3400
3600
3800

COMPUTER SYSTEMS FORTRAN

REFERENCE MANUAL

CONTROL DATA
CORPORATION

3400
3600
3800

COMPUTER SYSTEMS
FORTRAN

REFERENCE MANUAL

CONTROL DATA
CORPORATION

60132900

REVISION RECORD

REVISION	NOTES
	This printing includes the most current revision level.
(8-65)	Original printing.
A	
(12-66)	Reprint with revision.

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

Pub. No. 60132900

Address comments concerning this manual to:

Control Data Corporation
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

PREFACE

This manual describes the combined features of 3400 FORTRAN† (version 1.4) and 3600 FORTRAN (version 5.3). 3600 FORTRAN is also used on the 3800 computing system. 3400 FORTRAN source language is upward compatible with 3600 FORTRAN source language. Both languages contain the features of FORTRAN-63.

This reference manual was written as a text for advanced 3400 and 3600 FORTRAN classes and as a reference manual for programmers using the 3400 and 3600 FORTRAN system. The manual assumes a basic knowledge of the FORTRAN language, COMPASS language, and the SCOPE monitor system.

For additional information see:

	Publication No.
3400 SCOPE/COMPASS Reference	60057800
3600 SCOPE Reference	60053300
3400/3600 Instant FORTRAN	60057500
3400 FORTRAN/Library Routines	60057200
3600 FORTRAN/Library Routines	60056400

†FORTRAN is an abbreviation for FORmula TRANslation and was originally developed for International Business Machine equipment.

CONTENTS

1.1 QUANTITIES

FORTRAN handles floating point and integer quantities. Floating point quantities have an exponent and a fractional part. The following types of numbers are floating point quantities.

REAL	Exponent and sign 11 bits; fraction and sign 37 bits; range of number (in magnitude) $10^{-307} \leq \lceil N \rceil \leq 10^{307}$ and zero; precision approximately 10 decimal digits.
DOUBLE	Exponent and sign 11 bits; fraction and sign 85 bits; range of number (in magnitude) $10^{-307} \leq \lceil N \rceil \leq 10^{307}$ and zero; precision approximately 25 decimal digits.
COMPLEX	Two consecutive reals, as defined above, which constitute the real and imaginary parts respectively.

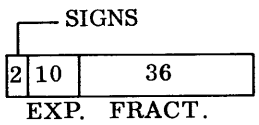
Integer quantities do not have a fractional part. The following types of numbers are integer quantities:

INTEGER	Represented by 48 bits, left most bit is the sign; range of number (in magnitude) $0 \leq \lceil N \rceil \leq 2^{47} - 1$; precision is up to 15 decimal digits.
LOGICAL	1 bit represents the value true. 0 bit represents the value false.
HOLLERITH	Binary coded decimal (BCD) representation treated as an integer number.

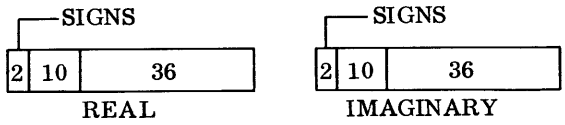
The FORTRAN program may contain any or all of these types of numbers in form of constants, variables, elements of arrays, evaluated functions and so forth. Variables, arrays and functions are associated with types assigned by the programmer. The type of a constant is determined by its form.

WORD STRUCTURE

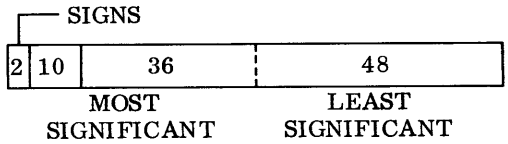
Floating Point Quantities



REAL - 1 word

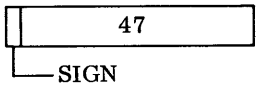


COMPLEX - 2 words



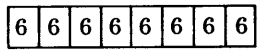
DOUBLE - 2 words

Integer Quantities



INTEGER - 1 word

LOGICAL - 1 word
(unsubscripted variable)



HOLLERITH - 1 word
6 bits/character

1.2 CONSTANTS

Five basic kinds of constants are used in FORTRAN: integer, octal, floating point, Hollerith, and logical. Complex and double precision constants can be formed from floating point constants. The type of a constant is determined by its form and context.

INTEGER

Integer constants may consist of up to 15 decimal digits. If the range of $0 \leq \lceil N \rceil \leq 2^{47}-1$ is exceeded, the constant is treated as zero and a compiler diagnostic is provided.

Examples:

63	3647631
247	2
314159265	464646464

OCTAL

Octal constants may consist of up to 16 octal digits. The form is:

n_1 --- n_iB

n_i are octal digits.

If the constant exceeds 16 digits, or if a non-octal digit appears, the constant is treated as zero and a compiler diagnostic is provided.

Examples:

```
7777777700000000B
   7777700077777B
2323232323232323B
              77B
777777777777700B
```

LOGICAL

Logical constants are in the form .TRUE. (1) and .FALSE. (0). Logical constants are treated as integers.

FLOATING POINT REAL

Real constants may be expressed with a decimal point or with a fraction and an exponent representing a power of ten. Forms of real constants:

nE n.n n. .n nE±s n.nE±s n.E±s .nE±s

n is the base; s is the exponent to the base 10. The plus sign may be omitted for positive s. The range of s is 0 through 308. If the range is exceeded, the constant is treated as zero and a compiler diagnostic occurs. When the exponent (E) is followed by a + or - sign, all characters between the sign and the next operator or delimiter are assumed to be part of the exponent expression.

Examples:

```
3.1415768          31.41592E-01
314.               .31415E01
.0749162           .31415E+ 01
314159E-05
```

DOUBLE

Double precision constant forms:

nD n.nD n.D .nD nD ± s .nD ± s n.nD ± s n.D ± s

The base is n; s is the exponent to the base 10. The D must always appear. The plus sign may be omitted for positive s; the range of s is 0 through 308. If the range is exceeded, the constant is treated as zero and a compiler diagnostic occurs. When the exponent (D) is followed by a + or - sign, all characters between the sign and the next operator or delimiter are assumed to be part of the exponent expression. Unnormalized double precision numbers are not allowed in 3400.

Examples:

3.14159265358979323846 26D

3.1415D

3.1415D0

3141.598D-03

31415.D-04

379867524430111D+01

COMPLEX

Complex constants are represented by pairs of real constants separated by a comma and enclosed in parentheses (R_1, R_2). R_1 represents the real part of the complex number and R_2, the imaginary part. Either constant may be preceded by a minus sign.

If the range of the reals forming the constant is exceeded, a compiler diagnostic is provided. Diagnostics also occur when the number pair consists of 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.)	-i

HOLLERITH

A Hollerith constant is a string of alphanumeric characters of the form hHf; h is an unsigned decimal integer between 1 and 136 representing the length of the field f. Spaces are significant in the field f. When h is not a multiple of 8, the last computer word is left-justified with BCD spaces filling the remainder of the word. When h is greater than 136, a diagnostic is provided.

An alternate form of a Hollerith constant is hRf. When h is less than or equal to 8, the computer word is right-justified with zero fill. When h is greater than 8 only the first 8 characters are retained and the excess characters are discarded, but no diagnostic is provided. h may not be zero.

Hollerith constants appearing in arithmetic replacement statements and as actual parameters will be treated as integers and only the first 8 characters will be used. Hollerith constants may also appear in DATA statements where the entire constant is used.

Examples:

6HCOGITO	8RCDC 3600
4HERGO	8R **
3HSUM	1H)

1.3 VARIABLES

Simple and subscripted variables are recognized. A simple variable represents a single quantity; a subscripted variable represents an array or a single quantity (element) within an array of quantities. Variables are identified by 1-8 alphanumeric characters; the first character must be alphabetic.

The variable type may be defined in a TYPE declaration (section 4.1) and may be integer, real, complex, double, logical, or an arbitrary mode (5.1).

If a variable is not declared by a TYPE declaration, it is determined by the first letter of the variable name. I, J, K, L, M, or N indicates a fixed point (integer) variable; any other first letter indicates a floating point (real) variable.

SIMPLE

A simple variable references the location in which values can be stored. The value specified by the name is always the current value stored in that location.

Examples:

VECTOR	A65302	
BAGELS	BATMAN	
N	NOODGE	
K2SO4	M58	Since spaces are ignored in variable names, M58 and M 58 are identical.
LOX	M 58	

SUBSCRIPTED VARIABLE

A subscripted variable is an alphanumeric identifier with one, two, or three associated subscripts enclosed in parentheses. If more than three subscripts appear, a compiler diagnostic is given. The identifier is the name of an array. The subscripts can be integer constants, variables, or expressions. Any other constant, variable, or expression will be truncated to an integer value.

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.

SUBSCRIPT FORMS

A standard subscript has one of the following forms; other forms are non-standard. I is a simple integer variable which must be defined before being used, and c and d are unsigned integer constants.

c * I ± d

I ± d

c * I

I

c

Examples:

Subscripted Variable (Standard)	Subscripted Variable (Non-standard)
A(I,J)	A(MAXF(I,J,M))
B(I+2,J+3,2*K+1)	B(J,SINF(J))
Q(14)	C(I+K)
P(KLIM,JLIM+5)	MOTZO(3*K*ILIM+3.5)
SAM(J-6)	WOW(I(J(K)))
B(1,2,3)	Q(1,-4,-2)

Standard subscripted variables are treated as non-standard subscripted variables whenever (array length - constant addend -1) ≥ 32768. Array length is the greater of number of words or number of elements. See page E-5 for definition of constant addend.

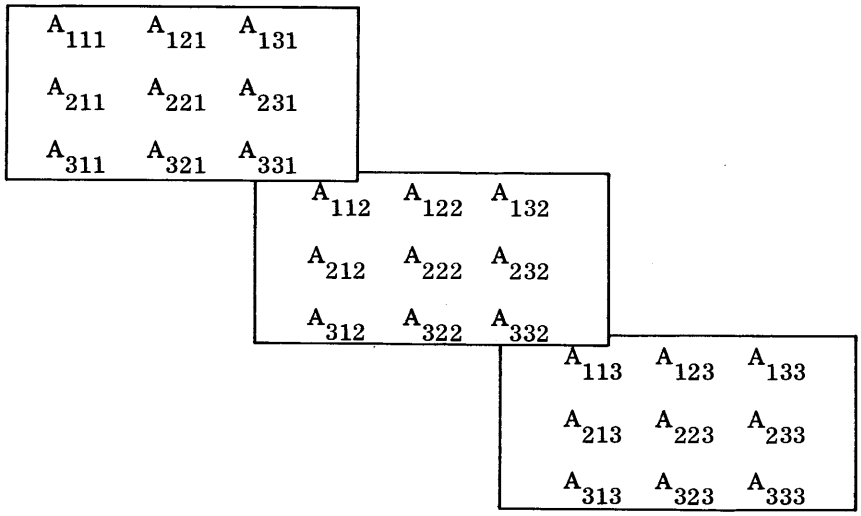
1.4 ARRAYS

An array is a block of successive memory locations for storage of variables. The entire array may be referenced by the array name without subscripts (I/O lists and implied DO loops, 9.1). Arrays may have one, two, or three dimensions; the array name and dimensions must be declared at the beginning of the program in a DIMENSION, COMMON, or TYPE (except 3400) statement (sections 4.1, 2, 3). The type of an array is determined by the array name or the TYPE declaration.

Each element of an array may be referenced by the array name plus the 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, the product of the dimensions, cannot exceed 32767. The maximum amount of storage reserved for an array cannot exceed 32767 words.

Array Structure

Elements of arrays are stored by columns in ascending order of storage location. In the array declared as A(3,3,3):



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

$A_{111} \rightarrow A$ $\quad A_{121} \rightarrow A+3 \ldots A_{133} \rightarrow A+24$

$A_{211} \rightarrow A+1$ $\quad A_{221} \rightarrow A+4 \ldots A_{233} \rightarrow A+25$

$A_{311} \rightarrow A+2$ $\quad A_{321} \rightarrow A+5 \ldots A_{333} \rightarrow A+26$

Array allocation is discussed in Chapter 4. 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. Addresses are computed modulo 2^{15}. 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 in braces is the subscript expression. If it is not an integer value, it is truncated after evaluation.

E is the element length, the number of storage words required for each element of the array; for real and integer arrays, E = 1.

Referring to the array A(3,3,3) the location of A (2,2,3) with respect to A (1,1,1) is

$$\text{Locn } \{A(2,2,3)\} = \text{Locn } \{A(1,1,1)\} + \{2\text{-}1\text{+}3(1\text{+}3(2))\}$$
$$= A + 22$$

Example:

Given DIMENSION Z (5,5,5) and I = 1, K = 2, X = 45^{o}, A = 7.29, B = 1.62. The location, z, of Z (I * K, TANF (X), A − B) with respect to Z (1,1,1) is:

$z = \text{Locn } \{Z(1,1,1)\} + \{2\text{-}1\text{+}5(1\text{-}1\text{+}5(4.67))\}$ Integer part

$= \text{Locn } \{Z(1,1,1)\} + \{117.75\}$ Integer part

$= \text{Locn } \{Z(1,1,1)\} + 117$

Given the location, A + constant of A (i, j, k) with respect to the first element of the array dimensioned A (L, M, N) with E words per element, i, j and k may be found by the following procedure:

Compute: X0 = (constant + (L*(M+1)+1)*E)/E

a) K = X0/(L*M) with remainder of X1

if X1 = 0, then k = K-1, j = M-1, i =L,
else, if X1 ≤ L, then k = K-1, X1 = X1 + (L*M)
or, if X1 > L, k = K and

b) J = X1/L with remainder of I
if I = 0, then j = J-1, i = L
else j = J, i = I

For two dimensional arrays such as A (L, M):

Compute: X1= (constant + (L+1)*E)/E

J = X1/L with remainder of I
if I = 0, then j = J-1, i= L
else j = J, i = I

Examples:

1. Given A + 54 for the real array A (2, 5, 7)
X0 = 54 + 2 (5 +1) + 1 = 67

a) K = 67/10 = 6 with X1 = 7
k = 6

b) J = 7/2 = 3 with I = 1
j = 3, i = 1

therefore A(1, 3, 6) = A + 54 of A (2, 5, 7)

2. Given B + 124 for the real array B (5, 13, 3)
X0 = 124 + 5 (13 + 1) + 1 = 195

a) K = 195/65 = 3 with X1 = 0
therefore k = 2, j = 12, i = 5

3. Given P + 27 for array P (7, 3) with E = 3
X 1 = (27 + (7 + 1)*3)/3 = 17

b) J = 17/7 = 2 with I = 3
therefore j = 2 , i = 3

An integer variable which appears as a subscript in any multi-subscripted array element reference should never be assigned a value greater than $2^{15}-1$ since address calculation depends on the previous value of the index function calculation. For example:

```
          .
          .
          .
J = 2525252525252526B
          .
          .
          .
J = 1
IARRAY (I, J) = 1
```

The index of IARRAY is miscalculated since the previous value of J exceeded $2^{15}-1$.

Array Notation

The following subscript notations are permitted for array elements:

For a 3-dimensional array $A(D_1, D_2, D_3)$,

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)

For a 2-dimensional array A (D_1, D_2),

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

For a single-dimension array A (D_1),

A(I)	implies A(I)
A	implies A(1)

However, the elements of a single-dimension array A(D_1) may not be referred to as A(I,J,K) or A(I,J), and elements of a two-dimensional array A (D_1, D_2) may not be referred to as A(I,J,K). Diagnostics occur if this is attempted.

1.5 STATEMENTS

Statements are the basic functional units of the language. An executable statement performs a calculation or directs control of the program; a nonexecutable statement provides the compiler with information regarding variable structure, array allocation, storage sharing requirements, etc.

1.6 EXPRESSIONS

An expression is a constant, variable, function or any combination of these separated by operators and parentheses, written to comply with the rules given for constructing a particular type of expression.

There are four kinds of expressions in FORTRAN: arithmetic and masking (Boolean) expressions which have numerical values, and logical and relational expressions which have truth values. For each type of expression there is an associated group of operators and operands.

EXPRESSIONS 2

2.1 ARITHMETIC EXPRESSIONS

An arithmetic expression may be a constant, variable (simple or subscripted), or an evaluated function (Chapter 7). Arithmetic expressions may be combined by arithmetic operators to form complicated arithmetic expressions.

Arithmetic operators are:

+	addition	/	division
-	subtraction	**	exponentiation
*	multiplication		

An arithmetic expression may not contain adjacent arithmetic operators: X op op Y is not permitted.

If X is an expression, (X), ((X)) are expressions. If X, Y are arithmetic expressions, then the following are expressions:

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

Expressions of the form X**Y and X**(-Y) are legitimate, subject to the restrictions in section 2.3, Masking Expressions.

The following forms of implied multiplication are permitted:

constant (...)	implies constant * (...)
(...) (...)	implies (...) * (...)
(...) constant	implies (...) * constant
(...) variable	implies (...) * variable

Complex constants are enclosed in two set of parentheses:

constant ((R_1, R_2))	implies constant * (R_1, R_2)

Expressions:

```
A

3.141592

B + 16.8946

(A - B(I, J + K))

G * C(J) + 4.1/ (Z(I+J, 3*K))*SINF(V)

(Q + V(M, MAXF(A, B))*Y**2)/(G*H-F(K + 3))

-C + D(I, J)*13.627
```

ORDER OF EVALUATION

The hierarchy of arithmetic operation 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 the operators are in different classes, evaluation proceeds in the above order. When expressions contain operators in the same class, evaluation proceeds from left to right. For example, A**B**C is evaluated as (A**B)**C.

