

12HCONTROL DATA

5LSUMbb = SUMbb00000

1H)

3LbTT = bTT0000000

A statement of the form: I=(+5HABCDE) is permitted as a Hollerith constant. A semicolon (display code 77) cannot appear in Hollerith constants since this bit configuration is recognized as a Hollerith field terminator.

2.3.7 LOGICAL CONSTANTS

A false constant is stored internally as binary zero. A true constant is stored internally as the one's complement of binary zero. Logical constants may be in the following forms:

.TRUE. or .T.

.FALSE. or .F.

2.4 VARIABLES

FORTRAN recognizes simple and subscripted variables; a simple variable represents a single quantity; it references a storage location. The value specified by the name is always the current value stored in the location. Variables are identified by a symbolic name of 1-7 alphanumeric characters, the first of which must be alphabetic.

The type of variable is defined in one of two ways:

Explicit. Variables may be declared a particular type with the FORTRAN type declarations.

Implicit. A variable not defined in a FORTRAN type declaration is assumed to be integer if the first character of the symbolic name is I, J, K, L, M, or N.

Example:

I15, JK26, KKK, NP362L, M

All other variables not declared in a FORTRAN type declaration are assumed to be real.

Examples:

TEMP, ROBIN, A55, R3P281

2.4.1 INTEGER VARIABLES

Integer variables can be defined explicitly or implicitly; values may be in the range $-(2^{59}-1) \leq I \leq (2^{59}-1)$. The maximum allowable value of an integer variable depends on usage. The result of conversion from integer to real, of integer multiplication, integer division or input/output under the I-format specification is limited to $2^{48}-1$; the result of integer addition or subtraction can be as great as $2^{59}-1$. Subscripts and DO-indexes are limited to $2^{17}-2$. Each integer variable occupies one word in storage.

Examples:

N
ITEM
M58A
NEGATE
K2S04
M58

2.4.2 REAL VARIABLES

The type of a real variable may be explicit or implicit; the value must be in the range $10^{-294} < |r| < 10^{+322}$ with approximately 14 significant digits. Each real variable is stored in 6000 Series floating-point format and occupies one word.

Examples:

VECTOR
YBAR
A62597
BARMIN
X
X74A

The variable, r, may have any of the following values:

$$-10^{+322} < r < -10^{-294},\ r = 0, 10^{-294} < r < 10^{+322}.$$

2.4.3 DOUBLE PRECISION VARIABLES

Double precision variables must be defined explicitly by a type declaration. Each double precision variable occupies two words of storage and can assume values in the range $10^{-294} \leq |d| \leq 10^{+322}$ with approximately 29 significant digits.

2.4.4 COMPLEX VARIABLES

Complex variables must be explicitly defined by a type declaration. A complex variable occupies two words in storage. Each word contains a number in real variable format. This ordered pair of real variables (C_1,C_2) represents the complex number: $C_1 + i\ C_2$

2.4.5 LOGICAL VARIABLES

Logical variables must be defined explicitly by a type declaration. Each logical variable occupies one word of storage; it can assume the value true or false. A logical variable with a positive zero value is false; any other value is true. When a logical variable appears in an expression whose dominant mode is real, double, or complex, it is not packed and normalized prior to its use in the evaluation of an expression (as is the case with an integer variable).

2.5 SUBSCRIPTED VARIABLE

A subscripted variable may have one, two, or three subscripts enclosed in parentheses; more than three produce a compiler diagnostic. Subscripts can be expressions in which operands are simple integer variables and integer constants and operators are addition, subtraction, multiplication, and division only. Such expressions must result in positive integers; use of other values such as zero, real, negative integer, complex, logical may invalidate results.

When a subscripted variable represents the entire array, the subscripts are the dimensions of the array. When a subscripted variable references a single element in an array, the subscripts describe the relative location of the element in the array.

Valid Subscripted Variables	Invalid Subscripted Variables
A(I, J)	FRAN (0)
B(I+2, J+3, 2*K+1)	P(3. 5)
Q(14)	Z14(-4)
STRING (3*K*ILIM+3)	EVAL(2+(3. 1, 2. 5))
Q(1, 4, 2)	MAX3(1. GE. K)
	I(2, -5, 3)

2.6 ARRAYS

An array is a block of successive storage locations. The entire array may be referenced by the array name without subscripts (I/O lists and Implied DO-loop notation). Arrays may have one, two, or three dimensions; the array name and dimensions must be declared in a DIMENSION, COMMON, or TYPE declaration prior to the first program reference to that array.

Each element in an array may be referenced by the array name plus a subscript notation. Program execution errors may result if subscripts are larger than the dimensions initially declared for the array. The maximum number of elements in an array is the product of the dimensions.

2.6.1 ARRAY STRUCTURE

Array elements are stored by columns in ascending locations. In the array declared as A(3,3,3):

A_{111}	A_{121}	A_{131}
A_{211}	A_{221}	A_{231}
A_{311}	A_{321}	A_{331}

A_{112}	A_{122}	A_{132}
A_{212}	A_{222}	A_{232}
A_{312}	A_{322}	A_{332}

A_{113}	A_{123}	A_{133}
A_{213}	A_{223}	A_{233}
A_{313}	A_{323}	A_{333}

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

$A_{111} \rightarrow L$ $\quad A_{121} \rightarrow L+3 \; . \; . \; . \; A_{133} \rightarrow L+24$

$A_{211} \rightarrow L+1$ $\quad A_{221} \rightarrow L+4 \; . \; . \; . \; A_{233} \rightarrow L+25$

$A_{311} \rightarrow L+2$ $\quad A_{321} \rightarrow L+5 \; . \; . \; . \; A_{333} \rightarrow L+26$

Array allocation is discussed 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 the array.

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

The quantity enclosed by the outer parentheses is the subscript expression. E is the element length—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), the location of A(2,2,3) with respect to A(1,1,1) is:
>
> $$\text{Locn } A(2,2,3) = (\text{Locn } A(1,1,1) + (2-1+3(1+3(2))))*1$$
> $$= (L + 22)*1 = L + 22$$

FORTRAN 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)
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)
and for	$A(D_1)$	
	A(I)	implies A(I)
	A	implies A(1)

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.

EXPRESSIONS 3

An expression is a constant, variable (simple or subscripted), function, 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 type of expression is associated with a group of operators and operands.

3.1 ARITHMETIC EXPRESSIONS

An arithmetic expression can contain the following operators:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Operands are:

Constants

Variables (simple or subscripted)

Evaluated functions

Any unsigned constant, variable, or function 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
```

If op is a valid operator and X and Y are valid expressions, then X op op Y is never a valid expression

Examples:

```
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) )
```

3.1.1 ARITHMETIC EVALUATION

The hierarchy of arithmetic evaluation is:

**	exponentiation	class 1
/	division	class 2
*	multiplication	
+	addition	class 3
-	subtraction	

In an expression with no parentheses or within a pair of parentheses in which unlike classes of operators appear, evaluation proceeds in the above order. In expressions containing like classes of operators, evaluation proceeds from left to right. For example, A**B**C is evaluated as (A**B)**C.

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

When writing an integer expression, it is important to remember not only the left-to-right scanning process but also that dividing an integer quantity by an integer quantity always yields a truncated result; thus 11/3 = 3. The expression I*J/K may yield a different result than the expression J/K*I.

For example, 4*3/2 = 6 but 3/2*4 = 4.

Examples:

In the following examples, R indicates an intermediate result in evaluation:

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

A**B → R_1

R_1/C → R_2

D*E → R_3

R_3*F → R_4

R_4+R_2 → R_5

R_5-G → R_6 evaluation completed

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

E*F-G → R_1

C+D → R_2

A**B → R_3

R_3/R_2 → R_4

R_4*R_1 → R_5 evaluation completed

H(I3)+C(I,J+2)*(COS(Z))**2 is evaluated:

COS(Z) → R_1

R_1**2 → R_2

R_2*C(I,J+2) → R_3

R_3+H(13) → R_4 evaluation completed

The following is an example of an expression with embedded parentheses.

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

C/D → R_1

R_1-E → R_2

R_2+B → R_3

R_3*A → R_4 evaluation completed

(A*(SIN(X)+1.)-Z)/(C*(D-(E+F))) is evaluated:

E+F → R_1

D-R_1 → R_2

C*R_2 → R_3

SIN(X) → R_4

$R_4+1. \rightarrow R_5$

$A*R_5 \rightarrow R_6$

$R_6-Z \rightarrow R_7$

$R_7/R_3 \rightarrow R_8$ evaluation completed

3.1.2 MIXED-MODE ARITHMETIC EXPRESSIONS

Mixed-mode arithmetic with the exception of exponentiation is completely general; however, most applications probably mix operand types, real and integer, real and double, or real and complex. The relationship between the mode of an evaluated expression and the types of operands it contains is established as follows.

Order of dominance of the operand types within an expression from highest to lowest:

Complex

Double

Real

Integer

Logical

Simple double precision expressions are not evaluated by closed subroutines, but by in-line arithmetic instructions.

The type of an evaluated arithmetic expression is the mode of the dominant operand type.

In expressions of the form A**B, the following rules apply:

If B is preceded by a unary minus operator, the form is A**(-B).

B is treated as an integer if type logical.

For the various operand types, the type relationships of A**B are:

Type of B

		I	R	D	C	L
Type of A	I	I	n	n	n	n
	R	R	R	D	n	n
	D	D	D	D	n	n
	C	C	n	n	n	n
	L	I	n	n	n	n

mode of A**B

n indicates an invalid operation

For example, if A is real and B is integer, the mode of A**B is real.

Examples:

1) Given real A, B; integer I, J. The type of expression A*B-I+J is real because the dominant operand type is real.

   The expression is evaluated:

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

2) The use of parentheses can change the evaluation. A,B,I,J are defined as above. A*B-(I-J) is evaluated:

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

3) Given complex C,D, real A,B. The type of the expression A* (C/D)+B is complex because the dominant operand type is complex. The expression is evaluated:

$C/D \rightarrow R_1$	complex
Convert A to	complex
$A*R_1 \rightarrow R_2$	complex
Convert B to	complex
$R_2+B \rightarrow R_3$	complex

4) Consider the expression C/D+(A-B) where the operands are defined in 3 above. The expression is evaluated:

$A-B \rightarrow R_1$	real
$C/D \rightarrow R_2$	complex
Convert R_1 to	complex
$R_1+R_2 \rightarrow R_3$	complex

5) Mixed-mode arithmetic with all types is illustrated by this example:

Given: the expression C*D+R/I-L

C	Complex
D	Double
R	Real
I	Integer
L	Logical

The dominant operand type in this expression is complex; therefore, the evaluated expression is complex.

Evaluation:

Round D to real and affix zero imaginary part.

Convert D to	complex
C*D → R_1	complex
Convert R to	complex
Convert I to	complex
R/I → R_2	complex
R_2+R_1 → R_3	complex
R_3-L → R_4	complex

If the same expression is rewritten with parentheses as C*D+(R/I-L) the evaluation proceeds:

Convert I to	real
R/I → R_1	real
R_1-L → R_2	real
Convert D to	complex
C*D → R_3	complex
Convert R_2 to	complex
R_2+R_3 → R_4	complex

3.2 RELATIONAL EXPRESSIONS

A relational expression has the form:

a_1 op a_2

The a's are arithmetic expressions; op is an operator belonging to the set:

.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to

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 the value plus zero; a true relational expression is assigned the value minus zero (all one bits).

Relations are evaluated as illustrated in the relation p.EQ.q. This is equivalent to the question: does p - q = 0 ?

The difference is computed and tested for zero. If the difference is zero or minus zero, the relation is true. If the difference is not zero or minus zero, the relation is false. Relational expressions are converted internally to arithmetic expressions according to the rules of mixed-mode arithmetic. These expressions are evaluated and compared with zero to determine the truth value of the corresponding relational expression. When complex expressions are tested for zero or minus zero, only the real part is used in the comparison. For double precision numbers, only the most significant part is used in the comparison.

Relational expressions of the following forms are allowed:

I.LT.R

I.LT.D

I.LT.C (Real part of C is used)

I is integer, R is real, D is double precision and C is complex.

Order of dominance of the operand types within an expression is the order stated in mixed-mode arithmetic expressions.

The relation of the form I.GE.0 is treated as true if I assumes the value -0.

a_1 op a_2 op a_3. . . is not a valid expression.

A relation of the form a_1 op a_2 is evaluated from left to right. The relations a_1 op a_2,a_1 op (a_2), (a_1) op a_2, (a_1) 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) )
```

3.3 LOGICAL EXPRESSIONS

A logical expression has the general form:

L_1 op L_2 op L_3. . .

The terms L_i are logical variables, logical constants, or relational expressions and op is the logical operator .AND. indicating conjunction or .OR. indicating disjunction.

The logical operator .NOT. indicating negation appears in the form:

.NOT. L_1

The value of the expression is examined. If the value is equal to plus zero, the logical expression has the value false. All other values are considered true.

The hierarchy of logical operations is:

First	.NOT.	or	.N.
then	.AND.	or	.A.
then	.OR.	or	.O.

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_1 op op L_2 is never legitimate.

.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 → Q) may be written in two ways:
.NOT.(P.AND.(.NOT.Q))
.N.(P.A.(.N.Q))

3.4 MASKING EXPRESSIONS

The masking expression is a generalized form of the logical expression in which the variables may be types other than logical.

In a FORTRAN masking expression, 60-bit logical arithmetic is performed bit-by-bit on the operands within the expression. The operands may be any type variable, constant, or expression. No mode conversion is performed during evaluation. If the operand is complex, operations are performed on the real part. 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.	complement the operand
.AND. or .A.	form the bit-by-bit logical product of two operands
.OR. or .O.	form the bit-by-bit logical sum of two operands

The operations are described below:

p	v	p .AND. v	p .OR. v	.NOT. p
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Let B_i be masking expressions, variables or constants of any type except 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 . . .

Arithmetic expressions appearing in masking statements must be enclosed in parentheses. e.g. E=(E*100B).OR.F.

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

REPLACEMENT STATEMENTS 4

4.1 ARITHMETIC REPLACEMENT

The general form of the arithmetic replacement statement is A = E, where E is an arithmetic expression and A is any variable name, simple or subscripted. The operator = means that A is replaced by the value of the evaluated expression, E, with conversion for mode if necessary.

Examples:

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

4.2 MIXED-MODE REPLACEMENT

The type of an evaluated expression is determined by the type of the dominant operand. This, however, does not restrict the types that identifier A may assume. A complex expression may replace A, even if A is real. The following chart shows the A = E relationship for all the standard modes. The mode of A determines the mode of the statement.

When all the operands in the expression E are logical, the expression is evaluated as if all the logical operands were integers.

For example, if L_1, L_2, L_3, L_4 are logical variables, R is a real variable, and I is an integer variable, then $I = L_1*L_2 + L_3 - L_4$ is evaluated as if the L_i were all integers and the resulting value is stored as an integer in I.

$R = L_1*L_2 + L_3 - L_4$ is evaluated as stated above, but the result is converted to a real (a floating point quantity) before it is stored in R.

When a mode conversion is made from real, double precision, or complex to integer and the real number (or the real portion of the complex number) is in the range $-1<R<0$ the following conversion is made:

Real Value	Resulting Integer
$-1 < R < -(2^{-17})$	-0
$-(2^{-17}) \leq R < 0$	0

Type of A	Type of Expression E			
	Complex	Double Precision	Real	Integer
Complex	A = E	Set A = most significant half of E A_{real} = E A_{image} = 0	A_{real} = E A_{imag} = 0	Convert E to Real A_{real} = E A_{imag} = 0
Double Precision	A = E_{real} less significant is set to zero	A = E	A = E less significant is set to zero	Convert E to Real A = E less significant is set to zero
Real	A = E_{real}	Set A = most significant half of E A = E	A = E	Convert E to Real A = E
Integer	Truncate E_{real} to Integer A = E	Truncate E to 48 bit integer A = E	Truncate E to Integer A = E	A = E
Logical	If $E_{real} \neq$ 0, A = 1 If E_{real} = 0, A = 0	If E ≠ 0, A = 1 If E = 0, A = 0	If E ≠ 0, A = 1 If E = 0, A = 0	If E ≠ 0, A = 1 If E = 0, A = 0

Examples:

Given:	C_i, A_1	Complex
	D_i, A_2	Double
	R_i, A_3	Real
	I_i, A_4	Integer
	L_i, A_5	Logical

1. $A_1 = C_1*C_2 - C_3/C_4$ (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. $A_3 = C_1$ 4.4000+000 = (4.4,2.1)

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

3. $A_3 = C_1*(0. ,-1.)$ 2.1000+000 = (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. $A_4 = R_1/R_2*(R_3-R_4)+I_1-(I_2*R_5)$ 13=8.4/4.2*(3.1-2.1)+ 14-(1*2.3)

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

5. $A_2 = D_1**2*(D_2+(D_3*D_4)) +(D_2*D_1*D_2)$ 4.96800000000000+001= 2.0D**2*(3.2D+(4.1D*1.0D)) +(3.2D*2.0D*3.2D)

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

6. $A_5 = C_1*R_1-R_2+I_1$ 1=(4.4,2.1)*8.4-4.2+14

   The expression is complex. Since A_5 is logical, the real part of the evaluated expression replaces A_5. If the real part is zero, zero replaces A_5.

4.3 LOGICAL REPLACEMENT

The general form of the logical replacement statement is L = E, where L is a logical variable and E may be a logical, relational, or arithmetic 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)
```

4.4 MASKING REPLACEMENT

The general form of the masking replacement statement is M = E. E is a masking expression, and M is a variable of any type except logical. No mode conversion is made during the replacement.

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
           D = (B.LE.C) .AND. (C .LE. E) .AND. .NOT.I
```

4.5 MULTIPLE REPLACEMENT

Expressions of the form

```
A=B=C=D=3.0*X
```

are permissible and result in code which is equivalent to the expressions:

```
D=3.0*X
C=D
B=C
A=B
```

TYPE DECLARATIONS AND STORAGE ALLOCATION 5

5.1 TYPE DECLARATION

The type declaration statement provides the compiler with information on the structure of variable and function identifiers.

Statement	Characteristics	
COMPLEX list	2 words/element	Floating Point
DOUBLE PRECISION list or DOUBLE list	2 words/element	Floating Point
REAL list	1 word/element	Floating Point
INTEGER list	1 word/element	Integer
LOGICAL list	1 word/element	Logical

TYPE may precede any of the above statements.

DOUBLE may replace DOUBLE PRECISION in any FORTRAN statement in which the latter is allowed.

List is a string of identifiers separated by commas; integer constant subscripts are permitted. For example:

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

The type declaration is non-executable and must precede the first reference to the variable or function in a given program. If an identifier is declared in two or more type declarations, the first declaration holds until the second is read, the second holds until the third, etc. However, the second and ensuing declarations will result in informative diagnostics.

An identifier not declared in a type declaration is type integer if the first letter of the name is I, J, K, L, M, N; for any other letter, it is type real.

When subscripts appear in the list, the associated identifier is the name of an array, and the product of the subscripts determines the amount of storage to be reserved for that array. By this means, dimension and type information are given in the same statement. In this case no DIMENSION statement is needed; in fact it is not allowed.

Examples:

```
COMPLEX A412,DATA,DRIVE,IMPORT
DOUBLE PRECISION PLATE,ALPHA(20,20),B2MAX,F60,JUNE
REAL I,J(20,50,2),LOGIC,MPH
INTEGER GAR(60),BETA,ZTANK,AGE,YEAR,DATE
LOGICAL DISJ,IMPL,STROKE,EQUIV,MODAL
DOUBLE RL,MASS(10,10)
```

5.2 DIMENSION DECLARATION

A subscripted variable represents an element of an array of variables. Storage is reserved for arrays by the non-executable statements DIMENSION, COMMON, or a type statement.

The standard form of the DIMENSION declaration is:

DIMENSION $v_1, v_2, \ldots, v_n$

The variable names v_i may have 1, 2, or 3 integer constant subscripts separated by commas, as in SPACE (5, 5, 5). Under certain conditions within subprograms only, the subscripts may be constants or variables.

Example:

```
DIMENSION A(10),B(20,3)
```

The DIMENSION declaration is non-executable and it must precede the first reference to the array in a given program. The DIMENSION statement should precede the first executable statement and will result in an informative diagnostic otherwise.

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 provided.

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

In the above declarations, the number of elements in the array ATOM is 200. Two words are used to contain a complex element; therefore, the number of computer words reserved is 400. 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, the first declaration holds and an informative diagnostic is provided.

Examples:

```
DIMENSION A(20,2,5)

DIMENSION MATRIX(10,10,10),VECTOR(100),ARRAY(16,27)
```

5.2.1 VARIABLE DIMENSIONS

When an array identifier and some or all dimensions appear as formal parameters in a function or subroutine, the dimensions may be assigned through the actual parameter list accompanying the function reference or subroutine call. Dimensions must not exceed the maximum array size specified by the DIMENSION declaration in the calling program.†

Example:

```
SUBROUTINE X(A, L, M)

DIMENSION    A(L, 10, M)
```

5.3 COMMON DECLARATION

The COMMON declaration provides up to 61 blocks of storage that can be referenced by more than one subprogram. The declaration reserves blank, numbered, and labeled blocks. Starting addresses for these blocks are indicated on the core map.

Areas of common information may be specified by the declaration:

$\text{COMMON}/i_1/\text{list}_1/i_2/\text{list}_2\ldots$

The common block identifier, i, may be 1-7 characters. If the first character is alphabetic, the identifier denotes a labeled common block; remaining characters may be alphabetic or numeric. If the first character is numeric, remaining characters must be numeric and the identifier denotes a numbered common block. Leading zeros in numeric identifiers are ignored. Zero by itself is an acceptable numbered common identifier.

† See Variable Dimensions in Subprograms in Chapter 7.

Example:

```
COMMON/200/A, B, C
```

The following are common identifiers:

Labeled	Numbered
AZ13	1
MAXIM	146
Z	6600
XRAY	0

A common statement without a label, or with just blanks between the separating slashes is treated as a blank common block, for example:

```
COMMON / / A, B, C    or    COMMON X, Y, Z(5)
```

List_i is a string of identifiers representing simple and subscripted variables; formal parameters are not allowed. If a non-subscripted array name appears in the list, the dimensions must be defined by a type or DIMENSION declaration in that program. If an array is dimensioned in more than one declaration, a compiler diagnostic is issued. The order of simple variables or array storage within a common block is determined by the sequence in which the variables appear in the COMMON statements.

Numbered common is treated as labeled common by the loader. The total of labeled and numbered common blocks is limited to 61. Labeled and numbered common blocks may be preset; data stored in them by DATA declarations is made available to any subprogram using the appropriate block. Data may not be entered into blank common blocks by the DATA declaration.

Examples:

1. COMMON/BLK/A(3)

   DATA A/1.,2.,3./

2. COMMON/100/I(4)

   DATA I/4,5,6,7/

COMMON is non-executable and can appear anywhere in the program. Any number of blank COMMON declarations may appear in a program. If DIMENSION, COMMON or type declarations appear together, the order is immaterial.

Since labeled and numbered common block identifiers are used only within the compiler, they may be used elsewhere in the program as other kinds of identifiers except subroutine names in the same job. An identifier in one common block may not appear in another common block. (If it does, the name is doubly defined.)

At the beginning of program execution, the contents of all common areas are unpredictable except labeled common areas specified in a DATA declaration.

Examples:

```
COMMON A, B, C        )
                      }  Blank Common
COMMON/ /E, F, G, H   )
COMMON/BLOCKA/A1(15), B1, C1/BLOCKD/DEL(5, 2), ECHO
COMMON/VECTOR/VECTOR(5), HECTOR, NECTOR
COMMON/9999/AX, BX, CX
```

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

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

	Block A	
origin	Q(1)	
	Q(2)	
	Q(3)	
	Q(4)	
	R(1)	
	R(2)	
	R(3)	
	R(4)	
	S (1)	real part
	S (1)	imaginary part
	S (2)	real part
	S (2)	imaginary part

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

```
COMMON/SUM/A,B,C,D    (main program)
COMMON/SUM/E(3),D     (subprogram)
```

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

Each subprogram using a common block assigns the allocation of words in the block. The identifiers used within the block may differ as to name, type, and number of elements; but the block identifier must remain the same.

Example:

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

The length of the block named TEST is 40 computer words. The length of the block numbered 36 is 3 computer words.

The subprogram may rearrange the allocation of words as in:

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

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

The length of a common block other than blank common must not be increased by subprograms using the block unless that subprogram is loaded first by the SCOPE loader. The symbolic names used within the block may differ, however, as shown above.

5.4 EQUIVALENCE DECLARATION

The EQUIVALENCE declaration permits variables to share locations in storage. The general form is:

```
EQUIVALENCE (A,B,...),(A1,B1,...),...
```

(A,B,...) is an equivalence group of two or more simple or subscripted variable names; formal parameters are not allowed. A multiply subscripted variable can be represented by a singly subscripted variable. The correspondence is:

A(i,j,k) is the same as A ((the value of (i+(j-1)*I+(k-1)*I*J))*E)

where E is 1 or 2 depending on A's word length. i, j, k are integer constants; I and J are the integer constants appearing in DIMENSION A(I,J,K). For example, in DIMENSION A(2,3,4), the element A(1,1,2) can be represented by A(7).

EQUIVALENCE is most commonly used when two or more arrays can share the same storage locations. The lengths need not be equal.

Example:

```
     DIMENSION A(10,10),I(100)
     EQUIVALENCE (A,I)
5    READ 10, A
              .
              .
              .
6    READ 20, I
```

The EQUIVALENCE declaration assigns the first element of array A and array I to the same storage location. The READ statement 5 stores the A array in consecutive locations. Before statement 6 is executed, all operations using A should be completed since the values of array I are read into the storage locations previously occupied by A.

Variables requiring two memory positions which appear in EQUIVALENCE statements must be declared to be COMPLEX or DOUBLE prior to their appearance in such statements.

Example:

```
COMPLEX DAT,BAT
DIMENSION DAT(10,10),BAT(10,10),CAT(10,10)
DOUBLE PRECISION CAT
COMMON/IFAT/FAT(2,2)
EQUIVALENCE (DAT(2,1),FAT(2,2)),(CAT,BAT)
.
.
.
```

EQUIVALENCE is non-executable and can appear anywhere in the program or subprogram. However, if it appears after the first executable statement, an informative diagnostic is provided.

Any variable may be made equivalent to any other variable, provided that no two variables in any one group are in COMMON. The variables may be with or without subscript. In FORTRAN II, equivalence groups can reorder the common variables and arrays, and more than one variable in an equivalence group may be in common. The following examples illustrate changes in block lengths caused by the EQUIVALENCE declaration.

Given: Arrays A and B

Sa subscript of A

Sb subscript of B

Examples:

A and C in common, B not in common

Sb ≤ Sa is a permissible subscript arrangement

Sb > Sa is not

The design of this compiler prevents the following use of EQUIVALENCE

```
DIMENSION FAT(6)
COMMON/IFAT/SKINNY
EQUIVALENCE (SKINNY, FAT(n))
```

The latter statement will be flagged fatally if n > 1.

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)	

5.5 DATA DECLARATION

Values may be assigned to program variables or labeled common variables with the DATA declaration:

$$\text{DATA } d_1,\ldots,d_n/a_1,k*a_2,\ldots,a_n/,d_1,\ldots,d_n/a_1,\ldots,a_n/,\ldots$$

d_i identifiers representing simple variables, array names, or variables with integer constant subscripts or integer variable subscripts (implied DO-loop notation).

a_i literals and signed or unsigned constants.

k integer constant repetition factor that causes the literal following the asterisk to be repeated k times. If k is non-integer, a compiler diagnostic occurs.

A semicolon cannot be used in the character string of data entered under L, R or H control.

Data is non-executable and can appear anywhere in the program or subprogram. When DATA appears with DIMENSION, COMMON, EQUIVALENCE, or a type declaration, the statement that dimensions any arrays used in the DATA statement must appear prior to the DATA statement. Variables in blank common or formal parameters may not be preset by a DATA declaration.

Only single-subscript, DO-loop-implying notation is permissible. This notation may be used for storing constant values in arrays.

Examples:

1.
```
DIMENSION GIB(10)
DATA (GIB(I), I=1, 10)/1.,2.,3.,7*4.32/

     Array GIB:   1.
                  2.
                  3.
                  4.32
                  4.32
                  4.32
                  4.32
                  4.32
                  4.32
                  4.32
```

2.
```
DIMENSION TWO(2,2)
DATA TWO(1,1),TWO(1,2),TWO(2,2),TWO(2,1)/1.,2.,3.,4./
```

Array TWO:	TWO(1,1)	1.
	TWO(2,1)	4.
	TWO(1,2)	2.
	TWO(2,2)	3.

3.
```
DIMENSION SINGLE(3,2)
DATA (SINGLE(I),I=1,6)/1.,2.,3.,1.,2.,3./
```

Array SINGLE:	SINGLE(1,1)	1.
	SINGLE(2,1)	2.
	SINGLE(3,1)	3.
	SINGLE(1,2)	1.
	SINGLE(2,2)	2.
	SINGLE(3. 2)	3.

In the DATA declaration, the type of the constant stored is determined by the structure of the constant rather than by the variable type in the statement. In DATA A/2/, an integer 2 replaces A, not a real 2 as might be expected from the form of the symbolic name A.

There should be a one-one correspondence between the variable names and the list. This is particularly important in arrays in labeled common. For instance:

```
COMMON/BLK/A(3),B

DATA A/1.,2.,3.,4./
```

The constants 1.,2.,3., are stored in array locations A,A+1,A+2; the constant 4. is discarded, B is unmodified and an error is issued. If this occurs unintentionally, errors may occur when B is referred to elsewhere in the program.

```
COMMON/TUP/C(3)

DATA C/1. ,2./
```

The constants 1. , 2. are stored in array locations C and C+1; the content of C(3), that is, location C+2, is not defined.

When the number of list elements exceeds the range of the implied DO, the excess list elements are not stored, and a diagnostic is issued.

```
DATA (A(I), I=1, 5, 1)/1. , 2. , . . . ,10./
```

The excess values 6. through 10. are discarded.

Examples:

```
1)   DATA LEDA, CASTOR, POLLUX/15,16.0,84.0/
          LEDA                          15
                                         .
                                         .
                                         .
          CASTOR                        16.0
                                         .
                                         .
                                         .
          POLLUX                        84.0

2)   DATA A(1,3)/16.239/
          ARRAY A
              A(1,3)                    16.239

3)   DIMENSION B(10)
          DATA B/O000077, O000064, 3*O000005, 5*O000200/
              ARRAY B                   O77
                                        O64
                                         O5
                                         O5
                                         O5
                                       O200
                                       O200
                                       O200
                                       O200
                                       O200

4)   COMMON/HERA/C(4)
          DATA C/3.6, 3*10.5/
              ARRAY C                   3.6
                                       10.5
                                       10.5
                                       10.5
```

```
5)  COMPLEX PROTER (4)
        DATA PROTER/4*(1.0,2.0)/
            ARRAY PROTER              1.0
                                      2.0
                                      1.0
                                      2.0
                                      1.0
                                      2.0
                                      1.0
                                      2.0
6)  DIMENSION MESAGE (3)
        DATA MESAGE/9HSTATEMENT,2HIS,10HINCOMPLETE/
            ARRAY MESAGE                         STATEMENT
                                                 IS
                                                 INCOMPLETE
```

Data declaration statements of the following forms may also be used to assign constant values to program or common variables at load time.

DATA (i_1=value list), (i_2=value list), . . .

DATA (i(j,k,l)=value list), . . .

j, k, l are integer constants.

The variable identifier, i, may be:

non-subscripted variable

array variable with constant subscripts

array name

array variable with integer variable quantifiers

The value list is either a single constant or set of constants whose number is equal to the number of elements in the named array.

List contains constants only and has the form:

$a_1,a_2, \ldots ,k(b_1,b_2, \ldots),c_1,c_2, \ldots$

k is an integer constant repetition factor that causes the parenthetical list following it to be repeated k times. If k is non-integer a compiler diagnostic is provided.

Examples:

```
COMMON/DATA/GIB
DATA ( (GIB(I), I=1,10)=1. , 2. , 3. , 7(4.32) )
COMMON/DATA/ROBIN(5,5,5)
DATA (ROBIN(4,3,2)=16.)
```

5.5.1 BLOCK DATA SUBPROGRAM

A block data subprogram may be used to enter data into labeled or numbered common prior to program execution in place of a DATA declaration and it may appear more than once in a FORTRAN program. If more than one BLOCK DATA subprogram is compiled, the user-supplied name is used to identify the binary records. If no name is specified, however, the binary records are named BLKDAT A, BLKDAT B, BLKDAT C, ... Also if segmentation is used, the SEGMENT card must specify which block data binary record is to be used.

```
BLOCK DATA n
    .
    .
    .
FORTRAN declaration statements only
    .
    .
    .
END
```

Where n is blank or any acceptable alphanumeric identifier beginning with a letter, all elements in the common blocks must appear in a COMMON declaration in the subprogram even if they are not in the DATA declaration.

Examples:

```
1)  BLOCK DATA
    COMMON/ABC/A(5),B,C/DEF/D,E,F
    COMPLEX D,E
    DOUBLE PRECISION F
    DATA (A(L),L=1,5)/2.3,3.4,3*7.1/,B/2034.756/,D,E,F/2*(1.0,2.5),
   17.86972415872D30/
    END

2)  BLOCK DATA HOODAR
    COMMON /DEF/G,H,I
        .
        .
        .
    END
```

CONTROL STATEMENTS 6

Program execution normally proceeds from one statement to the statement immediately following it in the program. Control statements can be used to alter this sequence or cause a number of iterations of a program section.

Control may be transferred to an executable statement only; a transfer to a non-executable statement results in a fatal diagnostic.

6.1 GO TO STATEMENTS

Program control is transferred to a statement other than the next statement in sequence by the GO TO statements.

6.1.1 UNCONDITIONAL GO TO

GO TO n

An unconditional transfer is made to the statement labeled n.

6.1.2 ASSIGNED GO TO

GO TO m, $(n_1, n_2, \ldots, n_m)$
GO TO m

This statement acts as a many-branch GO TO; m is a simple integer variable assigned an integer value n in a preceding ASSIGN statement. The n_i are statement labels. As shown, the parenthetical statement label list need not be present.

The comma after m is optional; however, when the list is omitted, the comma must be omitted. m cannot be defined as the result of a computation. No compiler diagnostic is given if m is computed, but the object code is incorrect. If an assignment has not been made for an assigned GO TO statement and m is equal to zero, a diagnostic is provided at object time. If m is non-zero, a valid assignment is assumed. FORTRAN does not preset all locations to zero.

6.1.3 ASSIGN STATEMENT

ASSIGN s TO m

This statement is used with the assigned GO TO statement; s is a statement label, m is a simple integer variable.

Example:

```
ASSIGN 10 TO LSWTCH
        .
        .
        .
GO TO LSWTCH, (5,10,15,20)
```

Control transfers to statement 10.

6.1.4 COMPUTED GO TO

GO TO $(n_1,n_2, . . . ,n_m)$,i

This statement acts as a many-branch GO TO; i is preset or computed prior to its use in the GO TO.

The n_i are statement labels and i is a simple integer variable. If $i < 1$ or if $i > m$, the transfer is undefined and an object time diagnostic will be issued indicating the point at which the error was detected. If $1 \leq i \leq m$, the transfer is to n_i.

The comma separating the statement number list and the index is optional.

Example:

```
N=3
  .
  .
  .
GO TO (100,101,102,103) N
```

Statement number 102 will be the selected control transfer.

For proper operations, i must not be specified by an ASSIGN statement. No compilation diagnostic is provided for this error, but the object code is incorrect.

Example:

```
    ISWICH = 1
    GO TO (10,20,30), ISWICH
          .
          .
          .
10  JSWICH = ISWICH + 1
    GO TO (11,21,31), JSWICH
```

Control transfers to statement 21.

6.2 IF STATEMENTS

Program control is transferred to a statement depending upon the condition of the computed results of the IF statements.

6.2.1 THREE-BRANCH ARITHMETIC IF

IF (c) n_1, n_2, n_3

c is an arithmetic expression, and the n_i are statement labels. This statement tests the evaluated expression c and jumps accordingly as follows:

$c < 0$	jump to statement n_1
$c = 0$	jump to statement n_2
$c > 0$	jump to statement n_3

In the test for zero, +0 = -0. When the mode of the evaluated expression is complex, only the real part is tested.

```
      IF(A*B-SINF(X) ) 10,20,10
      IF (I)5,6,7
  402 IF (A/B ** 2) 3, 6, 6
```

6.2.2 ONE-BRANCH LOGICAL IF

IF (ℓ) s

ℓ is a logical expression and s is any executable statement except another logical IF, a DO statement or an END. If ℓ is true (not plus zero), the statement s is executed. If ℓ is false (plus zero) the statement immediately following the IF statement is executed.

```
IF(A.LE.2.5) A=2.0
IF (VALUE*4.73.GT.PRICE.OR.VALUE.LT.150.0)BUY=.TRUE.
IF(P.AND.Q)GO TO 427
```

6.2.3 TWO-BRANCH LOGICAL IF

IF (ℓ) n_1, n_2

ℓ is a logical expression; n_i are statement labels.

The evaluated expression is tested for true (not plus zero) or false (plus zero) condition. If ℓ is true, the jump is to statement n_1. If ℓ is false, the jump is to statement n_2.

Example:

```
      IF(L)5,6
5     IF(K.EQ.100)70,60
6     IF(IJUMP.LT.K)10,11
```

6.3 DO STATEMENT

DO n i = m_1, m_2, m_3

This statement makes it possible to repeat groups of statements and to change the value of an integer variable during the repetition. n is the statement label ending the DO loop; i is the index variable (simple integer). m_i are the indexing parameters; they may be unsigned integer constants or simple integer variables no larger than $2^{17}-2$. m_1 is the initial value assigned to i, m_2 is the terminal value, and m_3 is the amount added to i after each time the DO loop is executed. If m_3 does not appear, it is assigned the value 1.

The DO statement (statement labeled n) and any intermediate statements constitute a DO loop; n may not be an arithmetic IF or GO TO statement, a two branch logical IF, a RETURN, another DO statement or a nonexecutable statement.

The indexing parameters m_1, m_2, m_3 are either unsigned integer constants or simple integer variables. Subscripted variables and negative or zero integer constants cause a diagnostic.

The indexing parameters m_1 and m_2, if variable, may assume positive or negative values or zero.

The values of m_1, m_2, and m_3 may be changed during the execution of the DO loop.

Examples:

1.
```
   DO 25 I=1,100
25 A(I)=A(I)+B(I)
```

The index variable I is incremented by one for each cycle until the DO loop is executed 100 times. The control is then transferred to the statement immediately following statement 25.

2.
```
    DO 12 I=1,10,2
    J=1+K
    X(J)=Y(J)
12  CONTINUE
```

I is set to the initial value of one and incremented by two on each of the following cycles. When the execution of the fifth cycle (I=9) is completed, control passes out of the DO loop.

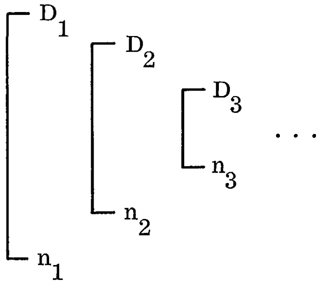
6.3.1 DO LOOP EXECUTION

The initial value of i, m_1, is increased by m_3 and compared with m_2 after executing the DO loop once, and if i does not exceed m_2, the loop is executed a second time. Then, i is again increased by m_3 and again compared with m_2; this process continues until i exceeds m_2. Control then passes to the statement immediately following n, and the DO loop is satisfied.

Should m_1 exceed m_2 on the initial entry to the loop, the loop is executed once and control is passed to the statement following n. When the DO loop is satisfied, the index variable i is no longer well defined. If a transfer out of the DO loop occurs before the DO is satisfied, the value of i is preserved and may be used in subsequent statements.

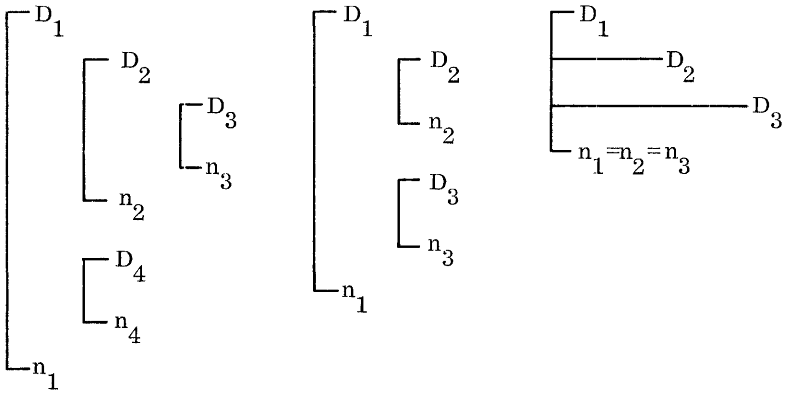
6.3.2 DO NESTS

When a DO loop contains another DO loop, the grouping is called a DO nest. Nesting may be to any level. The last statement of a nested DO loop must either be the same as the last statement of the outer DO loop or occur before it. If D_1,D_2, . . .D_m represent DO statements where the subscripts indicate that D_1 appears before D_2, D_2 appears before D_3 and $n_1, n_2, \ldots, n_m$ 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:



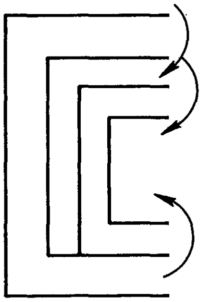
```
DO 1  I=1,10,2
      .
      .
      .
DO 2  J=1,5
      .
      .
      .
DO 3  K=2,8
      .
      .
      .
3 CONTINUE
      .
      .
      .
2 CONTINUE
      .
      .
      .
DO 4  L=1,3
      .
      .
      .
4  CONTINUE
      .
      .
      .
1  CONTINUE
```

```
DO  100  L=2,LIMIT
     .
     .
     .
DO  10  I=1,10
DO  10  J=1,10
     .
     .
     .
10 CONTINUE
     .
     .
     .
DO  20  K=K1,K2
     .
     .
     .
20  CONTINUE
     .
     .
     .
100  CONTINUE
```

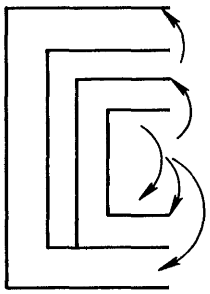
```
     DO  5  I=1,5
     DO  5  J=I,10
     DO  5  K=J,15
           .
           .
           .
5    A = B*C
```

6.3.3 DO LOOP TRANSFER

In a DO nest, a transfer may be made from one DO loop into a DO loop that contains it, but should not be made from the outer DO loop to the inner DO loop without first executing the DO statement of the inner DO loop.



Not Allowed



Allowed

One exception is allowed: once the DO statement has been executed and before the loop is satisfied, control may be transferred out of the DO range to perform some calculation and then transferred into the range of the DO.

Certain problems arise if the transfer from outside the range is to the terminal statement of the DO. A statement number terminating a DO loop not previously referenced except in a DO statement is ignored. A later reference to such a statement number causes a missing statement number indication. The statement number does not appear in the statement assignments list.

Examples 1 and 2 are permissible sets of statements, however, example 3 is not allowed.

Examples:

```
1)       K=0
         DO 3 I=1,10          (statement number 3 is referenced prior
         IF (K)3,2,3          to the end of the DO loop)
       2 X=1
         GO TO 1
       3 CONTINUE
       1 Y=M*X+B
         GO TO 3

2)       GO TO 2
       1 Y=M*X+B
         GO TO 3              (statement number 3 is referenced prior
       2 DO 3 I=1,10          to the end of the DO loop)
         X=I
         GO TO 1
       3 CONTINUE

3)     2 DO 3 I=1,10          (illegal)
         X=I
         GO TO 1
       3 CONTINUE
       1 Y=M*X+B
         GO TO 3              (statement number 3 is not referenced
                              prior to the end of the DO loop)
```

If more than one DO loop has the same terminal statement, the execution of the statements is properly satisfied only if the transfer occurred from the innermost DO.

Examples:

1) This example is acceptable since the statement GO TO 2 occurs from the innermost DO loop.

```
  GO TO 3
2 A(I)=A(I)+B(I,J)
  GO TO 1
3 DO 1 I=1,M
  A(I)=0
  DO 1 J=1,N
  GO TO 2
1 CONTINUE
```

2) This example is not acceptable since the statement GO TO 3 does not occur from the innermost DO loop.

```
3 IF(A(I))2,5,5      (statement number 2 causes index to
  DO 2 I=1,M         increment for inner DO loop, but not
  GO TO 3            for the outer DO)
5 DO 2 J=1,N
1 A(I)=A(I)+B(I,J)
2 CONTINUE
```

3) This example is acceptable since statement number 3 is in the range of the DO for I index and not in the range of the DO for J index.

```
4 IF(A(I))3,5,5
  DO 3 I=1,M
  GO TO 4
5 DO 2 J=1,N
1 A(I)=A(I)+B(I,J)
2 CONTINUE
3 CONTINUE
```

For the above examples, the terminal statement number of the DO loops must be referenced prior to the DO statement as a later reference to such a statement number produces a message indicating a missing statement number.

6.4 CONTINUE STATEMENT

n CONTINUE

The CONTINUE 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. The CONTINUE statement must contain a statement label n in column 1-5.

6.5 PAUSE STATEMENT

PAUSE

PAUSE n

n ≤ 5 octal digits without an O prefix or B suffix. PAUSE n stops program execution with the words PAUSE n displayed as a dayfile message. An operator entry from the console can continue or terminate the program. Program continuation proceeds with the statement immediately following PAUSE. If n is omitted, it is understood to be blank.

6.6 STOP STATEMENT

STOP

STOP n

n ≤ 5 octal digits without an O prefix or B suffix. When a STOP n statement is encountered, n is displayed in the dayfile, program execution is terminated and control is returned to the operating system. If n is omitted, it is assumed to be blank.

6.7 RETURN STATEMENT

A subprogram normally contains one or more RETURN statements to indicate the end of logic flow within the subprogram and return control to the calling program.

In function subprograms, control returns to the statement containing the function reference. In a subroutine subprogram, control returns to the next executable statement following the CALL. A RETURN statement in the main program causes an exit to the operating system.

6.8 END STATEMENT

END must be the final statement in a program or subprogram. It is executable in the sense that it effects termination of the program. The END statement may not be numbered.

The END statement may include the name of the program or subprogram which it terminates; however, any information appended to the END statement is ignored by the compiler.

In FORTRAN VI only, the END statement in a function or subroutine acts as a RETURN statement.

PROGRAM, FUNCTION, AND SUBROUTINE 7

A FORTRAN program consists of a main program with or without subprograms. Subprograms are of two kinds: subroutine and function. In the following discussions, the term subprogram refers to both. Subprograms may be compiled independently of the main program.

7.1 PROGRAM COMMUNICATION

The main program and subprograms communicate with each other via parameters and COMMON variables. Subprograms may call or be called by any other subprogram as long as the calls are nonrecursive; that is, if program A calls B, B may not call A. A calling program is a main program or subprogram that refers to another subprogram. A subroutine referenced by a program may not have the same name as the program.

7.2 SUBPROGRAM COMMUNICATION

Subprograms, functions, and subroutines use parameters as one means of communication. The parameters appearing in a subroutine call or a function reference are actual parameters. The corresponding arguments appearing with the program, subprogram, statement function, or library function name in the definition are formal parameters. One or more of the formal parameters or common variables can be used to return output to the calling program.

7.3 FORMAL PARAMETERS

Formal parameters may be the names of arrays, simple variables, library functions, and subprograms. Since formal parameters are local to the subprogram containing them, they may be the same as names appearing outside the procedure.

No element of a formal parameter list may appear in an EQUIVALENCE or DATA statement within the subroutine. If it does, a compiler diagnostic results.

When a formal parameter represents an array, it must be dimensioned within the subprogram. If it is not declared, the array name must appear without subscripts and only the first element of the array is available to the subprogram.

7.4 ACTUAL PARAMETERS

Permissible forms:

Arithmetic expression

Logical expression

Constant

Simple or subscripted variable

Array name

FUNCTION subprogram name

Library function and subroutine name

SUBROUTINE name

A calling program statement label, identified by suffixing the label with the character S. This form should be used only when calling DUMP or PDUMP.

A function name or a function reference may be used as an actual parameter. The function reference is a special case of an arithmetic expression.

Actual and formal parameters must agree in order, type and number.

I/O buffer names may not be used as actual parameters but the following is allowed:

```
PROGRAM MAIN (OUTPUT, TAPE 6 = OUTPUT)
        .
        .
        .
X = 50
        .
        .
        .
END
SUBROUTINE SUB(I, B)
        .
        .
        .
WRITE (I, 100)B
        .
        .
        .
END
```

7.5 MAIN PROGRAM

The first statement of a main program should be one of the following forms where name is an alphanumeric identifier of 1-7 characters. The parameter list is optional on all forms. If the first card of a program is not one of the following forms, a PROGRAM with a blank name and files of INPUT, OUTPUT, are assumed. If more files than INPUT, OUTPUT are necessary, a PROGRAM card is required.

For compilation in FORTRAN IV mode:

PROGRAM name ($f_1,\ldots,f_n$)
FORTRAN IV PROGRAM name ($f_1,\ldots,f_n$)
FORTRAN VI PROGRAM name ($f_1,\ldots,f_n$)

For compilation in FORTRAN II mode:

FORTRAN II PROGRAM name ($f_1,\ldots,f_n$)

Parameters f_i represent the names of all input/output files required by the main program and its subprograms; n must not exceed 50. Although these parameters may be changed at execution time; at compile time, they must satisfy the following conditions:

1. The file name INPUT (references standard input unit) must appear if any READ statement is included in the program or its subprograms.
2. The file name OUTPUT (references standard output unit) must appear if any PRINT statement is included in the program or its subprograms. OUTPUT is required for obtaining a listing of execution diagnostics.

   When logical file numbers are made equivalent to INPUT or OUTPUT, file names INPUT and OUTPUT must be declared in the PROGRAM statement card.

   Example:

   ```
   PROGRAM X (INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
   ```

3. The file name PUNCH must appear if any PUNCH statement is included in the program or its subprograms.
4. The file name TAPE i, must appear if a READ INPUT TAPE i, WRITE OUTPUT TAPE i, READ TAPE i, WRITE TAPE i, READ(i,n), WRITE (i,n), READ (i), or WRITE (i) statement is included in the program or its subprogram. (i is an integer.)
5. If I is an integer variable name for a READ INPUT TAPE I, WRITE OUTPUT TAPE I, READ TAPE I, WRITE TAPE I, READ (I,n) WRITE (I,n), READ (I), or WRITE (I) statement which appears in the program or its subprograms, the file names TAPE $i_1,\ldots,$ TAPE i_k must appear. The integers $i_1,\ldots,i_k$ must include all values which are assumed by the variable I. The file name TAPE I may not appear in the list of arguments to the main program.

File names may be made equivalent and their buffer lengths may be specified at compile time. See Appendix J for details on file name handling at execution time.

Example:

```
PROGRAM ORB (INPUT, OUTPUT=10000, TAPE1=INPUT, TAPE2=OUTPUT)
```

All input normally provided by TAPE 1 would be extracted from INPUT and all listable output normally recorded on TAPE 2 would be transmitted to the OUTPUT file. OUTPUT=10000 establishes an output buffer length of 10000_8. If buffer length is not indicated, a standard buffer size of 2022_8 is allocated.† Buffer length may not be less than 1002_8 words; for instance, PROGRAM X(INPUT=20) will cause a buffer of 1002_8 words to be formed. In the list of parameters, equivalenced file names must follow those to which they are made equivalent. Their corresponding parameter positions may not be changed at execution even though the names of the files to which they are made equivalent may be changed at that time.

7.6 SUBROUTINE SUBPROGRAM

A subroutine subprogram is a closed loop computational procedure which may return none, one or more values. A value or type is not associated with the subroutine name itself.

The first statement of a subroutine subprogram must have one of the following forms:

SUBROUTINE name $(p_1, \ldots, p_n)$

FORTRAN IV SUBROUTINE name $(p_1, \ldots, p_n)$

FORTRAN II SUBROUTINE name $(p_1, \ldots, p_n)$

FORTRAN VI SUBROUTINE name $(p_1, \ldots, p_n)$

name is an alphanumeric identifier and p_i are formal parameters; n may be 1 to 60.

The parameter list is optional. If the parameter list is not specified, the following form is allowed:

SUBROUTINE name

†The standard buffer size includes the 21_8 word file environment table of the buffer.

7.7 CALL STATEMENT

The executable statement in the calling program for referring to a subroutine is:

CALL name

or

CALL name $(p_1, \ldots, p_n)$

name is the name of the subroutine being called, and p_i are actual parameters; n is 1 to 60. The name should not appear in any declarative statement in the calling program, with the exception of the EXTERNAL statement when name is also an actual parameter.

The CALL statement transfers control to the subroutine. When a RETURN statement is encountered in the subroutine, control is returned to the next executable statement following the CALL statement in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until the DO loop is satisfied.

Examples:

1)
```
   SUBROUTINE BLDX(A,B,W)
   W=2.*B/A
   RETURN
   END
```

Calls

```
   CALL BLDX(X(I), Y(I),W)
   CALL BLDX(X(I)+H/2. ,Y(I)+C(J),PROX)
   CALL BLDX(SIN(Q5),EVEC(I+J),OVEC(L))
```

2)
```
   SUBROUTINE MATMULT
   COMMON/ITRARE/X(20,20),Y(20,20),Z(20,20)
   DO 10 I = 1,20
   DO 10 J = 1,20
   Z(I,J) = 0.
   DO 10 K=1,20
10 Z(I,J) = Z(I,J) + X(I,K)*Y(K,J)
   RETURN
   END
```

Operations in MATMULT are performed on variables contained in the common block ITRARE. This block must be defined in all calling programs.

```
COMMON/ITRARE/AB(20,20),CD(20,20),EF(20,20)
CALL MATMULT
```

3)
```
  SUBROUTINE AGMT(SUB,ARG)
  COMMON/ABL/XP(100)
  ARG = 0.
  DO 5 I = 1,100
5 ARG =ARG + XP(I)
  CALL SUB
  RETURN
  END
```

Here the formal parameter SUB is used to transmit another subprogram name. The call to subroutine AGMT might be call AGMT(MULT, FACTOR), where MULT is specified in an EXTERNAL statement. (section 7.8)

7.8 EXTERNAL STATEMENT

When the actual parameter list which calls a function or subroutine subprogram contains a function or subroutine name, that name must be declared in an EXTERNAL statement.

EXTERNAL name_1, name_2, . . .

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

Examples:

1) A function name used as an actual parameter requires an EXTERNAL statement.

<u>Calling Program Reference</u>

```
          .
          .
          .
EXTERNAL SIN
CALL PULL(SIN, R, Q)
          .
          .
          .
```

Called Subprogram

```
SUBROUTINE PULL(X,Y,Z)
                 .
                 .
                 .
Z=X(Y)
                 .
                 .
                 .
```

But a function reference used as an actual parameter does not need an EXTERNAL statement.

Calling Program Reference

```
                 .
                 .
                 .
CALL PULL(SIN(R),Q)
                 .
                 .
                 .
```

Called Subprogram

```
SUBROUTINE PULL(X,Z)
                 .
                 .
                 .
Z=X
                 .
                 .
                 .
END
```

2) A subroutine used as an actual parameter must have its name declared in an EXTERNAL statement in the calling program.

```
COMMON/ABL/ALST(100)
EXTERNAL RTENTA, RTENTB
CALL AGMT(RTENTA,V1)
CALL AGMT(RTENTB,V1)
```

When a subprogram name appears as an actual parameter, any parameters to be associated with a call of this subprogram must appear as separate actual parameters.

Example:

Calling Program

```
EXTERNAL ADDER
          .
          .
          .
CALL SUB (ADDER, A, B)
          .
          .
          .
```

Called Subprogram

```
SUBROUTINE SUB(X, Y, Z)
          .
          .
          .
CALL X(Y,Z)
          .
          .
          .
END
```

CALL SUB(ADDER(A, B)) would imply that ADDER is a function value, not a subroutine name.