Exponentiation is performed and parenthetical expressions are evaluated as they are encountered in the left to right scanning process. In nested parenthetical expressions, evaluation begins with the innermost expression.

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 $\rightarrow R_1$

R_1/C $\rightarrow R_2$

D*E $\rightarrow R_3$

R_3*F $\rightarrow R_4$

$R_4+R_2 \rightarrow R_5$

R_5-G $\rightarrow R_6$ evaluation completed

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

A**B $\rightarrow R_1$

C+D $\rightarrow R_2$

E*F-G $\rightarrow R_3$

$R_1/R_2 \rightarrow R_4$

R_4*R_3 $\rightarrow R_5$ evaluation completed

If the expression contains a function, the function is evaluated first.

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

COSF(Z) $\rightarrow R_1$

R_1**2 $\rightarrow R_2$

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

R_3+H(13) $\rightarrow R_4$ evaluation completed

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

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

C/D $\rightarrow R_1$

R_1-E $\rightarrow R_2$

R_2+B $\rightarrow R_3$

R_3*A $\rightarrow R_4$ evaluation completed

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

$SINF(X) \rightarrow R_1$

$R_1+1. \rightarrow R_2$

$E+F \rightarrow R_3$

$-R_3 \rightarrow R_3$

$R_3+D \rightarrow R_4$

$R_4*C \rightarrow R_5$

$-Z \rightarrow R_6$

$R_6/R_5 \rightarrow R_7$

$R_7+R_2 \rightarrow R_8$ evaluation completed

MIXED MODE ARITHMETIC

Full mixed mode arithmetic is permitted; mixed mode arithmetic is accomplished through the special library subroutines. In the 3400 computer system, these routines include complex and double precision arithmetic. In the 3600 computer some double arithmetic is provided by the hardware. The five standard operand types are complex, double, real, integer, and logical. The programmer may also define three non-standard types. (chapter 5)

Mixed mode arithmetic is completely general; however, most applications will probably mix operand types, real and integer, real and double, or real and complex. The following rules establish the relationship between the mode of an evaluated expression and the types of the operands it contains.

The mode of an evaluated arithmetic expression is referred to by the name of the dominant operand type. The order of dominance of the standard operand types within an expression from highest to lowest is:

COMPLEX
DOUBLE
REAL
INTEGER
LOGICAL

In mixed arithmetic expressions containing non-standard types the following restrictions hold:

The three non-standard types may never be mixed with each other; but any one of them may be mixed with any or all of the standard types. When this is done, the non-standard type dominates the hierarchy established above.

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

B may not be type logical or byte (non-standard) type. If A is logical or byte type, B must be an integer constant from 1 to 8.

B may be negative in which case the form is: A**(-B).

If A or B or both are of non-standard type, the programmer must provide subroutines for the evaluation of A**B.

For the standard types (except logical) the mode/type relationships are:

Type A \ Type B	I	R	D	C
I	I	R	D	C
R	R	R	D	C
D	D	D	D	C
C	C	C	C	C

} mode of A**B

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

The following exponentiation routines are provided:

real ** real	integer ** integer	double ** double
real ** integer	integer ** double	double ** complex
real ** double	integer ** complex	double ** real
real ** complex	integer ** real	double ** integer
		complex ** integer

complex ** complex
complex ** real
complex ** double
} Calls to these routines will give an error message.

EVALUATION EXAMPLES

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

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

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

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

3) Given C1, C2 type complex; A1, A2 type real. The mode of expression A1*(C1/C2)+A2 will be complex because its dominant operand is type complex. It is evaluated:

$C1/C2 \rightarrow R_1$		
Convert A1 to complex		
$A1*R_1 \rightarrow R_2$	complex	
Convert A2 to complex		
$R_2+A2 \rightarrow R_3$	complex	evaluation completed

4) Consider the expression C1/C2+(A1-A2) where the operands are defined as in 3 above. It is evaluated:

$A1-A2 \rightarrow R_1$	real	
Convert R_1 to complex $\rightarrow R_1$		
$C1/C2 \rightarrow R_2$	complex	
$R_2+R_1 \rightarrow R_3$	complex	evaluation completed

5) Mixed mode arithmetic with all standard types is illustrated by this example.

Given: C complex
D double
R real
I integer
L logical

and the expression C*D+R/I-L

The dominant operand type in this expression is type complex; therefore, the evaluated expression will be of mode complex. Evaluation:

Round D to a real and affix zero imaginary part:

$C*D \rightarrow R_1$ complex

Convert R to complex; convert I to complex

$R/I \rightarrow R_2$ complex

$R_2+R_1 \rightarrow R_3$ complex

Convert L to complex

$R_3-L \rightarrow R_4$ complex evaluation completed

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

Convert I to real

$R/I \rightarrow R_1$ real

Convert L to real

$R_1-L \rightarrow R_2$ real

Convert R_2 to complex

Round D to real and affix zero imaginary part

$C*D \rightarrow R_3$ complex

$R_3+R_2 \rightarrow R_4$ complex evaluation completed

2.2 LOGICAL EXPRESSIONS

Any logical operand by itself is a logical expression or may be combined by logical operators with other logical operands to form more complex logical expressions. The value of a logical expression is true if non-zero, false if zero.

When an arithmetic expression appears as a term of a logical expression, the value is examined. If the value is non-zero, the term is true. If the value is zero, the term is false.

If L is a logical expression, (L), ((L)) are logical expressions. Logical expressions are generally used in logical IF-statements (section 6.2).

A logical operand may be:

logical variable

logical constant (. TRUE. or . FALSE.)

arithmetic expression

arithmetic relation

ARITHMETIC RELATION

An arithmetic relation has the form:

q_1 op q_2

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

Operator	Meaning
. 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 q_1 and q_2 satisfy the relation specified by op, otherwise it is false.

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, the relation is true. If the difference is not zero, the relation is false. Relations

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 expressions of mode complex are tested for zero, only the real part is used in the comparison.

q_1 op q_2 op q_3... is <u>not</u> permissible

The evaluation of a relation of the form q_1 op q_2 is from left to right. The relations q_1 op q_2, q_1 op (q_2), (q_1) op q , (q_1) op (q_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)

LOGICAL OPERATIONS

In the general forms of logical expressions containing logical operators, L_i are logical operands:

L_1 op L_2 op L_3...	op is .AND. indicating conjunction or .OR. indicating disjunction
op L_1	op is .NOT. indicating negation

Logical expressions are scanned left to right and logical operations are performed according to the following precedence:

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

L_1 op op L_2 is not permitted if the logical operators are .AND. or .OR.

The following combinations of NOT are allowed:

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. . . . . Other combinations will 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

Incorrect usages such as the following will cause compiler diagnostics.

A. GT. (B. AND. C)

Q. NOT. . OR. R

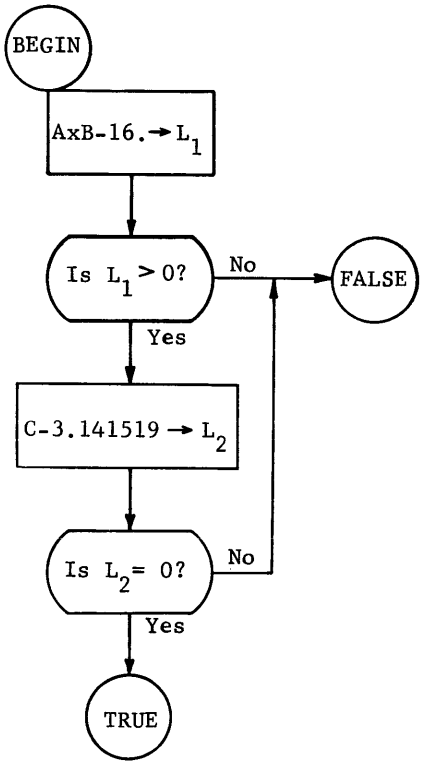
C. AND. . NOT. . NOT. B

The last expression is permissible in the form C. AND. . NOT. (. NOT. B).

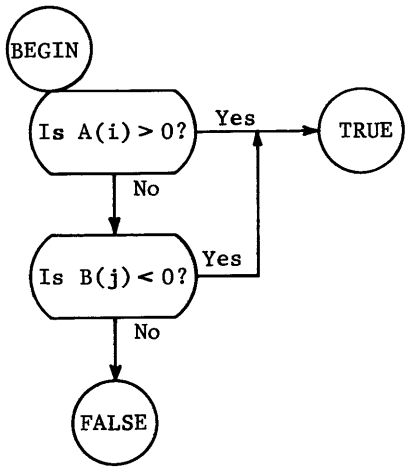
Examples: Logical Expressions:

{The product A*B greater than 16.} . AND. {C equals 3.141519}

A*B . GT. 16. . AND. C . EQ. 3.141519

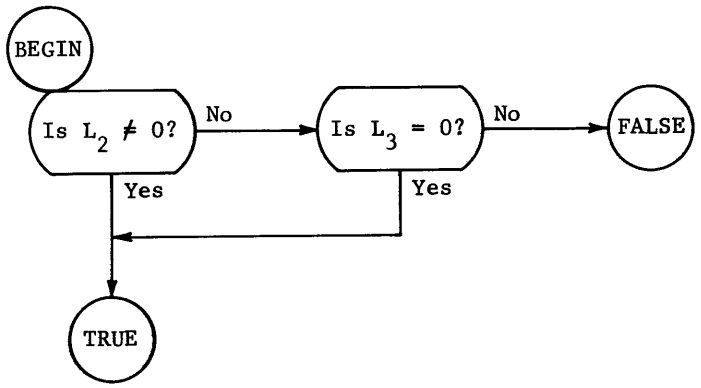


{A(I) greater than 0} .OR. {B(J) less than 0} A(I) .GT. 0 .OR. B(J) .LT. 0

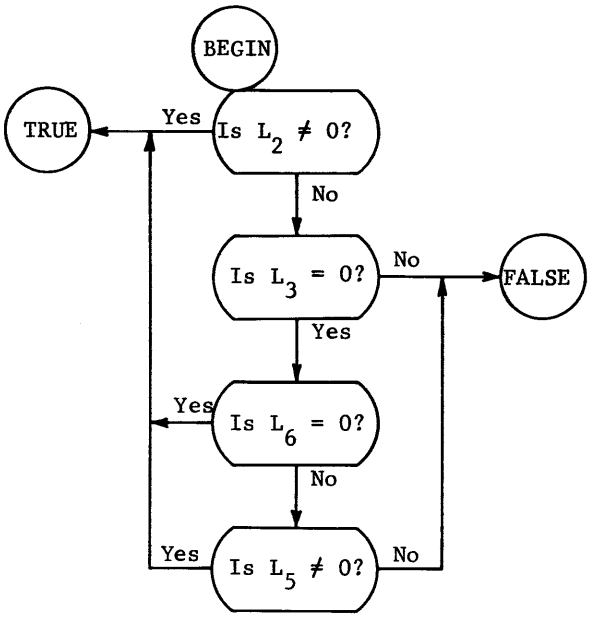


In the two examples below, all L_i are of TYPE LOGICAL

(L2 .OR. .NOT. L3)



L2 .OR. .NOT. L3 .AND. (.NOT. L6 .OR. L5)



2.3 MASKING EXPRESSIONS

In a masking expression, 48-bit arithmetic is performed bit-by-bit on the operands within the expression.

The following are masking expressions:

B_1 op B_2	op is .AND. or .OR.
op B_1	op is .NOT.

A masking expression may be enclosed in parentheses, (B), ((B)), etcetera.

The masking operand, B_i, may be:

variable (real or integer)	function (real or integer)
unsigned constant (real or integer)	masking statement function
masking expression	

Type integer includes octal and Hollerith constants. If operands of other types are used, a diagnostic will occur.

Mode conversion does not occur for mixed real and integer operands.

Although the masking operators are identical in appearance to the logical operators, their meanings are different. They are listed according to hierarchy, and the following definitions apply:

.NOT.	complement the operand
.AND.	form the bit-by-bit logical product of two operands
.OR.	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

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

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

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

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

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

Masking expressions must not contain arithmetic or relational operators, or statement functions other than masking statement functions.

A masking expression is valid only in a masking replacement statement. A masking expression will be interpreted as logical if the replacement variable is type logical or if used in a logical IF statement.

Examples:

A_1	7777000000000000		octal constant
A_2	0000000077777777		octal constant
B	0000000000001763		octal form of integer constant
C	2004500000000000		octal form of real constant
.NOT. A_1		is	0000777777777777
A_1 .AND. C		is	2004000000000000
A_1 .AND. .NOT. C		is	5773000000000000
B .OR. .NOT. A_2		is	7777777700001763

REPLACEMENT STATEMENTS 3

Values are assigned to variables by the replacement statement: V = E.

The = operator means that V is replaced by the value of the evaluated expression, E, with conversion for mode if necessary. The replacement variable, V, may be simple or subscripted. Replacement statements may be arithmetic, logical, or masking.

3.1 ARITHMETIC REPLACEMENT

In an arithmetic replacement statement, the replacement variable V is of any type, and E is any arithmetic expression.

Examples:

```
A = 1+BA(2)*3          C = D-(Y+3)
X(4) = (Y-Z(3,2))/J+K  L = (J+K(1,1,1))
```

MIXED MODE REPLACEMENT

The mode of the evaluated arithmetic expression and the type of the replacement variable may be mixed. However, if the mode of the replacement variable is of non-standard type, the mode of the evaluated arithmetic expression must not be a different non-standard type (see Chapter 5). The following chart shows the V, E relationship for all the standard modes.

Arithmetic Replacement Statement V = E

V is an identifier E is an arithmetic expression R is the evaluated arithmetic expression

Type of V	Mode of R			
	Complex	Double	Real	Integer
Complex	Store real & imaginary parts of R in real & imaginary parts of V.	Round R to real. Store in real part of V. Store zero in imaginary part of V.	Store R in real part of V. Store zero in imaginary part of V.	Convert R to real & store in real part of V. Store zero in imaginary part of V.
Double	Discard imaginary part of R & replace it with ±0 according to real part of R.	Store R (most & least significant parts) in V (most & least significant parts).	If R is ± affix ±0 as least significant part. Store in V, most & least significant parts.	Convert R to real. Fill out least significant half with binary zeros or ones accordingly as sign of R is plus or minus. Store in V, most and least significant parts.
Real	Store real part of R in V. Imaginary part is lost.	Round R to real & store in V. Least significant part of R is lost.	Store R in V.	Convert R to real. Store in V.
Integer	Truncate real part of R to INTEGER. Store in V. Imaginary part is lost.	Truncate R to INTEGER & store in V.	Truncate R to INTEGER. Store in V.	Store R in V.
Logical	If real part of R ≠0, 1 → V. If real part of R =0, 0 → V.	If R ≠0, 1 → V. If R =0, 0 → V.	Same as for double at left.	Same as for double at left.

When all of the operands in the expression E are of type 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

$I = L_1*L_2+L_3-L_4$

will be evaluated as if the L_i were all integers (0 or 1) and the resulting value will be 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.

Examples:

Given: C_i, A_1 complex — I_i, A_4 integer

D_i, A_2 double — L_i, A_5 logical

R_i, A_3 real

$A_1 = C_1 * C_2 - C_3/C_4$ — (6.907, 15.393) = (4.4, 2.1) * (3.0, 2.0) - (3.3, 6.8)/(1.1, 3.4)

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

$A_3 = C_1$ — 4.4000+000 = (4.4, 2.1)

The mode of the expression is complex. A_3 is type real; the real part of C_1 replaces A_3.

$A_3 = C_1*(0., -1.)$ — 2.1000+000 = (4.4, 2.1)*(0.,-1.)

The mode of the expression is complex. A_3 is type real; the imaginary part of C_1 replaces A_3.

$A_4 = R_1/R_2*(R_3-R_4)+I_1-(I_2*R_5)$ — 13 = 8.4/4.2 * (3.1-2.1) + 14 - (1*2.3)

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

$A_2 = D_1**2*(D_2+(D_3*D_4))$ +$(D_2*D_1*D_2)$	4.9680000000000000000000+001 = 2.0D**2*(3.2D+(4.1D*1.0D)) +(3.2D*2.0D*3.2D)

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

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

The mode of the expression is complex. Since A_5 is type logical, a 1 (true) will replace A_5 if the real part of the evaluated expression is not zero. A zero (false) will replace A_5 if the real part is zero.

3.2 LOGICAL REPLACEMENT

In a logical replacement statement, the replacement variable is type logical and the expression must be a logical expression.

Examples: L is type logical

L(3) = .NOT. A	L(1,3,3) = A(1)
L = B .OR. C	L = X .EQ. Y .AND. Q .GT. P

3.3 MASKING REPLACEMENT

In a masking replacement statement, the replacement variable must be type integer or real, and the expression must be a masking expression. Mode conversion does not occur for mixed real and integer operands.

Examples:

All variables are type real or integer.

```
A(I)    = B .OR. .NOT. C(I)
B       = D .AND. Q
C(I, J) = .NOT. Z(K) .AND. (Q1 .OR. .NOT. Q2)
TEST    = CELESTE .AND. 7HECLIPSE
AB      = D .OR. FUNC (X, T)
```

3.4 MULTIPLE REPLACEMENT

The multiple replacement statement is an extended form which may be used to assign the same value to more than one variable.

$$V_n = V_{n-1} = \ldots = V_2 = V_1 = \text{expression}$$

V_i are simple or subscripted variables; V_1 is subject to the following restrictions:

Replacement Statement: V_1 = EXP

if EXP is an arithmetic expression, V_1 may be any type.

if EXP is a logical expression, V_1 must be a logical variable only.

if EXP is a masking expression, V_1 must be type real or integer variable only.