7.9 ENTRY STATEMENT

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

```
ENTRY name
```

Name is an alphanumeric identifier, and 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. ENTRY may appear anywhere within the subprogram except it should not appear within a DO; ENTRY statement 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 may not be given 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)
```

7.10 LIBRARY SUBROUTINES

FORTRAN contains several built-in subroutine subprograms which may be referenced 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, the results are undefined and no diagnostic is provided.

CALL SLITET (i,j)

If sense light i is on, set j = 1, if sense light i is off, set j = 2; then turn sense light i off. i is 1 to 6. If i is out of the range, the results are undefined.

CALL SSWTCH (i, j)

If sense switch i is on, set j = 1; if sense switch i is off, set j = 2. i is 1 to 6. If i is out of the range, the results are undefined.

CALL OVERFL (j)[†]

If a floating point overflow condition exists, set j = 1; if no overflow exists, set j = 2; and set the machine to a no overflow condition.

CALL DVCHK (j)[†]

If division by zero occurred, set j = 1 and clear the indicator; if division by zero did not occur, set j = 2.

CALL FTNBIN (i, n, IRAY)

Sets the input/output format of the specified binary files according to the blocking flag, i. (appendix M)

IRAY is an integer array. IRAY(j) contains the logical unit designation of a binary file.

n is the number of elements in IRAY to be processed.

If n = 0, all files will be processed. IRAY has no effect in this case.

If i = 1, each processed file will have blocked input/output; if i = 0, each processed file will have nonblocked input/output.

Examples:

1) CALL FTNBIN (1,0,dummy)   The third parameter is used for compatibility with FTN and will not affect the routine.

Sets the format of all binary files in the program to be blocked.

2)
```
IRAY(1)=10
IRAY(2)=11
CALL FTNBIN(1,2,IRAY)
```

Sets the format of binary files TAPE10 and TAPE11 to be blocked.

Coded files are not affected by a call to FTNBIN.

[†] Currently J is always set to 2 (see LEGVAR in Appendix C).

CALL READEC (cm,ecs,n)

Transfers words from extended core storage (ECS) into central memory. cm is the central memory address; it can be an array name or a variable name. ecs is the ECS relative address. The extended core storage field length is declared in the JOB card (SCOPE card manual). n is the count of the number of words to be transferred; it must be an integer variable or integer constant.

If no parity error occurs, return is to the user's program. If a parity error is detected, *ECS READ PARITY ERROR AT RA = xxxxxB KEY IN GO OR DROP is output on the console and in the dayfile. If the operator keys in GO, return is to the user with the data as it was read; if DROP, the job is dropped.

CALL WRITEC (cm,ecs,n)

Transfers words from central memory to ECS. Parameters are the same as for READEC. If the ECS unit is off or in maintenance mode, a diagnostic *ECS UNIT DOWN is output on the console and dayfile; and the job is terminated.

Example:

```
      PROGRAM ECS(INPUT,OUTPUT)
      DIMENSION A(1000)
          .
          .
          .
      CALL WRITEC(A,0,1000)
C     TRANSFER 1000 CM WORDS BEGINNING AT CM LOCATION
C     A INTO ECS BEGINNING AT WORD 0 OF THE USERS
C     RESERVED ECS.
      END
```

CALL SECOND(t)

Returns central processor time from start of job in seconds in floating point format to three decimal places. t is a real variable.

CALL OPENMS (u,ix,ℓ,p)[†]

Opens a mass storage file, and informs SCOPE that it is a random access file. If the file already exists, the master index is read into the area specified by the program.

CALL READMS (u,fwa,n,i)[†]
CALL WRITMS (u,fwa,n,i)[†]

Perform data transfers between mass storage and central memory.

[†] Appendix I gives further information and examples for these routines.

CALL STINDX (u,ix,ℓ)[†]

Changes the file index to the base specified in the CALL.

u	Logical unit number
ix	First word address of the index (in central memory)
ℓ	Length of index; $\ell \geq 2$ (number of index entries) +1 for name index $\ell \geq$number of index entries +1 for number index
p=1	Indicates 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
i	Record number or address. When address, it is the address of record number or record name. Record number is right justified and record name is left justified display code 1-7 characters.

CALL EXIT terminates program execution and returns control to the monitor.

CALL REMARK (H) places a message, ≤ 40 characters, in the dayfile. H is a Hollerith specification. Program generated messages must terminate with a 12-bit byte of binary zeros.

CALL DISPLA(H,k) displays a variable name (≤ 40 characters, but should be restricted to 20) and its numerical value in the dayfile; k is displayed as an integer if not normalized, in floating point format if normalized. H is a Hollerith specification. Program generated messages must terminate with a 12-bit byte of binary zeros followed by a zero word.

CALL DUMP ($a_1, b_1, f_1, \ldots, a_n, b_n, f_n$)
CALL PDUMP($a_1, b_1, f_1, \ldots, a_n, b_n, f_n$) [n ≤ 20]

Dump storage on OUTPUT file in indicated format. For PDUMP control returns to the calling program; for DUMP execution terminates and control returns to the operating system. If no parameters are provided, an octal dump of all storage occurs.

a_i and b_i, identifiers or statement numbers, indicate first and last word of the storage area to be dumped. Statement numbers must be 1 to 5 digits with a trailing S, CALL DUMP (10S, 20S, 0). The last word of the storage area to be dumped cannot be contained in the last statement of a DO loop.

The dump format indicators are: f = 0 or 3 octal dump, f = 1 real dump, f = 2 integer dump; if bit 48 is set (normalize bit).

[†] Appendix I gives further information and examples for these routines.

7.11 FUNCTION SUBPROGRAM

A function is a computational procedure which returns a value associated with the function name. The mode of the function is determined by a type indicator or the name of the function. The first statement of a function subprogram must be one of the following forms where name is an alphanumeric identifier and p_i are formal parameters. A FUNCTION statement must have at least one parameter. $1 \leq n \leq 60$.

FUNCTION name $(p_1, \ldots, p_n)$

type FUNCTION name $(p_1, \ldots, p_n)$

FORTRAN IV FUNCTION name $(p_1, \ldots, p_n)$

FORTRAN IV type FUNCTIONname $(p_1, \ldots, p_n)$

FORTRAN II FUNCTION name $(p_1, \ldots, p_n)$

FORTRAN II type FUNCTION name $(p_1, \ldots, p_n)$

FORTRAN VI FUNCTION name $(p_1, \ldots, p_n)$

FORTRAN VI type FUNCTION name $(p_1, \ldots, p_n)$

Type is REAL, INTEGER, DOUBLE PRECISION, DOUBLE, COMPLEX, or LOGICAL. When the type indicator is omitted, the mode is determined by the first character of the function name.

The name of a function must not appear in a DIMENSION declaration. The name must appear, however, at least once as any of the following:

The left-hand identifier of a replacement statement

An element of an input list

An actual parameter of a subroutine reference

7.12 FUNCTION REFERENCE

In the general form, name identifies the function referenced, it is an alphanumeric identifier, and its type is determined in the same way as a variable identifier. p_i are actual parameters, n is 1 to 60.

name $(p_1, \ldots, p_n)$

A function reference may appear any place in an expression that an operand may be used. The evaluated function has a single value associated with the function name. When a function reference is encountered in an expression, control is transferred to the function indicated. When a RETURN statement in the function subprogram is encountered, control is returned to the statement containing the function reference.

Examples:

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

A reference to the function GRATER might be:

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

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

This function can be referenced:

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

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

7.13 STATEMENT FUNCTION

A statement function is defined by a single expression and applies only to the program or subprogram containing the definition. The name of the statement function is an alphanumeric identifier; a single value is always associated with the name.

name $(p_1, \ldots, p_n) = E$

p_i are formal parameters and must be simple variables; n is 1 to 60. The expression E may be any arithmetic or logical expression which may contain reference to library functions, statement functions, or function subprograms.

The nonparameter identifiers appearing in the expression have the same values as they have outside the function.

A statement function reference has the form:

name(p_1, . . . ,p_n)

> name is the name of the statement function; the actual parameters p_i may be any arithmetic expressions.

During compilation, the arithmetic statement function definition is compiled once at the beginning of the program and a transfer is made to this portion of the program whenever a reference is made to the arithmetic statement function.

The statement function name must not appear in a DIMENSION, EQUIVALENCE, COMMON, or EXTERNAL statement; the name can appear in a type declaration but cannot be dimensioned. Statement function names must not appear as actual or formal parameters.

Actual and formal parameters must agree in number, order, and mode. The mode of the evaluated statement function is determined by the name of the arithmetic statement function.

A statement function must precede the first statement in which it is used, but it must follow all declarative statements (DIMENSION, Type, etc.) which contain symbolic names referenced in the statement function. All statement functions should precede the first executable statement; otherwise, an informative diagnostic is provided.

A statement function may not reference itself and if such an attempt is made, a fatal diagnostic is provided.

Examples:

```
LOGICAL A, B
EQV(A, B)=(A. AND. B). OR. (. NOT. A. AND. . NOT. B)
COMPLEX Z
Z(X, Y)=(1., 0.)*EXP(X)*COS(Y)+(0., 1.)*EXP(X)*SIN(Y)
GROPAY (RATE, HRS, OTHRS)=RATE*HRS+RATE*. 5*OTHRS
```

Examples of use:

```
ANET=GROPAY(1.25,NOHRS(I),OVTIME(I))-DEDUCT(I)-TAX
RESULT=(Z(BETZ,GAMMA(I+K))**2-1.)/SQRT(TWOPIE)
```

7.14 LIBRARY FUNCTIONS

Function subprograms that are used frequently have been stored in a reference library and are available to the programmer through the compiler. Library function references may appear in the main program, subprograms, and statement functions.

FORTRAN contains the standard library functions available in earlier versions of FORTRAN. (Appendix C.) The parameter and result type of all library functions is also listed in Appendix C.

7.15 PROGRAM MODES

A FORTRAN program or subprogram is compiled in one of three modes:

FORTRAN IV

FORTRAN II

FORTRAN VI

When a mode is not indicated, the program or subprogram is compiled in FORTRAN IV mode.

The compiling mode for subprograms is assumed FORTRAN IV unless specific subprograms are declared to be of a different mode. A subprogram declared to be of a different mode is processed in its declared mode. The subprogram following it, unless declared to be of a different mode, is processed in FORTRAN IV mode.

FORTRAN II and FORTRAN IV statements which are not inherently incompatible may be intermixed in a program to be compiled in either mode (Appendix D). Inherently incompatible statements are those involving function subprogram references and EQUIVALENCE statements, causing a reordering of variables in COMMON. However, any standard FORTRAN II or FORTRAN IV library function or subroutine reference may appear in a program to be compiled in either mode.

FORTRAN VI causes FORTRAN IV type compilation except in the area of DO loops and END statements. Under FORTRAN VI, the 3600 FORTRAN DO procedure is used. That is, a DO loop is not executed if the initial value is greater than the terminal value. Also, in FORTRAN VI, the appearance of an END statement in a function or subroutine acts like a RETURN statement.

7.16 VARIABLE DIMENSIONS IN SUBPROGRAMS

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

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

The formal parameters representing the array dimensions must be simple integer variables. The array name must also be a formal parameter. The actual parameters representing the array dimensions must have integer values.

The total number of elements of the corresponding array in the subprogram may not exceed the total number of elements of a given array in the calling program.

Example:

Consider a simple matrix add routine written as a subroutine:

```
     SUBROUTINE MATADD (X,Y,Z,M,N)
     DIMENSION X (M,N),Y(M,N),Z(M,N)
     DO 10 I = 1,M
     DO 10 J = 1,N
10   Z(I,J)= X(I,J) + Y(I,J)
     END
```

The arrays X, Y, Z and the variable dimensions M, N must all appear as formal parameters in the SUBROUTINE statement and also in the DIMENSION statement as shown. If the calling program contains the array allocation declaration

```
DIMENSION A(10,10),B(10,10),C(10,10),E(5,5),F(5,5),G(5,5),H(10,10)
```

the program may call the subroutine MATADD from several places within the main program as follows:

```
     CALL MATADD(A,B,C,10,10)
     CALL MATADD(E,F,G,5,5)
     CALL MATADD(B,C,A,10,10)
     CALL MATADD(B,C,H,10,10)
```

The compiler does not check to see if the limits of the array established by the DIMENSION statement in the main program are exceeded.

7.17 PROGRAM ARRANGEMENT

FORTRAN assumes that all statements and comments appearing between a PROGRAM, SUBROUTINE, or FUNCTION statement and an END statement belong to one program. A typical arrangement of a set of main program and subprograms follows. (Also see appendix F.)

```
PROGRAM          WHAT
      .
      .
      .
END
FORTRAN II       SUBROUTINE S1(A,B)
      .
      .
      .
END
FORTRAN IV       SUBROUTINE S2
      .
      .
      .
END
REAL FUNCTION F1(P1)
      .
      .
      .
END
```

SEGMENTATION 8

8.1 SEGMENTS

Segmentation allows programs that exceed available storage to be divided into independent parts which may be called and executed as needed.

A segment is a group of relocatable subprograms or sections loaded and delinked as a unit. A section is a collection of relocatable programs with one section name; it is included in the loader scheme to reduce the number of program names in segment calls. The user defines the programs and sections to be included in a given segment. Segments allow the user to dynamically select programs which he requires in memory. Segment loading proceeds like normal loading and parameters are passed as in normal loading. However, when additional segments are called, they may destroy existing segments. The user defines a segment with a SEGMENT card. Segments are loaded by the monitor during initial load. A running program may load a segment with a user request.

LEVELS

The user assigns one priority or level number (0-77_8) to each segment. The level serves as a programmer's tool for rapid delinking of segments. Level zero is reserved for the initial or main segment which remains in memory during segment execution; subsequent segments may be loaded at any level. The number of segments in central memory at one time is limited only by the amount of memory available.

When a segment is loaded, its external references are linked to their corresponding entry points in subprograms and common blocks in previously loaded segments at lower levels. Unsatisfied references in the segments will remain unsatisfied. Subsequently loaded segments may include entry points to satisfy them; or the user may specify that they be satisfied from the system library. All external references which remain unsatisfied will contain out-of-bounds references.

If execution is attempted when unsatisfied externals exist, the job is terminated and a message output.

Levels are used to delink segments that are no longer needed. If a segment is 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 delinked and removed. If a segment is loaded at level 6, any segments previously loaded at levels 6, 7, 8 and upward will be delinked and removed. When a segment is delinked, the linkage of its entry points to external references in lower levels is destroyed and the externals are unsatisfied once again. Levels need not be consecutive. For instance, the user may request segments at levels 2, 3, 10 and they will be loaded as requested.

Example:

> A SINE routine is loaded in a segment at level 2. If any external symbols refer to entry points in level 1, they are linked. To try an experimental version of SINE, the user loads a segment containing new SINE at level 3. The original SINE remains at levels 2 and so do its links to level 1; but, any new segments loaded at higher levels will link to the new SINE at level 3. The linkage remains until a new level 3 is loaded, this would clear out SINE at level 3 and any references to it would be left unsatisfied. If a new SINE were loaded at a level higher than 3, any segments loaded afterward would be linked to it. It is permissible to have more than one version of a subprogram with the same name, however, the first version encountered from the current file position is the one which is loaded. It is up to the user to properly position the file to load the version he desires.

The first segment to declare blank common establishes its length. If subsequent segments declare larger blank common, it is truncated to fit the established length, a message is output on the DAYFILE, and loading continues. To change the size of blank common, the segment first declaring it must be delinked by loading a new segment at that level, which declares a new length. If that segment does not declare blank common, subsequently loaded segments may do so.

Segments are called with this statement; lib and m are optional:

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

fn variable name of location containing file name (left justified display code) from which the segment load should take place.

e level of the segment load

a simple or subscripted variable name of array containing a list of segments, sections, and/or subprograms to be loaded with this call. Names must be in either L or H form in the upper seven-character positions of each word and be terminated by a zero entry. If the first entry in the list is zero, all subprograms remaining in the file fn are loaded into the segment. Although contents of array a are modified by this call, they remain in a form suitable for a subsequent SEGMENT call.

lib if zero or missing, the system library will be searched for any unsatisfied externals. If non-zero, the system library will not be used. This parameter controls the c bit in the LOADER call.

m if zero or missing, no segment load map will be produced. If non-zero, a map of the segment load will be produced on file OUTPUT. This parameter controls the m bit in the LOADER call.

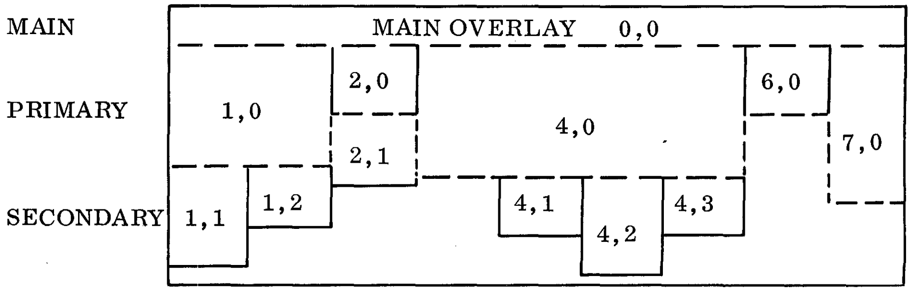
8.2 OVERLAYS

An overlay is a portion of a program written on a file in absolute form and loaded at execution time without relocation. As a result, the resident loader for overlays can be reduced substantially in size. Loading an overlay may destroy previously loaded overlays in much the same way as loading a segment may destroy previously loaded segments. The user defines an overlay with an OVERLAY card. The loader generates the overlay and writes it on a file to be called as needed for execution.

The above paragraph implies that during execution of the object programs, the structure of all overlays is fixed and predictable. This system is used when the organization of programs in core at object time is reasonably stable, and the most pressing need is for the fastest possible loading speed.

8.2.1 LEVELS

Levels are used to describe the sequence of loading overlays and to specify which sections of code are to overlay others. In 6000 SCOPE, there are three normal levels of overlaying, main, primary and secondary, and one floating method. Up to three overlays may be in core simultaneously; they are usually loaded contiguously. Any one or all of the overlays may be replaced by other overlays. The following diagram demonstrates the relationship of the levels when they are loaded into core. This example shows a number of different core loads, which might exist for a single job:



Normally only one overlay of each level will be in core at a time; secondary overlays usually replace secondary overlays, and primary overlays usually replace primary overlays.

8.2.2 USAGE

A typical overlay usage would be the case where a programmer desires to constrain field length to 40,000 words, but must execute programs requiring 100,000 words. Assuming a program which can be broken into three sequentially executed portions of approximately 20,000 words and a basic, common portion containing about 20,000 words, the common program portion would be designated as the MAIN OVERLAY. The common section would usually consist of the I/O routines, and other function subprograms which may be accessed by two or more of the portions, as well as some common data storage area. It would probably also contain the controlling CALL statements for loading and executing other portions of the program.

Since there is no interdependence between the other three program sections, each would be designated a PRIMARY OVERLAY. Only one of these sections would reside in core at a time, each being overlaid by its successor during processing.

When an overlay is loaded by a FORTRAN CALL statement, it is immediately entered and executed. Control is thus taken away from the calling program and some means must be established by the programmer for returning from the overlay. Usually return is to an entry point in the MAIN OVERLAY. The major differences between SEGMENTS and OVERLAYS are as follows:

> SEGMENTS are dynamically organized at execution time; the composition of OVERLAYS is fixed when they are generated.
>
> SEGMENTS are loaded by one CALL and entered at various points by additional CALL statements; OVERLAYS are loaded and entered with the same CALL.

8.2.3 IDENTIFICATION

Overlays may be loaded from the SCOPE library or from a specified file. A single overlay may be loaded only from a single file, although many files may be used for loading by a single job. When an overlay is loaded from the library, it is identified by its primary entry point name; when it is loaded from a file, it is identified by its level number. The level number is a pair of two-digit octal numbers ($0\text{-}77_8$) giving the primary and secondary overlay relationship. The first number is the primary level, the second is the secondary level. An overlay with a non-zero primary level and a zero secondary level (1,0) is a primary overlay. Any overlay with the same primary level and a non-zero secondary level (1,1) is associated with and subordinate to the corresponding primary and is called a secondary overlay. This difference is significant when overlays are loaded. Level 0,0 is reserved for the initial, or main, overlay which is neither primary nor secondary; it is a special case which remains in memory during overlay execution. Overlay numbers (0,1) to (0,77) are illegal.

The main overlay (0,0) is loaded first. All primary overlays are loaded at the same point immediately following the main overlay. Secondary overlays are loaded immediately following the primary overlay. Loading the next primary overlay destroys the first loaded primary overlay and any associated overlays. Likewise, the loading of a secondary overlay destroys a previously loaded secondary overlay.

8.2.4 COMPOSITION

An overlay may consist of one or more FORTRAN or COMPASS programs. The first program in the OVERLAY must have the characteristics of a FORTRAN main program (not a subprogram). The program name becomes the primary entry point for the overlay through which control passes when the overlay is called. An overlay cannot reference entry points in higher level overlays. The only method of reference for a MAIN overlay to primary and secondary overlays is through the CALL OVERLAY statement. However, the primary overlay may reference any entry point in the MAIN overlay, while the secondary overlay may reference any entry point in the primary or MAIN overlay.

Blank common and labeled common may be defined in any level overlay and referenced by that overlay and higher level overlays. (The same rules apply as for entry points.) Unlike SEGMENT jobs, labeled common is linked between overlays.

An OVERLAY is established by an OVERLAY card which precedes the program cards (section 8.3). The overlay consists of all programs appearing between the OVERLAY card and the next OVERLAY card or an end-of-file.

8.2.5 CALL

Overlays are called by the following statement:

CALL OVERLAY (fn, ℓ_1, ℓ_2,p)

OVERLAY FORTRAN subroutine which translates the FORTRAN call into a call to the loader

fn variable name of the location containing the name of the file (left justified display code) which includes the overlay

ℓ_1 primary level of the overlay

ℓ_2 secondary level of the overlay

p recall parameter. If p equals 6HRECALL, the overlay is not reloaded if it is in memory.

All four parameters must be specified; the absence of any one could result in a MODE error at execution time. The levels appearing on the OVERLAY card are always octal. The normal mode for parameters in FORTRAN calls is decimal. This fact should be considered when coding the ℓ_1, ℓ_2 parameters.

If uniqueness is ensured at execution time, more than one overlay may be created with the same level numbers. Uniqueness is determined by the level numbers, the file name from which the OVERLAY is to be loaded, and the position of the overlay on the file. Since the loader selects the first overlay encountered on the specified file with level numbers which match those in the call, it is possible to position a number of overlays on a file with the same identifier and by properly sequencing the calls thereto, have available a number of different overlays.

Loading from a file requires an end-around search of the file for the specified overlay; this can be time consuming in large files. When speed is essential, each overlay should be written to a separate file.

8.2.6 OVERLAY FORMAT

Each overlay consists of a logical record in the following format:

Word 1

*	ℓ_2	ℓ_1	fwa	ea	
59	47	41	35	17	0

* 50_8 (specified on overlay header)

ℓ_1 Primary overlay level

ℓ_2 Secondary overlay level

ea Entry point to the overlay

fwa First word address of overlay (overlay is loaded at fwa)

Word 2 through end of record: 60-bit data words.

8.3 LOADER CARDS

Loader cards are processed directly by the loader rather than by the monitor. They provide the loader with information necessary for generating overlays and segments. All loader cards must precede the subprogram text to be loaded. Formats are the same as for SCOPE control cards. However, if they are in the FORTRAN decks, the loader cards must begin in column 7.

8.3.1 SEGMENT CARDS

All subprograms named in a segment must reside in the same file.

SEGMENT

Segments other than segment zero may be defined by a segment card or in the user's program.

SEGMENT(sn, $pn_1, pn_2, \ldots, pn_i$)

sn and pn_i are defined as in SEGZERO

SEGZERO

All programs requiring segment loading must have a SEGZERO card defining the first segment. There may be only one SEGZERO card in the initial load.

SEGZERO(sn, $pn_1, pn_2, \ldots, pn_i$)

sn Segment name

pn_i Names of subprograms or sections

SECTION

This card defines a section, or group of programs within a SEGMENT.

SECTION(sname, $pn_1, pn_2, \ldots, pn_i$)

sname Name of the section

pn_i Name of a subprogram belonging to the section

If more than one card is necessary to define a section, consecutive SECTION cards with the same sname may follow. Whenever the named section is loaded, all subprograms within a section will be loaded.

All SECTION cards must appear prior to any SEGMENT cards.

8.3.2 OVERLAY CARDS

OVERLAY (fn,ℓ_1,ℓ_2,cnnnnnn)

fn File name onto which the generated overlay is to be written

ℓ_1 Primary level number

ℓ_2 Secondary level number

(for ℓ_1 and ℓ_2:) must be 0,0 for first overlay card and must be in octal[†]

cnnnnnn optional; nnnnnn is 6-octal digits. If absent, overlay is loaded normally
If present, overlay is loaded nnnnnn words from the start of blank common. This provides a method for changing the size of blank common at execution time.

The first overlay card must have an fn. Subsequent cards may omit fn, and the overlay is written on the same fn.

Each OVERLAY card must be followed by a program card. The program card for the <u>main overlay</u> must specify all needed file names, such as INPUT, OUTPUT, TAPE 1, etc, for all overlay levels. File names should not appear in program cards for other than the (0,0) OVERLAY.

[†] Level numbers given in the CALL OVERLAY, however, are decimal; e.g., the overlay card for overlay 1,9 would be OVERLAY(fn,1,11) and its call would be CALL OVERLAY(fn,1,9)

Example:

```
OVERLAY(XFILE,0,0)
PROGRAM ONE(INPUT,OUTPUT,PUNCH)
                .
                .
                .
CALL OVERLAY(5HXFILE,1,0,0)
                .
                .
                .
STOP
END
OVERLAY(XFILE,1,0)
PROGRAM ONE ZERO
CALL OVERLAY(5HXFILE,1,1)
                .
                .
                .
RETURN
END
OVERLAY(XFILE,1,1)
PROGRAM ONE ONE
                .
                .
                .
RETURN
END
```

INPUT/OUTPUT FORMATS 9

Data transmission between storage and external units requires the FORMAT statement (BCD only) and the I/O control statement (chapter 10). The I/O statement specifies the input/output device and process READ, WRITE, etc., and a list of data to be moved. The FORMAT statement specifies the manner in which the data is to be moved. In binary statements no FORMAT statement is used.

9.1 INPUT/OUTPUT LIST

The list portion of an input/output statement indicates the data items and the order, from left to right, of transmission. The input/output list can contain any number of elements; list items may be array names, simple or subscripted variables, or an implied DO loop. Items are separated by commas, and their order must correspond to any FORMAT specification associated with the list. External records are always read or written until the list is satisfied.

Subscripts in an I/O list may be in the following forms:

```
(c*I ± d)

(I ± d)

(c*I)

(I)

(c)
```

c and d are unsigned integer constants, and I is a simple integer variable, previously defined, or defined within an implied DO loop.

Examples:

```
READ 100, A,B,C,D
READ 200, A,B,C(I),D(3,4),E(I,J,7),H
READ 101, J,A(J),I,B(I,J)
READ 102, DELTA(5*J+2,5*I-3,5*K),C,D(I+7)
READ 202, DELTA
READ 300, A,B,C,(D(I),I=1,10),E(5,7),F(J),(G(I),H(I),I=2,6,2)
READ 400, I,J,K,( ( (A(II,JJ,KK),II=1,I),JJ=1,J),KK=1,K)
READ 500, ( (A(I,J),I=1,10,2),B(J,1),J=1,5),E,F,G(L+5,M-7)
```

9.1.1 ARRAY TRANSMISSION

Part or all of an array can be represented for transmission as a single I/O list item by using an implied DO notation in the form:

$(((A(I,J,K),L_1=m_1,m_2,m_3),L_2=n_1,n_2,n_3),L_3=p_1,p_2,p_3)$

m_i,n_i,p_i Unsigned integer constants or simple integer variables. If m_3, n_3, or p_3 is omitted, it is assumed equal to 1.

I,J,K Subscripts of A

L_1, L_2, L_3 Index variables I, J, K in same order

During execution, each subscript (index variable) is set to the initial index value: $L_1=m_1$, $L_2=n_1$, $L_3=p_1$. The first index variable defined in the list is incremented first, following the rules for a DO loop execution. When the first index variable reaches the maximum value, it is reset; the next index variable to the right is incremented, and the process is repeated until the last index variable has been incremented. If m_1 is greater than m_2 initially, one read is executed.

An array name which appears without subscripts in an I/O list causes transmission of the entire array by columns.

Example:

```
DIMENSION B(10,15)
```

the statement

```
READ 13, B
```

is equivalent to

```
READ 13,((B(I,J),I=1,10),J=1,15)
```

An implied DO loop can be used to transmit a simple variable more than one time. For example, the list item (A(K),B,K=1,5) causes the transmission of variable B five times. A list of the form K, (A(I),I=1,K) is permitted and the input value of K is used in the implied DO loop. The index variable in an implied DO list in a DATA statement should be an implicit integer.

Examples:

1) Simple implied DO loop list items.

```
    READ 400,(A(I),I=1,10)
400 FORMAT (E20.10)
```

This statement is equivalent to the following DO loop.

```
      DO 5 I=1,10
    5 READ 400, A(I)

      READ 100, ((A(JV,JX),JV=2,20,2),JX=1,30)
      READ 200, (BETA(3*JON+7),JON = JONA,JONB,JONC)
      READ 300, ((ITMSLST(I,J+1,K-2),I=1,25),J=2,N),K=IVAR,IVMAX,4)

      READ 600, (A(I),B(I),I=1,10)
  600 FORMAT (F10.2,E6.1)
```

The previous statement is equivalent to the DO loop:

```
      DO 17 I = 1,10
   17 READ 600, A(I),B(I)
```

2) Nested implied DO list items.

```
      READ 100,(((((A(I,J,K),B(I,L),C(J,N),I=1,10),J=1,5),
      K=1,8),L=1,15),N=2,7)
```

Data is transmitted in the following sequence:

```
      A(1,1,1),B(1,1),C(1,2),A(2,1,1),B(2,1),C(1,2)...
      ...A(10,1,1),B(10,1),C(1,2),A(1,2,1),B(1,1),C(2,2)...
      ...A(10,2,1),B(10,1),C(2,2)...A(10,5,1),B(10,1),C(5,2)...
      ...A(10,5,8),B(10,1),C(5,2)...A(10,5,8),B(10,15),C(5,2)...
      ...A(10,5,8),B(10,15),C(5,7)
```

The following list item will transmit the array E(3,3) by columns:

```
      READ 100,((E(I,J),I=1,3),J=1,3)
```

The following list item will transmit the array E(3,3) by rows:

```
      READ 100,((E(I,J),J=1,3),I=1,3)
```

3)

```
      DIMENSION MATRIX(3,4,7)
      READ 100, MATRIX
  100 FORMAT (I6)
```

The above items are equivalent to the following statements:

```
      DIMENSION MATRIX(3,4,7)
      READ 100,(((MATRIX(I,J,K),I=1,3),J=1,4),K=1,7)
```

The list is equivalent to the nest of DO loops:

```
      DO 5 K=1,7
      DO 5 J=1,4
      DO 5 I=1,3
    5 READ 100, MATRIX(I,J,K)
```

9.2 FORMAT DECLARATION

BCD input/output statements require a FORMAT declaration which contains conversion and editing information relating to internal/external structure of the corresponding I/O list items. A FORMAT declaration has the following form:

FORMAT (spec$_1$,...,k(spec$_m$,...),spec$_n$,...)

Spec$_i$	format specification
k	optional repetition factor, must be unsigned integer constant.

The FORMAT declaration is non-executable and may appear anywhere in the program. FORMAT declarations must have a statement label in columns 1-5.

The data items in an I/O list are converted from one representation to another (external/internal) according to FORMAT conversion specifications. FORMAT specifications may also contain editing codes.

Conversion specifications:

Ew.d	Single precision floating point with exponent	Iw	Decimal integer conversion
		Ow	Octal integer conversion
Fw.d	Single precision floating point without exponent	Aw	Alphanumeric conversion
Dw.d	Double precision floating point with exponent	Rw	Alphanumeric conversion
		Lw	Logical conversion
Gw.d	Single precision floating with or without exponent	nP	Scaling factor

Complex data items are converted on input/output according to a pair of consecutive Ew.d or Fw.d specifications.

Example:

```
      COMPLEX A,B
      PRINT 10,A
  10  FORMAT (F7.2,F9.2)
      READ 11,B
  11  FORMAT (E10.3,E10.3)
```

Editing specifications:

wX	Intraline spacing	/	Begin new record
wH	Heading and labeling	*...*	Heading and labeling

Both w and d are unsigned integer constants; w specifies the field width in number of character positions in the external record, and d specifies the number of digits to the right of the decimal within the field.

9.3 CONVERSION SPECIFICATIONS

9.3.1 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:

```
ba.a. . .a±eee          100 ≤ eee ≤ 322
     or
ba.a. . .aE±ee          0 ≤ ee ≤ 99
```

b indicates a blank or a minus sign. a's are the most significant digits of the integer and fractional part and eee are the digits in the exponent. If d is zero or blank, the decimal point and digits to the right of the decimal do not appear as shown above. Field w must be wide enough to contain the significant digits, signs, decimal point, E, and the exponent. Generally, w ≥d+7. Positive numbers need not reserve a space for the sign of the number.

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.

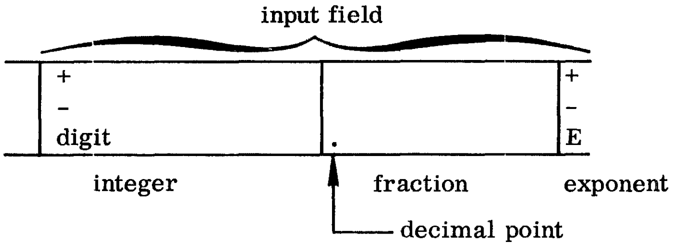
Examples:

```
Ew.d Output
     PRINT 10,A                                A contains -67.32
10   FORMAT(*bA=*E10.3)                               or  +67.32
          Result:  A = -6.732E+01 or b6.732E+01

     PRINT 10,A
10   FORMAT(*bA=*E12.4)
          Result:  A = b-6.7320E+01            bb6.7320E+01

     PRINT 10,A                                A contains -67.32
10   FORMAT(*bA=*E9.3)                         provision not made
          Result:  A = *.732E+01               for sign

     PRINT 10,A
10   FORMAT(*bA=*E10.4)
          Result:  A = *.7320E+01
```

9.3.2 Ew.d INPUT

The E specification converts the number in the input field to a real number and stores it in the proper location.

Subfield structure of the input field:



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. An input field consisting entirely of blanks is interpreted as minus zero.

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 -. When it begins with D or E, the + 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
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:

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

For example, if the specification is E7.8, the input quantity 3267+05 is converted and stored as: $3267x10^{-8}x10^{5} = 3.267$.

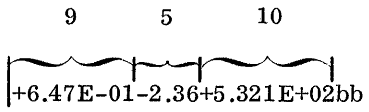
A decimal point in the input field overrides d. The input quantity 3.67294+5 read by an E9.d specification is always stored as $3.6729x10^{5}$. When d does not appear, it is assumed to be zero.

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 the significant digits, signs, decimal point, E or D, and exponent.

Example:

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

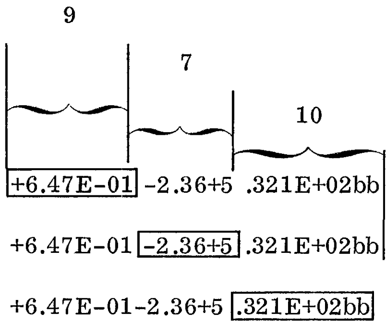
Input quantities on the card are in three contiguous fields columns 1 through 24:



```
+6.47E-01-2.36+5.321E+02bb
```

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

Reading proceeds as follows:



```
+6.47E-01 -2.36+5 .321E+02bb
+6.47E-01 -2.36+5 .321E+02bb
+6.47E-01-2.36+5 .321E+02bb
```

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:

Ew.d Input

Input Field	Specification	Converted Value	Remarks
+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. x 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.x10^1
E+06	E10.6	0.	No integer or fraction subfield; zero stored regardless of exponent field contents
1.bEb1	E6.3	10.	Blanks are interpreted as zeros

9.3.3 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:

$\underline{b}$a. . .a.a. . .a

b indicates a blank. The 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, the decimal point and digits to the right 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. If the field is longer than required to accommodate the number, the number is right justified with blank fill to the left.

Contents of A	Format Statement	Print Statement	Printed Result
+32.694	10 FORMAT (F7.3)	PRINT 10,A	b32.694
+32.694	11 FORMAT (F10.3)	PRINT 11,A	bbbb32.694
-32.694	12 FORMAT (F6.3)	PRINT 12,A	*2.694 no provision for - sign and most significant digit)
.32694	13 FORMAT (F4.3,F6.3)	PRINT 12,A,A	.327b0.327

9.3.4 Fw.d INPUT

This specification is a modification of Ew.d. The input field consists of an integer and a fraction subfield. An omitted subfield is assumed to be zero. The restrictions described under Ew.d input apply.

Examples:

Input Field	Specification	Converted Value	Remarks
367.2593	F8.4	367.2593	Integer and fraction field
37925	F5.7	.0037925	No fraction subfield; input number converted as 37925 x 10^{-7}
-4.7366	F7	-4.7366	No d in specification
.62543	F6.5	.62543	No integer subfield
.62543	F6.2	.62543	Decimal point overrides d of specification
+144.15E-03	F11.2	.14415	Exponents are legitimate in F input and may have P-scaling
5bbbb	F5.2	500.00	No fraction subfield; input number converted as 50000x 10^{-2}
bbbbb	F5.2	-0.00	Blanks in input field interpreted as -0

9.3.5 Gw.d OUTPUT

The field occupies w positions of the output record, with d signficant digits. The real data will be represented by F conversion unless the magnitude of the data exceeds the range that permits effective use of F conversion. In this case, the E conversion will represent the external output. Therefore, the effect of the scale factor is not implemented unless the magnitude of the data requires E conversion.

When F conversion is used under Gw.d output specification, 4 blanks are inserted within the field, right justified. Therefore, for effective use of F conversion, d must be $\leq$ w-6.

The method of representation in the output record is a function of the magnitude N of the real data being converted. The following table gives a correspondence between N and the method of conversion:

$0.1 \leq N < 1$	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

Examples:

```
Gw.d  OUTPUT

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:  b1.215E+06
```

9.3.6 Gw.d INPUT

Gw.d specification is similar to the Fw.d input specification.

9.3.7 Dw.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:

$\underline{b}$a.a. . .a±eee $100 \leq eee \leq 512$

or

$\underline{b}$a.a. . .aD±ee $0 \leq ee \leq 99$

b indicates blank. D conversion corresponds to Ew.d Output.

9.3.8 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:

$$\underbrace{\text{-6.31675298443E-03}}_{18}\underbrace{\text{+2.718926453147}}_{15}\underbrace{\text{6293477528869D-09}}_{17}$$

9.3.9 Iw OUTPUT

I specification is used to output decimal integer values. The output quantity occupies w output record positions, right justified:

$\underline{b}$a. . .a

b is a blank. The a's are the most significant decimal digits (maximum 15) of the integer. If the integer is positive, the + sign is suppressed. The range of numbers permitted is roughly $-2^{48}+1 \leq n \leq 2^{48}-1$.

If the field 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.

Example:

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

Result: $\underbrace{\text{bbb-3762}}_{8}\underbrace{\text{bbb4762937}}_{10}\underbrace{\text{bbb13}}_{5}$

9.3.10 Iw INPUT

The field is w characters in length, and the list item is a decimal integer constant. The input field w consists of an integer subfield, containing +, -, 0 through 9, 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)
```

Input Card:

$\underbrace{\text{139}}_{3}\underbrace{\text{bb-15bb}}_{7}\underbrace{\text{18}}_{2}\underbrace{\text{bb7}}_{3}\underbrace{\text{bb}}_{2}\underbrace{\text{b1b4}}_{4}$

In storage:

I contains 139

J	-1500
K	18
L	7
M	-0
N	104

9.3.11 Ow OUTPUT

O specification is used to output octal integer values. The output quantity occupies w output record 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 negative number is output in its one's complement internal form.

9.3.12 Ow INPUT

Octal integer values are converted under O specification. The field is w characters in length, and the list item must be an integer variable.

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. Blanks are interpreted as zeros.

Example:

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

Input Card:

$$\underbrace{\texttt{3737373737}}_{10}\underbrace{\texttt{666b6644b444}}_{12}\underset{2}{\texttt{-0}}$$

In storage:

```
P    00000000003737373737
Q    00000000666066440444
R    77777777777777777777
```

A negative number is represented in one's complement form.

A negative octal number is represented internally in seven'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 77777777777777777074.

```
That is,  77777777777777777777
         -00000000000000000703
          77777777777777777074
```

9.3.13 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.

9.3.14 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:

$$\underbrace{\text{LUX MENT}}_{8}\underbrace{\text{IS LUX O}}_{8}\underbrace{\text{RBIS}}_{4}$$

In storage:

Q	LUXbMENTbb
P	ISbLUXbObb
O	RBISbbbbbb

9.3.15 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.

9.3.16 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:

$$\underbrace{\text{LUX MENT}}_{8}\underbrace{\text{IS LUX O}}_{8}\underbrace{\text{RBIS}}_{4}$$

In storage:

Q	00LUXbMENT
P	00ISbLUXbO
O	000000RBIS

9.3.17 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  contains -0      J  contains 0
      PRINT 5, I, J, K, L         K  contains -0      L  contains -0
   5  FORMAT (4L3)
                  Result:  bbTbbFbbTbbT
```

9.3.18 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.

9.4 nP SCALE FACTOR

The D, E, F, and G conversion may be preceded by a scale factor whose effect is defined by: External number = Internal number $x10^{\text{scale factor}}$. The scale factor applies to Fw.d and Gw.d on both input and output and to Ew.d and Dw.d on output only. A scaled specification is written as shown below; n is a signed integer constant.

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

The scale factor is assumed to be zero if no other value has been given; however, once a value has been given, it holds for all D, E, F, and G specifications. To nullify this effect in subsequent D, E, F, and G specifications, a zero scale factor, 0P, must precede a D, E, F, or G specification. Scale factors for D, E, F, and G output specifications must be in the range $-8 \leq n \leq 8$.

Scale factors on D or E input specifications are ignored. For USASI compatible scale factor see section 9.8.

The scaling specification nP may appear independently of a D, E, F, or G specification; it holds for all subsequent D, E, F, and G specifications within the same FORMAT statement unless changed by another nP.

Example:

```
FORMAT(3PE12.6,F10.3,OPD18.7,-1P,F5.2)
```

The E12.6 and F10.3 specifications are scaled by 10^3, the D18.7 specification is not scaled, and the F5.2 specification is scaled by 10^{-1}.

The specification (3P,3I9,F10.2) is the same as the specification (3I9,3PF10.2).

9.4.1 Fw.d SCALING

Input

The number in the input field is divided by 10^n and stored. For example, if the input quantity 314.1592 is read under the specification 2PF8.4, the internal number is $314.1592x10^{-2} = 3.141592$.

Output

The number in the output field is the internal number multiplied by 10^n. In the output representation, the decimal point is fixed; the number moves to the left or right, depending on whether the scale factor is plus or minus. For example, the internal number 3.145926538 may be represented on output under scaled F specifications as follows:

Specification	Output Representation
F13.6	3.141593
1PF13.6	31.415927
3PF13.6	3141.592654
-1PF13.6	.314159

9.4.2 Ew.d ORDw.d SCALING

Output

The scale factor has the effect of shifting the output number left n places while reducing the exponent by n. Using 3.1415926538, some output representations corresponding to scaled E specifications are:

Specification	Output Representation	
E20.2	3.14	E+00
1PE20.2	31.42	E-01
2PE20.2	314.16	E-02
3PE20.2	3141.59	E-03
4PE20.2	31415.93	E-04
5PE20.2	314159.27	E-05
-1PE20.2	0.31	E+01

9.4.3 Gw.d SCALING

Input

Gw.d scaling on input is the same as Fw.d scaling on input.

Output

The effect of the scale factor is suspended unless the magnitude of the data to be converted is outside the range that permits the effective use of F conversion.

9.5 EDITING SPECIFICATIONS

9.5.1 wX

This specification may be used to include w blanks in an output record or to skip w characters on an input record to permit spacing of input/output quantities. 0X is not permitted; bX is interpreted as 1X. In the specification list, the 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:  b7bbbbbbb13.60bbbbbbb1.46237E+03
        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
```

9.5.2 wH OUTPUT

With this specification 6-bit characters, including blanks may be output in the form of comments, titles, and headings. w, an unsigned integer, specifies the number of characters to the right of H that are transmitted to the output record; w may specify a maximum of 136 characters. H denotes a Hollerith field; the comma following H is optional.

Examples:

Source program:

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

produces output record:

```
bBLANKSbCOUNTbINbANbHbFIELD.
```

Source program:

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

produces output record:

```
bLMAX = b1.50
```

9.5.3 wH INPUT

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 (27Hbbbbbbbbbbbbbbbbbbbbbbbbbbb)
```

Input Card:

$$\underbrace{\text{bTHIS IS A VARIABLE HEADING}}_{\text{27 cols}}$$

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:

```
bTHIS IS A VARIABLE HEADING
```

9.5.4 NEW RECORD

The slash (/) signals the end of a record anywhere in the specifications list. Consecutive slashes may appear in a list and they need not be separated from the other list elements by commas. During output, the slash is used to skip lines, cards, or tape records. During input, it specifies that control passes to the next record or card. K(/) results in K-1 lines being skipped.

Examples:

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

Printout:

```
     HEADING ____________________  line 1
             ______ (blank) ______  line 2
             ______ (blank) ______  line 3
INPUTbbOUTPUT_____________________  line 4
```

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

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

In storage:

```
A   -11.6
B   .325
C   46.327
D   -14.261
```

Printout:

```
b-1.16E+01bb3.25E-01
b46.327-14.261
```

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

Printout:

```
b-1.16E+01bb3.25E-01 __________  line 1
                     — (blank) —  line 2
b46.327-14.261 _________________  line 3
```

```
4)        PRINT 15, (A(I), I=1, 9)
      15  FORMAT (8HbRESULTS2(/) (3F8.2) ,
```

Printout:

```
      RESULTS ____________________________________  line 1
                                 _____ (blank)_______ line 2
           3.62      -4.03      -9.78 ______________________  line 3
          -6.33       7.12       3.49 ______________________  line 4
           6.21      -6.74      -1.18 ______________________  line 5
```

9.5.5 * . . . *

The specification *...* can be used as an alternate form of wH to output headings, titles, and comments. Any 6-bit character (except asterisk) between the asterisks will be output. The asterisks delineate the Hollerith field. This specification need not be separated from other specifications by commas.

Output Examples:

1) Source program:

```
                    PRINT 10
                 10 FORMAT (*bSUBTOTALS*)
```

produces the output record: bSUBTOTALS

2) Improper source program to output ABC*BE:

```
        PRINT 1
      1 FORMAT(*ABC*BE*)
```

The * in the output causes the specification to be interpreted as *ABC* and BE*. BE* is an improper specification; therefore, the wH specification must be used to output ABC*BE.

For input, this specification may be used in place of wH to read a new heading into an existing Hollerith field. Characters are stored in the heading until an asterisk is encountered in the input field or until all the spaces in the format specification are filled. If the format specification contains n spaces and the mth character (m≤n) in the input field is an asterisk, all characters to the left of the asterisk will be stored in the heading and the remaining character positions in the heading will be filled with blanks.

Input Examples:

1) Source program:

```
          READ 10
     10   FORMAT (*bbbbbbbbbbbbbbbbbbbbb*)
```

Input card:

```
bFORTRAN FOR THE 6600
```

A subsequent reference to statement 10 in an output control statement:

```
PRINT   10   produces:        FORTRAN FOR THE 6600
```

2) Source program:

```
          READ 10
     10   FORMAT (*bbbbbbbb*)
```

```
bHEAD*LINE
```

```
PRINT   10   produces:    HEAD bbb
```

9.6 REPEATED FORMAT SPECIFICATIONS

FORMAT specifications may be repeated by using an unsigned integer constant repetition factor, k, as follows: k(spec), spec is any conversion specification except nP. For example, to print two quantities K, L:

```
      PRINT 10,K,L
 10   FORMAT (I2,I2)
```

Specifications for K, L are identical; the FORMAT statement may also be:

```
 10   FORMAT (2I2)
```

When a group of FORMAT specifications repeats itself as in: FORMAT (E15.3, F6.1,I4,I4,E15.3,F6.1,I4,I4), the use of k produces: FORMAT(2(E15.3,F6.1, 2I4))

Nesting of parenthetical groups preceded by repeat constants beyond two levels is not permitted in FORMAT specifications.

9.6.1 UNLIMITED GROUPS

FORMAT specifications may be repeated without using a repetition factor. The innermost parenthetical group that has no repetition factor is unlimited and will be used repeatedly until the I/O list is exhausted. Parentheses are the controlling factors in repetition. The right parenthesis of an unlimited group is equivalent to a slash. Specifications to the right of an unlimited group can never be reached, as in the following example:

Format specifications for output data:

```
(E16.3,F20.7,2(I4),(I3,F7.1),F8.2)
```

The first two fields are printed according to E16.3 and F20.7. Since 2(I4) is a repeated parenthetical group, the next two fields are printed according to I4 format. The remaining print fields follow (I3,F7.1), which does not have a repetition factor, until the list elements are exhausted. F8.2 is never reached.

9.7 VARIABLE FORMAT

FORMAT specifications may be specified at the time of program execution. The specification, including left and right parentheses but not the statement label or the word FORMAT, is read under A conversion or in a DATA statement and stored in an array or a simple variable. 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 that appears with or without subscript specifies the location of the first word of the FORMAT information.

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)
   ```

9.6.1 UNLIMITED GROUPS

FORMAT specifications may be repeated without using a repetition factor. The innermost parenthetical group that has no repetition factor is unlimited and will be used repeatedly until the I/O list is exhausted. Parentheses are the controlling factors in repetition. The right parenthesis of an unlimited group is equivalent to a slash. Specifications to the right of an unlimited group can never be reached, as in the following example:

Format specifications for output data:

```
(E16.3,F20.7,2(I4),(I3,F7.1),F8.2)
```

The first two fields are printed according to E16.3 and F20.7. Since 2(I4) is a repeated parenthetical group, the next two fields are printed according to I4 format. The remaining print fields follow (I3,F7.1), which does not have a repetition factor, until the list elements are exhausted. F8.2 is never reached.

9.7 VARIABLE FORMAT

FORMAT specifications may be specified at the time of program execution. The specification, including left and right parentheses but not the statement label or the word FORMAT, is read under A conversion or in a DATA statement and stored in an array or a simple variable. 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 that appears with or without subscript specifies the location of the first word of the FORMAT information.

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:

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

9.8 USASI COMPATIBILITY

During compilation, a compiler parameter is available to select either the USASI compatibility features of the execution time routines or retain the present method of FORMAT/list interaction and output format. The switch is enabled by initialization of the main program and has effect only when a program is being compiled.

9.8.1 UNLIMITED GROUPS FOR USASI

Unlimited group repeat is implemented according to the USASI specification. An innermost parenthetical group that has no group repeat count specified in a FORMAT statement assumes a group repeat count of one. If the last outer right parenthesis of the format specification is encountered and the I/O list is not exhausted, 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 statement.

9.8.2 SCALE FACTOR FOR USASI

A scale factor is allowed for F, E, D, and G on input. On input, the scale factor has no effect if there is an exponent in the external field. G output makes use of the scale factor only if E conversion is necessary to convert the data.

INPUT/OUTPUT STATEMENTS 10

The following definitions apply to all I/O statements:

i logical I/O unit number:

an integer constant of one or two digits (the first must not be zero)

integer variable names of no more than 7 characters, with a value of 1 to 99

n FORTRAN declaration identifier:

statement number

variable identifier which references the starting storage location of FORMAT information

L I/O list

The logical BCD record and the logical binary record for each I/O device as used with the 6000 Series Computer System are defined as follows:

Printer Card Reader Card Punch	Logical BCD record is a one-card image (80 characters) for the card I/O devices and 136† characters (1 print line) for the printer. Logical binary record, for the card I/O devices, is a number of cards between the EOR cards (7,8,9 punch in column 1); for the printer, it is a number of print-lines between the EOR marks.
One Inch Tape	Logical BCD record is a character string ending with a zero byte blocked in 512-word physical records. Logical binary record is a number of chained blocks (512 words).
One-half Inch Tape	Logical BCD record is a physical record (even parity) containing up to 136† characters. Logical binary record is a set of chained blocks (512 words) with odd parity.
Disk	Logical BCD record is a character string ending with a zero byte (blocked in sectors). Logical binary record is a number of chained sectors.

†As the first character specifies carriage control, only 135 characters are printed.

10.1 OUTPUT STATEMENTS

PRINT n, L

Information in the list (L) is transferred from the storage locations to the standard output unit as line printer images, 136 characters or less per line in accordance with the FORMAT declaration, n. The maximum record length is 136 characters, but the first character of every record is not printed as it is used for carriage control† when printing on-line. Characters in excess of the print line appear on the succeeding line. Each new record starts a new print line.

For off-line printing, the printer control is determined by the installation's printer routine.

PUNCH n,L

Information is transferred from the storage locations given by the list (L) identifiers to the standard punch unit. Information is transferred as Hollerith images, 80 characters or less per card in accordance with the FORMAT declaration, n.

WRITE (i,n)L

WRITE OUTPUT TAPE i,n,L

These forms are equivalent; they transfer information from storage locations given by the list (L) to a specified output unit (i) according to the FORMAT declaration (n).

With a half inch tape unit, a logical record containing up to 136 characters is recorded in even parity (BCD mode). The number of words in the list and the FORMAT declaration determine the number of records that are written on a unit. If the logical record is less than 136 characters, the remainder of the record is filled with blanks.

†See Appendix O for carriage control character conventions.

With a one-inch tape unit, a packed 5120-character physical record is recorded in odd parity. Each physical record consists of as many logical record characters as required to fill the physical record. The information is recorded in 6000 series display code with no special control characters added, and it represents a continuous stream of logical output records. Trailing blanks on each logical record are removed and two consecutive characters with a value of zero separate logical records on the tape.

If the tape is to be printed, the first character of a record is not printed as it is a printer control. If the programmer fails to allow for a printer control character, the first character of the output data is lost on the printed listing.

WRITE (i)L

WRITE TAPE i,L

These equivalent forms transfer information from storage locations given by the list (L) to a specified output unit (i). If L is omitted, the WRITE (i) statement acts as a do-nothing statement. See READ (i)L.

If blocked binary option is not used, the number of words in the list determines the number of physical records written on the unit. A physical record contains a maximum of 512 central storage words; the last physical record may contain from 1 to 512 words. Physical records written by one WRITE (i)L statement constitute one logical record. The information is recorded in odd parity (binary mode). For blocked binary I/O files see FTNBIN routine in section 7-10 and appendix M.

A logical record which is an exact multiple of 512 words is followed by a physical record of eight zero characters called a zero length record.

Examples:

```
1.        DIMENSION A(260),B(4000)
          WRITE(10)A,B
2.        DO 5 I = 1,10
       5  WRITE TAPE 6, AMAX (I),(M(I,J),J=1,5)
3.        PRINT 50,A,B,C(I,J)
      50  FORMAT (X 8HMINIMUM=F17.7,2X8HMAXIMUM=F17.7,
          2X10HVALUE IS $F8.2)
4.        PRINT 51, (A(I),I=1,20)
      51  FORMAT(X23HTRUTH MATRIX VALUES ARE/(3X4L3))
5.        PUNCH 52,ACCT, LSTNME, FSTNME, TELNO, SHPDTE,ITMNO
      52  FORMAT (F8.2,3X4A10,XI5)
```

The format on the previous page assumes the following dimension statement:

```
      DIMENSION LISTNME(2),FSTNME(2)
      WRITE (2,53)A,B,C,D
53    FORMAT (4E21.9)
      WRITE OUTPUT TAPE 2,52,A,B,C,D
      WRITE (2,54)
54    FORMAT (32HTHIS STATEMENT HAS NO DATA LIST.)
```

10.2 READ STATEMENTS

A check should be made for the end of the file (either by counting records or by an IF EOF statement after each read). If this check is not made and the EOF is reached, the data used for processing will remain unchanged from the last read. If a read is issued after the EOF is reached, the job will be terminated unless the EOF flag has been cleared by an IF EOF statement.

READ n, L

One or more card images are read from the standard input unit. Information is converted from left to right in accordance with FORMAT specification (n), and it is stored in the locations named by the list (L). Input may be on 80-column Hollerith cards or magnetic tapes prepared off-line, containing 80-character records in BCD mode.

Example:

```
      READ 10,A,B,C
10    FORMAT (3F10.4)
```

READ (i,n)L

READ INPUT TAPE i,n,L

These equivalent forms transfer one logical record of information from logical unit (i) to storage locations named by the list (L), according to FORMAT specification (n). The number of words in the list and the FORMAT specifications must conform to the record structure on the logical unit.

READ (i)L

READ TAPE i,L

These equivalent forms transfer one logical record of information from a specified unit (i) to storage locations named by the list (L).

Records to be read by READ (i) should be written in binary mode. The number of words in the list of READ (i)L must not exceed the number of words in the corresponding WRITE statement.

If L is omitted, READ (i) spaces over one logical record. See WRITE (i)L.

Examples:

```
1)        DIMENSION C(264)
          READ (10)C
          DIMENSION BMAX (10), M2 (10,5)
          DO7I=1, 10
      7   READ TAPE 6, BMAX (I), (M2(I,J),J=1,5)
          READ (5) (skip one logical record on unit 5)
          READ (6) ((A(I,J),I=1,100),J=1, 50)
          READ TAPE 6,((A(I,J),I=1,100),J=1, 50)

2)        READ INPUT TAPE 10,50,X,Y,Z
     50   FORMAT (3F10.6)
          DOUBLE PRECISION DB(4)
          READ (10,51) DB
     51   FORMAT (4D20.12)
          READ 51,DB
          READ (2,52) (Z(J),J=1,8)
     52   FORMAT (F10.4)
```

10.3 NAMELIST STATEMENT

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

NAMELIST /y_1/a_1/y_2/a_2/.../y_n/a_n

When NAMELIST appears with a DIMENSION, COMMON, EQUIVALENCE, or a type declaration, any arrays used in NAMELIST must have been dimensioned prior to the NAMELIST statement.

Each $\underline{y}$ is a NAMELIST name consisting of 1-7 characters which must be unique within the program unit in which it is used. Each $\underline{a}$ is a list of the form $b_1, b_2, \ldots, b_n$; each being a variable or array name.

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

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, $\underline{b}$ may be a dummy parameter identifying a variable or an array, but the array may not have variable dimensions.