The remaining n-1 V_i may be variables of any type, and the multiple replacement statement replaces each of the variables $V_2, \ldots, V_n$ with the value of V_1 in a manner analogous to that employed in mixed mode arithmetic statements.

Examples:

A	real	The numbers in the examples represent the evaluations of expressions.
E, F	complex	
G	double	
I	integer	
K	logical	

```
A = G = 3.1415926535897932384626D
        3.1415926535897932384626D→G
        3.141592654                →A
```

I = A = 4.6	4.6 → A	
	4 → I	
A = I = 4.6	4 → I	
	4.0 → A	
I = A = E = (10.2,3.0)	10.2 → E	real
	3.0 → E	imaginary
	10.2 → A	
	10 → I	
F = A = I = E = (13.4,16.2)	13.4 → E	real
	16.2 → E	imaginary
	13 → I	
	13.0 → A	
	13.0 → F	real
	0.0 → F	imaginary
K = I = -14.6	-14 → I	
	(true) 1 → K	
I = K = -14.6	(true) 1 → K	
	1 → I	

TYPE DECLARATIONS AND STORAGE ALLOCATIONS 4

TYPE, DIMENSION, COMMON, EQUIVALENCE, and DATA declarations may appear in the same program in any order. They are non-executable and must appear before the first executable statement.

4.1 TYPE DECLARATIONS

The TYPE declaration provides the compiler with information on the structure of variable and function identifiers. There are five standard variable types (non-standard types are explained in Chapter 5). The type of a variable is declared by one of the following statements:

FORTRAN-63 Statement		Characteristics
TYPE COMPLEX list	2 words/element	Floating point
TYPE DOUBLE list	2 words/element	Floating point
TYPE REAL list	1 word/element	Floating point
TYPE INTEGER list	1 word/element	Integer
TYPE LOGICAL list	1 word/element	Logical (non-dimensioned)
	32 elements/word	Logical (dimensioned)

FORTRAN-IV Alternate Statement Forms

COMPLEX list

DOUBLE PRECISION list

REAL list

INTEGER list

LOGICAL list

A list is a string of identifiers separated by commas. List identifiers may be simple variables, array names, function names, formal parameters, ASSIGN variables, ENTRY names, and so forth. An example of a list is:

A, B1, CAT, D36F, EUPHORIA

Except for 3400, dimensioning in TYPE statements is allowed anywhere in FORTRAN-IV TYPE declarations and TYPE-other declarations. The first variable of each FORTRAN-63 TYPE declaration may not be dimensioned; all others may be dimensioned. Examples:

```
TYPE COMPLEX A, B(10,10), C          FORTRAN-63
COMPLEX A(10,20), B(10,10), C        FORTRAN-IV
REAL E(20), F, G(10)                 FORTRAN-IV
TYPE REAL E, F(20), G(10)            FORTRAN-63
```

The TYPE declaration is non-executable and must precede the first executable statement in a given program. Any number of TYPE declarations may appear in a program section.

If an identifier is declared in two or more TYPE declarations, a compilation diagnostic will occur. An identifier not declared in a TYPE statement will be an integer if the first letter of the identifier is I, J, K, L, M, N; for any other letter, it will be real.

Examples:

```
COMPLEX A(147), RIGGISH, ATILL2
TYPE DOUBLE TEEPEE, B2BAZ(10,10)
REAL EL, CAMINO, REAL, IDE63
TYPE INTEGER QUID, PRO, QUO
TYPE LOGICAL GEORGE6
```

4.2 DIMENSION

Storage may be reserved for arrays by the non-executable statements DIMENSION, TYPE (except 3400), or COMMON. The standard form of the DIMENSION statement is:

DIMENSION V_1, V_2, ..., 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 integer variables. This is explained in section 7.12.

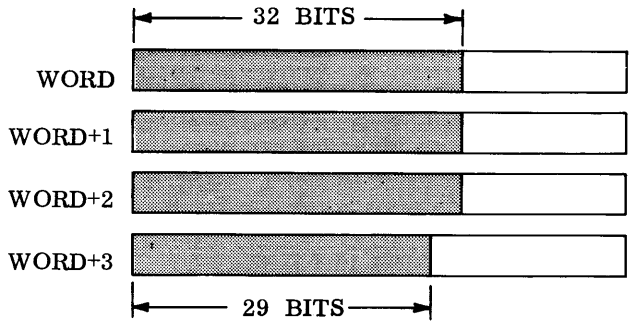
The number of computer words reserved for a given array is determined by the product of the subscripts in the subscript string, and the type of the variable. A maximum of 32767 elements or storage locations may be reserved for any given array. If the maximum is exceeded, a diagnostic is given. In the following statements, the number of elements in the array HERCULES is 200.

```
TYPE COMPLEX HERCULES
DIMENSION HERCULES (10, 20)
```

Two words are used to store a complex element; therefore, the number of computer words reserved is 400. The argument is the same for double precision. For reals and integers the number of words in an array equals the number of elements in the array.

For subscripted logical variables, up to 32 bits of a computer word are used; each bit represents an element of the logical variable array. The elements are stored left to right in a computer word starting with the most significant bit position. In the following statements the 125 elements in the array XERXES will occupy four sequential words as shown below.

```
TYPE LOGICAL XERXES
DIMENSION XERXES (5, 5, 5)
```



VARIABLE DIMENSIONS When an array identifier and its dimensions appear as formal parameters in a function or subroutine, some or all of the dimensions may be assigned through the actual parameter list accompanying the function reference or subroutine call. The dimensions must not exceed the maximum array size specified by the dimensioning statement in the calling program. See section 7.12 for details and examples.

4.3 COMMON

The COMMON statement reserves blocks of storage, numbered or labeled, that can be referenced by more than one subprogram. Only labeled common blocks may be preset; that is, data may be stored in labeled common blocks by the DATA statement and is made available to any subprogram using the appropriate labeled block. The areas of common information are specified by the statement form.

COMMON/I_1/list/I_2/list...

I is a common block identifier, up to 8 characters, which designates either a labeled or numbered common block. An alphabetic first character denotes a labeled common block; the remaining characters may be alphabetic or numeric. If the first character is numeric, the 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 block identifier. The following are common block identifiers:

Labeled	Numbered
AZ13	1
MAXIMUS	146
Z	3600
XRAY	0

List is composed of simple variable identifiers and array identifiers (subscripted or non-subscripted). If a non-subscripted array name appears in the list, the dimensions must be defined by a DIMENSION or TYPE (except 3400) statement in that program.

Arrays may also be dimensioned in the COMMON statement when a subscript string appears with the identifier. An array declared in COMMON may be dimensioned in either a DIMENSION or TYPE (except 3400) statement (not both). A diagnostic results if an array is dimensioned more than once in a DIMENSION and/or TYPE statement. If an array is dimensioned in a COMMON and either a DIMENSION or TYPE statement, the dimensions in the DIMENSION or TYPE statement are used and a warning diagnostic is issued.

The common block identifier with or without the separating slashes may be omitted for blank common. Blank common is treated as numbered common by the compiler. Blank common without separating slashes may immediately follow another blank common statement, but any other multiple use of blank common will cause a diagnostic to be issued.

Any number of COMMON statements may appear in a program section. COMMON is non-executable and must precede the first executable statement in the program, otherwise a diagnostic will occur. If TYPE, DIMENSION or COMMON appear together, the order is immaterial. The following arrangements are equivalent:

```
TYPE DOUBLE A                TYPE DOUBLE A
DIMENSION A (10)             COMMON A
COMMON A                     DIMENSION A (10)

DIMENSION A (10)             TYPE DOUBLE A
TYPE DOUBLE A                COMMON A (10)
COMMON A
```

An identifier (block name or common variable) in one common block may not appear in another common block. If it does the identifier is doubly defined and a diagnostic occurs. However, since labeled common block identifiers are used only by the LOADER at load time, they may be used elsewhere in the program as other kinds of identifiers except as program or subprogram names on the 3400. The following is permissible:

```
COMMON /A/A(10)/B/B(5,5)/C/C(5,5,5)
```

The order of the arrays in a common block is determined by their appearance in a COMMON statement.

At the beginning of the program execution, the contents of common are undefined unless preset using the DATA statement.

Integer variables in COMMON and those used in subscript expressions must be assigned values before the subprogram in which the variables appear is called for execution.

Subscripts which are defined as common area variables should be preset with a DATA statement in the first subprogram which declares the common block. This applies to labeled common only, and only when the subscript variable is used on a multi-subscripted variable. Declaring subscript variables as elements of blank or numbered common may cause undiagnosed errors.

Examples of general COMMON statements:

```
COMMON A, B,C

COMMON // A, B,C,D

COMMON/BLOCK1/ A, B/1234/C(10), D(10, 10), E(10,10, 10)

COMMON/BLOCK A/D (15), F(3,3), GOSH(2, 3, 4), Q1
```

COMMON BLOCK RULES The length of a common block is determined from the number and type of the list identifiers. In the following statement, the length of the common block A is 12 computer words. The origin of the common block is Q(1). (Q and R are real variables and S is complex).

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

	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

The length of the common block must not be changed by the subprograms using the block; declaring different sizes for a common block causes a loader error.

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

Example:

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

The length of TEST is 40 computer words.

The subprogram may rearrange the allocation of words as in:

```
SUBPROG1       COMMON/TEST/A(10), G(10), K(10)
               TYPE 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.

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

```
MAIN PROG      COMMON/SUM/A, B, C
SUB PROG       COMMON/SUM/E, F, G
```

In the above example, only the variables E and G are used in the subprogram. The unused variable F is necessary to space over the area reserved by B.

4.4 EQUIVALENCE

The EQUIVALENCE statement is designed to permit sharing of storage by two or more variables. It should not be used to mathematically equate these variables. The general form is:

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

(A,B, ...) is an equivalence group of two or more simple or singly subscripted variable identifiers. A multi-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))

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

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

Example:

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

The EQUIVALENCE statement 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 as the values of array I will be read into the storage locations previously occupied by A.

EQUIVALENCE is non-executable and must precede the first executable statement in the program or subprogram. If TYPE, DIMENSION, COMMON, or EQUIVALENCE appear together, the order is immaterial.

Any full or multi-word variable, standard or non-standard type, may be made equivalent to any other full or multi-word variable. The variables may be with or without subscript. Any partial word variable, standard logical or non-standard byte, may be made equivalent to any type of partial, full, or multi-word variable. The partial word variable must not be subscripted.

Storage is allocated differently to equivalenced arrays depending on whether the storage area is a common block or not.

If two arrays, not in common, are equivalenced:

```
TYPE INTEGER A, B, C
DIMENSION A(3), B(2), C(4)
EQUIVALENCE (A(3), C(2))
```

storage locations are assigned as follows:

L	A(1)	
L+1	A(2)	C(1)
L+2	A(3)	C(2)
L+3		C(3)
L+4		C(4)
L+5	B(1)	
L+6	B(2)	

However, if two arrays in common are equivalenced:

```
COMMON A(3), B(2), C(4)
EQUIVALENCE (A(3), C(2))
```

storage locations are assigned as follows:

L	A(1)	
L+1	A(2)	C(1)
L+2	A(3)	C(2)
L+3	B(1)	C(3)
L+4	B(2)	C(4)

When equivalenced integer variables are used in subscript expressions, proper address calculation is insured if each equivalenced variable is assigned an explicit value.

Example:

```
      EQUIVALENCE (JP, J)
      DO 1 K = 1, 3
      JP = K + 1
      J = JP
1     IA (I, J) = 1
```

The EQUIVALENCE statement does not rearrange common, but arrays may be defined as equivalent so that the length of the common block is changed. The origin of the common block must not be changed by the EQUIVALENCE statement.

The following simple cases illustrate changes in block lengths caused by the EQUIVALENCE statement.

Given: Arrays A and B

Sa = subscript of A

Sb = subscript of B

CASE I A, B both in COMMON

a) If A appears before B in the COMMON statement:

Sa ≥ Sb is a permissible subscript arrangement

Sa < Sb is not

Block 1

origin →	A(1)		COMMON/1/ A(5), B(7)
	A(2)	B(1)	EQUIVALENCE (A(4), B(3))
	A(3)	B(2)	
	A(4)	B(3)	
	A(5)	B(4)	
		B(5)	
		B(6)	
		B(7)	

Statement EQUIVALENCE (A(3), B(4)) changes the origin of block 1. This is not permitted.

		B(1) ← origin changed
origin →	A(1)	B(2)
	A(2)	B(3)
	A(3)	B(4)
	A(4)	B(5)

b) If B appears before A in the COMMON statement:

Sa ≤ Sb is a permissible subscript arrangement

Sa > Sb is not

CASE II A in COMMON, B not in COMMON (corresponds to CASE Ia)

Sb ≤ Sa is a permissible subscript arrangement

Sb > Sa is not

Block 1

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

CASE III B in COMMON, A not in COMMON (corresponds to CASE Ib)

Sa ≤ Sb is a permissible subscript

Sa > Sb is not

Block 1

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

CASE IV A, B not in COMMON

No subscript arrangement restrictions.

4.5 DATA

The programmer may assign constant values to variables and arrays in the source program by using the DATA statement either by itself or with a dimensioning statement. It may be used to store constant values in variables and arrays contained in a labeled common block.

DATA(I_1 = list), (I_2 = list), ...

I is an identifier representing a simple variable, array name, or a variable with integer constant subscripts or integer variable subscripts (implied DO-loop notation).

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 occurs.

DATA is non-executable and must precede the first executable statement in any program or subprogram in which it appears. When DATA appears with TYPE, DIMENSION, COMMON or EQUIVALENCE statements, the order is immaterial.

DO loop-implying notation is permissible with the restriction that the third indexing parameter, m_3, cannot appear. This notation may be used for storing constant values in arrays.

```
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
```

The order of the DO loop-implying notation is interpreted according to the order of the subscripts regardless of the order in which it is written. For instance,

((V(I, J), J=1, 10), I=1, 5) is interpreted as ((V(I, J), I=1, 5), J= 1, 10)

Variable dimensioned arrays may not be preset in a DATA statement. Violation of this rule causes an assembly error.

The list may contain constants (floating, integer, octal, or Hollerith), either unsigned or signed. Use of . NOT. will cause a compiler diagnostic.

In the DATA statement, the type of the constant stored is determined by the structure of the constant rather than by the identifier 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 identifier. There should be a one-one correspondence between the identifiers and the list. This is particularly important in arrays. 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 stored in location B. 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 contents 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 stored in consecutive locations starting with the first location specified in the DO-loop.

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

The excess values 6. through 10. are stored in locations A through A + 4.

For a logical or a byte size variable, the identifier may only be a simple variable or array name and the constant value in the list must completely fill each computer word.

```
TYPE OTHER5 (/6)A
DIMENSION A(14)
DATA (A= 4142434445464761B, 6263646566676060B) or
DATA (A= 14HJKLMNOP/STUVWX)
```

Use of DATA with a logical variable constitutes a special case, as shown in the following example.

Given:
```
TYPE LOGICAL L
COMMON / NETWORK / L (4, 8)
```

Store the following matrix of logical elements:

$$L = \begin{vmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{vmatrix}$$

Arrays are stored by columns. Elements of logical arrays are stored 32 bits to the word, left to right, left justified with zero fill.

The matrix fits into one computer word as follows:

111 110 101 111 011 010 000 100 101 110 100 0... 0

and its octal equivalent is

7657320456400000

Therefore, the appropriate DATA statement is:

DATA (L = 7657320456400000B)

Examples:

1) DATA (LEDA=15), (CASTOR=16.0), (POLLUX=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 = 77B, -77B, 4(776B, -774B))

ARRAY B	77B
	-77B
	776B
	-774B
	776B
	-774B
	776B
	-774B
	776B
	-774B

4) COMMON /HERA/ C(4)
DATA (C = 3.6, 3(10.5))

ARRAY C	3.6
	10.5
	10.5
	10.5

5) TYPE COMPLEX PROTEUS
DIMENSION PROTEUS (4)
DATA (PROTEUS = 4((1.0, 2.0)))

ARRAY PROTEUS	1.0
	2.0
	1.0
	2.0
	1.0
	2.0
	1.0
	2.0

6) DIMENSION MESSAGE (3)
DATA (MESSAGE = 3HWHO, 2HIS, 6HSYLVIA)

ARRAY MESSAGE	WHO
	IS
	SYLVIA

4.6 BANK STATEMENT FOR 3600 FORTRAN

The 3600 FORTRAN programmer has the option of specifying the banks to which common blocks or subprograms are to be assigned, or of allowing the SCOPE loader to determine the bank storage assignment. If no bank assignment is made, the subprograms and common blocks are assumed to be bank relocatable; the loader places the subprograms in the bank in which they fit most tightly. If the subprograms and common blocks are assigned the same general bank, the loader will place the subprograms in the bank having the largest amount of available memory.

The general form of the statement is:

BANK, (b_1), $identifier_1$, ..., (b_2), $identifier_2$, ...

b may be a bank designator, 0-7, a subprogram name or entry point, or the name of a common block enclosed in slashes, (/block/).

identifier may be a subprogram name, entry point, or a common block name enclosed in slashes.

The BANK statement is ignored by the 3400 FORTRAN compiler.

A program may contain any number of BANK statements. The BANK statement must appear after the PROGRAM, SUBPROGRAM or FUNCTION statement and before the first executable statement. When BANK appears with TYPE, DIMENSION COMMON, DATA or EQUIVALENCE, the order is immaterial.

If b is a bank designator, all identifiers following will be assigned to that specific bank. If b is a name, all succeeding identifiers will be assigned to the same bank as b. The specific bank will be determined by the monitor when the program is loaded or by another BANK statement.

The subprogram names, entry points, and common block names need not be defined or referenced in the same subprogram containing the BANK statement.

Execution time can be decreased by using the BANK statement to assign subprograms and common blocks to the same bank. This eliminates the use of augmented instructions at references to variables located in the same bank as the current operand bank.

```
    .                          ENO     $C
    .                          LDA     C
    .                          ENO     $B
A=B+C          generates       FAD     B
    .                          ENO     $A
    .                          STA     A

    .
    .
    .
BANK, (C), B
    .                          LDA     C
    .                          FAD     B
A=B+C          generates       ENO     $A
    .                          STA     A
    .
```

Bank assignments for 3600 FORTRAN programs may also be made by a SCOPE BANK control card or by a COMPASS BANK pseudo instruction. Details are given in 3600 SCOPE/Reference Manual, Pub. No. 533, and 3600 COMPASS/Reference Manual, Pub. No. 525.

```
IDENT DRIVE
BANK (2), PLUS, /A/,/B/
    .
    .
    .
END
SUBROUTINE PLUS              Subprogram PLUS and common blocks
BANK, (PLUS), /A/,/B/        /A/ and /B/ will be assigned to bank 2.
COMMON /A/.../B/....
    .
    .
    .
```

SCOPE BANK statements may be mixed with FORTRAN BANK statements. Inconsistent BANK statements may give loader errors or may be loaded without any indication of an erroneous result. Bank statements within a FORTRAN program may not include subprogram names containing periods or other special characters normally considered as delimiters. A SCOPE bank card must be used for these names. (See Instant 3600 SCOPE).

Examples:

```
PROGRAM SAM
BANK, (2), SAM, /AL/, (1), LENNY
COMMON /AL/A, B
    .
    .
    .
END
SUBROUTINE LENNY
    .
    .
    .
END
```

Program SAM and common block AL will be loaded into bank 2. Subprogram LENNY will be loaded into bank 1.

```
PROGRAM ONE
BANK, (ONE), /A/,/B/,/C/
COMMON /A/.../B/...
    .
    .
    .
END
SUBROUTINE UNO
BANK, (UNO), /A/
COMMON /A/.../B/.../C/...
    .
    .
    .
END
SUBROUTINE DUO
BANK, (DUO), /B/
COMMON /B/...
    .
    .
    .
END
```

Program ONE, subprograms UNO and DUO, and common blocks /A/,/B/, and /C/ will be assigned to the same bank by the loader.

NON-STANDARD TYPE DECLARATIONS AND EXPRESSIONS 5

FORTRAN allows eight distinct modes of arithmetic. The mode and the size of the operand is fixed for the five standard types - real, integer, double, complex and logical. The routines or instructions required to handle these arithmetic modes are provided with the system. For further details see Appendix E.

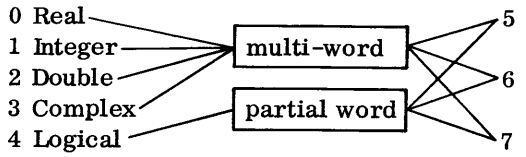
The programmer can define up to three modes of non-standard arithmetic arbitrarily identified as types 5, 6, 7. A non-standard type is arbitrary both in mode and execution and may specify multi-word elements (operands) or partial word elements, called bytes.

The mode and structure of the operand is defined in the TYPE-other declaration. Execution of all expressions containing non-standard variables must be defined in routines supplied by the user (Appendix E).

Non-standard types may be used to introduce a new kind of arithmetic by giving new meaning to the basic arithmetic operators. In a standard arithmetic expression, a + symbol has the fixed interpretation "to add". In a non-standard expression, the programmer may, for example, define + to mean "shift" or "cube".

Non-standard types also may be used to extend precision up to seven computer words or to manipulate bytes in arithmetic operations.

	Standard	Non-Standard
Number of types	5	3
Mode and structure	Fixed	Arbitrary (defined in TYPE-other)
Arithmetic operations	Fixed (defined in system routines)	Arbitrary (defined in user routines)



The steps in solving a non-standard operation are:

1. Define a problem
2. Write and compile a program to solve the problem

   Define non-standard variables in TYPE-other declarations
3. Analyze the calls to subroutines generated by the compiler (Appendix E).
4. Provide subroutines with the calls as entry points; the subroutines will perform the operations desired by the programmer (Appendix E).
5. Compile and execute the program and subroutine (Chapter 11, Deck Structure).

5.1 TYPE-OTHER DECLARATIONS

The TYPE-other declaration provides the compiler with information regarding the structure of the non-standard identifier that names variables and functions.

The general form of a non-standard declaration is:

```
TYPE name#    (/b) list
   or
TYPE name#    (w) list
```

name# is an arbitrary alphanumeric identifier, 2-8 characters. The last character, #, must be one of the type indicators 5, 6, or 7.

(/b) specifies the number of bits in a partial word element. b must be a divisor of 48; if it is not, a compilation diagnostic will be given.

```
TYPE BYTE5 (/6) A          A is a 6-bit element
TYPE PARTS6 (/3) MAX       MAX is a 3-bit element
```

When simple partial word elements are specified, the leftmost b characters of a word are used. When partial word element arrays are specified, the elements are in consecutive locations, left to right, in the word. The number of elements in a word is 48/b.

(w) specifies the number of words in a multi-word element. w must be in the range 1-7; otherwise, a compilation diagnostic will be given.

```
TYPE DOUBLE7 (4) OX        OX is a 4-word element
```

list is a string of simple variable identifiers, or array names, separated by commas. Identifiers have w words per element or b bits per element. Both multi-word elements and partial word element identifiers may be dimensioned in the TYPE-other declaration (except in 3400) or in separate DIMENSION or COMMON statements.

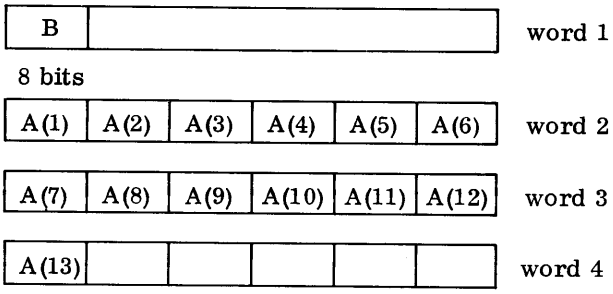
.3600/3800 Example: TYPE BYTE5 (/8)A(10),B

An identifier is doubly defined if it occurs in more than one TYPE-other declaration:

```
TYPE BYTE5 (3)    A,B  }
TYPE BYTE6 (/2)   A, B } This causes a compilation diagnostic.
```

Example:

```
DIMENSION A (13)
TYPE BYTE5 (/8) A, B
```



A program may contain a maximum of three non-standard types (type 5, 6, 7). Any number of TYPE-other declarations with the same name, type, and element length may appear in a program section.

Two or more TYPE-other declarations of the same name and type with multi-word elements of different lengths may appear in the same program.

Examples:

TYPE SAM5 (6) A,B TYPE SAM5 (3) C,D	will compile correctly; the programmer must provide a way to determine element length of variables which are the same type.
TYPE LIEBE6 (6) E,F TYPE LIEBE6 (/5) G,H	will cause a compilation diagnostic; only full word elements may be used.
TYPE PATTI7 (1) M TYPE BARBI7 (3) B	will cause a compilation diagnostic; the name must be the same.

5.2 EVALUATION OF NON-STANDARD ARITHMETIC EXPRESSIONS

The translation of a non-standard arithmetic expression follows the same rules of precedence as for standard arithmetic expressions: exponentiation, multiplication-division, addition-subtraction. The scanning order of the expression is left to right.

The non-standard types (5, 6, 7) may not be mixed within an expression. Non-standard variables of the same type but with different element lengths may be mixed with each other, and any one of the types 5, 6, 7 may be mixed with any of the standard types in arithmetic expressions.

An evaluated expression assumes the mode of the non-standard type variable in the expression.

A non-standard type variable of byte size may not participate in exponentiation unless the exponent is an integer constant 1-8. If A or B or both are non-standard multi-word elements (and B is not an integer constant 1-8), the programmer must provide subroutines for the evaluation of A**B. For exponentiation, if the exponent is an integer constant 1-8, the value is calculated by successive multiplications which may or may not be calculated in a separate subroutine.

Further information on non-standard types in mixed mode arithmetic is given in Chapter 2 under Mixed Mode Arithmetic Expression.

5.3 SAMPLE PROGRAM

The following is a simple example of using non-standard variables in a non-standard arithmetic operation.

Step 1 Define problem:
Add B to A by using a multiply operator, *. Store the value in C and print value in the form: C =

Step 2 Define variables:
A and B are non-standard and are defined in the TYPE-other declaration; C is real:

```
TYPE OTHER5(1) A, B
```

Step 3 Write a FORTRAN program and compile it:

```
      PROGRAM OTHER
2     TYPE REAL C
3     TYPE OTHER5 (1) A, B
4     A=4.1 $ B=5.4 $ C=A*B
5     PRINT 1, C
1     FORMAT (2HC=E14.8)
      END
```

Step 4 Analyze the calls to subroutines generated by the COMPASS assembler:

3400 OBJECT CODE

```
                                    IDENT       OTHER
PROGRAM LENGTH              00027
ENTRY POINTS
                  OTHER     00002
EXTERNAL SYMBOLS
                  Q8QEXITS
                  Q1Q10510
                  Q1Q04550
                  Q1Q00550
                  Q1Q10550
                  Q1Q00510
                   CONVERT
                     THEND
                       STH
                                          PROGRAM          OTHER
00000                       FORMAT.  BSS        2
                                     ENTRY      OTHER
00002                       ENDING.  BSS        0
00002    75   0   P00002   OTHER    SLJ        OTHER
         50   0    00000            ENI        0
00003    75   4   X77777   .4       RTJ        Q1Q00510          Load accumulator
                                                                 with 4.1
         00   0   P00023            00         =02003406314631463
00004    75   4   X77777   +        RTJ        Q1Q10550          Store accumulator
                                                                 in A
         00   0   P00021            00         A
00005    75   4   X00004   +        RTJ        Q1Q00510          Load accumulator
                                                                 with 5.4
         00   0   P00024            00         =02003531463146315
00006    75   4   X00005   +        RTJ        Q1Q10550          Store accumulator
                                                                 in B
         00   0   P00020            00         B
00007    75   4   X77777   +        RTJ        Q1Q00550          Load accumulator
                                                                 with A
         00   0   P00021            00         A
00010    75   4   X77777   +        RTJ        Q1Q04550          Multiply (add) A
                                                                 and B
         00   0   P00020            00         B
00011    75   4   X77777   +        RTJ        Q1Q10510          Store product in C
         00   0   P00022            00         C
```

3400 (cont.)

```
00012    10  0   00075  .5       ENA       +61
         04  0  P00016            ENQ       GG00000.
00013    50  0  P00000  +         50        ..1
         75  4  X77777            RTJ       STH
00014    12  0  P00022  +         LDA       C
         75  4  X77777            RTJ       CONVERT.
00015    75  4  X77777  +         RTJ       THEND.
         50  0   00000
00016                   GG00000.  BSS       0
                P00000            ORGR      FORMAT.
00000    74  0   23023  ..1       BCD       2,(2HC=E14.8)
         13  2   50104
00001    33  1   03460
         60  6   06060
                P00016            ORGR      *
00016    75  4  X77777  EXIT..    RTJ       Q8QEXITS
         50  0   00000
00017    75  0  P00002  +         SLJ       ENDING.
         50  0   00000
00020    00  0   00000  B         OCT       0000000000000000
         00  0   00000
00021    00  0   00000  A         OCT       0000000000000000
         00  0   00000
00022    00  0   00000  C         OCT       0000000000000000
         00  0   00000
                                  EXT       Q8QEXITS
                                  EXT       Q1Q10510
                                  EXT       Q1Q04550
                                  EXT       Q1Q00550
                                  EXT       Q1Q10550
                                  EXT       Q1Q00510
                                  EXT       CONVERT
                                  EXT       THEND
                                  EXT       STH
00023    20  0   34063
         14  0   31463
00024    20  0   35314
         63  1   46315
                                  END       OTHER
         NULLS                        .4              .5      EXIT..
```

(5.1) OTHER 09/13/66

```
                                        IDENT     OTHER
PROGRAM LENGTH                00053
ENTRY POINTS    OTHER         00010
EXTERNAL SYMBOLS
                Q8QENTRY
                Q1Q00510
                Q1Q10550
                Q1Q00550
                Q1Q04550
                Q1Q10510
                THEND.
                Q8QDICT.
                STH.
                QNSINGL.
 00000   63  0 P00000  EXIT.    63        ($)*
         20  0 X77777            20        ($)Q8QDICT.
 00001   00  0  00000  DICT.     OCT       0
         00  0  00000
 00002   46  6  33025            OCT       4663302551606060
         51  6  06060
 00003                  FORMAT.  BSS       5
                                 ENTRY     OTHER
 00010   63  0  00000  OTHER     UBJP      ($)OTHER,,*
         01  0 P00010
 00011   63  0 P00010  +         63        ($)*-1
         20  0 X00000            20        ($)Q8QDICT.
 00012   63  0 P00010            63        ($)*-2
         20  0 P00001            20        ($)DICT.
 00013   63  0  00000            BRTJ      ($)Q8QENTRY,,*
         03  0 X77777
 00014   75  0 P00015            SLJ       *+1
         00  0 P00001            00        DICT.
                                 EXT       Q8QENTRY
 00015                  .4       BSS       0
                                 EXT       Q1Q00510
 00015   63  0  00000            BRTJ      ($)Q1Q00510,,*
         03  0 X77777
```

```
00016   77  1  04000          ENO     *
        12  0 P00051          LDA     =02003406314631463
                               EXT     Q1Q10550
00017   63  0  00000          BRTJ    ($)Q1Q10550,,*
        03  0 X77777
00020   77  1  04000          ENO     *
        20  0 P00044          STA     A
                               EXT     Q1Q00510
00021   63  0  00000          BRTJ    ($)Q1Q00510,,*
        03  0 X00015
00022   77  1  04000          ENO     *
        12  0 P00052          LDA     =02003531463146315
                               EXT     Q1Q10550
00023   63  0  00000          BRTJ    ($)Q1Q10550,,*
        03  0 X00017
00024   77  1  04000          ENO     *
        20  0 P00043          STA     B
                               EXT     Q1Q00550
00025   63  0  00000          BRTJ    ($)Q1Q00550,,*
        03  0 X77777
00026   77  1  04000          ENO     *
        12  0 P00044          LDA     A
                               EXT     Q1Q04550
00027   63  0  00000          BRTJ    ($)Q1Q04550,,*
        03  0 X77777
00030   77  1  04000          ENO     *
        12  0 P00043          LDA     B
                               EXT     Q1Q10510
00031   63  0  00000          BRTJ    ($)Q1Q10510,,*
        03  0 X77777
00032   77  1  04000          ENO     *
        20  0 P00045          STA     C
00033   10  0  00075  .5       ENA     61
        04  0 P00042          ENQ     GG00000.
00034   63  0  00000          BRTJ    ($)STH.,,*
        03  0 X77777
00035   75  0 P00037          SLJ     *+2
        01  0 P00001          01      DICT.
00036   00  0 P00003  +        00      ($)..1
        00  0  00000  -        00      0
00037   12  0 P00045          LDA     C
        75  4 P00047          RTJ     CNVRT1.
00040   63  0  00000          BRTJ    ($)THEND.,,*
        03  0 X77777
00041   75  0 P00042          SLJ     *+1
        00  0 P00001          00      DICT.
                               EXT     THEND.
            P00003  CRFMT.     EQU     FORMAT.
00042               GG00000.   BSS     0
            P00003             ORGR    CRFMT.
00003   77  3  30160  ..1      OCT     7733016060606060
        60  6  06060
00004   30  0  00000          OCT     3000000000001002
        00  0  01002
```

```
00005   23  1  30000          OCT     2313000000000000
        00  0  00000
00006   25  0  00000          OCT     2500000000004016
        00  0  04016
00007   37  0  00000          OCT     3700000000000000
        00  0  00000
               P00042         ORGR    *
00042   75  0 P00050          SLJ     ENDING.
        50  0  00000
00043                    B     BSS     1
00044                    A     BSS     1
00045                    C     BSS     1
                               EXT     Q8QDICT.
                               EXT     STH.
00046   63  0  00000          BRTJ    ($)QNSINGL.,,*
        03  0 X77777
00047   75  0  77777  CNVRT1.  SLJ     **
        75  0 P00046          SLJ     *-1
                               EXT     QNSINGL.
00050   75  0 P00000  ENDING.  SLJ     EXIT.
        50  0  00000
00051   20  0  34063
        14  6  31463
00052   20  0  35314
        63  1  46315
                               END     OTHER
 00030 SYMBOLS
```

Step 5 Provide a subroutine with the external calls as entry points to perform the desired operation:

3400

```
            IDENT   JOE
            ENTRY   Q1Q00510
Q1Q00510    SLJ     **

            LDA     *
+           ARS     24
            INA     -1
+           SAU     *+1

+           LDA     **,7

            SLJ     Q1Q00510
-           ENTRY   Q1Q10550
Q1Q10550    SLJ     **
            STA     TEMP
+           LDA     *-1
            ARS     24
+           INA     -1
            SAL     *+1

+           LDA     TEMP
            STA     **,7
            SLJ     Q1Q10550
            ENTRY   Q1Q00550

Q1Q00550    SLJ     **
            LDA     *
+           ARS     24

            INA     -1
+           SAU     *+1

+           LDA     **,7

            SLJ     Q1Q00550
            ENTRY   Q1Q04550
Q1Q04550    SLJ     **
            STA     TEMP
+           LDA     *-1
            ARS     24
+           INA     -1
            SAL     *+1

+           LDA     TEMP
            FAD     **,7
            SLJ     Q1Q04550
            ENTRY   Q1Q10510
```

3400 (cont.)