A NAMELIST 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 only in a READ or WRITE statement. The form of the input/output statements used with NAMELIST is as follows:

```
READ (u,x)

WRITE (u,x)
```

$\underline{u}$ is an integer variable or integer constant denoting a logical unit, and $\underline{x}$ is a NAMELIST 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

$a=d_1,\ldots,d_j$

$a(n) = d_1,\ldots,d_m$

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:

```
$HURRY I1=1,L=.TRUE.,I2=2,I3=3.5,Y(3,5)=26,Y(1,1)=11,12.0E1,13,4*14,
Z=(1.,2.),K=16,M=17$
```

produce the following values:

```
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 $\underline{v}$ is set to $\underline{c}$

$a=d_1,\ldots,d_j$ the values $\underline{d}_1,\ldots,\underline{d}_j$ are stored in consecutive elements of array $\underline{a}$ in the order in which the array is stored internally.

$a(n)=d_1,\ldots,d_m$ 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 the order in which they appear in the defining NAMELIST statement.

OUTPUT DATA

Output to unit u of BCD information is as follows:

One record consisting of a $ in column 2 immediately followed by the identifier x. As many records as are needed to output the current values of all variables in the list associated with x. Simple variables are output as v=c.

Elements of dimensioned variables are output in the order in which they are stored internally.

The data fields are made large enough to include all significant digits. Logical constants appear as T and F. No data appears in column 1 of any record.

One record consisting of a $ in column 2 immediately followed by the letters END.

The records output by such a WRITE statement may be read by a READ (u,x) statement where x is the same NAMELIST identifier.

If unit u is the standard punch unit and a record is longer than 80 characters, the remaining characters are punched on the next card.

The maximum length of a record written by a WRITE (u,x) statement is 130 characters.

10.4 TAPE HANDLING STATEMENTS

REWIND i

Magnetic tape unit i is rewound to load point. If the tape is already rewound, the statement acts as a do-nothing statement.

BACKSPACE i

Unit i is backspaced one logical record in a binary file or a BUFFER IN/OUT file or one BCD record in a normal BCD file. If tape is at load point (rewound), this statement acts as a do-nothing statement.

END FILE i

An end-of-file is written on magnetic tape unit i.

IF (ENDFILE i)n_1,n_2

IF (EOF,i)n_1,n_2

These statements check the previous read operation to determine if an end-of-file has been encountered on unit i. If so, control is transferred to statement n_1; if not, control is transferred to statement n_2.

IF (UNIT,i)n_1,n_2,n_3,n_4

n_1	not ready
n_2	ready and no previous error
n_3	EOF sensed on last input operation
n_4	parity error (buffered I/O operations)

With the present system, a write parity error on a unit being tested is not detected, as the operator is notified of the existing condition by the SCOPE Version 3.0 operating system.

10.5 BUFFER STATEMENTS

The primary differences between buffer I/O and read/write I/O statements are given below:

1. The mode of transmission (BCD or binary) is tacitly implied by the form of the read/write control statement. In a buffer control statement, parity must be specified by a parity indicator.

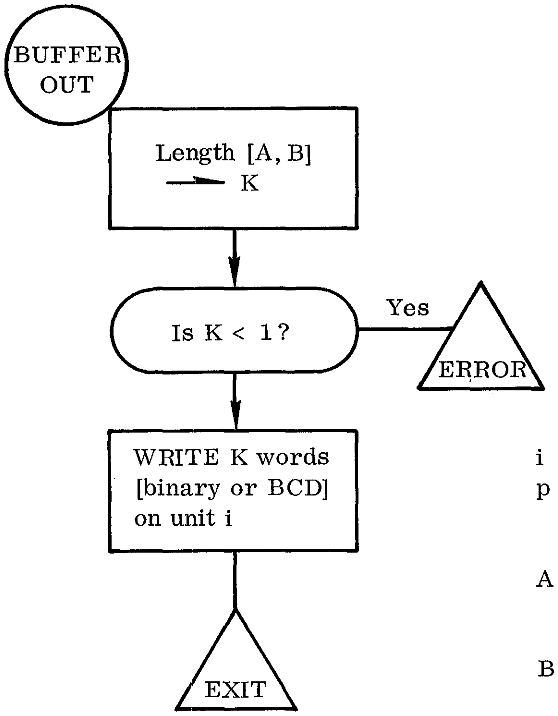
2. The read/write control statements are associated with a list and, in BCD transmission, with a FORMAT statement. The buffer control statements are not associated with a list; data transmission is to or from one area in storage.

3. A buffer control statement initiates data transmission, and then returns control to the program, permitting the program to perform other tasks while data transmission is in progress. Before buffered data is used, the status of the buffer operation should be checked. A read/write control statement completes the operation before returning control to the program.

In the descriptions that follow, these definitions apply.

u logical unit number

p parity key. May be specified by a simple variable (not subscripted). 0 for even parity (coded characters); non-zero for odd parity.



i logical unit number
p recording mode
0 even-BCD
1 odd-binary
A variable identifier: first word of data block to be transmitted.
B variable identifier: last word of data block to be transmitted.

In the BUFFER statements the address of B must be greater than that of A. A unit referenced in a BUFFER statement may not be referenced in other I/O statements.

BUFFER IN (u,p) (A,B)

Information is transmitted from unit u in mode p to storage locations A through B. The use of this statement is described in detail under BUFFEI in Appendix I.

BUFFER OUT (u,p) (A,B)

Information is transmitted from storage locations A through B and one logical record is written on unit u in mode p containing all the words from A to B inclusive. The use of this statement is described in detail under BUFFEO in Appendix I.

Examples:

1)
```
COMMON/BUFF/DATA(10),CAL(50)
PAR=0
BUFFER IN(9,PAR) (DATA(1),CAL(50))
```

BCD information is input from unit 9 to the labeled common area BUFF beginning at DATA(1), the first word of the block, and extending through CAL(50), the last word of the block.

2)
```
DIMENSION A(100)
N=6
BUFFER OUT(N,1) (A(1),A(100))
```

Binary information is transmitted to unit N from the block area defined by A(1) and A(100), that is, all of array A is transmitted.

10.6 ENCODE/DECODE STATEMENTS

The ENCODE/DECODE statements are comparable to the BCD 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:

ENCODE (c, n, v) L where

n statement number, variable identifier, or formal parameters representing the FORMAT statement

L input/output list

v variable identifier or an array identifier which supplies the starting location of the BCD record.

c unsigned integer constant or a simple integer variable (not subscripted) specifying the number of characters in the record. c may be an arbitrary number of BCD characters.

When encoding or decoding is performed, the first record begins with the leftmost character position specified by v and continues c BCD characters (10 BCD characters per computer word). For ENCODE, if c is not a multiple of 10, the record ends in the middle of a word and the remainder of the word is blank filled. For DECODE, if the record ends with a partial word the balance of the word is ignored.

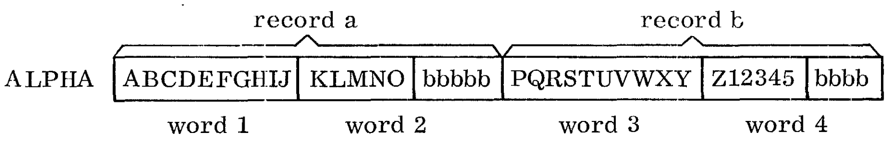
Since each succeeding record begins with a new computer word, an integral number of computer words is allocated for each record with $\frac{c+9}{10}$ words. The number of characters allocated for any single record in the encoded area must not exceed 150.

Example:

```
A(1) = ABCDEFGHIJ
A(2) = KLMNObbbbb
B(1) = PQRSTUVWXY
B(2) = Z12345bbbb
```

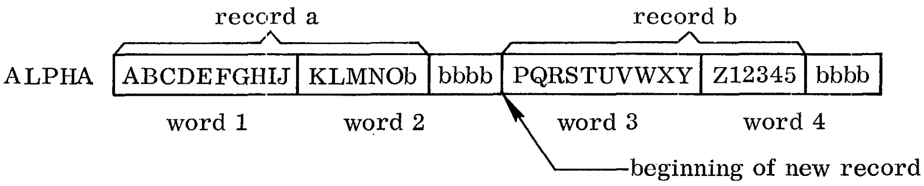
1) c = multiple of 10

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



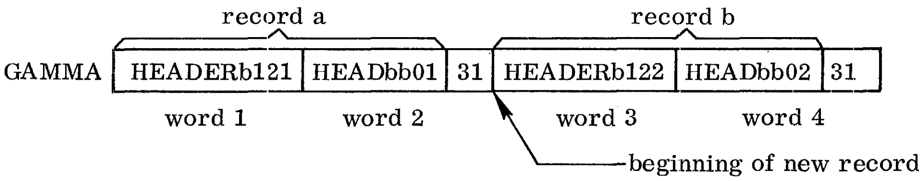
2) c ≠ multiple of 10

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



3) c ≠ multiple of 10

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



A6(1) = HEADERb121

A6(2) = HEADbb01bb

B6(1) = HEADERb122

B6(2) = HEADbb02bb

ENCODE (c,n,v)L

The information of the list variables, L, is transmitted according to the FORMAT (n) and stored in locations starting at v, c BCD characters per record. If the I/O list (L) and specification list (n) translate more than c characters per record, an execution diagnostic occurs. If the number of characters converted is less than c, the remainder of the record is filled with blanks.

DECODE (c,n,v)L

The information in c consecutive BCD characters (starting at address v) is transmitted according to the FORMAT n and stored in the list variables. If the number of characters specified by the I/O list and the specification list (n) is greater than c (record length) per record, an execution diagnostic occurs. If DECODE attempts to process an illegal BCD code or a character illegal under a given conversion specification, an execution diagnostic occurs.

Examples:

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

```
     LOC(1)  SSSSSxxxxx
               .
               .
               .
     LOC(6)  xxxxxddddd
             10 BCD ch/wd
```

To form SSSSSddddd 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)
```

2) DECODE may be used to calculate a field definition in a FORMAT specification at object time. Assume that in the statement FORMAT (2A8,Im) the programmer wishes to specify m at some point in the program, subject to the restriction $2 \leq m \leq 9$. The following program permits m to vary.

```
    IF(M.LT.10.AND.M.GT.1)1,2
  1 ENCODE (8,100,SPECMAT) M
100 FORMAT (6H(2A8,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: (2A8,Im). This packed FORMAT is stored in SPECMAT. SPECMAT contains (2A8,Im).

A and B will be printed under specification A8, and the quantity J under specification I2, or I3, or . . . or I9 according to the value of m.

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.

```
    I = 10HV = bbFT/SEC
    IA = 16
    ENCODE (10,1,I)I,IA,I
1   FORMAT (A2,I2,R6)
```

Before executing the above code

```
I = 26545555062450230503
```

After execution

```
I = 26543441062450230503
```

4) ENCODE/DECODE handles randomly formatted input data. The following example illustrates how ENCODE/DECODE is used to interpret format and reduce data to a desired format.

Problem: To search for items of information when a given word in the input string keys the format for the rest of the string. In the example, EFORMAT is the key word and EF is unique in that, if EF starts an input record, ORMAT is assumed to follow. The appearance of EF indicates a number in the string is in E format.

```
      DIMENSION IDATLIN (80),INTER(3)
      READ 1, IDATLIN
    1 FORMAT (80R1)
      .
      .
      .
      IF((IDATLIN(1).EQ.1RE).AND.(IDATLIN(2).EQ.1RF))
       GO TO 5000
      .
      .
      .
 5000 ENCODE(23,900,INTER) (IDATLIN(J),J=33,55)
  900 FORMAT (23R1)
      DECODE (23,901,INTER) NUMBER
  901 FORMAT (E23.14)
      .
      .
      .
      PRINT 1000,NUMBER
 1000 FORMAT (* NUMBER = *F8.3)
      .
      .
      .
      END
 7
  8
   9
 .
 .
 .
EFORMAT                    3.47315999999997    E+002
 .
 .
 .
 6
  7
   8
    9
```

Output will be

```
      NUMBER = b347.316
```

APPENDIX SECTION

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	67	∧	0-8-7	12	37
30	X	0-7	0-7	27	70	↑	11-8-5	8-4	55
31	Y	0-8	0-8	30	71	↓	11-8-6	0-8-7	56
32	Z	0-9	0-9	31	72	<	12-0 or 12-8-2	12-0 or 12-8-2	72
33	0	0	0	12	73	>	11-8-7	0-8-6	57
34	1	1	1	01	74	≤	8-5	12-8-4	15
35	2	2	2	02	75	≥	12-8-5	0-8-2	75
36	3	3	3	03	76	¬	12-8-6	11-8-7	76
37	4	4	4	04	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
00	: †	8-2	8-2	072
01	A	12-1	12-1	101
02	B	12-2	12-2	102
03	C	12-3	12-3	103
04	D	12-4	12-4	104
05	E	12-5	12-5	105
06	F	12-6	12-6	106
07	G	12-7	12-7	107
10	H	12-8	12-8	110
11	I	12-9	12-9	111
12	J	11-1	11-1	112
13	K	11-2	11-2	113
14	L	11-3	11-3	114
15	M	11-4	11-4	115
16	N	11-5	11-5	116
17	O	11-6	11-6	117
20	P	11-7	11-7	120
21	Q	11-8	11-8	121
22	R	11-9	11-9	122
23	S	0-2	0-2	123
24	T	0-3	0-3	124
25	U	0-4	0-4	125
26	V	0-5	0-5	126
27	W	0-6	0-6	127
30	X	0-7	0-7	130
31	Y	0-8	0-8	131
32	Z	0-9	0-9	132
33	0	0	0	060
34	1	1	1	061
35	2	2	2	062
36	3	3	3	063
37	4	4	4	064
40	5	5	5	065
41	6	6	6	066
42	7	7	7	067
43	8	8	8	070
44	9	9	9	071
45	+	12	12-8-6	053
46	–	11	11	055
47	*	11-8-4	11-8-4	052
50	/	0-1	0-1	057
51	(	0-8-4	12-8-5	050
52	)	12-8-4	11-8-5	051
53	$	11-8-3	11-8-3	044
54	=	8-3	8-6	075
55	blank	no punch	no punch	040
56	, (comma)	0-8-3	0-8-3	054
57	. (period)	12-8-3	12-8-3	056
60	#	0-8-6	8-3	043
61	' (apostrophe)	8-7	8-5	047
62	!	0-8-2	12-8-7	041
63	%	8-6	0-8-4	045
64	" (quote)	8-4	8-7	042
65	_ (underline)	0-8-5	0-8-5	137
66	]	11-0 or 11-8-2	11-0 or 11-8-2	175
67	&	0-8-7	12	046
70	@	11-8-5	8-4	100
71	?	11-8-6	0-8-7	077
72	[	12-0 or 12-8-2	12-0 or 12-8-2	173
73	>	11-8-7	0-8-6	076
74	<	8-5	12-8-4	074
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 11-8-2	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	73	>	11-8-7	0-8-6	57
36	3	3	3	03	74	≤	8-5	12-8-4	15
37	4	4	4	04	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_8 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 keypunch) 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).

FORTRAN STATEMENT LIST B

SUBPROGRAM STATEMENTS				Page
Entry Points		ENTRY name	N	7-8
		PROGRAM name $(f_1,\ldots,f_n)$	N	7-3
		FORTRAN VI PROGRAM name $(f_1,\ldots,f_n)$	N	7-3
		FORTRAN IV PROGRAM name $(f_1,\ldots,f_n)$	N	7-3
		FORTRAN II PROGRAM name $(f_1,\ldots,f_n)$	N	7-3
		SUBROUTINE name $(p_1,\ldots,p_n)$	N	7-4
		FORTRAN VI SUBROUTINE name $(p_1,\ldots,p_n)$	N	7-4
		FORTRAN IV SUBROUTINE name $(p_1,\ldots,p_n)$	N	7-4
		FORTRAN II SUBROUTINE name $(p_1,\ldots,p_n)$	N	7-4
		FUNCTION name $(p_1,\ldots,p_n)$	N	7-13
		type FUNCTION name $(p_1,\ldots,p_n)$	N	7-13, 7-15
		FORTRAN VI FUNCTION name $(p_1,\ldots,p_n)$	N	7-13
		FORTRAN IV FUNCTION name $(p_1,\ldots,p_n)$	N	7-13
		FORTRAN II FUNCTION name $(p_1,\ldots,p_n)$	N	7-13
		FORTRAN IV type FUNCTION name $(p_1,\ldots,p_n)$	N	7-13
		FORTRAN II type FUNCTION name $(p_1,\ldots,p_n)$	N	7-13
		FORTRAN VI type FUNCTION name $(p_1,\ldots,p_n)$	N	7-13
Intersubroutine		EXTERNAL $\text{name}_1,\text{name}_2\ldots$	N	7-6
	F	$\text{name}_1,\text{name}_2,\ldots$†	N	D-2
Transfer Statements		CALL name	E	7-5
		CALL name $(p_1,\ldots,p_n)$	E	7-5
		RETURN	E	6-10

N = Non-executable E = Executable

† Col. 1 indicator (F) is used in FORTRAN II modes.

DATA DECLARATION AND STORAGE ALLOCATION			Page
Type Declaration	COMPLEX List	N	5-1
	DOUBLE PRECISION List	N	5-1
	DOUBLE List	N	5-1
	REAL List	N	5-1
	INTEGER List	N	5-1
	LOGICAL List	N	5-1
	TYPE DOUBLE List	N	5-1
	TYPE COMPLEX List	N	5-1
	TYPE REAL List	N	5-1
	TYPE INTEGER List	N	5-1
	TYPE LOGICAL List	N	5-1
Storage Allocations	DIMENSION $v_1,v_2,\ldots,v_n$	N	5-2
	COMMON/i_1/list$_1$/i_2/list$_2$.../i_n/list$_n$	N	5-3
	EQUIVALENCE (A,B,...),(A1,B1,...)...	N	5-7
	DATA $d_1,\ldots,d_n/a_1,k*a_2,\ldots,a_n/,d_1,\ldots,d_n/$	N	5-9
	DATA (i_1=value list),(i_2=value list),...	N	5-12
	BLOCK DATA	N	5-13
	BLOCK DATA name	N	5-13
ARITHMETIC STATEMENT FUNCTION			
	name ($p_1,p_2,\ldots p_n$) = Expressions	E	7-14
SYMBOL MANIPULATION, CONTROL AND I/O			
Replacement	A=E Arithmetic	E	4-2
	D A=E†		
	I A=E†		
	L=E Logical/Relational	E	4-4
	M=E Masking	E	4-4
	B M=E†		

†Col. 1 indicators (D, I, B) are used in FORTRAN II mode.

			Page
Intraprogram Transfers	GO TO n	E	6-1
	GO TO m	E	6-1
	GO TO m, $(n_1,...n_m)$	E	6-1
	GO TO $(n_1,...,n_m)$,i	E	6-2
	IF ACCUMULATOR OVERFLOW n_1,n_2†	E	D-2
	IF QUOTIENT OVERFLOW n_1,n_2†	E	D-2
	IF (c)n_1,n_2,n_3	E	6-3
	IF (ℓ)n_1,n_2	E	6-3
	IF (ℓ)s	E	6-3
	IF DIVIDE CHECK n_1,n_2†	E	D-2
	IF (ENDFILE i)n_1,n_2	E	10-10
	IF (EOF,i)n_1,n_2	E	10-10
	IF (SENSE LIGHT i)n_1,n_2†	E	D-2
	IF (SENSE SWITCH i)n_1,n_2†	E	D-2
	IF (UNIT,i)n_1,n_2,n_3,n_4	E	10-10
LOOP CONTROL	DO n i = m_1,m_2,m_3	E	6-4
MISCELLANEOUS PROGRAM CONTROLS			
	ASSIGN s to m	E	6-1
	n CONTINUE	E	6-10
	PAUSE	E	6-10
	PAUSE n	E	6-10
	SENSE LIGHT i†	E	D-1
	STOP	E	6-10
	STOP n	E	6-10
I/O FORMAT			
	FORMAT ($spec_1$,...,k($spec_m$,...),...	N	9-4
	NAMELIST/y_1/a_1/y_2/a_2/.../y_n/a_n	N	10-6

† FORTRAN II

			Page
I/O CONTROL STATEMENTS			
	BUFFER IN (u,p) (A,B)	E	10-12
	BUFFER OUT (u,p) (A,B)	E	10-12
	DECODE (c,n,v)L	E	10-15
	ENCODE (c,n,v)L	E	10-13
	PRINT n,L	E	10-2
	PUNCH n,L	E	10-2
	READ (i)L	E	10-5
	READ (i,n)L	E	10-4
	READ INPUT TAPE i,n,L	E	10-4
	READ n,L	E	10-4
	READ TAPE i,L	E	10-5
	WRITE (i)L	E	10-3
	WRITE (i,n)L	E	10-2
	WRITE OUTPUT TAPE i,n,L	E	10-2
	WRITE TAPE i,L	E	10-3
I/O Tape Handling Handling	END FILE i	E	10-9
	REWIND i	E	10-9
	BACKSPACE i	E	10-9
PROGRAM AND SUBPROGRAM TERMINATION			
	END	E	6-11
	END name	E	6-11
SEGMENTATION			
	CALL OVERLAY (f_n,l_1,l_2,p)	E	8-5
	CALL SEGMENT (f_n,e,a,lib,m)	E	8-2
	SEGZERO (sn,pn_1,pn_2,...,pn_i)	E	8-7
	SECTION (sname,pn_1,pn_2,...,pn_i)	E	8-7

FORTRAN FUNCTIONS C

IN-LINE FUNCTIONS

Form	Definition	Actual Parameter Type	Mode of Result
ABS(X)	Absolute value	Real	Real
AIMAG(C)	Obtain the imaginary part of a complex argument	Complex	Real
AINT(X)	Truncation integer. Sign of X times largest integer $\leq \lvert X \rvert$.	Real	Real
AMAX0(I_1,I_2, ...)	Determine maximum argument	Integer	Real
AMAX1(X_1,X_2, ...)	Determine maximum argument	Real	Real
AMIN0(I_1,I_2, ...)	Determine minimum argument	Integer	Real
AMIN1(X_1,X_2, ...)	Determine minimum argument	Real	Real
AMOD(X_1,X_2)†		Real	Real
AND(X_1, ..., X_n)	Boolean AND of X_1, ..., X_n	-	Logical
CMPLX(X_1,X_2)	Convert real to complex ($X_1 + iX_2$)	Real	Complex
COMPL(X)	Complement of X	-	Logical
CONJG(C)	Conjugate of C	Complex	Complex
DIM(X_1,X_2)	If $X_1 > X_2$:$X_1 - X_2$ If $X_1 \leq X_2$:0	Real	Real
DMAX1(D_1,D_2, ...)	Determine maximum argument	Double	Double
DMIN1(D_1,D_2, ...)	Determine minimum argument	Double	Double
FLOAT(I)	Integer to real conversion	Integer	Real
IABS(I)	Absolute value	Integer	Integer
IDIM(I_1,I_2)	If $I_1 > I_2$: $I_1 - I_2$ If $I_1 \leq I_2$: 0	Integer	Integer

†AMOD(X_1,X_2) is defined as $X_1-[X_1/X_2]X_2$, where [x] is an integer with magnitude of not more than the magnitude of x and sign the same as x.

Form	Definition	Actual Parameter Type	Mode of Result
IFIX(X)	Real to integer conversion	Real	Integer
INT(X)	Truncation, integer. Sign of X times largest integer $\leq \lvert X \rvert$.	Real	Integer
ISIGN(I_1,I_2)	Sign of I_2 times absolute value of I_1.	Integer	Integer
MAX0($I_1,I_2,\ldots$)	Determine maximum argument	Integer	Integer
MAX1($X_1,X_2,\ldots$)	Determine maximum argument	Real	Integer
MIN0($I_1,I_2,\ldots$)	Determine minimum argument	Integer	Integer
MIN1($X_1,X_2,\ldots$)	Determine minimum argument	Real	Integer
MOD(I_1,I_2)†		Integer	Integer
OR($X_1,\ldots,X_n$)	Boolean OR of $X_1,\ldots,X_n$	-	Logical
REAL(C)	Obtain the real part of a complex argument	Complex	Real
SIGN(X_1,X_2)	Sign of X_2 times absolute value of X_1.	Real	Real

LIBRARY FUNCTIONS

ACOS(X)	Arccosine in radians	Real	Real
ALOG(X)	Natural log of X	Real	Real
ALOG10(X)	Log to the base 10 of X	Real	Real
ASIN(X)	Arcsine in radians	Real	Real
ATAN(X)	Arctangent in radians	Real	Real
ATAN2(X_1,X_2)	Arctangent (X_1/X_2) in radians	Real	Real
CABS(C)	Absolute value	Complex	Real
CCOS(C)	Complex cosine, argument in radians	Complex	Complex
CEXP(C)	Complex exponent	Complex	Complex
CLOG(C)	Complex log	Complex	Complex
COS(X)	Cosine X radians	Real	Real
CSIN(C)	Complex sine, argument in radians	Complex	Complex
CSQRT(C)	Complex square root	Complex	Complex
DABS(D)	Absolute value	Double	Double
DATAN(D)	Double arctangent in radians	Double	Double
DATAN2(D_1,D_2)	Double arctangent: D_1/D_2 in radians	Double	Double

† MOD(I_1,I_2) is defined as $I_1-[I_1/I_2]I_2$, where [x] is an integer with magnitude not more than the magnitude of x and sign the same as x.

Form	Definition	Actual Parameter Type	Mode of Result
DBLE(X)	Real to double	Real	Double
DCOS(D)	Double cosine, argument in radians	Double	Double
DEXP(D)	Double exponent	Double	Double
DLOG(D)	Natural log of D	Double	Double
DLOG10(D)	Log to the base 10 of D	Double	Double
DMOD(D_1, D_2)†		Double	Double
DSIGN(D_1, D_2)	Sign of: D_2 times absolute value of D_1	Double	Double
DSIN(D)	Sine of double precision argument in radians	Double	Double
DSQRT(D)	Double square root	Double	Double
EXP(X)	e to Xth power	Real	Real
IDINT(D)	Double to integer. Sign of D times largest integer $\leq \|D\|$.	Double	Integer
LEGVAR(A)	Returns -1 if variable is indefinite, +1 if out of range, and 0 if normal	Real	Integer
LENGTH(I)	Returns number of words transferred to CM from unit I after BUFFER IN	Integer	Integer
RANF(X)	Random number generator; typical call follows: Y=RANF(X) where X is type real. Three conditions exist on X. 1. If X is zero, the next random number is generated and returned. 2. If X is negative, a random number is not generated but the last previously generated random number (or the seed if no random number has been generated) is returned. 3. If X is positive, the exponent part of X is set to 1717_8 and the low order bit is set to one. This result is returned as the seed of a new sequence, and any additional calls to RANF will be based on a sequence using this seed.	Real	Real
SNGL(D)	Double to real (unrounded)	Double	Real
SIN(X)	Sine X radians	Real	Real
SQRT(X)	Square root of X	Real	Real
TAN(X)	Tangent X radians	Real	Real
TANH(X)	Hyperbolic tangent X radians	Real	Real

† DMOD(D_1, D_2) is defined as $D_1-[D_1/D_2]D_2$, where [x] is an integer with magnitude of not more than the magnitude of x and sign the same as x.

Following functions accept A as a variable address name for an actual parameter:

Form	Definition	Actual Parameter Type	Mode of Result
LOCF(A)	Returns address of argument A	-	Integer
XLOCF(A)	Returns address of argument A	-	Integer

SOME FORTRAN II, 63, IV DIFFERENCES

The following FORTRAN II statements are accepted by FORTRAN:

1. In FORTRAN II arithmetic replacement statements, column 1 may contain either of the following characters.

D	Double Precision mode
I	Complex mode

   When these characters are encountered, all variables and constants in the statement are assumed to be of the same type (double precision or complex).

2. FORTRAN II statements which contain a B in column 1 (Boolean) are evaluated as masking expressions. The operator equivalences are:

FORTRAN	FORTRAN II
.AND.	*
.NOT.	-
.OR.	+
none	/

   Exclusive OR function defined as:

p	v	p/v
1	1	0
1	0	1
0	1	1
0	0	0

3. Mixed mode variables may appear in any FORTRAN II Boolean, B-type, Statement.

4. SENSE LIGHT STATEMENTS

   SENSE LIGHT i

   The statement turns on SENSE light i; i must be an integer constant in the range 1 to 6.

SENSE LIGHT 0 turns off all sense lights.

IF (SENSE LIGHT i)n_1,n_2

The statement tests sense light i. If it is on, it is turned off, and a jump occurs to statement n_1. If it is off, a jump occurs to statement n_2. The n_i are statement labels; i must be an integer constant in the range 1 to 6.

5. IF SENSE SWITCH STATEMENT

IF (SENSE SWITCH i)n_1,n_2

If sense switch i is set (on), a jump occurs to statement n_1. If it is not set (off), a jump occurs to statement n_2; i may be a simple integer variable constant (1 to 6).

6. FAULT CONDITION STATEMENTS

At execute time, the computer may be set to interrupt on divide overflow or exponent fault. The fault indicator must be checked immediately after any statement that could possibly cause a fault condition.

IF DIVIDE CHECK n_1,n_2

A divide check occurs following division by zero. The statement checks for this condition: if it has occurred jump to statement n_1 takes place. If no check exists, a jump to statement n_2 takes place.

IF QUOTIENT OVERFLOW n_1,n_2

IF ACCUMULATOR OVERFLOW n_1,n_2

An overflow occurs when the result of a real, double precision, or complex arithmetic operation exceeds the upper limits specified for these types. Results that are less than the lower limits are set to zero without indication. This statement is therefore a test for floating point overflow only. If the condition has occurred, a jump to statement n_1 takes place. If the condition does not exist, a jump to statement n_2 takes place.

7. FORTRAN accepts the FORTRAN II version of the EXTERNAL statement. This form contains the same name list, but the word EXTERNAL has been replaced by the character F in column 1 of the statement.

1   7

F   name$_1$,name$_2$, ...

8. The only inherently incompatible areas are the following:

COMMON-EQUIVALENCE Statement Relationships

In FORTRAN II, equivalence groups can reorder the common variables and arrays, and more than one variable in an equivalence group can be in common.

In FORTRAN, equivalence groups do not reorder common, but may only extend the length of a common block.

Function-Naming Conventions

In FORTRAN II, the following rules apply for function subprogram, library function and statement function names:

The name is 4-7 alphanumeric characters, ending with the character F.

The first character must be X if, and only if, the value of the function is integer; for any other first character, the value of the function is real.

In FORTRAN, the number of characters in the function name is 1-7; the first character must be alphabetic.

FORTRAN 63 and FORTRAN IV DIFFERENCES

DO STATEMENT

If the terminal value of the DO statement (m_2) is less than the initial value (m_1):

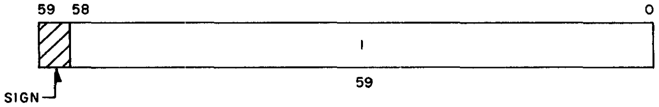
FORTRAN IV	DO loop is executed once
FORTRAN 63	DO loop is not executed

END STATEMENT

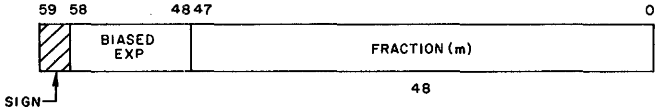
FORTRAN IV	in subprograms, the END statement not preceded by a RETURN statement will cause the compilation of termination instruction (STOP)
FORTRAN 63	similar condition is compiled with an assumed RETURN statement

COMPUTER WORD STRUCTURE OF CONSTANTS-6600 E

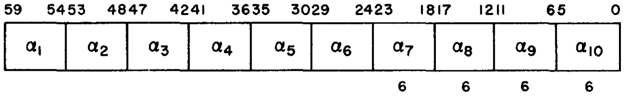
INTEGER



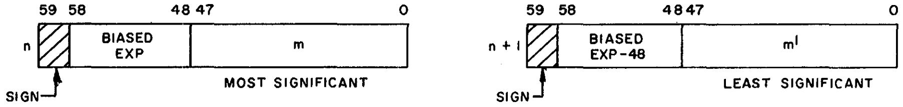
REAL



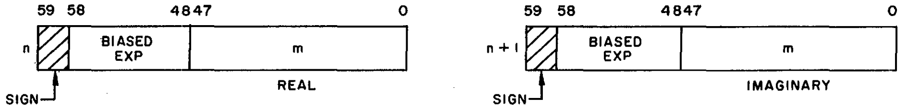
HOLLERITH BCD AND DISPLAY CODE


59 54 53 48 47 42 41 36 35 30 29 24 23 18 17 12 11 6 5 0
α_1 α_2 α_3 α_4 α_5 α_6 α_7 α_8 α_9 α_{10}
6 6 6 6

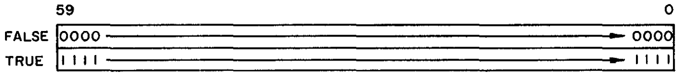
DOUBLE-PRECISION



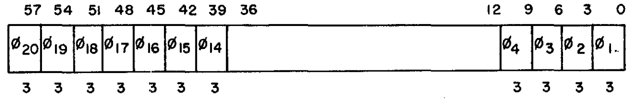
COMPLEX



LOGICAL



OCTAL


57 54 51 48 45 42 39 36
12 9 6 3 0
ϕ_{20} ϕ_{19} ϕ_{18} ϕ_{17} ϕ_{16} ϕ_{15} ϕ_{14}
ϕ_4 ϕ_3 ϕ_2 ϕ_1
3 3 3 3 3 3 3
3 3 3 3

COMPILATION AND EXECUTION

FORTRAN Control Card

The FORTRAN compiler is called by the control card:

```
RUN(cm, fl, bl, if, of, rf, lc, as, cs)
```

cm Compiler mode option; (if omitted, assume G; if unrecognized, assume S)

- G compile and execute, nolist unless explicit LIST cards appear in the deck
- S compile with source list, no execute
- †P compile with source list and punch deck on file PUNCHB, no execute
- L compile with source and object list which contains mnemonics, no execute
- †M compile with source and object list which contains mnemonics, produce a punch deck on file PUNCHB, no execute

fl object program field length (octal); if omitted, it is set equal to the field length at compile time.

bl object program I/O buffer lengths (octal); if omitted, assumed to be 2022_8

if file name for compiler input; if omitted assumed to be INPUT

of file name for compiler output; if omitted, assumed to be OUTPUT

rf file name on which the binary information is always written; if omitted, assumed to be LGO.

lc line-limit (octal) on the OUTPUT file of an object program. If omitted, assumed to be 10000_8. If the line count exceeds the specified line limit, the job is terminated.

as if non-zero or non-blank, the USASI switch causes the USASI I/O list/format interaction at execution time. It has no effect on the compilation method.

cs cross-reference switch. If non-zero a cross reference listing is produced.

A dayfile message indicates that an error has been detected in the RUN card fields 2-7:

```
RUN CARD FIELD IN ERROR- xxxxxxxxxx
```

xxxxxxxxxx is the input field in error.

† Because COMPASS allows only one binary output file to be written, a RUN (P or M) and LGO. will result in only the FORTRAN code of a FORTRAN-COMPASS job being placed on LGO. If P or M compile mode options are used and the sixth parameter is PUNCHB, two binary decks will be punched.

Compiler output, except in the G mode, includes a reproduction of the source program, a variable map, and indications of fatal and non-fatal errors detected during compilation. If the G mode is selected, all output is suppressed unless fatal errors are detected in which case the output is the same as indicated for the S mode of compilation. If the L or M mode is selected, the output includes an object list which contains mnemonics.

A copy of the compiled programs is always left in disk storage as a binary record on a file named either LGO or the name specified as the rf parameter in the call to the compiler. The compiled program may be called and executed repeatedly, until the end of job occurs, by using the name of the load-and-go file. In the output file at the end of compilation of each subprogram, the compiler indicates the amount of unused compilation space.

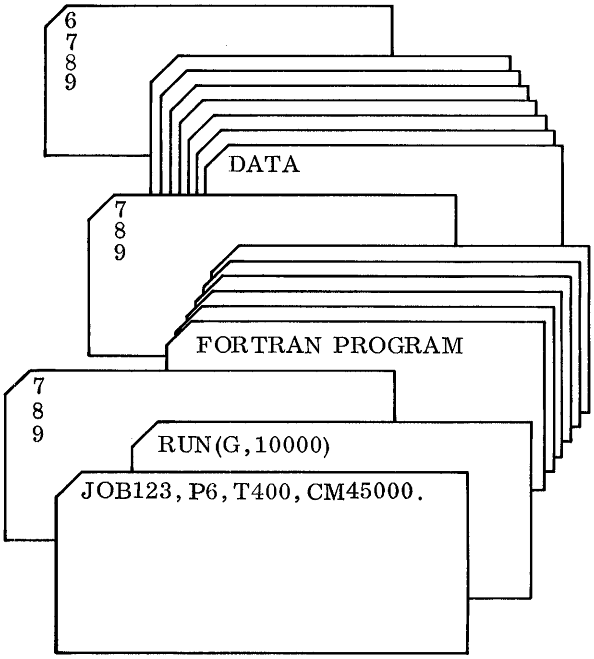
Two control cards LIST and NOLIST are available to allow the programmer more flexibility in requesting a list of his programs. These cards are free field beyond column 6 and appear between subprograms. When the LIST card is detected, the source cards of the following programs are listed. If the compiler mode was L, the object code is also listed. When the NOLIST card is detected, no more listing takes place until a LIST card is detected. Each LIST card will probably cause repositioning of the output file, therefore excessive LIST-NOLIST sequences should be avoided.

To aid in the preparation of overlay and segment files, the FORTRAN compiler, upon detecting an OVERLAY or SEGMENT card between subprograms, transfers them to the load-and-go file, and to the PUNCHB file if the P or M option is selected. They also are transferred to the output file.

'The following types of control cards are transferred to the load-and-go and PUNCHB file if mode is P or M.

```
     OVERLAY (...
     SECTION (...
     SEGZERO (         (These statements must begin after column 6)
     SEGMENT (
```

COMPILE AND EXECUTE



The above control card sequence will compile in a field of 45000_8 words and run in a field of 10000_8 words.

DECK STRUCTURE FOR A NORMAL COMPILE AND EXECUTE

Job name	JOB123
Priority	6
Time limit	approx. 4 minutes
Field length	45000_8 words

Compile and execute with no list and no binary deck.

FORTRAN Load and Run

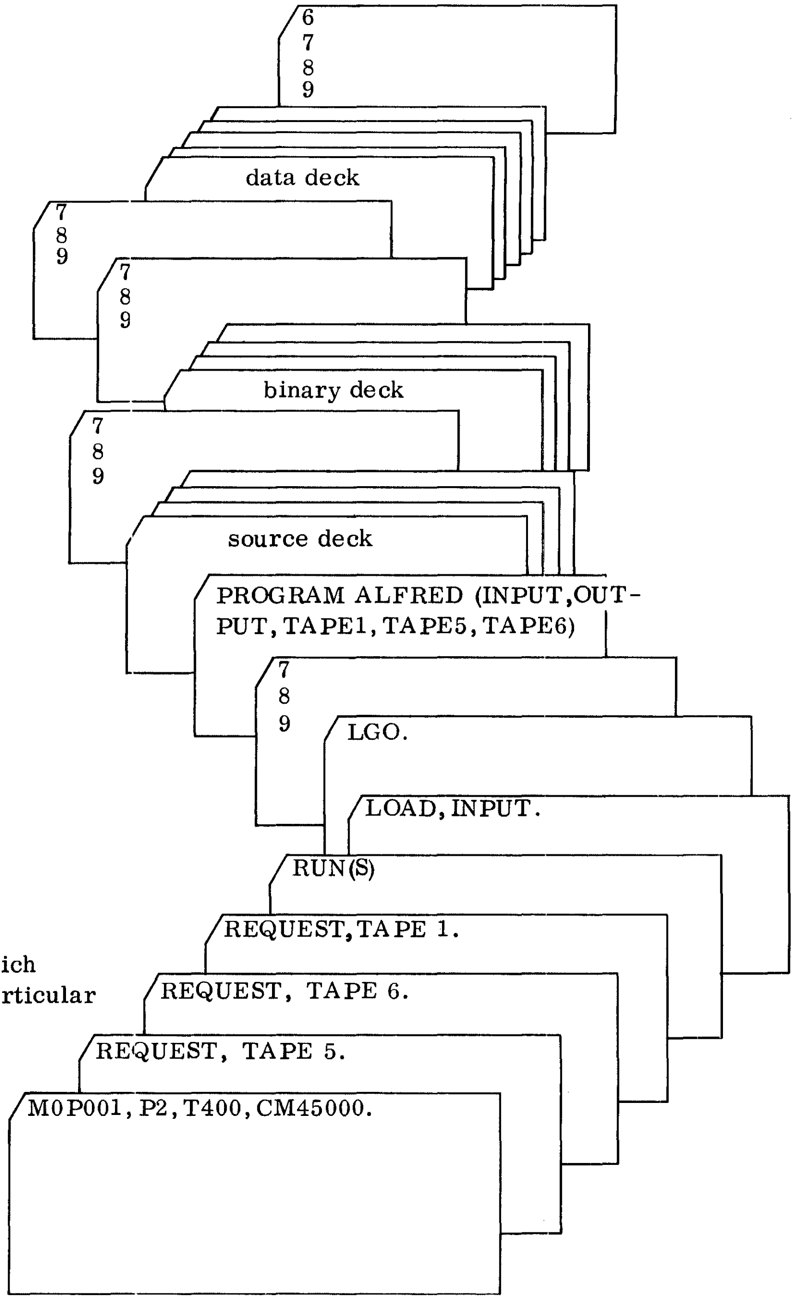
Job name: M0P001

Three tape references:

TAPE 1 assumed input tape which operator loads on a particular unit

TAPE 5 } output scratch tapes
TAPE 6 } drawn from tape pool

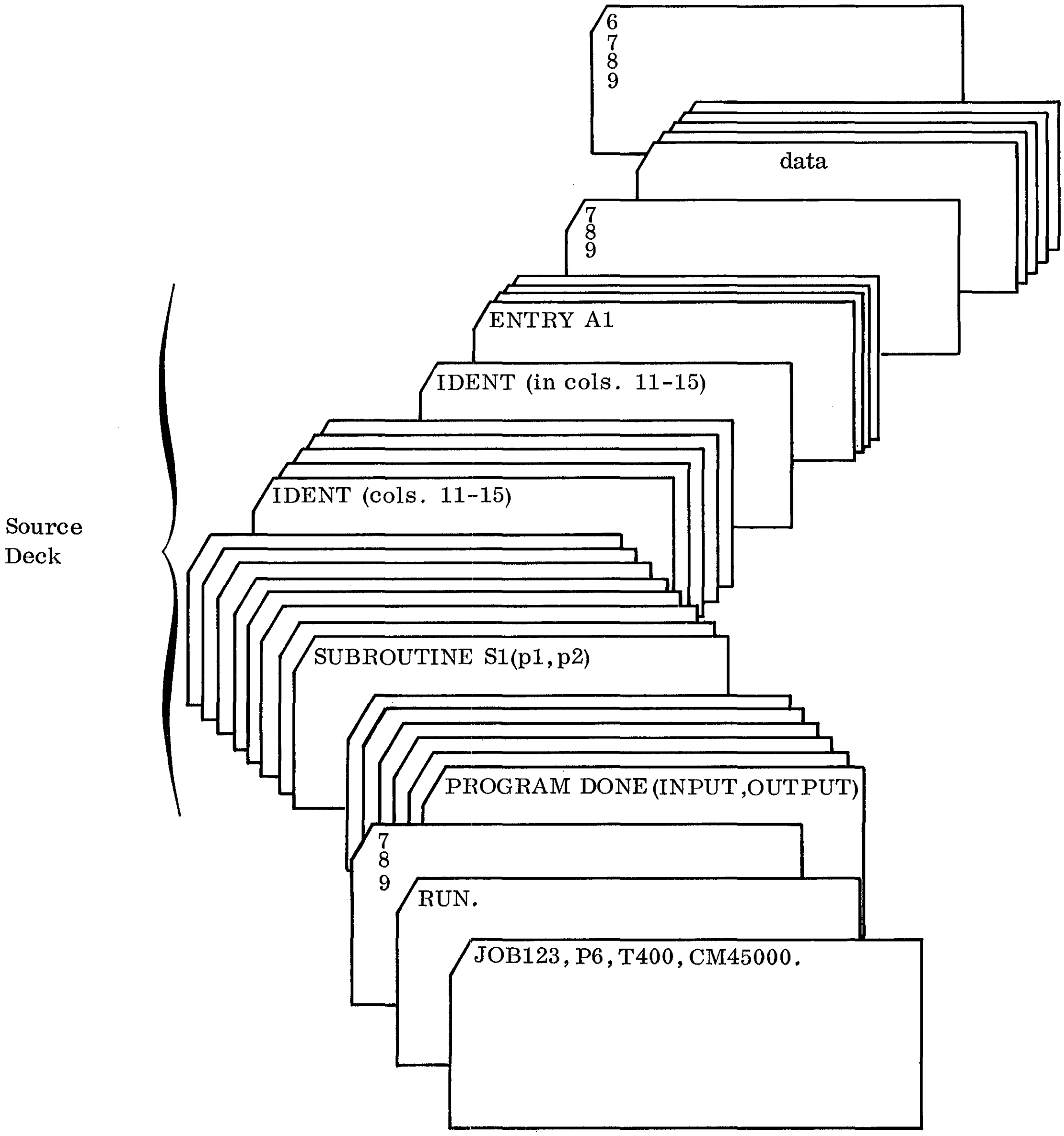
Some binary decks on INPUT



```
M0P001,P2,T400,CM45000.
REQUEST, TAPE 5.
REQUEST, TAPE 6.
REQUEST,TAPE 1.
RUN(S)
LOAD,INPUT.
LGO.
7
8
9
PROGRAM ALFRED (INPUT,OUT-
PUT,TAPE1,TAPE5,TAPE6)
source deck
7
8
9
binary deck
7
8
9
7
8
9
data deck
6
7
8
9
```

The FORTRAN compiler, RUN, causes the COMPASS assembler to be loaded and transfers control to it upon the detection of a header card that has IDENT beginning in column 11. When a header card is found that the COMPASS assembler does not recognize, control returns to the RUN compiler which continues processing. COMPASS is directed to produce the type of output specified by the compiler mode (note limitations of P and M on page F-1).

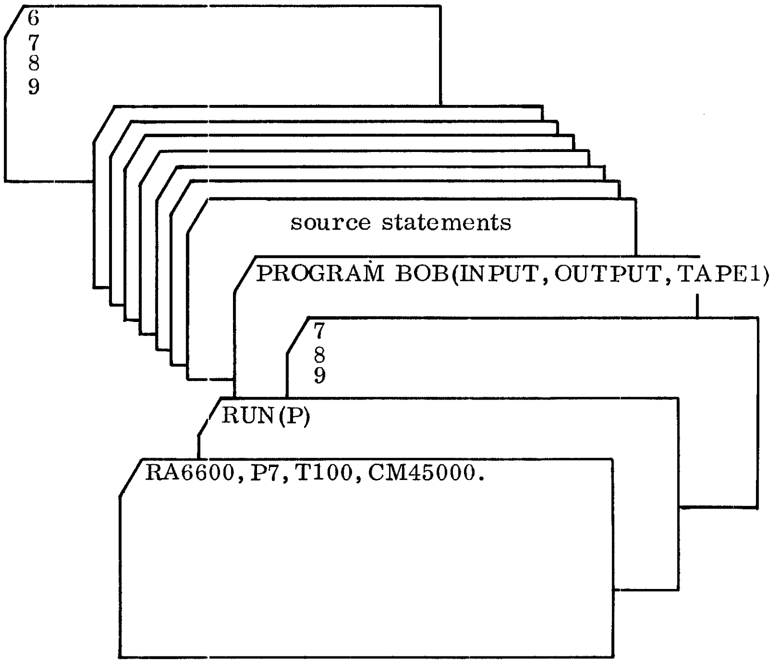
FORTRAN Compile and Execute with Mixed Deck



FORTRAN Compile and Produce Binary Card; no execution.

Three files of I/O - INPUT, OUTPUT AND TAPE1

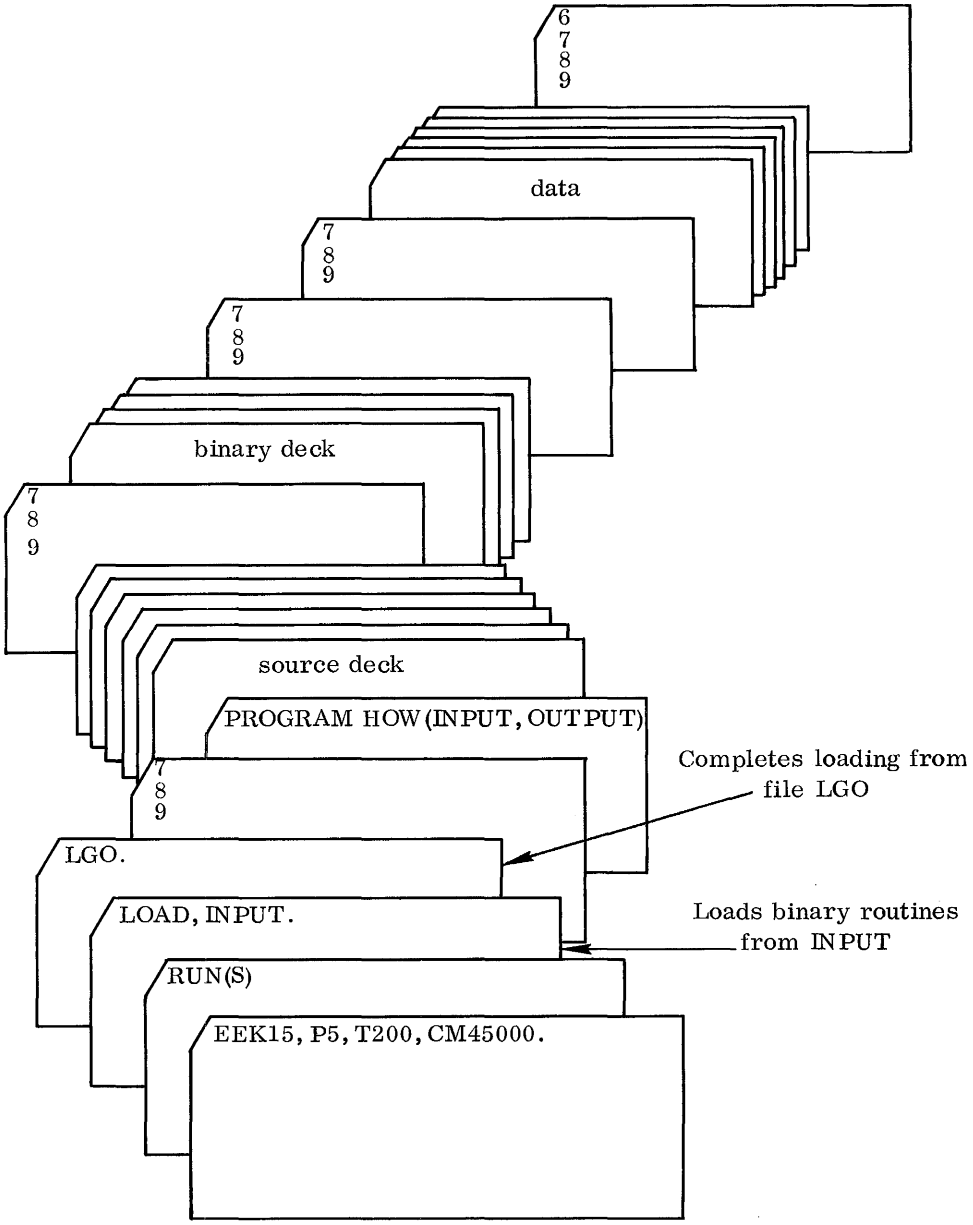
Job name	RA6600
Priority	7
Time limit	approximately 1 minute
field length	45000_8 words



```
PROGRAM BOB(INPUT,OUTPUT,TAPE1)
RUN(P)
RA6600,P7,T100,CM45000.
```

FORTRAN Compile and Execute (plus a prepunched binary subroutine deck)

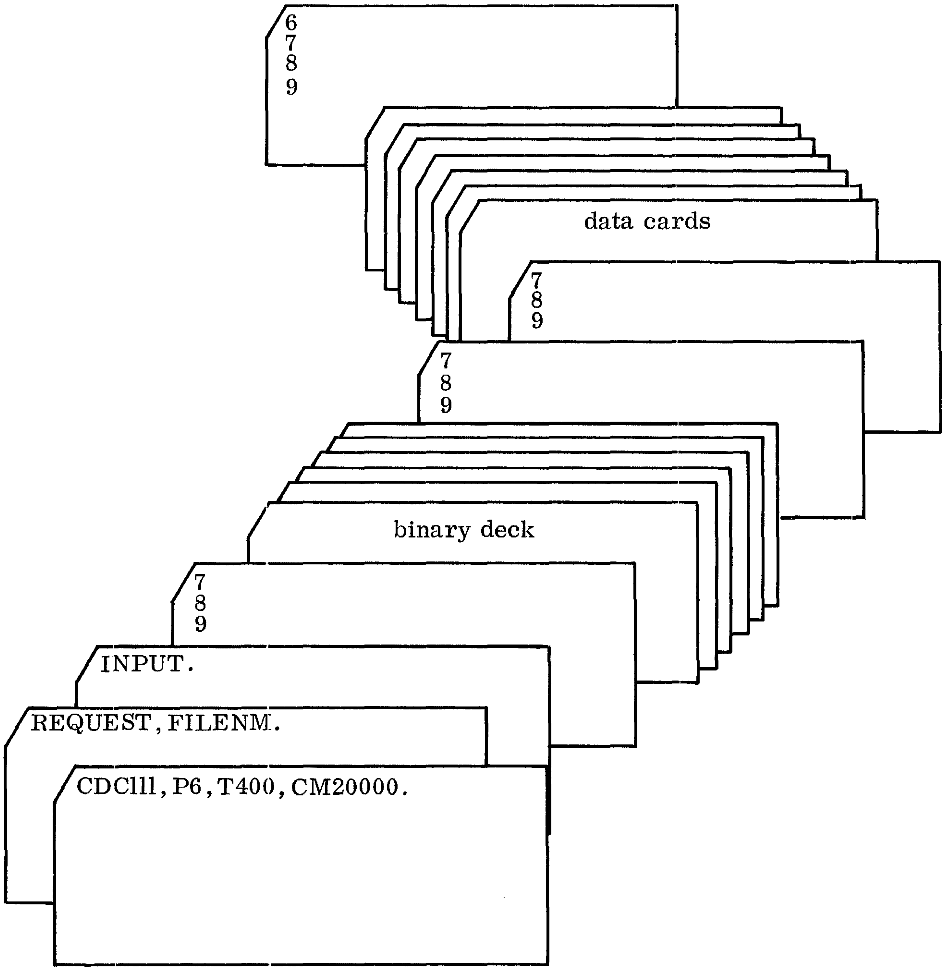
A binary deck to be loaded with a compiled routine must be preceded by 7,8,9 card.



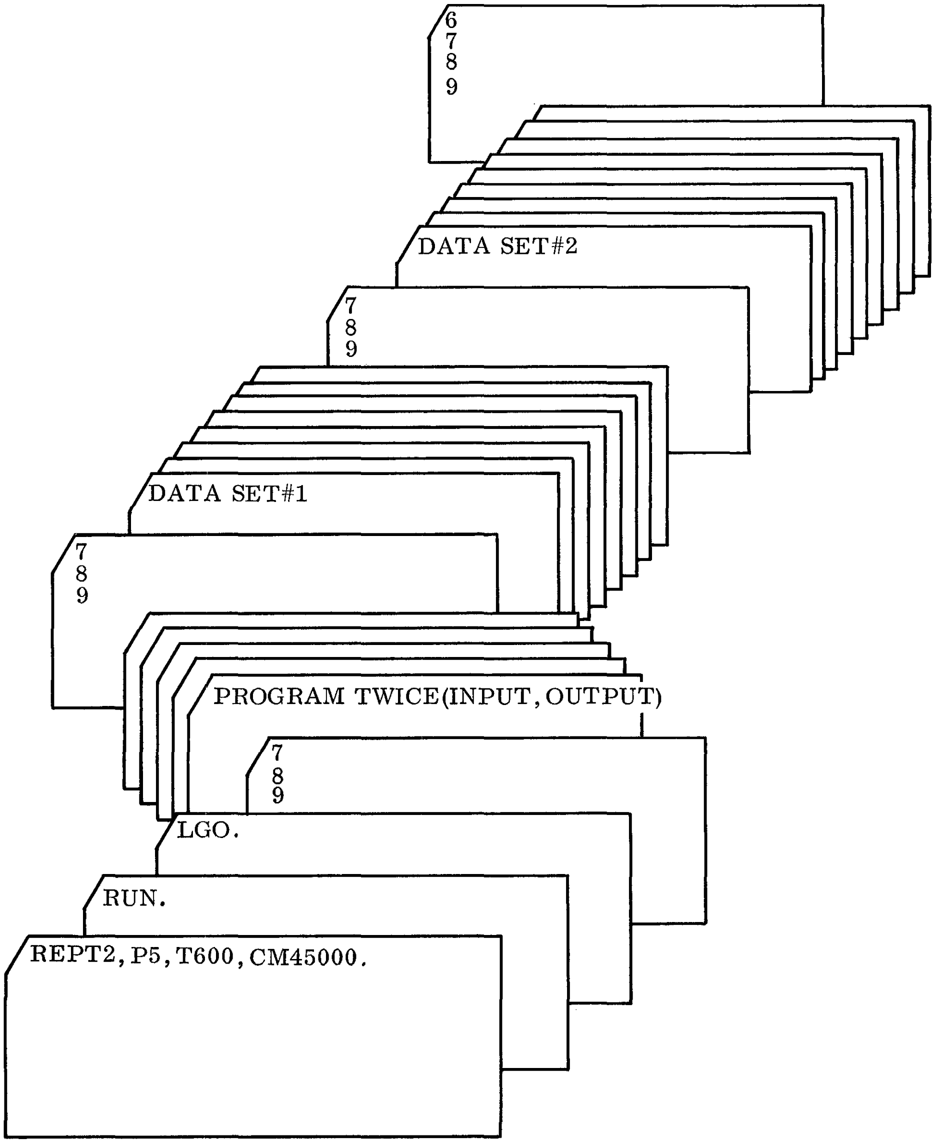
6
7
8
9
data
7
8
9
7
8
9
binary deck
7
8
9
source deck
PROGRAM HOW(INPUT, OUTPUT)
7
8
9
LGO.
LOAD, INPUT.
RUN(S)
EEK15, P5, T200, CM45000.
Completes loading from file LGO
Loads binary routines from INPUT

Load and Execute a Prepunched Binary Program

The binary cards in the input file following the record separator are loaded into central memory when the program call card INPUT is encountered.

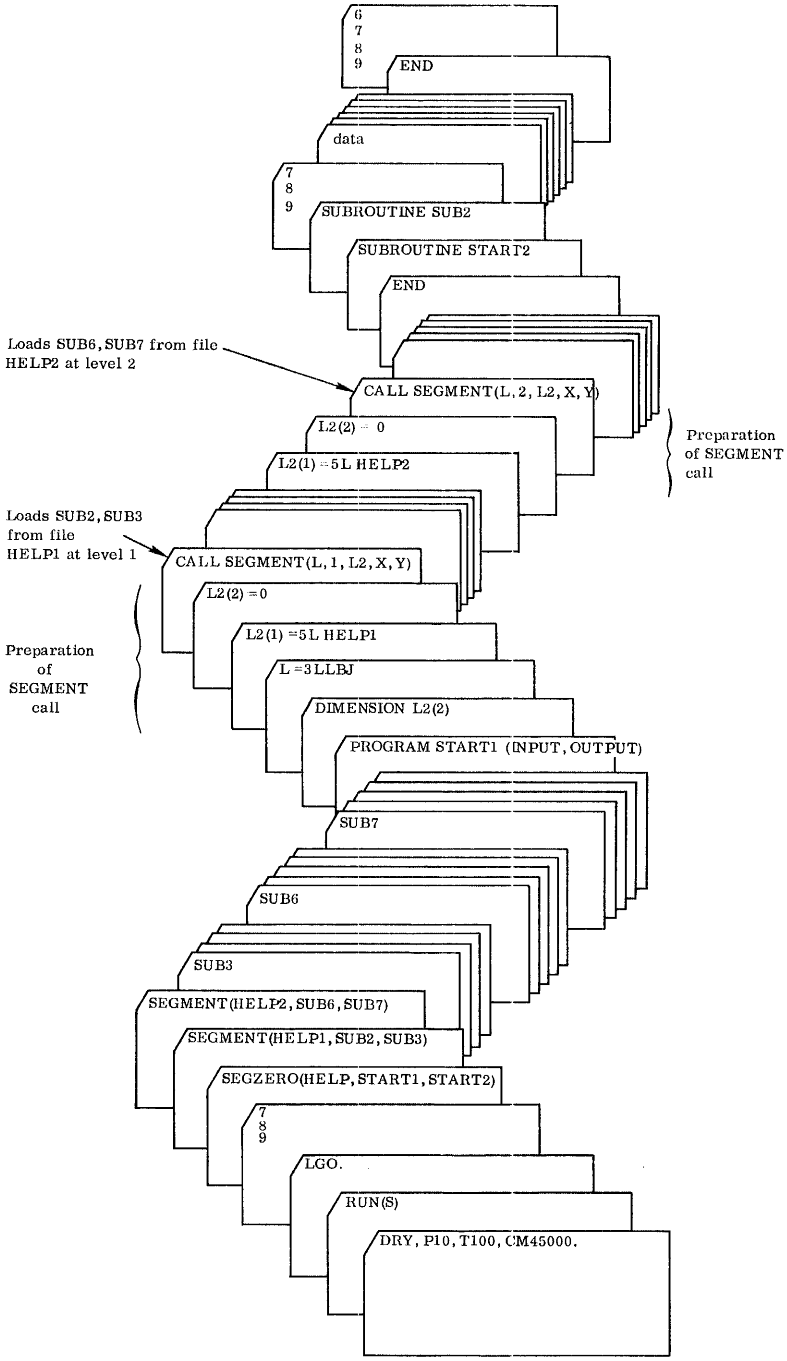


Compile Once and Execute Twice[†] with Different Data Decks



[†] Program TWICE must read the end of record card (7/8/9).

FORTRAN Load and Execute Segments