```
Q1Q10510  SLJ     **
          STA     TEMP
+         LDA     *-1
          ARS     24
          INA     -1
          SAL     *+1
+         LDA     TEMP
          STA     **,7
          SLJ     Q1Q10510

TEMP      BSS     1

          END
```

3600

```
          IDENT   JOE
          ENTRY   Q1Q00510,Q1Q10550,Q1Q00550,Q1Q04550,Q1Q10510
          REM     ROUTINE TO LOAD REAL CONSTANT AS TYPE 5
    GET   EXEC    **        GO LOAD CONSTANT AND RETURN
Q1Q00510  UBJP    (*)**     ADDR. WILL BE SET FOR RETURN TO MAIN
                              PROGRAM
          LDA     (*)*-1    PICK UP ADDR. STORED BY BRTJ INST.
          SBYT,A0,E19 GET   STORE ADDR, TAG, BANK; IN GET 0-18
          RAO     Q1Q00510  INC. RETURN ADDR. BY ONE
          SLJ     GET       JUMP TO GET
          REM     ROUTINE TO STORE CONSTANT
    PUT   EXEC    **        GO STORE A-REG. AND RETURN
Q1Q10550  UBJP    (*)**
          RXT     A,D       SAVE A-REG.
          RXT     IB,OB     SAVE INST. BANK REG.
          LDQ     Q1Q10550
          RAO     Q1Q10550
          SBYT,Q0,E19 PUT
          RXT     D,A       RESTORE A-REG.
          SLJ     PUT
          REM     ROUTINE TO LOAD TYPE 5 CONSTANT AS TYPE 5
   GET2   EXEC    **
Q1Q00550  UBJP    (*)**
          LDA     (*)*-1
          SBYT,A0,E19 GET2
```

3600 (cont.)

```
                RAO         Q1Q00550
                SLJ         GET2
                REM       ROUTINE TO OBTAIN PRODUCT OF CONSTANTS
Q1Q04550        UBJP        (*)**
                STA         (*) TEMP  SAVE A-REG.
                LDQ         Q1Q04550
                RAO         Q1Q04550
                SBYT,Q0,E19 COMP
     COMP       EXEC        **        OBTAIN SUM
                FAD         (*) TEMP  OF CONSTANTS
                SLJ         Q1Q04550
                REM       ROUTINE TO STORE PRODUCT AS REAL NUMBER
     PUT2       EXEC        **
Q1Q10510        UBJP        (*)**
                RXT         A,D
                RXT         IB,OB
                LDQ         Q1Q10510
                RAO         Q1Q10510
                SBYT,Q0,E19 PUT2
                RXT         D,A
                SLJ         PUT2
     TEMP       DEC         0
                END
```

Step 6 Compile and execute FORTRAN program and COMPASS subprogram.

CONTROL STATEMENTS 6

Program execution normally proceeds from statement to statement as they appear 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 will result in a program error which is usually recognized during assembly. With the DO statement, a predetermined sequence of instructions can be repeated any number of times with the stepping of a simple integer variable after each iteration.

Statements are identified by unsigned numbers, 1 to 99999, which can be referred to from other sections of the program. An identifier up to 5 digits long may occupy any of the first five columns of the coding form; blanks are squeezed out and leading zeros are ignored, 1, 01, 001, 0001, are identical.

6.1 GO TO STATEMENTS

GO TO statements provide transfer of control.

UNCONDITIONAL

GO TO n
This statement causes an unconditional transfer to the statement labeled n; n is a statement identifier.

ASSIGNED

GO TO m, (n_1, n_2, ..., n_m)
This statement acts as a many-branch GO TO. m is a simple integer variable assigned an integer value n_i in a preceding ASSIGN statement. The n_i are statement numbers. Although a parenthetical list need not be present, it should appear when the statement is used in a DO-loop for more efficient object code.

The comma after m is optional when the list is omitted. m cannot be the result of a computation. No compiler diagnostic is given if m is computed, but the object code will be incorrect.

ASSIGN STATEMENT

ASSIGN s TO m

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

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

Control will transfer to statement 10.

COMPUTED

GO TO (n_1, n_2, ..., n_m), E
GO TO (n_1, n_2, ..., n_m) E

This statement acts as a many-branch GO TO where the expression, E, is evaluated prior to its use in the GO TO.

The n_i are statement numbers; E is an arithmetic expression which will be reduced to an integer value, i. If $i \leq 1$, a transfer to n_1 occurs; if $i \geq m$, a transfer to n_m occurs. Otherwise, transfer is to n_i.

Example:

```
      A=1, B=2, C=1
      GO TO (10,20,30), A*B-C
          .
          .
          .
10    A=A+1
      GO TO (11,21,31), A*B-C
```

control will transfer to statement 31.

6.2 IF STATEMENTS

The following IF statements and the status of sense lights and switches provide conditional transfer of control. Masking cannot be done in any IF statement; the masking expression will be interpreted as logical.

THREE BRANCH IF (arithmetic)

IF (A) n_1, n_2, n_3
A is an arithmetic expression and the n_i are statement numbers. This statement tests the evaluated quantity A and jumps accordingly.

A < 0	jump to n_1
A = 0	jump to n_2
A > 0	jump to n_3

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

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

TWO BRANCH IF (logical)

IF (L) n_1, n_2
L is a logical expression. The n_i are statement numbers.

The evaluated expression is tested for true (non-zero) or false (zero). If L is true jump to statement n_1. If L is false jump to statement n_2.

```
IF (A .GT. 16. .OR. I .EQ. 0)5,10
IF (L)1,2                          (L is TYPE LOGICAL)
IF (A*B-C)1,2                      (A*B-C is arithmetic)
IF (A*B/C .LE. 14.32)4,6
```

ONE BRANCH IF (logical)

IF (L) s
L is a logical expression and s is a statement. s must not be a DO, another one branch IF, END, or FORMAT statement. If L is true (non-zero), execute statement s. If L is false (zero), continue in sequence to the statement following the IF logical.

```
IF (L) GO TO 3 (L is logical)
IF (L) Y = SINF (X) /2
```

SENSE LIGHT

SENSE LIGHT i
Execution of this statement turns on the sense light i. SENSE LIGHT 0 turns off all sense lights. i may be a simple integer variable or constant (1 to 48). Sense lights are simulated internally.

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. i is a sense light

and the n_i are statement numbers; i may be a simple integer variable or constant.

```
IF (SENSE LIGHT 4)10,20
```

SENSE SWITCH

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 must be a simple integer variable or constant (1 to 6). Sense switches are simulated as a SCOPE monitor function in the 3400.

```
N = 5
IF (SENSE SWITCH N)5,10
```

6.3 FAULT CONDITIONS

At execute time, the computer is set to interrupt on divide, overflow or exponent fault. The fault condition statements should be placed immediately after any statement for which the check is intended. If not, erroneous indications may be returned.

IF DIVIDE CHECK n_1, n_2
IF DIVIDE FAULT n_1, n_2

The above statements are equivalent. A divide fault occurs following division by zero. The statement checks for this fault; if it has occurred, the indicator is turned off and a jump to statement n_1 takes place. If no fault exists, a jump to statement n_2 takes place.

IF EXPONENT FAULT n_1, n_2

An exponent fault occurs when the result of a real 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 fault has occurred, the indicator is turned off, and a jump to statement n_1 takes place. If no fault exists a jump to statement n_2 takes place.

IF OVERFLOW FAULT n_1, n_2

An overflow fault occurs when the magnitude of the result of an integer sum or difference exceeds $2^{47}-1$. This fault does not occur in division and it is not indicated in multiplication. If the fault occurs, the indicator is turned off and a jump to statement n_1 takes place. If no fault exists, a jump to statement n_2 takes place.

6.4 DO STATEMENT

DO n i = m_1, m_2, m_3

This statement makes it possible to repeat a group of subsequent statements and to change the value of a fixed point variable during the repetition. n is the number of the statement ending the DO loop. A comma separating n and i is permissible. i is the index variable (simple integer). The m_i are the indexing parameters; they may be unsigned non-zero integer constants or simple integer variables. The initial value assigned to i is m_1; m_2 is the largest value assigned to i, and m_3 is the increment added to i after each iteration of the DO loop. If m_3 does not appear, it is assigned the value 1.

The DO statement, the statement labeled n, and any intermediate statements constitute a DO loop. Statement n may not be a GO TO, FORMAT, another DO statement or an IF statement (except IF (L) S). See Transmission of Arrays (section 9.1) and DATA statement (section 4.5) for usage of implied DO loops.

The indexing parameters m_1, m_2, m_3 are either unsigned non-zero integer constants or simple integer variables. Subscripted variables and negative or zero integer constants cause a diagnostic. If the value of the DO variable is to be used outside the DO loop, it must appear as an operand in any statement within the DO loop.

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

The indexing parameter m_3, if variable, should be a positive value.

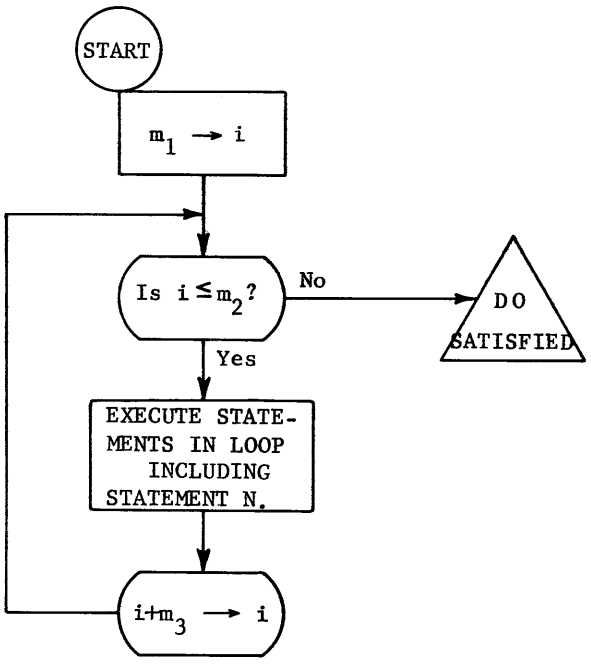
The values of m_2 and m_3 may be changed during the execution of the DO loop.

i is initially m_1. As soon as i exceeds m_2, the loop is terminated.

DO loops may be nested 50 deep.

DO LOOPS

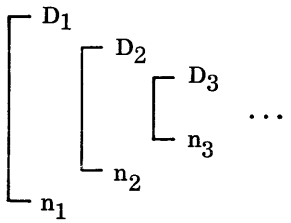
The initial value of i, m_1, is compared with m_2 before executing the DO loop and, if it does not exceed m_2, the loop is executed. After this step, i is increased by m_3. Again i is compared with m_2; this process continues until i exceeds m_2 as shown below. Control then passes to the statement immediately following n, and the DO loop is satisfied. Should m_1 exceed m_2 on the initial entry to the loop, the loop is not executed and control passes to the next statement after n.



When the DO loop is satisfied, the value of 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.

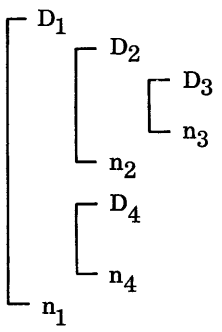
DO NESTS

When a DO loop contains another DO loop, the grouping is called a DO nest. 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 appears before D_3, and n_1, n_2, ..., n_m represent the corresponding limits of the D_i, then n_m must appear before (or coincide with) n_m-1...n_2 must appear before (or coincide with) n_1.

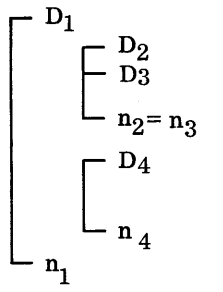


Examples:

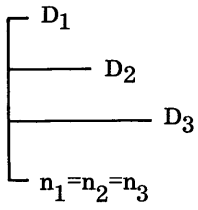
DO loops may be nested in common with other DO loops:




```
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
```



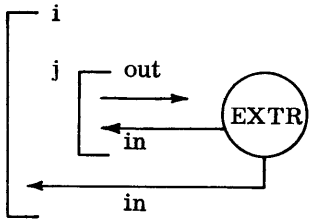

```
DO 5 I = 1,5
DO 5 J=I,10
DO 5 K=J,15
    :
5 CONTINUE
```

DO LOOP TRANSFER

In a DO nest, a transfer may be made from one DO loop into a DO loop that contains it; and a transfer out of a DO nest is permissible.

The special case is transferring out of a nested DO loop and then transferring back to the nest. In a DO nest, if the range of i includes the range of j and a transfer out of the range of j occurs, then a transfer into the range of i or j is permissible.

In the following diagram, EXTR represents a portion of the program outside of the DO nest.



If two or more DO loops terminate at the same statement and a transfer is made to the terminal statement for the outer DO loop, then the inner DO should have its own terminal statement.

Example:

```
      DO 20 ITEM=1,5
      IF (DATA(ITEM)) 11,20
11    STORE=DATA(ITEM)
      DO 10 JOE= 1,6
            .
            .
            .
10    CONTINUE
20    CONTINUE
```

6.5 OTHER CONTROL STATEMENTS

CONTINUE

CONTINUE

This statement is most frequently used as the last statement of a DO loop to provide a loop termination when a GO TO or IF would normally be the last statement of the loop. If CONTINUE is used elsewhere in the source program it acts as a do-nothing instruction and control passes to the next sequential program statement. Alphanumeric characters following a CONTINUE statement are ignored (treated as a comment). Non-alphanumeric characters may cause errors.

PAUSE

PAUSE

PAUSE n

n $\leq$ 5 octal digits without a B suffix. PAUSE n generates a jump to the library subroutine Q8QPAUSE where PAUSE n is typed out on the console typewriter. When OK (for 3400) or any letter, digit, or symbol (for 3600) is typed on the console and the carriage return key is pressed, program execution proceeds with the statement immediately following PAUSE n. PAUSE (n omitted) generates a jump to Q8QPAUSE, but acts as a do-nothing. Although n is octal, a B suffix will cause a diagnostic. In DRUM SCOPE the operator types @on the console and presses the carriage return key to return control to the program.

STOP

STOP

STOP n

n ≤ 5 octal digits without a B suffix. STOP n generates a jump to the library subroutine Q8QSTOP with n typed out on the console typewriter. When OK (on 3400) or any letter, digit or symbol (on 3600) is typed on the console and the carriage return key is pressed, an exit will be made to the SCOPE monitor. STOP (n omitted) generates a jump to Q8QSTOP which causes immediate exit to monitor. A B suffix will cause a diagnostic if used with n. In Drum SCOPE when the operator types @ in the console and presses the carriage return key, an exit will be made to the monitor (3600/3800).

END

END

This statement may include the name of the program or subprogram which it terminates. This name is ignored. However, the name FILE may not be used. END marks the physical end of a program or subprogram. It is executable in the sense that it will effect return from a subprogram in the absence of a RETURN.

PROGRAM, FUNCTION, SUBROUTINE 7

A FORTRAN program consists of a main program with or without subprograms. The main program and subprograms communicate with each other via parameters and common variables and may call or may be called by any other subprogram within the program as long as the calls are non-recursive. That is, if program A calls subprogram B, subprogram B may not call program A. Furthermore, a program or subprogram may not call itself. A calling program is a main program or subprogram that refers to another subprogram.

There are two kinds of subprograms, subroutine and function. In the following discussions, the term subprogram refers to both. Subprograms are compiled independently of the main program.

In addition to multi-statement function subprograms, a function may be defined by a single statement in the program (arithmetic statement function) or may be defined in the compiler (library function).

An arithmetic statement function definition may appear only in a main program or subprogram body and is available only to the subprogram or main program containing it. The meanings of the identifiers appearing in a statement function are the same as those assigned to them in the subprogram. A statement function may contain references to function subprograms, library functions, or other statement functions in the same subprogram.

Library function references may appear in the main program, subprograms, and statement functions.

Main programs (when overlays and segments are used), subprograms, statement functions, and library functions 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.

7.1 PROGRAM AND SUBPROGRAM PARAMETERS

Actual and formal parameters must agree in order, type and number. If they do not agree in type no conversion takes place. For the 3600, parameters must agree in number for the first call of the subprogram only. Missing parameters may only have been simple or subscripted variables, array names or unsigned constants. (See example on page 7-4.) Missing parameters are not allowed on the 3400.

FORMAL PARAMETERS

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

No element of a formal parameter list may appear in a COMMON, DATA, or EQUIVALENCE statement within the subroutine. If it does, a compiler diagnostic results. When a formal parameter represents an array, it should be dimensioned in a DIMENSION or TYPE (except 3400) statement within the subprogram; otherwise, only the first element of the array will be available to the subprogram and the array name must appear without subscripts.

ACTUAL PARAMETERS

Permissible forms:

arithmetic expression	function subprogram name
constant	library function name
simple or subscripted variable	subroutine name
array name	

When an actual parameter is a constant, the corresponding formal parameter should not be used as the object of a replacement or input statement in the called subprogram. This would change the value of the constant in the calling program or subprogram.

Since a function always returns a single value, a function used as an actual parameter may appear as one parameter or two parameters:

Two Parameters

```
FUNCTION PULL (X,Y)
      .
      .
      .
B= X(Y)
      .
      .
      .
```

Function Subprogram Reference

```
      .
      .
      .
EXTERNAL SINF
A= PULL (SINF,X)
      .
      .
      .
```

The function SINF is passed as a parameter.

One Parameter