```
6
7
8
9
END
data
7
8
9
SUBROUTINE SUB2
SUBROUTINE START2
END
CALL SEGMENT(L,2,L2,X,Y)
L2(2) = 0
L2(1) =5L HELP2
CALL SEGMENT(L,1,L2,X,Y)
L2(2) =0
L2(1) =5L HELP1
L =3LLBJ
DIMENSION L2(2)
PROGRAM START1 (INPUT,OUTPUT)
SUB7
SUB6
SUB3
SEGMENT(HELP2,SUB6,SUB7)
SEGMENT(HELP1,SUB2,SUB3)
SEGZERO(HELP,START1,START2)
7
8
9
LGO.
RUN(S)
DRY,P10,T100,CM45000.
```

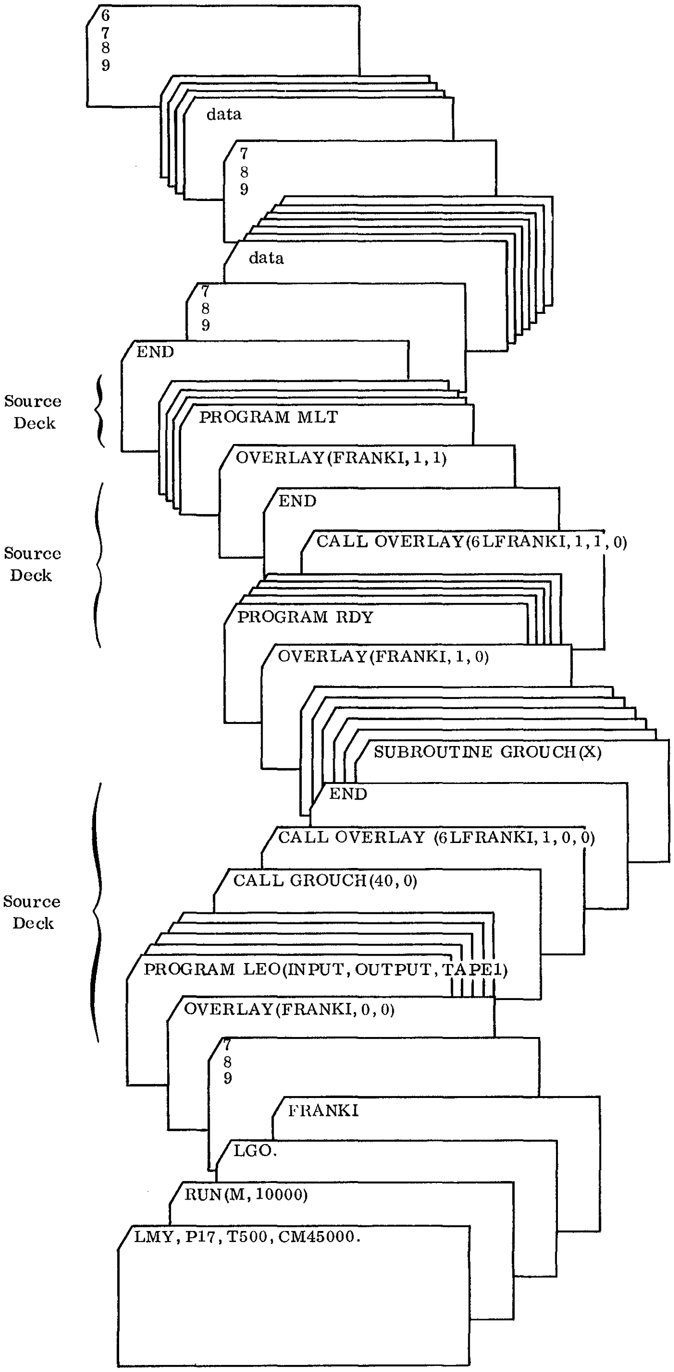
Loads SUB6,SUB7 from file HELP2 at level 2

Preparation of SEGMENT call

Loads SUB2,SUB3 from file HELP1 at level 1

Preparation of SEGMENT call

Overlay Preparation of 0,0;1,0 ;1,1



COMPILATION DIAGNOSTICS

During a FORTRAN compilation, 2- or 3-character error printouts follow statements which are incorrect; other printouts may follow the end statement indicating types of errors in the program. The short printouts produce a more descriptive full line diagnostic which is printed after the subprogram has been compiled.

The full line diagnostic contains a number to identify the statement in which the error was detected. In the short diagnostic an F as the third letter indicates a fatal error.

Fatal errors force a listing of the subprogram with diagnostics and inhibit the production of a relocatable record of the subprogram.

Non-fatal errors do not force a listing and will only appear if some type of listing has been specified or if fatal errors have been detected.

The two-character error indicators are defined below:

*****AC** Argument count too high.

Indicates that the number of arguments in this reference to a subroutine differs from the number which occurred in a prior reference.

*****AE** ARITHMETIC STATEMENT FUNCTION CALLS ITSELF

The arithmetic statement function being compiled references itself.

*****AF** ARITHMETIC STATEMENT FUNCTION ERROR

The arithmetic statement function has a statement number or appears after the first executable statement.

*****AL** SYNTAX ERROR IN ARGUMENT LIST

Indicates a format error in a list of arguments.

*****AS** SYNTAX ERROR IN ASSIGNMENT STATEMENT

Indicates a format error in an ASSIGN statement.

*****BX** SYNTAX ERROR IN BOOLEAN CONSTANT

Indicates a format error in the designation of a FORTRAN boolean constant in a B-type expression.

*****CB** LABELED COMMON BLOCKS EXCEED MAX OF 61

Attempt made to use more than 61 labeled common blocks.

*****CD** VARIABLE DUPLICATED IN A COMMON REGION

Indicates that a variable currently being assigned to the COMMON region has been previously assigned to this region.

*****CE** VARIABLES ASSIGNED TO COMMON ARE IMPROPERLY EQUIVALENCED

Indicates that two variables assigned to COMMON are improperly equivalenced.

*****CL** SYNTAX ERROR IN CALL STATEMENT

Indicates a format error in a call statement.

*****CM** SYNTAX ERROR IN COMMON STATEMENT

Indicates a format error on a COMMON statement.

*****CN** TOO MANY CONTINUATION CARDS

Indicates that more than 19 continuation cards appear in succession or that one such card appears in an illogical sequence.

*****CO** COMMON STORAGE EXCEEDED

Indicates that the amount of COMMON storage required by the main program or specified to the compiler is less than required by the current program or subroutine.

*****CT** CONTINUE STATEMENT IS MISSING A STATEMENT NUMBER

Indicates a CONTINUE statement with no statement number.

*****DA** DUPLICATE ARGUMENTS IN A FUNCTION DEFINITION STATEMENT

*****DB** ARRAY SIZE OUT OF RANGE

Indicates the requested array size exceeds 131K. A constant used as an array subscript cannot be contained in 17 bits.

*****DC** SYNTAX ERROR IN A DECIMAL CONSTANT

Indicates a format error in the expression of a FORTRAN decimal constant.

*****DD** VARIABLE BEING DIMENSIONED HAS BEEN PREVIOUSLY DIMENSIONED

Indicates a variable has appeared in more than one DIMENSION statement.

*****DF** DUPLICATE FUNCTION NAME

Indicates that the function name in the current function-definition statement has occurred as the name of a previously defined function.

*****DI** DO TERMINATOR PREVIOUSLY DEFINED

The terminator of this DO loop has already been defined.

*****DL** DECLARATIVE APPEARS AFTER FIRST EXECUTABLE STATEMENT

The declarative statement appears after the first executable statement.

*****DM** SYNTAX ERROR IN DIMENSION STATEMENT

Indicates an error in the format of a DIMENSION statement.

*****DN** ILLEGAL DO TERMINATOR

This statement cannot be used as a DO terminator. Indicates the attempt to use a FORMAT, GO TO, arithmetic IF, or another DO statement as the termination statement of a DO.

*****DO** SYNTAX ERROR IN A DO STATEMENT

Indicates an error in the format of a DO statement.

*****DP** MULTIPLY DEFINED STATEMENT NUMBER

Indicates the current statement has previously appeared in the statement number field.

*****DR** DATA RANGE ERROR

Attempt to store data out of range.

*****DS** UNDEFINED STATEMENT NUMBER IN A DO LOOP

Indicates that references have been made in DO statements to statement numbers which did not appear anywhere in the statement number field.

*****DT** SYNTAX ERROR IN DATA STATEMENT

Indicates an error in the format of a DATA statement.

*****DU** ATTEMPT HAS BEEN MADE TO PRESTORE VALUE INTO BLANK COMMON.

*****EC** CONTRADICTION IN EQUIVALENCE STATEMENT

Indicates that a variable currently appearing in an EQUIVALENCE statement cannot be equivalenced because of an inherent contradiction in the statement.

*****EF** END OF FILE CARD ENCOUNTERED, END CARD ASSUMED

Indicates that an end of file card is detected before the last END card is encountered.

*****EM** SYNTAX ERROR IN INDICATED EXPONENTIATION

Indicates the mode of the base or the exponent of an indicated exponentiation process is improper.

*****EQ** SYNTAX ERROR IN EQUIVALENCE STATEMENT

Indicates an error in the format of an EQUIVALENCE statement.

*****EX** SYNTAX ERROR IN EXPONENT

Indicates an error in the exponent portion of an indicated exponentiation process.

*****FA** FUNCTION HAS NO ARGUMENT

The FUNCTION has a void parameter list; at least one argument is required.

*****FL** SYNTAX ERROR IN EXTERNAL OR F-TYPE STATEMENT

Indicates an error in an EXTERNAL statement of F-TYPE statement.

*****FM** UNRECOGNIZABLE STATEMENT

Indicates a statement whose type cannot be determined.

*****FN** NO STATEMENT NUMBER ON FORMAT STATEMENT

Indicates that a FORMAT statement is missing a statement number.

*****FS** ERROR IN SPECIFICATION PORTION OF FORMAT STATEMENT

Indicates a format error in the specification portion of a FORMAT statement.

*****FT** SYNTAX ERROR IN FUNCTION TYPE STATEMENT

Indicates an error in a FUNCTION TYPE statement.

*****GF** FORMAT NUMBER REFERENCED BY CONTROL STATEMENT

A statement number attached to a preceding FORMAT statement is being referenced by a control statement, e.g., a GOTO statement.

*****GO** SYNTAX ERROR IN A GO TO STATEMENT

Indicates an error in the format of a GO TO statement.

*****IC** CHARACTER NOT IN FORTRAN CHARACTER SET

Attempt was made to use a character other than those listed in appendix A.

*****ID** IMPROPERLY NESTED DO LOOPS

The sequence of DO loops is improper.

*****IF** SYNTAX ERROR IN AN IF STATEMENT

Indicates an error in the format of an IF statement.

*****IL** SYNTAX ERROR IN AN INDEXED LIST OF I/O STATEMENT

Indicates a format error in an indexed list of the current input/output statement.

*****IN** ILLEGAL FUNCTION NAME

The name of a function reference starts with a number.

*****IO** ILLEGAL I/O DESIGNATOR

I/O designator has a variable name of more than six characters or a numeric value of more than two digits or is alphanumeric and begins with a number.

*****IT** ILLEGAL TRANSFER TO DO TERMINATOR

A transfer to a DO terminator is not allowed if the terminator has already been defined and no transfer to it appeared before it was defined.

*****LN** NAMELIST ERROR

*****LP** EXPONENTIATION TO A LOGICAL POWER

*****LS** SYNTAX ERROR IN INPUT/OUTPUT LIST

Indicates an error in the format of an input/output list.

*****MA** MISUSED SUBROUTINE ARGUMENT IN EQUIVALENCE STATEMENT

Indicates that an argument of the subroutine or function being compiled has been misused in an EQUIVALENCE statement.

*****MO** MEMORY OVERFLOW, FIELD LENGTH TOO SHORT

Indicates that the compiler field length, as specified on the JOB card, is too short.

*****MS* UNDEFINED STATEMENT NUMBER

Indicates that references have been made to statement labels which did not appear somewhere in the statement-label field of a line.

*****NC** SUBROUTINE OR FUNCTION NAME CONFLICTS WITH A PRIOR USAGE

*****NM** IMPROPER HEADER CARD

Indicates an error in the formatting of the name (header) card.

*****NO** NO OBJECT CODE GENERATED

Source program has generated no object code. This error will occur if a void file is input to the compiler.

*****NP** NO PATH TO THIS STATEMENT

The flagged statement cannot be executed at object time; program continues.

*****PM** FUNCTION PARAMETER MODE INCONSISTENCY

Indicates the parameters in a function reference do not agree in mode with the formal parameters of the statement function.

*****PN** UNBALANCED PARENTHESIS

Indicates an unequal number of open and closed parentheses in a statement.

*****PT** SYNTAX ERROR IN AN ENTRY STATEMENT

The entry statement being processed is labeled or has more than one name or is in a DO loop or name started with a number.

*****RN** SYNTAX ERROR IN A RETURN STATEMENT

Indicates an error in the format of a RETURN statement.

*****SB** ERROR IN AN ARRAY SUBSCRIPT

Indicates a format error in a subscript of an array reference currently being processed.

*****SE** SYNTAX ERROR IN SENSE STATEMENT

Indicates an error in the format of a sense statement.

*****SF** FIELD LENGTH OF ROUTINE BEING COMPILED EXCEEDS THE SPECIFIED FIELD LENGTH

*****SM** SYNTAX ERROR IN STATEMENT NUMBER

Indicates an error in the format of the statement label field.

*****SN** ILLEGAL CHARACTER IN STATEMENT NUMBER USAGE

Indicates an error in the format of the position where the statement label should appear.

*****SY** SYSTEM ERROR IN FORTRAN COMPILER

*****TM** SUBROUTINE HAS MORE THAN 60 ARGUMENTS

Indicates that a subroutine reference has more than 60 arguments or that the routine being compiled has more than 60 parameters.

*****TT** VARIABLE GIVEN CONFLICTING TYPES

A variable has appeared in more than one type statement.

*****TN** PROGRAM HAS MORE THAN 50 ARGUMENTS

*****TY** SYNTAX ERROR IN A TYPE STATEMENT

Indicates an error in the format of a TYPE statement.

*****UA** REFERENCE MADE TO AN AS YET UNDIMENSIONED ARRAY

Indicates reference was made to an array which has not previously appeared in a DIMENSION statement.

*****UE** LOGICAL UNIT NUMBER IS NOT AN INTEGER

*****VC** VARIABLE NAME CONFLICTS WITH A PRIOR USAGE

Indicates that a variable name appears which conflicts with some prior use.

*****VD** ARRAY WHOSE DIMENSIONS ARE ARGUMENTS TO THE SUBROUTINE OR FUNCTION HAS BEEN MISUSED

Indicates improper use of an array with variable dimensions.

*****XF** SYNTAX ERROR IN THE EXPRESSIONS BEING PROCESSED

Indicates an error in the format of the expression currently being processed.

*****ZY** SYSTEM ERROR-UNKNOWN TWO LETTER CODE

PROGRAM - SUBPROGRAM FORMAT H

The starting address of all programs is RA+100_8 with the first 77_8 locations containing file and loader information. Only 50 files may be declared for any one program and the file names along with their associated buffer addresses begin at RA+2. An object time routine, Q8NTRY, transfers the file information to RA+2+n, where n is the number of declared files, at execution time. The I/O buffers are reserved as a portion of the main program and Q8NTRY also initializes the buffer parameters during execution.

The first word of a main program contains the name of the program in left justified display code and a parameter count greater than 77_8 in the right most position. Since no more than 74_8 parameters may be passed to a subprogram, a count of greater than 77_8 terminates trace back information. The second word of a main program is the entry point. It contains instructions to preset the parameters for Q8NTRY which performs initiation only once per execution. Therefore, entry into an overlay is through this word destroying its contents. Since Q8NTRY does not perform any function after the first entry, the destruction of the preset parameters for an overlay entry does not matter.

The addresses of the first six parameters to a subprogram are passed by B registers 1-6. One word is reserved for each parameter greater than six so that the address of the parameter is actually passed through this reserved word. Immediately following these reserved words is a location containing the name of the subprogram in left justified display code and a parameter count in the lower six bits. Next is the entry/exit line for the subprogram. Therefore, a subprogram will have as few as two reserved words if the parameter count is six or less. Otherwise, there will be a reserved word for each parameter over six plus the name and entry words.

Subroutines written in the COMPASS assembly language that will operate in conjunction with FORTRAN coded routines should be formatted as in the following examples to take full advantage of the error tracing facility of FORTRAN Version 2.0.

Examples:

```
PROGRAM PETE (INPUT, OUTPUT, TAPE 1)
    DATA  0          L00002    Name & argument count plus $100_8$
    SB1   L00002     L00001    Entry/Exit line
    SB2   C00001
    RJ    Q8NTRY     L00003
```

```
SUBROUTINE PHD     (A, B, C)

    DATA  0                    L00002      Name & argument count
    DATA  0                    L00001      Entry/Exit line

SUBROUTINE PEN   (A, B, C, D, E, F, G, H, I, J)

    DATA  0                    L00011      Reserved word for G
    DATA  0                    L00012                        H
    DATA  0                    L00013                        I
    DATA  0                    L00014                        J
    DATA  0                    L00002      Name & argument count
    DATA  0                    L00001      Entry/Exit line
```

<u>Calling Sequence to PEN</u>

```
CALL PEN   (M, N, O, P, Q, R, S, T, U, V)

    SB1    M
    SB2    N
    SB3    O
    SB4    P
    SB5    Q
    SB6    R
    SX6    Entry line of PEN
    SA1    X6-1     Name & argument count
    SB7    X1-6     Number of arguments less 6
    SX6    S
    SA6    A1-B7    Reserved word for S
    SX7    T
    SA7    A6+1     Reserved word for T
    SX6    U
    SA6    A7+1     Reserved word for U
    SX7    V
    SA7    A6+1     Reserved word for V
    RJ     PEN
```

0712L00002   Where 12_8 is argument count and L00002 is word containing the name of calling routine.

FORTRAN I/O ROUTINES I

A new set of FORTRAN I/O routines is supplied with Version 2.1. The re-written routines are BACKSP, BUFFEI, BUFFEO, ENDFIL, IFENDF, INPUTB, INPUTC, IOCHEK, OUTPTB, OUTPTC, and REWINM. In addition, there is a new routine, SION. SION is used for all communication between the FORTRAN I/O routines and the PP routine CIO. Also, SION includes routines to backspace physical records and read physical records. The characteristics of these routines are as follows:

Structure of I/O Files

Logical record	is composed of physical records which terminate in a short or zero length physical record.
Physical record	is composed of a pre-determined maximum number of characters. On magnetic tape the physical record separator is the record mark, and on disk a physical record is defined to be one sector.
Unit record	is analogous to a card image or print line.

Coded physical records on disk are composed of 640_{10} characters (maximum); on magnetic tape, 1280_{10} characters (maximum). Binary physical records are composed of 5120_{10} characters (maximum) on tape, 640_{10} characters (maximum) on disk.

Records created by FORTRAN coded writes are unit records. Unit records are packed into physical records.

Records created by FORTRAN binary writes (including BUFFER OUT) are logical records.

Records read using FORTRAN coded reads are unit records.

Records read using FORTRAN binary reads (including BUFFER IN) are logical records.

BUFFER IN coded and BUFFER OUT coded read and create a logical record.

Multi-file files are permitted on all files except INPUT and OUTPUT.

BUFFER I/O

BUFFEI

Only one logical record is read each time BUFFEI is called. If the block length specified by the call is longer than the logical record, the excess block locations are not changed by the read. If the logical record is longer than the block, the 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 statement on unit i, and prior to a subsequent reference to unit i or to the information being read in, the status of the BUFFER operation must be checked by an IF UNIT statement. This insures that the data has actually been transferred, and the buffer parameters have been properly restored.

BUFFEO

One logical record is written each time the routine is called. The length of the record is LWA-FWA+1.

The IF UNIT statement must be used similarly with BUFFEO as with BUFFEI. Since BUFFEO 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 IF UNIT statement.

Random Access Files (Mass Storage)

Two degrees of sophistication are available using the mass storage subroutines. It is possible to utilize the routines in a normal fashion having just one master index, or it is possible to have a master index and many sub-indexes. A file may have a name or a number index.

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 CM area specified in the call (the ix parameter).

The STINDX subroutine causes no transfer of data; it merely changes the file index in the FET to the base specified in the call. Only the master indexes are managed by the system. The user must keep track of all sub-indexes. A user may open a master index, for example, I, do a call READMS onto B, assuming B to be a sub-index, then call STINDX to assure that subsequent operations on the logical unit will use B as the index. If the master index is to be changed to some sub-index, a call to STINDX should be made prior to CALL WRITMS. Master indexes should be restored prior to job conclusion. After making a call to STINDX, if the next operation on that file is to be a random access write (WRITMS) and if the file is being referenced through a name index, the programmer must zero out the area reserved for the new index buffer (whose first word address is specified by the ix parameter in the call to STINDX) prior to calling WRITMS. The master index must be reset before termination of the job so that the correct index will be written on the file.

Upon termination of the job, the mass storage file is closed automatically by FORTRAN. At this time the index as specified in the FET is dumped to the file.

Examples:

1.
```
I     PROGRAM MS (TAPE5)
      DIMENSION I(10), B(20), C(30)
C READ MASTER INDEX INTO I
      CALL OPENMS(5,I,10,0)
          .
          .
          .
C READ RECORD 4 INTO B (ASSUME THIS RECORD IS A SUB-INDEX)
      CALL READMS(5,B,20,4)
      CALL STINDX(5,B,20)
C ALL SUBSEQUENT OPERATIONS ON UNIT 5 WILL USE
C B AS THE INDEX FOR THE FILE
          .
          .
          .
C RESTORE MASTER INDEX
      CALL STINDX(5,I,10)
      END
```

2.
```
      PROGRAM MS (TAPE5)
C PROGRAM FOR CREATING RANDOM FILE
      DIMENSION J(10),B(7),XYZ(20),ZXY(10),YXZ(50)
      DATA SUB 1/4L SUB1/
      DATA JOE,SAM,PETE/3H JOE,3L SAM,4H PETE/
      CALL OPENMS(5,J,10,1)
C USE INDEX B
      CALL STINDX(5,B,7)
      CALL WRITMS(5,XYZ,20,JOE)
      CALL WRITMS(5,ZXY,10,SAM)
      CALL WRITMS(5,YXZ,50,PETE)
      CALL STINDX(5,J,10)
C WRITE OUT THE SUB-INDEX
      CALL WRITMS(5,B,7,SUB1)
      END
```

3.
```
      PROGRAM MS (TAPE5)
C THIS MS FILE HAS NO SUB-INDEXES
      DIMENSION I(10)
C READ MASTER INDEX INTO I
      CALL OPENMS(5,I,10,0)
          .
          .
          .
C ANY READ OR WRITE ON THIS FILE WILL USE THE INDEX IN
C ARRAY I
          .
          .
          .
      END
```

Execution of the END routine will close the file, causing the index at I to be rewritten on the file.

Status Checking

The IF UNIT statement checks the status of a buffered operation on logical unit i. The status returned is unit not ready, unit ready, no previous error, or previous read encountered an end of file.

Example:

```
IF (UNIT,5) 10, 20, 30
```

Control would transfer to statement 10, 20, or 30 if the unit was still busy, if the unit was not busy and there were no previous errors, or if an end of file was read on the previous read, respectively. Since the optional fourth branch is ineffective, it is omitted.

When the IF UNIT statement references a non-buffered unit, the second branch is always taken.

The IF EOF statement tests for an end of file read (non-buffered) on unit i.

Example:

```
IF (EOF,i) 10, 20
```

Control would transfer to statement 10 if the preceding READ statement had encountered an EOF, or to 20 otherwise. If an EOF had been read, the indicator would be cleared before proceeding.

Backspace

When a BACKSPACE is requested on a coded file (except file created by the BUFFER OUT statement) the file is logically moved back one unit record. The backspace will be attempted within the I/O buffer; if not possible the external I/O device will be repositioned.

Backspace on binary files and files created by the use of the 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 and backspaced over; and then the requested backspace is processed.

Labeled files (tapes)

Only files recorded on 1/2" magnetic tape may be labeled. FORTRAN will accept labeled tapes, but the FORTRAN program cannot access the label.

A labeled tape written by a FORTRAN program will be given a default label by the system. A standard label cannot be written on a labeled tape from a FORTRAN program.

A labeled tape prepared elsewhere with a standard label can be read by a FORTRAN program only if special instructions are given to the operator. The information in the label will not compare with the information in the FORTRAN prepared FET and SCOPE will not allow the job to proceed unless overridden from the console.

Undefined Operation

Meaningful results are not guaranteed in the following circumstances:

- Mixed mode files
- Mixing buffer I/O statements and standard READ/WRITE statements referencing the same logical file.
- Two consecutive buffer I/O statements referencing the same logical file without an intervening IF UNIT statement.
- Requesting a LENGTH function on a buffered unit before checking status on the unit with an IF UNIT statement.

Summary of Differences from Previous Versions

Although this version includes a complete re-write of the FORTRAN I/O routines, the external characteristics remain essentially the same. Disk I/O is the same as in version 2.0 and, for the most part, tape I/O is now the same as Disk I/O. The most important change is the BCD record structure on tape. Previously,unit records existed on tape as separate physical records and a logical record was the entire file; now one or more unit records are packed into logical records as on the disk and a file may consist of one or more logical records.

Note that BUFFER IN/OUT read and write logical records regardless of device type. READ's and WRITE's read and write unit records. While writing, no end-of-logical record is written unless a backward operation occurs, causing the emptying of the buffer. Thus, if no backward operation occurs on a write file, it will consist of a single logical record. Encountering an end-of-logical-record while reading from a file other than INPUT is without special significance. The EOR is passed over and the next unit record is read.

Increasing the buffer size will speed up I/O operations until the size becomes as large as the largest logical record on the file. Beyond that point, no advantage is gained. Under no circumstances will more than one logical record be read or written at any one time.

SYSTEM ROUTINE J

The SYSTEM routine handles the following extensions for the mathematical routines of FORTRAN Version 2.1: error tracing, diagnostic printing, termination of output buffers and transfer to specified non-standard error procedures. The END processor also uses SYSTEM to dump the output buffers and print an error summary. Since SYSTEM, along with the initialization routine, Q8NTRY, and the end processors, END, STOP, EXIT, must always be available, these routines are combined into one with multiple entry points. Any of the parameters used by SYSTEM relating to a specific error may be changed by a user routine during execution by calling SYSTEMC.

CALLING SYSTEM

The calling sequence to SYSTEM from an assembly language routine passes the error number X1 and an error message address in X2. Therefore, one error number may have several different messages associated with it. The error summary at the end of the program lists the total number of times each error number was encountered.

FORTRAN routines call SYSTEM via a RJ to SYSTEMP, a special entry point. Because the addresses of the subprogram arguments must be passed to a non-standard recovery routine if one is specified, SYSTEMP must be called with eight parameters. The first six parameters are the first six formal parameters of the subprogram. If the subprogram does not have six parameters, dummy parameters must be supplied. The seventh parameter to SYSTEMP is the error number specified as an integer constant or integer variable. The array or simple variable containing the diagnostic message is the eighth parameter. After adjusting the parameters, SYSTEMP transfers to SYSTEM for error processing.

ERROR PROCESSING

If an error number of zero is accepted, this is a special call to end the output buffers and return. If no OUTPUT file is defined before SYSTEM is called, there is no error printing and an appropriate message appears in the Dayfile. Each line printed is subjected to the line limit of the OUTPUT buffer. When limit is exceeded, the job is terminated. The error table is ordered serially; the first error corresponds to the error number 1, and is expandable at assembly time. The last entry in the table is a catch-all for any error number that exceeds the table length. Following is an entry in the error table.

Error Table

	PRINT		ERROR			
PRINT	FREQUENCY	PRINT	DETECTION	F/	A/	NON-STANDARD
FREQUENCY	INCREMENT	LIMIT	TOTAL	NF	NA	RECOVERY ADDRESS
8	8	12	12	1	1	18

Use of PRINT FREQUENCY

PRINT FREQUENCY = PF

PRINT FREQUENCY INCREMENT = PFI

1. If PF = 0 and PFI = 0, diagnostic and trace back information are never listed.
2. If PF = 0 and PFI = 1, diagnostic and trace back information are always listed until the print limit is reached.
3. If PF = 0 and PFI = n, diagnostic and trace back information are listed only the first n times unless the print limit is reached first.
4. If PF = n, diagnostic and trace back information are listed every nth time until the print limit is reached.

Use of FATAL (F)/ non-FATAL (NF)

1. If the error is non-fatal and no non-standard recovery address is specified, the error messages are printed according to PRINT FREQUENCY and control is returned to the calling routine.
2. If the error is fatal and no non-standard recovery address is specified, the error messages are printed according to PRINT FREQUENCY, an error summary is listed, all the output buffers are terminated, and the job is terminated.

TRACEBACK EXAMPLE

```
    DATA  0    L00002      Name and number of parameters
    DATA  0    L00001      Entry/exit line
      .
      .
      .
+   RJ         SYSTEM
-   07         L00002      07 is number of parameters passed to SYSTEM and L00002 is
                           address of word containing name of calling routine
```

The name of the routine always precedes the entry/exit line.

Use of NON-STANDARD RECOVERY

SYSTEM will supply the non-standard recovery routine with the following information:

B1-B6	address of the first six parameters passed to the routine that detected the error
X1	error number passed to SYSTEM
X2	address of the diagnostic message available to SYSTEM
X3	address within an auxiliary table if A/NA bit is set
X4	instruction word consisting of the return jump to SYSTEM in the upper 30 bits and trace back information in the lower 30 bits for the routine which detected the error
A0	address of error number entry within SYSTEM's error table.

1. Non-fatal error

The entry/exit line of the routine which called SYSTEM is set into the entry/exit line of the recovery routine. Control is then passed to the word immediately following the entry/exit line of the recovery routine. The traceback information available to SYSTEM from the routine which detected the error is passed to the recovery routine in X4.

Any faulty parameters may be corrected, and the recovery routine is allowed to call the routine which detected the error with corrected parameters. Upon exit from the recovery routine, control is turned not to SYSTEM nor to the routine which detected the error, but rather back another level (see example). By not correcting the faulty parameters in the recovery routine, a three routine loop could develop between the routine which detects the error, SYSTEM, and the recovery routine. No checking is done for this case.

Example:

```
            MAIN
    E/E     □

            .
            .
            .
            CALL MATH (A, B, C)
    RTN1    .                       Point of return from MATH, if no errors detected,
            .                       or from RECOVERY.
            .
            END
```

```
        MATH
E/E     jump to RTN1          May be reentered from RECOVERY with corrected
                              parameters
        .
        .
        .
        RJ SYSTEM
        07XXAAAAAA.........trace-back information

RTN2    .
        .
        .
        END

        SYSTEM
        jump to RTN2          transfers E/E line of MATH to E/E
        .                     line of RECOVERY and gives control
        .                     to RECOVERY
        JUMP TO RECOVERY
        .
        .
        END

        RECOVERY
E/E     jump to RTN1          corrects faulty parameters and may
                              recall MATH
        .
        .
        RJ MATH
        .
        .
        jump to E/E           returns to MAIN following reference
        END                   to MATH
```

2. Fatal error:

Into the entry/exit line of the recovery routine is set a return address back to SYSTEM. Control is then passed to the word immediately following the entry/exit line of the recovery routine. The traceback information available to SYSTEM from the routine which detects the error is set in X4. If control is returned to SYSTEM from the recovery routine, then an error summary is listed, all output buffers are terminated and the job is aborted.

```
        SYSTEM
E/E     □

        .
        .
        .
TAG     jump to RECOVERY

        07XXAAAAAA            trace back information
RTN3    .
        .
        .
        END
        RECOVERY
E/E     jump to RTN3
        .
        .
        .
        jump to E/E
        END
```

Jse of the A/NA Bit

The A/NA bit is for use only when a non-standard recovery address is specified. If this bit is set, the address within an auxiliary table is passed in X3 to the recovery routine. This bit allows more information than is normally supplied by SYSTEM to be passed to the recovery routine. Only during assembly of SYSTEM may this bit be set, because 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 variables is detected:

Calling routine is a program (the number of arguments > 77B).

Maximum trace back limit is reached.

No trace back information is supplied; a 07 instruction does not follow the return jump as is the case with I/O operations.

To change an error table during execution, a FORTRAN type call is made to SYSTEMC with the addresses of the following parameters in B1 and B2:

B1 error number

B2 parameter list in consecutive locations containing:

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 encountered
word 4	print limit
word 5	non-standard recovery address
word 6	maximum trace back limit

If any word within the parameter list is negative, the value already in table entry will not be altered.

(Since the 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
CALLED FROM cccccc at wwwwww
.
.
.
```

zzzzzzz and ccccccc are routine names, yyyyyy and wwwwww are absolute addresses and error number is decimal

```
ERROR SUMMARY

     ERROR TIMES

     xxxx     yyyy
.    xxxx     yyyy
:    xxxx     yyyy
```

all numbers are decimal

```
NO OUTPUT FILE FOUND

OUTPUT FILE LINE LIMIT EXCEEDED
```

Functions of entry points:

Q8NTRY — initialize I/O buffer parameters

STOP — enter STOP in the Dayfile and begin END processing

EXIT — enter EXIT in the Dayfile and begin END processing

END — terminate all output buffers, print an error summary; transfer control to main overlay if within an overlay or 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, aborts the job, or returns to calling routine depending on type of error

SYSTEMP adjusts arguments for use by SYSTEM and transfers control to SYSTEM

SYSTEMC changes entry in SYSTEM's error table according to arguments passed.

ABNORML gains control from an execution routine when an error had been assembled as fatal and during the processing of the job was changed to non-fatal with no non-standard error recovery. An abnormal termination is given.

FILE NAME HANDLING BY SYSTEM

SYSTEM(Q8NTRY) places in RA+2 and the locations immediately following, the file names from the FORTRAN PROGRAM card. The file name is left justified and the file's FET address is right justified in the word. (Thus the declared file names replace any actual file names at execution time in the RA area.)

The logical file name (LFN) which appears in the first word of the FET is determined in one of the three following ways:

CASE 1: If no actual parameters are specified, the LFN will be the file name from the PROGRAM card.

```
Example:    .
            .
            .
            RUN(S)
            LGO.
            .
            .
            .
            PROGRAM TEST1(INPUT,OUTPUT,TAPE1,TAPE2)

Before      SYSTEM(Q8NTRY)
RA+2        000----------000
            000----------000

After                                           LFN in FET
RA+2        INPUT--------FET address            INPUT
            OUTPUT-------FET address            OUTPUT
            TAPE1--------FET address            TAPE1
            TAPE2--------FET address            TAPE2
```

CASE 2: If actual parameters are specified, the LFN will be that specified by the corresponding actual parameter, or the file name from the PROGRAM card if no actual parameter was specified. A one-to-one correspondence exists between the actual parameters and the file names found on the PROGRAM card.

Example:

```
.
.
.
RUN(S)
LGO(,,DATA,ANSW)
.
.
.
PROGRAM TEST2(INPUT,OUTPUT,TAPE1,TAPE2,TAPE3=TAPE1)
```

Before
RA+2

```
000 ———————— 000
000 ———————— 000
DATA ——————— 000
ANSW ——————— 000
```

After
RA+2

```
                                            LFN in FET
INPUT ———————— FET address                  INPUT
OUTPUT ——————— FET address                  OUTPUT
TAPE1 ———————— FET address                  DATA
TAPE2 ———————— FET address                  ANSW
TAPE3 ———————— FET address of TAPE1         Uses TAPE1 FET
```

CASE 3: An equivalenced file name from the PROGRAM card will ignore an actual parameter. The LFN will be that of the file to the right of the equivalence and no new FET will be created.

Example:

```
.
.
.
RUN(S)
LGO(,,DATA,ANSW)
.
.
.
PROGRAM TEST3(INPUT,OUTPUT,TAPE1=OUTPUT,TAPE2,TAPE3)
```

Before
RA+2

```
000 ———————— 000
000 ———————— 000
DATA ——————— 000
ANSW ——————— 000
```

After
RA+2

```
                                            LFN in FET
INPUT ———————— FET address                  INPUT
OUTPUT ——————— FET address                  OUTPUT
TAPE1 ———————— FET address of OUTPUT        uses OUTPUT FET
TAPE2 ———————— FET address                  ANSW
TAPE3 ———————— FET address                  TAPE3
```

EXECUTION DIAGNOSTICS K

The format of these error listings is shown on page J-6.

The symbol INF denotes infinite and IND denotes indefinite internal words.

Some error conditions are preceded by "also". The routine in question calls on a subordinate library routine, giving it the arguments indicated; therefore the subordinate routine may detect some errors of its own and report them under its own error number.

Routine	Condition	Standard Recovery	Error Number
ACGOER	Called only upon detection of a computed or assigned GO TO error	Fatal	1
ACOS (R)	R = INF or R = IND or abs (R) .GT. 1.0	+IND +IND	2

ALOG (R)	R = INF or R = IND or R .LT. 0
	R = 0
		+IND
		-INF
			3

ALOG10 (R)	R = INF or R = IND or R .LT. 0
	R = 0
		+IND
		-INF
			4

ASIN (R)	R = INF or R = IND or abs (R) .GT. 1.0	+IND	5
ATAN (R)	R = INF or R = IND	+IND	6
ATAN2 (R1, R2)	(R1 or R2) = (INF or IND) R1 = R2 = 0	+IND +IND	7
CABS (Z)	(real (Z) or imag (Z) = (INF or IND)	+IND	8
CBAIEX:Z**I	(real (Z) or imag (Z)) = (INF or IND) Z = (0,0) and I .LE. 0	(+IND, +IND) (+IND, +IND)	9
CCOS (Z)	(real (Z) or imag (Z)) = (INF or IND) also: COS (real (Z)) and EXP (imag (Z)) and imag (Z) .LT. -675.82	(+IND, +IND)	10
CEXP (Z)	(real (Z) or imag (Z)) = (INF or IND) also: SIN(inag (Z)) and EXP (real (Z))	(+IND, +IND)	11
CLOG (Z)	(real (Z) or imag (Z)) = (INF or IND) also: ALOG (CABS(Z)) and ATAN2 (imag (Z), real (Z))	(+IND, +IND)	12
COS (R)	R = INF or R = IND or abs (R) .GT. 2.2E14	+IND	13

Routine	Condition	Standard Recovery	Error Number
CSIN (Z)	(real (Z) or imag (Z)) = (INF or IND)	(+IND, +IND)	14
	also: SIN(real(Z)) and EXP (imag (Z))		
	and imag (Z) . LT. -675.82		
CSQRT (Z)	(real (Z) or imag(Z)) = (INF or IND)	(+IND, +IND)	15
DABS (D)	D = INF	+INF	16
	D = IND	+IND	
DATAN (D)	D = INF or D = IND	+IND	17
DATAN2 (D1,	(D1 or D2) = (INF or IND)	+IND	18
D2)	D1 = D2 = 0	+IND	
DBADEX:	(D1 or D2) = (INF or IND)	+IND	19
D1**D2	D1 = 0 and D2 . LE. 0	+IND	
	D1 .LT. 0	+IND	
DBAIEX:	D1 = INF or D1 = IND	+IND	20
D1**I2	D1 = 0 and I2 . LE. 0	+IND	
INPUT B	Illegal file input	Proceed	20
DBAREX:	(D1 or R2) = (INF or IND)	+IND	21
D1**R2	D1 = 0 and R2 . LE. 0	+IND	
	D1 .LT. 0	+IND	
DCOS (D)	D = INF or D = IND or abs (D) . GT. 2. 2E14	+IND	22
DEXP (D)	D = INF or D = IND	+IND	23
	D .GT. 741.67	+INF	
DLOG (D)	D = INF or D = IND or D . LT. 0	+IND	24
	D = 0	-INF	
DLOG10 (D)	D = INF or D = IND or D . LT. 0	+IND	25
	D = 0	-INF	
DMOD (D1,D2)	(D1 or D2) = (INF or IND)	+IND	26
	D2 = 0	+IND	
	D1 / D2 .GE. 2 ** 96	+IND	
DSIGN (D1,D2)	D1 = IND or D2 = (0 or INF or IND)	+IND	27
	D1 = INF	INF with sign of D2	
DSIN (D)	D = INF or D = IND or abs (D) .GT.2.2E14	+IND	28
DSQRT (D)	D = INF or D = IND or D . LT. 0	+IND	29
EXP (R)	R = INF or R = IND	+IND	30
	R .GT. 741.67	+INF	
IBAIEX:	I1 = 0 and I2 .LE. 0	0	31
I1**I2	I1 ** I2 .GE. 2** 48	0	
IDINT (D)	D = +INF or D = IND or D .GE. 2**59	2**59-1	32
	D = -INF or D . LE. -2**59	1-2**59	

Routine	Condition	Standard Recovery	Error Number
RBADEX:	(R1 or D2) = (INF or IND)	+IND	33
R1**D2	R1 = 0 and D2 .LE. 0	+IND	
	R1 .LT. 0	+IND	
RBAIEX:	R1 = INF or R1 = IND	+IND	34
R1**I2	R1 = 0 and I2 .LE. 0	+IND	
	R1**I2 = INF	+INF	
RBAREX:	(R1 or R2) = (INF or IND)	+IND	35
R1**R2	R1 = 0 and R2 .LE. 0	+IND	
	R1 .LT. 0	+IND	
SIN (R)	R = INF or R = IND or abs (R) .GT.2.2E14	+IND	36
SLITE (I)	I .GT. 6 or I .LT. 0	Proceed	37
SLITET (I1,I2)	I1 .GT. 6 or I1 .LE. 0	I2 = 2	38
SQRT (R)	R = INF or R = IND or R .LT. 0	+IND	39
SSWTCH (I1,I2)	I1 .GT. 6 or I1 .LE. 0	I2 = 2	40
TAN (R)	R = INF or R = IND or abs (R) .GT.8.4E14	+IND	41
TANH (R)	R = INF or R = IND	+IND	42
INPUTN	Precision lost in floating integer constant	Proceed	49
	Namelist data terminated by EOF not $	Proceed	
	Too few constants for unsubscripted array	Proceed	
OVERLAY	Fatal error reported by LOADER	Fatal	50
SEGMENT	Fatal error reported by LOADER	Fatal	51
	Non-fatal error reported by LOADER	Proceed	52
BACKSP	Unassigned medium†	Fatal	53
BUFFEI	Unassigned medium†	Fatal	54
	Attempt to read past EOF on Buffer In.	Fatal	55
	Last operation was a write, no data available to read.	Fatal	56
	Starting address greater than terminal address.	Fatal	57
BUFFEO	Unassigned medium†	Fatal	58
	Starting address greater than terminal address.	Fatal	59
ENDFIL	Unassigned medium†	Fatal	60
IFENDF	Unassigned medium †	Fatal	61

†Execution time diagnostic occurs when a variable file name is undefined. It is printed as Unassigned medium, file xxxxxxx (where xxxxxxx is the name of the undefined file).

Routine	Condition	Standard Recovery	Error Number
FTNBIN	Unassigned medium [†]	Fatal	62
INPUTB	Unassigned medium[†]	Fatal	62
	Attempt to read past EOF - binary input.	Fatal	63
OUTPTN	Unassigned medium variable file name is undefined.	Fatal	64
INPUTC	Unassigned medium,[†] variable file name is undefined.	Fatal	64
	Attempt to read past EOF - coded input	Fatal	65
INPUTN	Namelist name not found.	Fatal	66
	No I/O medium assigned.	Fatal	
	Wrong type constant.	Fatal	
	Incorrect subscript.	Fatal	
	Too many constants.	Fatal	
	(,$,OR = expected, missing.	Fatal	
	Variable name not found.	Fatal	
	Bad numeric constant.	Fatal	
	Missing constant after *.	Fatal	
	Uncleared EOF on read.	Fatal	
	Attempted read after write.	Fatal	
INPUTS	Attempt to transfer more than 150 characters/record on DECODE processing.	Fatal	66
	DECODE * character per record count less than or equal to zero.	Fatal	
IOCHEK	Unassigned medium[†] for IF UNIT statement.	Fatal	67
KODER (Coded output)	Illegal letter as format specification	Fatal	68
	Format specification has more than 2 levels of parentheses (3 levels under ASA).	Fatal	69
	Exceeded record size (format specified more than 136 characters per line).	Fatal	70
	Field width specified as zero.	Fatal	71
	Field width specified is less than or equal to the specified decimal width.	Fatal	72
	Attempt to output data under Hollerith format.	Fatal	73

[†] Execution time diagnostic occurs when a variable file name is undefined. It is printed as Unassigned medium, file xxxxxxx (where xxxxxxx is the name of the undefined file).

Routine	Condition[††]	Standard Recovery	Error Number
KRAKER (Coded input)	Illegal letter used as format specification.	Fatal	74
	Format specification with more than 2 levels of parentheses.	Fatal	75
	Field width specified as zero.	Fatal	76
	Coded read past end of record.	Fatal	77
	Illegal data in the external field.[†††]	Fatal	78
	Data converted is out of range.[†††]	Fatal	79
	Attempt to input data under Hollerith format.	Fatal	80
LENGTH	Unassigned medium[†]	Fatal	81
OUTPTB	Unassigned medium[†]	Fatal	82
OUTPTC	Unassigned medium[†]	Fatal	83
	Line limit as specified on RUN card exceeded.	Fatal	84
OUTPTS	Attempt to transfer more than 150 characters/record on ENCODE processing.	Fatal	85
REWINM	Unassigned medium[†]	Fatal	86
KODER (Coded output)	Attempt to output a single array under "D" format specification.	Fatal	87
INPUTC	Last operation was a write, no data available to read.	Fatal	88
INPUTB	List exceeds data on file, attempt to read more data than exists in the logical record.	Fatal	89
INPUTB	Last operation was a write, no data available to read.	Fatal	90
OUTPTB	Mixed mode operation	Proceed	91
IOCHEC	Unassigned medium variable file name file is undefined.	Fatal	95
	Status of buffer I/O must be checked by the unit function.	Fatal	96

[†] Execution time diagnostic occurs when a variable file name is undefined. It is printed as Unassigned medium, file xxxxxxx (where xxxxxxx is the name of the undefined file).

[††] All input/output errors at execution time are fatal. Standard error recovery for all the above cases is to terminate the job after standard error tracing.

[†††] Card image will be printed.

Routines	Condition	Standard Recovery	Error Number
INITMS, READMS, WRITMS	Unassigned medium;† variable file name is undefined.	Fatal	97
INITMS	File does not reside on a random access device.	Fatal	98
READMS, WRITMS	File not opened by a call to subroutine OPENMS	Fatal	99
READMS	Record name referred to in call is not in file index.	Fatal	100
INITMS, WRITMS	Index buffer is too small.	Fatal	101
READMS	Read parity error	Fatal	102
READMS	Index specified in this mass storage call is greater than master index or is zero.	Fatal	110
WRITEC	ECS unit has lost power or is in maintenance mode.	Fatal	112
READEC	ECS read parity error.	Proceed††	113
BUFFEO	Array too large. Buffer too small.	Fatal Fatal	114 115

† Execution time diagnostic occurs when a variable file name is undefined. It is printed as Unassigned medium, file xxxxxxx (where xxxxxxx is the name of the undefined file).

†† Fatal or non-fatal depending upon operators response.

FORTRAN LIBRARY ROUTINE ENTRY POINTS

Routine	Entry Points	Externals
ACGOER	ACGOER	SYSTEM, ABNORML
ALNLOG	ALOG, ALOG10	SYSTEM
ASINCOS	ASIN, ACOS	SYSTEM
ATAN	ATAN	SYSTEM
ATAN2	ATAN2	SYSTEM
BACKSP	BACKSP	SYSTEM, ABNORML, GETBA, CIO1., BKSPRU, FIZBAK
BUFFEI	BUFFEI	SYSTEM, ABNORML, GETBA, OPEN., CZO1.
BUFFEO	BUFFEO	SYSTEM, ABNORML, GETBA, OPEN., CIO1.
CABS	CABS	SYSTEM
CBAIEX	CBAIEX	SYSTEM
CCOS	CCOS	COS, SIN, EXP, SYSTEM
CEXP	CEXP	COS, SIN, EXP, SYSTEM
CLOG	CLOG	ALOG, ATAN2, CABS, SYSTEM
CSIN	CSIN	COS, SIN, EXP, SYSTEM
CSQRT	CSQRT	CABS, SQRT, SYSTEM
DABS	DABS	SYSTEM
DATAN	DATAN, DATAN2	SYSTEM

Routine	Entry Points	Externals
DBADEX	DBADEX, DBAREX, RBADEX	DLOG, DEXP, SYSTEM
DBAIEX	DBAIEX	SYSTEM
DBLE	DBLE	
DEXP	DEXP	SYSTEM
DISPLA	DISPLA	
DLNLOG	DLOG, DLOG10	SYSTEM
DMOD	DMOD	SYSTEM
DSIGN	DSIGN	SYSTEM
DSINCOS	DSIN, DCOS	SYSTEM
DSQRT	DSQRT	SYSTEM
DUMP	DUMP, PDUMP	OUTPUTC, STOP
DVCHK	DVCHK	
ENDFIL	ENDFIL	SYSTEM, ABNORML, GETBA, FIZBAK., ØPEN., CIØM.
FTNBIN	FTNBIN	SYSTEM, ABNORML, GETBA
EXP	EXP	SYSTEM
GETBA	GETBA	SYSTEM, ABNORML
IBAIEX	IBAIEX	SYSTEM
IDINT	IDINT	SYSTEM
IFENDF	IFENDF	SYSTEM, ABNORML, GETBA
INITMS	IXTYPE, OPENMS, STINDX	GETBA, OPEN., ABNORML, SYSTEM
INPUTB	INPUTB	SYSTEM, ABNORML, GETBA, OPEN., CIO1., RDWDS.
INPUTC	INPUTC	SYSTEM, ABNORML, GETBA, KRAKER, OPEN., RDCARD., DAT.

Routine	Entry Points	Externals
INPUTS	INPUTS	SYSTEM, ABNORML, KRAKER
IOCHEK	IOCHEK	SYSTEM, ABNORML, GETBA, RDWDS., CIØ1.
IOCHEC	IOCHEC	SYSTEM, ABNORMAL, GETBA
KODER	KODER	SYSTEM, ABNORML
KRAKER	KRAKER	SYSTEM, ABNORML
LEGVAR	LEGVAR	
LENGTH	LENGTH	SYSTEM, ABNORML, GETBA
LOCF	LOCF, XLOCF	
OUTPTB	OUTPTB	SYSTEM, ABNORML, GETBA, ØPEN., WRWDS., CIØ1.
OUTPTC	OUTPTC	SYSTEM, ABNORML, GETBA, KODER, ØPEN., WRWDS., DAT., FIZBAK.
OUTPTS	OUTPTS	SYSTEM, ABNORML, KODER
OVERFL	OVERFL	
OVERLAY	OVERLAY	LOADER, SYSTEM, ABNORML
PAUSE	PAUSE	
RANF	RANF	
RBAIEX	RBAIEX	SYSTEM
RBAREX	RBAREX	ALOG, EXP, SYSTEM
READEC	READEC	SYSTEM
READMS	READMS	GETBA, SIO., CIO1., IXTYPE, ABNORML, SYSTEM
REMARK	REMARK	
REWINM	REWINM	SYSTEM, ABNORML, GETBA, CIO1.
SECOND	SECOND	

Routine	Entry Points	Externals
SEGMENT	SEGMENT	LOADER, SYSTEM, ABNORML
SIØN	SIØ., CIØ1., ØPEN., BKSPRU., FIZBAK	
SINCOS	SIN, COS	SYSTEM
SLITE	SLITE	SYSTEM
SLITET	SLITET	SYSTEM
SNGL	SNGL	
SQRT	SQRT	SYSTEM
SSWTCH	SSWTCH	SYSTEM
SYSTEM	SYSTEM, SYSTEMC, SYSTEMP, Q8NTRY, STOP, END, EXIT, ABNORML	
TAN	TAN	SYSTEM
TANH	TANH	EXP, SYSTEM
TIME	TIME	
WRITEC	WRITEC	
WRITMS	WRITMS	

FORTRAN FILE STRUCTURE AND BLOCKED BINARY INPUT/OUTPUT — M

STRUCTURE OF FILES

A file is an ordered sequence of user logical records. Each type of input/output that a FORTRAN programmer can use has a user logical record definition.

FORMATTED I/O

```
READ        f,k
PRINT       f,k
PUNCH       f,k
READ(u,f)   k
READ(u,f)
WRITE(u,f)  k
WRITE(u,f)
```

The user logical record (also referred to as a unit record) corresponds to a card image or a print line. User logical records may be a maximum of 150_{10} characters for input, but no more than 136_{10} are transferred on output records. (On S and L tapes a user logical record corresponds to a tape block.) For X tapes, user logical records may have a maximum of 136 characters for input/output records (only 135 will be printed).

UNFORMATTED I/O

```
READ(u)     k
WRITE(u)    k
```

When I/O is unformatted, the user logical record is the same as a SCOPE logical records on internal files or X magnetic tape files. On an S and L magnetic tape the physical representation of user logical records is the same as that on a SCOPE internal tape even though there is no SCOPE-logical-record definition.

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 (137 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.

UNFORMATTED I/O

Since the physical representation of FORTRAN unformatted user logical records is the same on S tapes as that on SCOPE internal tapes, the files created may be used interchangeably; a tape created as a SCOPE internal tape may be read as an S tape. Likewise, a tape created as an S tape may be read as a SCOPE internal tape (tapes written as X tapes must be read as X tapes). On L tapes the maximum physical record (user logical record) is determined by the size of the user's buffer area.

Throughput of small user logical records can be increased if S magnetic tapes are used instead of SCOPE internal or L tapes. Non-stop tape motion can often be achieved when the buffer size is in excess of $2,048_{10}$ words, which is four tape physical record units.

With use of blocked binary input/output files, logical records are grouped into blocks and accesses are made in terms of blocks rather than records. Since with each access several logical records, rather than only one, are transmitted to or from the file buffer, the speed of job throughput is increased by decreasing the number of accesses made to an input/output device.

FILE FORMAT

Blocked binary files have the following format: control word (CW), one logical record or part of one logical record (LR), CW, LR, etc. Logical records are packed in the order in which they are referenced, but the end of a block need not correspond to the end of a record. Records may extend across several blocks depending on their lengths and positions in the file.

At least one control word is associated with each logical record. Control words created during output indicate the beginning of a logical record, the continuation of a logical record, and the EOF.

Each control word contains:

End of record flag to indicate whether or not the next control word is at the beginning of a logical record

Number of words between it and the control word at the beginning of the previous logical record (current record if it is the control word of a continuation).

Number of words between it and the next control word.

INPUT/OUTPUT OPERATIONS

Blocked binary input/output operations are performed under the assumption that the file is in blocked format; nonblocked binary operations assume a nonblocked file format. If the mode of the operation is not the same as that of the file, the operation will not be executed correctly.

The decision on whether blocked or unblocked binary files will be output is made by each installation. However, to change the format of one or more binary files, the library subroutine FTNBIN (section 7.10) may be called to override the installation selection.

FORTRAN CROSS-REFERENCE MAP

If the ninth field of the run control card is non-zero, FORTRAN supplies the programmer with a cross-reference map after each PROGRAM, SUBROUTINE, or FUNCTION, purely as an aid to program debugging. The following information is furnished:

Program length including I/O Buffers

Statement function references with the relative core locations, general compiler tag assigned, symbolic tag given in the program and the references to the statement function

Statement number references with the same information as above

Block names and lengths

Variable references — also with location, general tag, symbolic tag, and a list of references

Start of constants (relative address)

Start of temporaries (relative address)

Start of indirects (relative address)

Unused compiler space

The programmer should bear in mind that because of the operation of the compiler not all references will be listed. An actual physical reference is necessary before the reference is placed in the reference map. If the required variable address is already in a register, the compiler will use the address in the register and not make an actual variable reference by name. A reference to a statement number will not be listed if an actual jump is not necessary, such as when the code simply falls through to the next statement and the compilation of a jump instruction is therefore unnecessary.

The following cross reference map was produced by a main program compiled in $40,000_8$ words of memory.

```
STORAGE MAP FOR MAIN PROGRAM

PROGRAM LENGTH INCLUDING I/O BUFFERS
002453
```

Length of main program; includes (for this example 2022_8 words for I/O buffers.

```
STATEMENT FUNCTION REFERENCES

         LOCATION  GEN TAG   SYM TAG   REFERENCES

STATEMENT NUMBER REFERENCES

         LOCATION  GEN TAG   SYM TAG   REFERENCES
         000201    L00057    2         000177
         000214    C00007    90        000003
         000223    C00016    100       000007
         000225    C00020    101       000015
         000233    C00026    110       000117
         000255    C00050    120       000151
         000267    C00062    130       000155
         000247    C00042    140       000137
         000230    C00023    150       000024
```

Statement labels and locations. The statement labeled 2 is generated object code at location 201. FORMAT statements are transmitted and stored in BCD code in object program. FORMAT statement 100 is stored at location 223 which follows the main program code and constants.

```
BLOCK NAMES AND LENGTHS
      -  000002     LAB1    -  000001
```

Length of blank and labeled common.

```
VARIABLE REFERENCES

         LOCATION  GEM TAG   SYM TAG   REFERENCES
         000356    V00011    AB        000053    000071    000111    000114    000142
         000357    V00012    AC        000062    000070    000075    000110    000114    000144
         000000C01 V00020    ALPHA     000166
         000362    V00015    AREA      000105    000107    000113    000160
         000360    V00013    BC        000071    000077    000110    000115    000146
         000001C01 V00021    BETA      000170
         000000C02 V00002    GAMMA     000172
```

Location of all variables. AREA is stored at locati 362. BETA is stored in second word of common.

```
START OF CONSTANTS
000205
```

All constants are stored immediately following program object code.

```
START OF TEMPORARIES
000274

START OF INDIRECTS
000346

UNUSED COMPILER SPACE

002700
```

This program used (40000-2700) or 35100_8 memory positions.

Unused compiler space is 2700_8 for this program.

PRINT FILE CONVENTIONS

O

Files with a print disposition (including OUTPUT) and files assigned to a printer, must adhere to specific format rules as follows:

1. All characters must be in display code.
2. The end of a print line must be indicated by a zero byte in the lower 12 bits of the last central memory word of the line. Any other unused characters in the last word should be filled with display code blanks (55_8). For example, if the line has 136 characters (including carriage control), the last word would be aabbccddeeffgg550000 in octal; the letters represent the last seven characters to be printed in the line. No line should be longer than 136 characters.
3. Each line must start in the upper 6 bits of a central memory word.
4. The first character of a line is the carriage control, which specifies spacing as shown in the following table. It will never be printed, and the second character in the line will appear in the first print position; therefore a maximum of 136 characters can be specified for a line, but only 135 characters will be printed. All characters apply to both the 501 printer and the 512 printer unless they are specifically designated otherwise.

Carriage Control Characters

Character	Action Before Printing	Action After Printing
A	Space 1	Eject to top of next page
B	Space 1	Space 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	No space

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)
T	Select 8 vertical lines per inch (512)
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 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.

INDEX

COMMENT SHEET

CONTROL DATA
CORPORATION

TITLE: FORTRAN Reference Manual

PUBLICATION NO. 60174900 REVISION E

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 do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

FROM NAME: ______________________ POSITION: ______________________

BUSINESS
ADDRESS: ____

____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

CUT ON THIS LINE

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