```
FUNCTION PULL (X)
      .
      .
      .
B= X
      .
      .
      .
```

Function Subprogram Reference

```
      .
      .
      .
A= PULL (SINF(X))
      .
      .
      .
```

The Value of SINF(X) is passed as a parameter.

When a subroutine appears as an actual parameter, the subroutine name may appear alone or with a parameter list. When a subroutine appears with a parameter list, the subroutine name and its parameters must appear as separate actual parameters:

```
FUNCTION PULL (X,Y,Z)
      .
      .
      .
CALL X(Y,Z)
      .
      .
      .
```

Subroutine Subprogram Reference

```
      .
      .
      .
EXTERNAL DIS
A= PULL(DIS,A,B)
      .
      .
      .
```

When an actual parameter is the name of a function or subroutine, that name must also appear in an EXTERNAL statement in the calling program.

Example of missing parameter calls:

Subroutine Subprogram

```
      SUBROUTINE PIP (A, B, C)

      A = B**C
          .
          .
          .
      END
```

Calling Program Reference

```
          .
          .
          .
1     CALL PIP (V(1), X, 3)              parameters must agree in number
          .
          .
          .
2     CALL PIP (V(2), Y,) or (V(2), Y)   implies CALL PIP (V(2), Y, 3)
          .
          .
          .
3     CALL PIP (V(3),, 4)                implies CALL PIP (V(3), Y, 4)
          .
          .
          .
4     CALL PIP (V(4),,) or (V(4))        implies CALL PIP (V(4), Y, 4)
          .
          .
          .
```

7.2 MAIN PROGRAM

The first statement of a main program must be of the following form; name is an alphanumeric identifier, 1-8 characters:

```
PROGRAM name
```

When the PROGRAM statement appears in an overlay or segment, it has the form:

```
PROGRAM name (p1, ..., pn)        1 ≤ n ≤ 59
```

The overlay or segment program is treated as a subroutine except the name becomes the transfer name on the transfer (TRA) card. The formal parameters, p_i, correspond to the actual parameters in an overlay or segment call (section 8.1).

7.3 SUBROUTINE SUBPROGRAM

A subroutine is a computational procedure which may return none, one or more values. No value or type is associated with the name of a subroutine.

The first statement of subroutine subprograms must have the following form; name is an alphanumeric identifier and p_i are formal parameters.

SUBROUTINE name
or
SUBROUTINE name (p_1, ..., p_n) $1 \leq n \leq 63$

The name of the subprogram must not appear in any declarative statement, as an identifier in a replacement statement, in an input/output list, or as an argument of a CALL statement.

7.4 CALL STATEMENT

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

CALL name
or
CALL name (p_1, ..., p_n) $1 \leq n \leq 63$

name is the name of the subroutine being called, and p_i are the actual parameters. The name may not appear in any declarative statement in the calling program with the exception of the EXTERNAL statement when name is also an actual parameter. Name should not be a Library Function Name (Appendix C).

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

Examples:

1) Subroutine Subprogram

```
SUBROUTINE BLVDLDR (A, B, W)
W = 2. *B/A
END
```

Calling Program References

```
CALL BLVDLDR (X(I), Y(I), W)
     .
     .
CALL BLVDLDR (X(I)+H/2., Y(I)+C(1)/2., W)
     .
     .
CALL BLVDLDR (X(I)+H, Y(I)+C(3), Z)
```

2) Subroutine Subprogram (Matrix Multiply)

```
      SUBROUTINE MATMULT
      COMMON/BLK1/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
```

Calling Program References

```
COMMON/BLK1/A(20,20), B(20,20), C(20,20)
     .
     .
CALL MATMULT
     .
     .
```

7.5 FUNCTION SUBPROGRAM

A function is a computational procedure which returns a single 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 have the following form:

type FUNCTION name (p_1, ..., p_n)
or
FUNCTION name (p_1, ..., p_n)

$1 \leq n \leq 63$

Type is REAL, INTEGER, DOUBLE PRECISION, COMPLEX, LOGICAL. When the type indicator is omitted, the mode is determined by the first character of the function name or a TYPE declaration. Name is an alphanumeric identifier and p_i are formal parameters.

The name of a function must not be dimensioned. The name must appear, however, at least once within the function subprogram as any of the following:

the left-hand identifier of a replacement statement

an element of an input list

an actual parameter of a subprogram call

7.6 FUNCTION REFERENCE

name(p_1, ..., p_n) $\qquad 1 \leq n \leq 63$

name identifies the function being referenced; it is an alphanumeric identifier and its type is determined in the same way as a variable identifier. p_i are actual parameters. The name of a function subprogram should not be a Library Function Name (see Appendix C).

A function reference may appear any place in an expression that an operand may be used. The evaluated function will have 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 or END statement in the function subprogram is encountered, control is returned to the statement containing the function reference.

Example (1)

Function Subprogram

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

Calling Program Reference

```
Z(I,J)=F1+F2-GREATER(C-D,3.*I/J)
```

When a variable is used as an index which is subsequently modified in a function subprogram, a statement, variable = variable, must follow the statement containing the function reference to insure proper updating of index functions depending on this variable.

Example (2)

Function Subprogram

```
FUNCTION FUS (C,D)
COMMON L1
    .
    .
    .
L1= L1+2
    .
    .
    .
END
```

Calling Program Reference

```
    .
    .
    .
COMMON L1
W=F(L1,3)
A=FUS(C,D)
L1= L1
W=F(L1,J)
```

7.7 STATEMENT FUNCTION

A statement function is defined by a single expression and applies to the program or subprogram containing the definition.

name $(p_1, \ldots, p_n)$=E $\qquad 1 \leq n \leq 63$

The name of the statement function is an alphanumeric identifier; a single value is associated with the name. The formal parameters p_i must be simple variables. The expression E may contain references to library functions, statement functions, or function subprograms. Formal parameters should appear in the expression. Non-parameter identifiers in the expression have the same values as they have outside the function.

During compilation, the statement function definition is inserted in the code wherever the statement function reference appears as an operand in an expression. A statement function reference has the form:

name $(p_1, \ldots, p_n)$

Name is the name of the statement function; the actual parameters p_i may be any arithmetic expressions. The statement function name must not be dimensioned or appear in an EQUIVALENCE, COMMON or EXTERNAL statement.

All statement functions must precede the first executable statement of the program or subprogram, but they must follow all declarative statements.

Actual and formal parameters must agree only in number and order. The type of the statement function name or formal parameters is ignored; the mode of the evaluated statement function is determined by the expression E evaluated with the actual parameters (chapter 2). Statement function references may not have missing parameters.

Statement function names must not appear as actual or formal parameters. Statement function names may be contained in a parameter list if they appear with parameters, e.g., CALL SUB (ASF (1.,2.),X).

Example: (1)

```
      SUBROUTINE SMAT
      Z(X,Y)=(1.,0.)*EXPF(X)*COSF(Y)+(0.,1.) *EXPF(X)*SINF(Y)
            .
            .
            .
3     X=Z(3.,8.)/R+S
            .
            .
            .
      END
```

This arithmetic statement function computes the complex exponential $Z(X,Y) = e^{x+iy}$. The formal parameters X and Y assume the values 3. and 8. and the statement function expression replaces the statement function reference in statement 3.

```
X=((1.,0.)*EXPF(3.)*COSF(8.)+(0.,1.)*EXPF(3.)*SINF(8.))/R+S
```

Example: (2)

```
      SUBROUTINE EXAM
      XOR (X,Y)=(X.OR.Y). AND..NOT.(X.AND.Y)
            .
            .
            .
      A=XOR(R,S)
            .
            .
            .
      END
```

This masking statement function computes the exclusive OR of the formal parameters X, Y.

7.8 LIBRARY FUNCTIONS

The standard library functions are available as listed in Appendix C. When one of these appears as an operand in an expression, the compiler identifies it as a library function and generates the appropriate calling sequence within the object program.

The modes of the library functions have been established through usage. The compiler recognizes the library functions and associates the established type with the results. The actual parameters must be the type indicated in Appendix C; if they are not, no conversion is done.

For example, in the function identifier LOGF, the first letter, L, would normally cause that function to return an integer result. The compiler recognizes LOGF as a standard library function with the return of a real result.

The FORTRAN IV function name mnemonics that differ from 3400/3600 FORTRAN function names will have an object code reference to the corresponding 3400/3600 FORTRAN library name. For example, SIN and COS will reference SINF and COSF. Consequently, no 3400/3600 function subprogram may have a FORTRAN IV name.

7.9 RETURN AND END

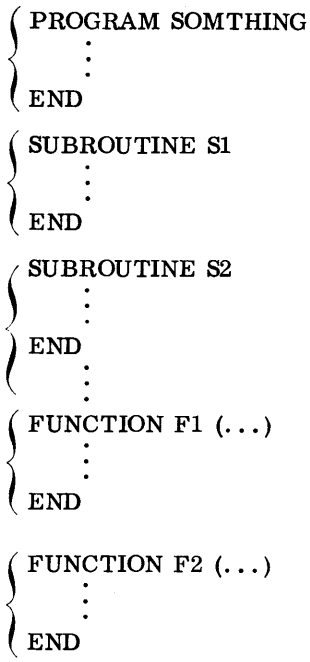
A subprogram normally contains RETURN statements that indicate the end of logic flow within the subprogram and return control to the calling program.

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

The END statement marks the physical end of a program, subroutine subprogram or function subprogram. If the RETURN statement is omitted, END acts as a RETURN.

PROGRAM ARRANGEMENT

FORTRAN assumes that all statements and comments appearing between a PROGRAM, SUBROUTINE or FUNCTION statement and an END statement belong to that subprogram. Comment statements located after END are associated with the subprogram containing the END statement. A typical arrangement of a set of main program and subprograms follows.



7.10 ENTRY STATEMENT

This 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 maximum number of entry points, including the subprogram name, is 20. 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 executable statement range except within a DO; it cannot be labeled.

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.

ENTRY names must agree in type with the function or subroutine name.

Examples:

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

7.11 EXTERNAL STATEMENT

When the actual parameter list of a given function or subroutine reference contains a function or subroutine name, that name must be declared in an EXTERNAL statement.

EXTERNAL identifier$_1$, identifier$_2$, ...

Identifier$_i$ are the names of functions or subroutines. The EXTERNAL statement must precede the first executable statement of any program in which it appears. When it is used, EXTERNAL always appears in the calling program; it must not be used with arithmetic statement functions. If it is, a compiler diagnostic is given.

The EXTERNAL statement is optional in the following case:

In a subprogram, if the function or subroutine name appearing as a parameter has been declared as a formal parameter of that subprogram.

```
PROGRAM TEST
EXTERNAL ONE
CALL TEST2 (ONE)
END

SUBROUTINE TEST2 (FUN)
EXTERNAL FUN                    may be omitted
CALL TEST3 (FUN)
END
```

Examples:

1) Function Subprogram

```
FUNCTION PHI(ALFA,PHI2)
PHI=PHI2(ALFA)
END
```

Calling Program Reference

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

From its call in the main program, the formal parameter ALFA is replaced by Q(K), and the formal parameter PHI2 is replaced by SINF. PHI will be replaced by the value of the sine of Q(K).

2) Function Subprogram

```
FUNCTION PSYCHE (A,B,X)
CALL X
PSYCHE = A/B*2.*(A-B)
END
```

Calling Program Reference

```
EXTERNAL EROS
      .
      .
      .
R=S-PSYCHE (TLIM, ULIM, EROS)
```

In the function subprogram, TLIM, ULIM replaces A,B. The CALL X is a call to a subroutine named EROS. EROS appears in an EXTERNAL statement so that the compiler recognizes it as a subroutine name rather than a variable identifier.

3) Function Subprogram

```
FUNCTION AL(W,X,Y,Z)
CALL W(X,Y,Z)
AL=Z**4
RETURN
END
```

Calling Program Reference

```
EXTERNAL SUM
      .
      .
      .
G=AL(SUM,E,V,H)
```

In the function subprogram the name of the subroutine (SUM) and its parameters (E,V,H) replace W and X,Y,Z. SUM appears in the EXTERNAL statement so that the compiler will treat it as a subroutine name rather than a variable identifier.

4) Subroutine Subprogram

```
      SUBROUTINE ISHTAR (Y,Z)
      COMMON/1/X(100)
      Z=0.
      DO 5 I=1,100
5     Z=Z+X(I)
      CALL Y
      RETURN
      END
```

Calling Program Reference

```
COMMON/1/A(100)
EXTERNAL PRNTIT
      .
      .
      .
CALL ISHTAR (PRNTIT,SUM)
```

7.12 VARIABLE DIMENSIONS

In many subprograms, especially those performing matrix manipulation, the programmer may wish to vary the dimension of the arrays each time the subprogram is called. This is accomplished by specifying the array identifier and its dimensions as formal parameters in the FUNCTION or SUBROUTINE statement heading a subprogram. In the subroutine call from the calling program, the corresponding actual parameters specified in the calling program reference are used by the called subprogram. The maximum dimension that any given array may assume is determined by a dimensioning statement in the calling program at compile time.

The formal parameters representing the array variable dimensions must be simple integer variables. The array identifier must also be a formal parameter. The actual parameters representing the array dimensions may be unsigned non-zero integer constants or integer variables.

If the total number of elements of a given array in the calling program is N, then then the total number of elements of the corresponding array in the subprogram may not exceed N.

Examples:

1) 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)
      RETURN
      END
```

The arrays X, Y, Z and the variable dimensions M, N must all appear as formal parameters in the SUBROUTINE statement and be declared in a DIMENSION or TYPE (except 3400) statement. If the calling program contains the array allocation declaration:

```
      DIMENSION A(10, 10), B(10, 10), C(10, 10),
     1E(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 whether the limits of the array established by the dimensioning statement in the main program are exceeded.

2) $Y = \begin{cases} y_{11} \cdots y_{1n} \\ y_{21} \cdots y_{2n} \\ y_{31} \cdots y_{3n} \\ y_{41} \cdots y_{4n} \end{cases}$

Its transpose Y' is:

$$Y' = \begin{cases} y_{11} & y_{21} & y_{31} & y_{41} \\ \vdots & \vdots & \vdots & \vdots \\ y_{1n} & y_{2n} & y_{3n} & y_{4n} \end{cases}$$

The following FORTRAN program permits variation of N from call to call:

```
      SUBROUTINE MATRAN (Y, YPRIME, N)
      DIMENSION Y(4, N), YPRIME (N, 4)
      DO 7 I=1, N
      DO 7 J=1, 4
    7 YPRIME (I, J)=Y(J, I)
      END
```

OVERLAYS AND SEGMENTS 8

Programs that exceed available memory may be divided into independent parts which may be called and executed as needed. Such programs consist of a main program, overlays of the main program and segments of overlays. The main program may contain subprograms; overlays and segments are one or more subprograms. One subprogram in each overlay and segment must begin with the following statement:

PROGRAM name
or
PROGRAM name (p_1, ..., p_n) $1 \leq n \leq 59$

Name is the transfer address for the overlay or segment, and p_i are formal parameters corresponding to the actual parameters p_i in the call statement.

The main subprogram remains in core storage during execution; overlays and segments are loaded from an overlay tape when called. Only main, one overlay, and one segment may occupy storage at a given time. The main program may call overlays and segments. An overlay may not call another overlay; overlays may only call their associated segments. A segment belonging to an overlay not currently in storage cannot be loaded. Overlays, and segments within an overlay, may be called in any order.

When a call is encountered, control is transferred to a FORTRAN subroutine which has an entry point for overlays, OVERLAY, and an entry point for segments, SEGMENT. The FORTRAN subroutine passes the parameters to LOVER, a SCOPE subroutine, which loads the overlay or segment from the overlay tape.

An overlay or segment, entered via the return jump instruction (3400) or bank return jump instruction (3600) when LOVER is used, should exit with a normal return. The overlay or segment may also exit directly to the calling overlay or main program; however, because of the bookkeeping in LOVER, a new overlay or segment cannot be loaded until the previous one has exited with a normal return.

For information on preparation and format of overlay tapes, and deck structure for processing overlays see 3600 SCOPE/Reference Manual, Pub. No. 60053300.

8.1 FORTRAN CALL

FORTRAN source subprograms use the following call statement to load and execute overlays and segments:

CALL $\left\{\begin{matrix}\text{SEGMENT}\\\text{OVERLAY}\end{matrix}\right\}$ (o,s,u,d,p_1, ..., p_n)

- o — overlay identification number; specified for both segment and overlay. Overlays are numbered sequentially, starting at 1, on each overlay tape. Segments are numbered sequentially, starting at 1, for each overlay.
- s — segment identification number, blank (3600) or zero (3400) if the call references an overlay.
- u — logical unit number of the overlay tape.
- d — dummy parameter which must be present if any actual parameters appear, otherwise it may be blank. d may also be the literal 8H.RECALL. or a variable containing that literal if the last loaded overlay or segment is to be recalled. In recall mode, the overlay or segment is not reloaded; initialization of variables local to the overlay or segment is the programmers responsibility. In RECALL mode, the overlay, segment, and unit parameters should appear.
- p_i — actual parameters, if any, to be passed to the overlay or segment routine; no more than 59 may appear.

If o, s, u, or d is blank, the comma must appear; when they are present, the order is fixed.

8.2 COMPASS CALLING SEQUENCE

This calling sequence is generated during compilation for the CALL statement:

3400

	RTJ	$\left\{\begin{matrix}\text{OVERLAY}\\\text{SEGMENT}\end{matrix}\right\}$
	n	*+m
+	00	o
	00	s
+	00	u
	00	d
+	00	p_1
	:	:
	00	p_n

3600

	BRTJ	($) $\left\{\begin{matrix}\text{SEGMENT}\\\text{OVERLAY}\end{matrix}\right\}$,,*
+	SLJ	*+m
	n	DICT.
+	00	($)o
	00	($)s
+	00	($)u
	00	($)d
+	00	($)$p_1$
	:	:
	00	($)$p_n$

n is the total number of parameters in the FORTRAN call statement (o...p_n)

m is $\left(\frac{n+1}{2}\right)+1$; *+m is the return address

DICT. contains the entry point into the subroutine called and is used by the standard error procedure

The above calling sequence jumps to the OVERLAY or SEGMENT subroutine which passes the parameters o, s, and u to LOVER. LOVER loads the segment or overlay and returns either a loading error code (in the A register) or the transfer address (in the Q register) for the overlay or segment loaded.

If no errors occur during loading, the following call to the transfer address is generated by the SEGMENT or OVERLAY subroutine. Control is then given to the overlay or segment subroutine.

3400

```
      RTJ     name
      n       *+m
+     00      p1
        :       :
      00      pn
```

3600

```
      BRTJ    ($) name,,*
+     SLJ     *+m
      n       DICT.
+     00      p1
        :       :
      00      pn
```

name is the transfer address for the loaded overlay or segment

n is the total number of parameters $p_1 \ldots p_n$.

m, DICT. are defined above

p_i are the actual parameters in the FORTRAN call

Example:

```
PROGRAM SUB2 (X, Y, Z)
    :
CALL SEGMENT (3,2,25,,A,B,C)
```

The transfer address in segment 2 of overlay 3 is SUB2. The call for 3600 FORTRAN to load the segment is:

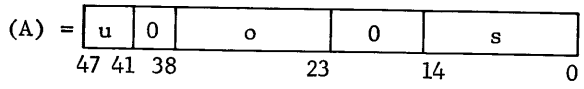
```
BRTJ    ($) SEGMENT,,*
SLJ     *+5
07      DICT.
00      ($)=D3
00      ($)=D2
00      ($)=D25
00      ($)0
00      ($)A
00      ($)B
00      ($)C
nop
```

The call from SEGMENT to the transfer address SUB2 in segment 2 of overlay 3 is:

```
BRTJ    ($) SUB2,,*
SLJ     *+3
03      DICT.
00      ($)A
00      ($)B
00      ($)C
nop
```

8.3 ERRORS

If errors occur during the loading of the overlays or segments, the job is terminated. The A register will contain the contents of the parameters for the last LOVER call specified.



u = logical unit number
o = overlay number
s = segment number

The contents of the A register is written out followed by one of the following messages:

3400	3600
NON-RECOVERABLE PARITY ERROR Non-recoverable parity error encountered when loading overlay or segment record.	READ PARITY ERROR
ILLEGAL LOGICAL UNIT USED Specified logical unit is not 1-49.	LUN OUT OF RANGE
TOO MANY LOGICAL UNITS USED More than four logical units addressed in reading overlay and segment records.	USE OF TOO MANY LUN
RECORD CALLED NOT ON THIS UNIT Overlay or segment in calling sequence not on specified logical unit.	RECORD NOT ON THIS LUN
INCONSISTENT SEQUENCE Overlay or segment in calling sequence not consistent with last overlay or segment loaded.	ILLEGAL SEQUENCE
OUT OF BOUNDS LOAD ATTEMPTED Attempt was made to load an overlay or segment out of bounds.	OUT OF BOUNDS LOAD

FORMAT SPECIFICATIONS 9

Data transmission between storage and external units requires an I/O control statement and for BCD, a FORMAT statement. The I/O statement specifies the input/output device and process--READ, WRITE, and a list of data to be moved. The FORMAT statement specifies the manner in which the data is to be converted, edited and moved. In binary tape statements, no FORMAT statement is used.

9.1 I/O LIST

The list portion of an I/O control statement indicates the data elements and the order, from left to right, of transmission. Elements may be simple variables or array names (subscripted or non-subscripted). They may be unsigned constants on output only. All variables in the list must be standard type. List elements are separated by commas, and the order must correspond to the order of the FORMAT specifications.

Subscripts in an I/O list may be one of the following standard 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.

Non-standard subscripted variables are allowed in an I/O list which does not contain an implied DO loop. (See Subscript Forms, p. 1-6.)

Examples:

```
A, B, H(I), Q(3, 4)                          ((BUZ(K, 2*L), K= 1, 5), L=1, 13, 2)
SPECS                                        Q(3), Z(2, 2), (TUP(3*I-4), I=2, 10)
A, DELTAX(J+ 1)                              (RAZ(K), K= 1, LIM1, LIM2)
3.2, 10, 1.56D0, 5HEND                       ((B, I(J), Z, X(L), J= 1, 5), L= 1, 10)
A(I+ J/K-INTF(SINF(X))), B(A(I))
```

DO-IMPLYING SEGMENTS

A DO-implying segment consists of one or more list elements and indexing values. Multi-dimensioned arrays may appear in the list, with values specified for the range of the subscripts in an implied DO loop. The general form is:

(((A(I, J, K), γ_1=m_1, m_2, m_3), γ_2=n_1, n_2, n_3), γ_3=p_1, p_2, p_3)

m_i, n_i, p_i	unsigned integer constants or predefined positive integer variables. If m_3, n_3 or p_3 is omitted, it is construed as 1.
I, J, K	are subscripts of A and must be of the standard form.
$\gamma_1, \gamma_2, \gamma_3$	I, J, or K; $\gamma_1 \neq \gamma_2 \neq \gamma_3$

During execution, each subscript (index variable) is set to the initial index value: $\gamma_1 = m_1, \gamma_2 = n_1, \gamma_3 = p_1$.

The first index variable defined in the list is incremented first. Data named in the implied DO loops is transmitted in increments according to m_3 until the m_2 is exceeded. If the third parameter is omitted, the increment value is 1. 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 initially exceeds m_2, a card is read but no data is transmitted.

The DO-implying segment replaces a nest of DO loops of the form:

```
      DO    10    γ3 =p1, p2, p3
      DO    10    γ2 =n1, n2, n3
      DO    10    γ1 =m1, m2, m3
      Transmit A(I, J, K)
10    CONTINUE
```

An implied DO loop may also be used to transmit a variable or sequence of variables more than one time. In (A,K= 1,10), A will be transmitted 10 times.

The I/O list may contain nested implied DO loops to a maximum depth of 10.

Examples:

1) DO loops nested 5 deep:

(((((A(I, J, K), B(M), C(N), N=n_1, n_2, n_3), M=m_1, m_2, m_3),
K=k_1, k_2, k_3), J=j_1, j_2, j_3) I=i_1, i_2, i_3)

During execution, each subscript (index variable) is set to the initial index value: I=i_1, J=j_1, K=k_1, M=m_1, N=n_1.

2) To transmit the elements of a 3 by 3 matrix by columns:

```
( (A(I, J), I=1,3), J=1,3)
```

3) To transmit the elements of a 3 by 3 matrix by rows:

```
( (A(I, J), J=1,3), I=1,3)
```

4) (B(J), L, (A(I, L), I= 1, L), J= 3,9,3)

In this input I/O list, L appears before it is used as an index function.

5) (CAT, DOG, RAT, I= 1,10)

The variables CAT, DOG, RAT will be transmitted 10 times. If any of the variables are arrays, the entire array will be transmitted 10 times.

TRANSMITTING ENTIRE ARRAYS

If a dimensioned array name appears in a list without subscripts the entire array is transmitted.

Example:

```
      :
DIMENSION SPECS (7,5,3)
      :
Transmit SPECS
      :
```

transmits the array SPECS as if under control of the nested DO loops:

```
      DO    10    K=1,3
      DO    10    J=1,5
      DO    10    I=1,7
      Transmit SPECS(I, J, K)
10    CONTINUE
```

or as if under control of an implied DO loop;

```
..., ( (SPECS(I, J, K), I=1,7), J=1,5), K=1,3), ...
```

9.2 FORMAT STATEMENT

Each BCD I/O control statement references a FORMAT statement which contains the specifications relating to the internal-external structure of the corresponding I/O list elements.

FORMAT ($spec_1$, ..., k($spec_m$, ...), $spec_n$, ...)

$Spec_i$ are format specifications and k is an optional repetition factor which must be an unsigned integer constant. The FORMAT statement is non-executable, and may appear anywhere in the program.

If the control card option, F, does not appear, the FORMAT statement is checked and converted to an interpretive list at compile time. Any erroneous specifications or improper form in the statement will be diagnosed. FORMAT statements read in at object time are still acceptable (section 9.7).

9.3 CONVERSION SPECIFICATIONS

The data elements in I/O lists are converted from external to internal or from internal to external representation according to FORMAT conversion specifications. FORMAT specifications may also contain editing codes.

Conversion Specifications

Ew.d	Single precision floating point with exponent
Fw.d	Single precision floating point without exponent
Dw.d	Double precision floating point with exponent
C(Zw.d, Zw.d)	Complex conversion; Z may be E or F conversion
Iw	Decimal integer conversion
Ow	Octal integer conversion
Aw	Alphanumeric conversion
Rw	Alphanumeric conversion
Lw	Logical conversion
nP	Scaling factor

Editing specifications

wX	Intra-line spacing
$\left\{ \begin{matrix} *...* \\ wH \end{matrix} \right\}$	Heading and labeling
/	Begin new record

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

Ew.d OUTPUT

E conversion converts floating point numbers in storage to the BCD character form for output. The field occupies w positions in the output record; the corresponding floating point number will appear right justified in the field as

$\Delta\alpha.\ \alpha- - -\ \alpha\pm eee \qquad\qquad 0 \leq eee \leq 308$

$\alpha.\ \alpha---\alpha$ 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 fraction to the right of the decimal do not appear as shown above. Field w must be wide enough to contain significant digits, signs, decimal point, and exponent. Generally, w is greater than or equal to d+7.

If the field is not long enough to contain the output value, asterisks are inserted for the entire field. If the field is longer than the output value, the quantity is right justified with blank fill to the left.

For P-scaling on output, see section 9.5.

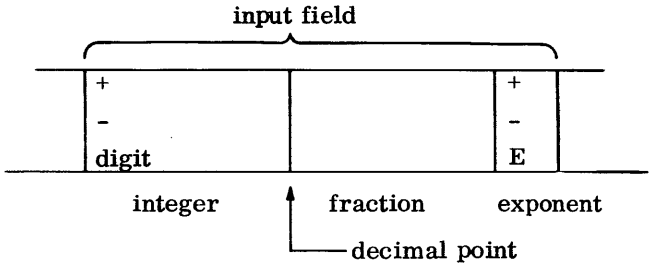
Examples Ew.d Output:

```
    PRINT 10,A                          A contains -67.32
10  FORMAT(E10.3)                             or +67.32
    Result: -6.732+001 or ∧6.732+001

    PRINT 10,A
10  FORMAT(E13.3)
    Result:  ∧∧∧-6.732+001 or     ∧∧∧∧6.732+001

    PRINT 10,A                          A contains -67.32
10  FORMAT(E9.3)
    Result: *********

                                        provision not made for sign
    PRINT 10,A
10  FORMAT(E10.4)
    Result: **********
```

Ew.d INPUT

The E specification converts the number in the input field (specified by w) to a real and stores it in the appropriate storage 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. Leading blanks in the input field are ignored.

An integer subfield begins with a sign (+ or -) or a digit and may contain a string of digits terminated by a decimal point, an E or D, a + or -, or the end of the input field.

A fraction subfield begins with a decimal point and may contain a string of digits terminated by an E or D, a + or -, or the end of the input field.

An exponent subfield may begin with an E or D, a + 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 a string of digits in this subfield must not exceed 308.

Permissible subfield conbinations:

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

In the Ew.d specification, d acts as a negative power of ten scaling factor when the decimal point is not present. The internal representation of the input quantity will be:

(integer subfield) $x10^{-d}x10^{(\text{exponent subfield})}$

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

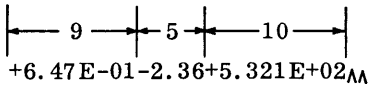
When d does not appear it is assumed to be zero.

If E conversion is specified, but a decimal point occurs in the input constants, the decimal point will override d. The input quantity 3.67294+5 may be read by any E9.d specification but will always be stored as 3.67294×10^5.

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

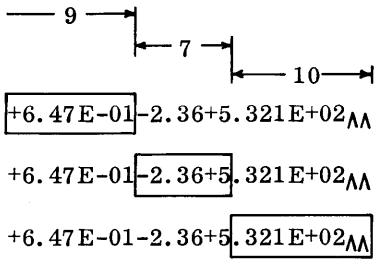
```
      READ 20, A, B, C
20    FORMAT (E9.3, E7.2, E10.3)
```

The input quantities appear on a card in three contiguous field columns 1 through 24:



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

Reading proceeds as follows:



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 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 converted as $8936.x10^{-10+4}$
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 converted as $1.x10^{1}$
E+06	E10.6	0.	No integer or fraction subfield. Zero stored regardless of exponent field contents.
1.∧E∧1	E6.3	10.	Blanks interpreted as zeros.

Fw.d

The field occupies w positions in the output record; the corresponding list element must be a floating point quantity, and it will appear as a decimal number, right justified in the field w, as:

$\Delta\delta\text{---}\delta.\delta\text{---}\delta$

δ represents the most significant digits. 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. On output, w is generally greater than or equal to d+ 3 to allow for the decimal point, at least one digit, and a possible minus sign.

If the field is too short to accommodate the number, asterisks will appear in the output field. If the field w is longer than required, it is right justified with blanks occupying the excess positions to the left.

For F conversion output, the magnitude of the internal number representation after P-scaling must not exceed 10^{14}.

Output Examples:

A contains +32.694

```
      PRINT 10,A
10    FORMAT(F7.3)
      Result: ∧32.694

      PRINT 11,A
11    FORMAT(F10.3)
      Result: ∧∧∧∧32.694
```

A contains -32.694

```
      PRINT 12,A
12    FORMAT (F6.3)
      Result: ******
```

no provision for - sign

Fw.d input is a modification of Ew.d. The input field consists of an integer and a fraction subfield. An omitted subfield is assumed to be zero. All the restrictions for Ew.d input apply. P scaling is permissible.

Input 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 converted as 37925×10^{-7}
-4.7366	F7	-4.7366	No d in specification
.62543	F6.5	.62543	No integer subfield
.62543	F6.d	.62543	Decimal point overrides d
+144.15E-03	F11.2	.14415	Exponents allowed in F input and may have P-scaling
blanks	Fw.d	-0.	All subfields empty

Dw.d

D conversion corresponds to Ew.d input except that 25 (84 binary bits) is the maximum number of significant digits in the combined integer-fraction field; excess digits are read and discarded. P-scaling is not applicable for input.

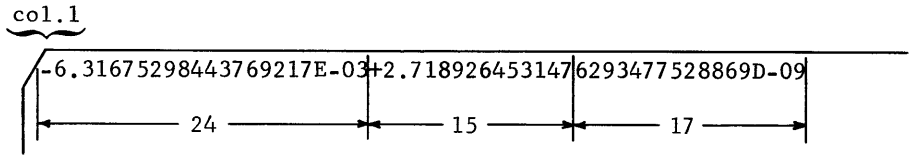
The field occupies w positions of the output record, the corresponding list element which must be a double precision quantity will appear as a decimal number, right justified in the field w as:

$\Delta\alpha.\alpha---\alpha\pm eee$ $\qquad 0 \leq eee \leq 308$

Input Example:

```
      TYPE DOUBLE Z, Y, X
      READ1, Z, Y, X
1     FORMAT (D24.17, D15, D17.4)
```

Input card:



C(Z$_1$w$_1$.d$_1$,Z$_2$w$_2$.d$_2$)

Z is either E or F. The field occupies $w_1 + w_2$ character positions, and the corresponding list element must be complex. w_1 and w_2 are two real values; w_1 represents the real part of the complex number and w_2 represents the imaginary part. The value may be one of the following forms:

$\pm\delta.\delta---\delta$ Exp. $\pm\delta.\delta---\delta$ Exp.	(Ew.d, Ew.d)
$\pm\delta.\delta---\delta$ Exp. $\pm\delta---\delta.\delta---\delta$	(Ew.d, Fw.d)
$\pm\delta---\delta.\delta---\delta$ $\pm\delta.\delta---\delta$ Exp.	(Fw.d, Ew.d)
$\pm\delta---\delta.\delta---\delta$ $\pm\delta---\delta.\delta---\delta$	(Fw.d, Fw.d)

Exp is $\pm e_1e_2e_3$.

The restrictions for Ew.d and Fw.d apply. If spaces are to appear between the two output numbers, the second specification should indicate a field (w_2) larger than required.

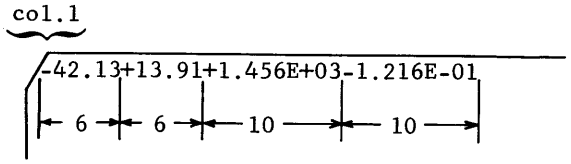
Output Example:

```
      TYPE COMPLEX A                    real part of A is 362.92
      PRINT 10, A                       imaginary part is -46.73
10    FORMAT (C(F7.2, F9.2) )

      Result: ∧362.92∧∧∧-46.73
```

Input Example:

```
      TYPE COMPLEX A, B
      READ 10, A, B
10    FORMAT (C(F6.2, F6.2), C(E10.3, E10.3) )
```

Input card:



Iw

I specification is used to output decimal integer values; and the corresponding list element must be a decimal integer quantity. The output quantity occupies w output record positions right justified in the field w, as:

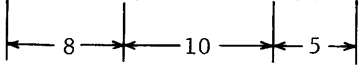
$\triangle\delta$---δ

δ are the most significant decimal digits (maximum 15) of the integer; if positive the + sign is suppressed.

If the field w is larger than required, the output quantity is right justified with blanks occupying excess positions to the left. If the field is too short, asterisks will appear in the output field in 3400 FORTRAN; in 3600 FORTRAN, excess characters are discarded from the left.

Output Example:

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

Result: ∧∧∧-3762∧∧∧4762937∧∧∧13
```

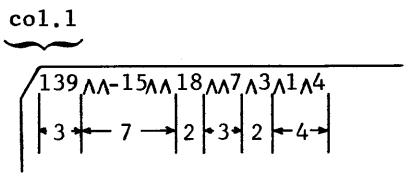


The input field w which consists of an integer subfield may contain only the characters +, -, 0 through 9, or blank. When a sign appears, it must precede the first digit in the field; blanks are interpreted as zeros. The value is stored right-justified in the specified variable. An all blank field is interpreted as -0. The integer subfield may not exceed $2^{47} - 1$.

Input Example:

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

Input card:



I contains 139
J contains -1500
K contains 18
L contains 7
M contains 3
N contains 104

Ow

Octal integer values are converted under O specification. The field is w octal integer characters in length and the corresponding list element should be an integer quantity.

The output, unsigned octal integer values, occupies w output record positions, and the octal digits will appear right justified in the field as: $\delta\ \delta \cdots \delta$

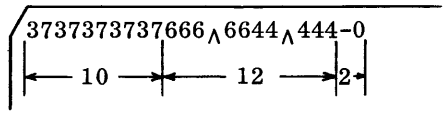
In 3600, if w is 16 or less, the rightmost w digits appear; in 3400, asterisks will appear in the output field. If w is greater than 16, the number is right justified in the field with blanks to the left of the output quantity. A negative number is output in its machine form (one's complement).

The input field w consists of an integer subfield only (maximum of 16 octal digits). In 3600, if w > 16, the first w - 16 characters are ignored and the last 16 characters are read. In 3400, the first w - 16 characters must be blank or an error occurs when w > 16. The only characters that may appear in the field are + or -, blank and 0 through 7. Only one sign is permitted; it must precede the first digit in the field. Blanks are interpreted as zeros. An all blank field is converted as -0.

Input Example:

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

Input Card:



P contains 0000003737373737
Q contains 0000666066440444
R contains 7777777777777777

A negative octal number is represented internally in 16-digit seven's complement form obtained by subtracting each digit of an octal number from seven. For example, if -703 is an input quantity, its internal representation is 7777777777777074. That is,

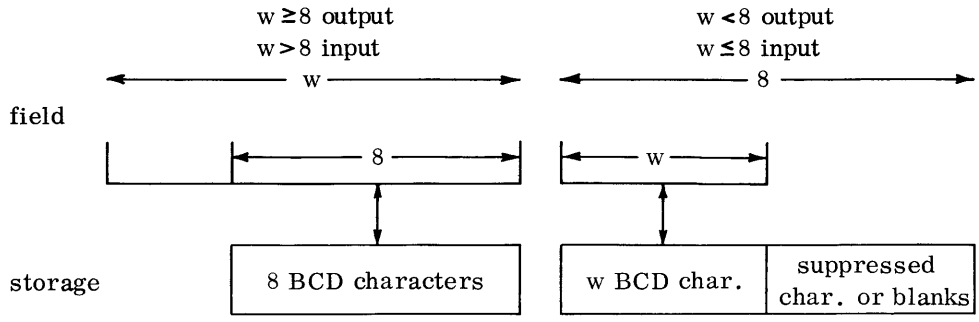
```
 7777777777777777
-0000000000000703
 7777777777777074
```

Aw

This conversion outputs alphanumeric characters. If w is 8 or more, the output quantity appears right justified in the output field, blank fill to left. If w is less than 8, the output quantity represents the leftmost w characters, left justified in the field.

On input, this specification will accept as list elements any set of 6-bit characters including blanks. The internal representation is BCD; the field width is w characters.

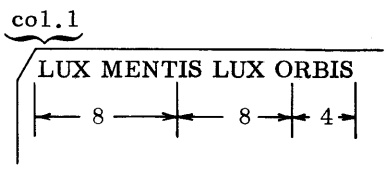
If w exceeds 8, the input quantity will be the rightmost 8 characters. If w is 8 or less, the input quantity goes to the designated storage location as a left justified BCD word, the remaining spaces are blank-filled.



Input Example: (Compare with next example)

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

Input card:

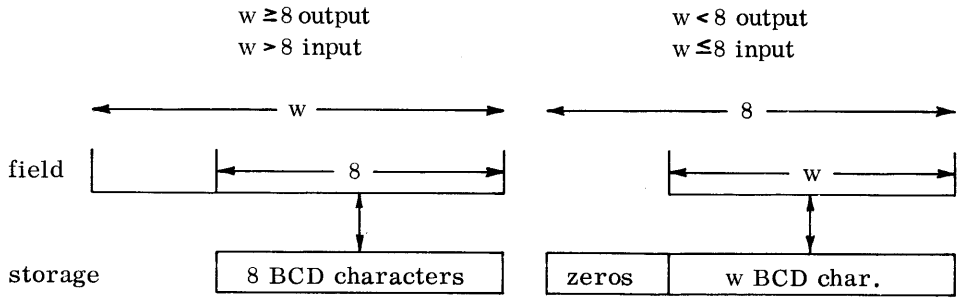


Q contains LUXbMENT
P contains ISbLUXbO
O contains RBISbbbb

Rw This specification is the same as the Aw specification with the following exception.

On output, if w is less than 8, the output quantity represents the rightmost characters.

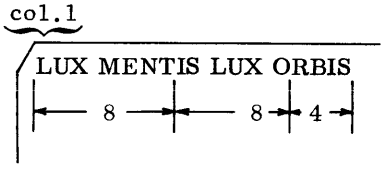
On input, if w is less than 8, the input quantity goes to the designated storage location as a right justified binary zero filled word.



Input Example: (Compare with previous example)

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

Input card:



Q contains LUXbMENT
P contains ISbLUXbO
O contains 0000RBIS

Lw L specification is used to output logical values. The output field is w characters long and the corresponding list element must be a logical element. If w is greater than 1, 1 (for true) or 0 (for false) is printed right justified in the field w with blank fill to the left.

Output Example:

```
      TYPE LOGICAL I, J, K, L      I is true, J is false, K is true,
      PRINT 5, I, J, K, L          L is true
5     FORMAT (4L3)
```

Result: ∧∧^{1}∧∧^{0} ∧∧^{1}∧∧^{1}

Input

For the 3400, this specification accepts logical quantities as list elements. Zero or blank in the field w is false. One in the field w is true. Only one character (0 or 1) may appear in any input field. Any other character is incorrect.

For the 3600, a zero or blank field is false and a non-zero field is true; the value may not exceed $\pm 2^{47}-1$.

9.4 EDITING SPECIFICATIONS

wX

This specification may be used to include w blanks in an output record or to skip w characters on input to permit spacing of input/output quantities. ∧X is interpreted as 1X. 0X is not permitted.

Examples:

```
      PRINT 10, A, B, C                 A contains 7, B contains 13.6,
10    FORMAT(I2, 6X, F6.2, 6X, E12.5)   C contains 1462.37
```

Result: ∧^{7}∧∧∧∧∧∧ $^{13.60}$∧∧∧∧∧∧ $^{1.46237+003}$

```
      READ 11, R, S, T
11    FORMAT(F5.2, 3X, F5.2, 6X, F5.2) or FORMAT (F5.2, 3XF5.2, 6XF5.2)
```

Input card:

```
col. 1
14.62∧∧$13.78∧COST∧15.97
```

R contains 14.62
S contains 13.78
T contains 15.97

In the specification list, the comma following X is optional.

wH

This specification provides for the output of any set of 6-bit characters, including blanks in the form of comments, titles, and headings. w is an unsigned integer specifying the number of characters to the right of the H that will be transmitted to the output record. H denotes a Hollerith field. The comma following the H specification is optional.

Output Examples:

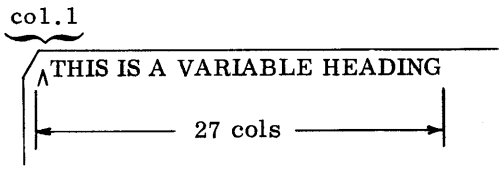
```
Source program:          PRINT 20
                    20   FORMAT(28H BLANKS COUNT IN AN H FIELD.)

produces output record:  ΛBLANKS COUNT IN AN H FIELD.


Source program:          PRINT 30,A                    A contains 1.5
                    30   FORMAT(6H LMAX=,F5.2) comma is optional

produces output record:  ΛLMAX=Λ1.50
```

On input, the H editing specification may be used to read a new heading into an existing H field.

Input Example:

```
Source program:      READ 10
                10   FORMAT (27HΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛΛ)
```

Input card:



After READ, the FORMAT statement labeled 10 will contain the alphanumeric information read from the input card; a subsequent reference to statement 10 in an output control statement would act as follows:

```
PRINT 10
```

produces the printer line: ΛTHIS IS A VARIABLE HEADING

... 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 (*ΛSUBTOTALS*)

produces the output record:  ΛSUBTOTALS
```

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. On the 3600, 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. On the 3400, characters are stored in the heading until all the spaces in the format specification are filled as in the wH specification.

Input Examples:

1) **Source program:**

```
      READ 10
10    FORMAT (*^^^^^^^^^^^^^^^^^^*)
```

Input card:

```
FORTRAN FOR THE 3800
```

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

PRINT 10 produces: FORTRAN FOR THE 3800

2) **Source program:**

```
      READ 10
10    FORMAT (*^^^^^^^*)
```

```
HEAD*LINE
```

PRINT 10 produces: HEAD^^^

NEW RECORD

The slash / which signals the end of a BCD record may occur anywhere in the specifications list. It need not be separated from the other list elements by commas; consecutive slashes may appear in a list. During output, it is used to skip lines, cards, or records. During input, it specifies that control passes to the next record or card. k-1 lines will be skipped for (k(/)).

Examples:

```
      PRINT 10
10    FORMAT (20X, 7HHEADING///6X, 5HINPUT, 19X, 6HOUTPUT)
```

Print-out: HEADING — line 1

line 2

line 3

INPUT OUTPUT — line 4

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

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

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

Result: -1.16+001 ∧∧ 3.25-001 — line 1

46.327-14.261 — line 2

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

Result: -1.16+001 ∧∧ 3.25-001 — line 1

line 2

46.327-14.261 — line 3

```
      PRINT 15, (A(I), I=1, 9)
15    FORMAT (8H RESULTS2(/) (3F8.2) )
```

RESULTS — line 1

line 2

```
 3.62   -4.03   -9.78
-6.33    7.12    3.49
 6.21   -6.74   -1.18
```

line 3
line 4
line 5

9.5 nP SCALE FACTOR

The C, D, E, and F conversion may be preceded by the scale factor: external number = internal number $x10^{\text{scale factor}}$. The scale factor applies to Fw.d on both input and output and to Ew.d and Dw.d on output only. The nP specification may appear with complex conversion. C(Zw.d, Zw.d); each word is scaled separately according to Fw.d or Ew.d scaling. A scaled specification is written as:

nPDw.d nPEw.d nPFw.d nPC(Zw.d, Zw.d)	n is a signed integer constant which cannot exceed 13 for output.

Fw.d SCALING

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

For 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 3.1415926536 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

Ew.d SCALING

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+000
1PE20.2	31.42-001
2PE20.2	314.16-002
3PE20.2	3141.59-003
4PE20.2	31415.93-004
5PE20.2	314159.27-005
-1PE20.2	0.31+001

SCALING RESTRICTIONS The scale factor is assumed to be zero if no other value has been given; however, once a value has been given, it will hold for all D, E and F specifications following the scale factor within the same FORMAT statement. To nullify this effect in subsequent D, E and F specifications, a zero scale factor, 0P, must precede a D, E or F specification. Scale factors for D, E and F output specifications must be in the range $-13 \leq n \leq 13$.

Scale factors on D or E input specifications are ignored.

The scaling specification nP may appear independently of a D, E or F specification, but it will hold for all D, E and F specifications that follow within the same FORMAT statement unless changed by another nP.

(3P, 3I9, F10.2) same as (3I9, 3PF10.2)

9.6 REPEATED FORMAT SPECIFICATIONS

Any FORMAT specification may be repeated by using an unsigned non-zero integer constant repetition factor, k, as follows: k(spec); spec is any specification except nP. The parentheses are optional when spec is a conversion specification. 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 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))

A repeated group, the parenthetical grouping of FORMAT specifications, may be nested to 10 levels.

FORMAT (k_1(...k_2(...k_{10}(...)))))))))))

Therefore, FORMAT statements like the following example are legal.

FORMAT (1H1, 2(25X, 3(5X, F6.2)))

UNLIMITED GROUPS

FORMAT specifications may be repeated without the use of 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. Each repetition of an unlimited group will start a new record. Parentheses are the controlling factors in repetition. The right parentheses of an unlimited group terminates the specifications. Specifications to the right of an unlimited group can never be reached.

The following are format specifications for output data:

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

Print fields according to E16.3 and F20.7. Since 2(I3, F7.1) is a repeated parenthetical group, print fields according to (2I4,2(I3, F7.1)), which does not have a repetition operator, until the last elements are exhausted. F8.2 will never be reached.

9.7 VARIABLE FORMAT

FORMAT lists may be specified at the time of execution. The specification list including left and right parentheses, but not the statement number or the word FORMAT, is defined in a DATA statement or read under the A conversion and stored in an array. The name of the array containing the specifications may be used in place of the FORMAT statement number 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 could be punched in an input card and read by a program such as:

```
      DIMENSION IVAR(4)
      READ 1, (IVAR(I), I=1,4)
1     FORMAT(3A8, A6)
```

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

```
IVAR   : (E12.2,F
IVAR+1 : 8.2,I7,2
IVAR+2 : E20.3,F9
IVAR+3 : .3,I4)^^
```

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

```
    PRINT IVAR(1), A, B, I, C, D, E, J
or
    PRINT IVAR, A, B, I, C, D, E, J
```

This would produce 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 LAIS(4)
DATA (LAIS= 32H(E12.2, F8.2, 2I7)(F8.2, E12.2, 2I7))

Output statements:

```
    I = 1
    PRINT LAIS(I), A, B, I, J
or  PRINT LAIS, A, B, I, J
```

which is the same as:

```
    PRINT 1, A, B, I, J
  1 FORMAT (E12.2, F8.2, 2I7)
```

```
    I = 3
    PRINT LAIS(I), C, D, I, J
```

which is the same as:

```
    PRINT 2, C, D, I, J
  2 FORMAT (F8.2, E12.2, 2I7)
```

Input/output statements control and transfer information between the storage unit and an external device. The particular device used is selected by SCOPE control cards, SCOPE standard usage, or by specific FORTRAN statements (READ, PUNCH, PRINT).

The following definitions for i, n, L apply for all standard I/O control statements.

The logical unit number, i, must be an integer variable or an unsigned, non-zero constant. Logical numbers are assigned to physical units by SCOPE. The standard input unit is 60; standard output unit is 61; standard punch unit is 62. Records on standard input are buffered; records on standard output are blocked.

The FORMAT statement describing the data is represented by n which may be the statement number, a variable identifier (section 9.7) or a formal parameter. Binary data transmission does not require a related FORMAT statement.

The input/output list is specified by L. Binary information is transmitted with odd parity checking bits. BCD information is transmitted with even parity checking bits.

10.1 WRITE STATEMENTS

PRINT n,L transfers information from the storage locations given by the list (L) to the standard output unit. This information is transferred as line printer images, 136 characters or less per line in accordance with the FORMAT statement, n. The maximum record length is 136 characters, but the first character of every record is used for carriage control on the printer and is not printed. (Carriage Control, Appendix A). Characters in excess of the print line are lost; each new record starts a new line. The PRINT statement is equivalent to WRITE (61,n) L.

If the standard output unit (61) is assigned to a magnetic tape, a blocked output record will be written.

PUNCH n,L transfers information from the memory locations given by the list (L) identifiers to the standard punch unit. This information is transferred as Hollerith images, 80 characters or less per card in accordance with the FORMAT statement, n. A maximum of 80 characters are permitted on one card.

WRITE(i,n)L

WRITE OUTPUT TAPE i,n,L

are equivalent forms which transfer information from storage locations given by identifiers in the list (L) to a specified unit (i) according to the FORMAT statement (n); i may be 1 to 59 or 61, 62, 64.

A logical record containing up to 136 characters is recorded on a unit in even parity (BCD mode). Each logical record is one physical record. The number of words in the list (L) and the FORMAT statement (n) determine the number of records that will be written on a unit. If the logical record is less than 136 characters, blanks will be filled to the nearest multiple of 8 characters.

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

Examples:

```
1)      WRITE OUTPUT TAPE 10,20,A,B,C
    20  FORMAT (3F10.6)

2)      TYPE DOUBLE D
        DIMENSION D (4)
        WRITE (10,30) D
    30  FORMAT (4D25.16)

3)      WRITE OUTPUT TAPE 4, 21
    21  FORMAT (33H THIS STATEMENT HAS NO DATA LIST.)
```

WRITE(i)L

WRITE TAPE i,L

are equivalent forms which transfer information from storage locations given by the list (L) identifiers to a specified unit (i); i may be 1 to 59. If the list (L) is omitted, the WRITE (i) statement acts as a do-nothing statement.

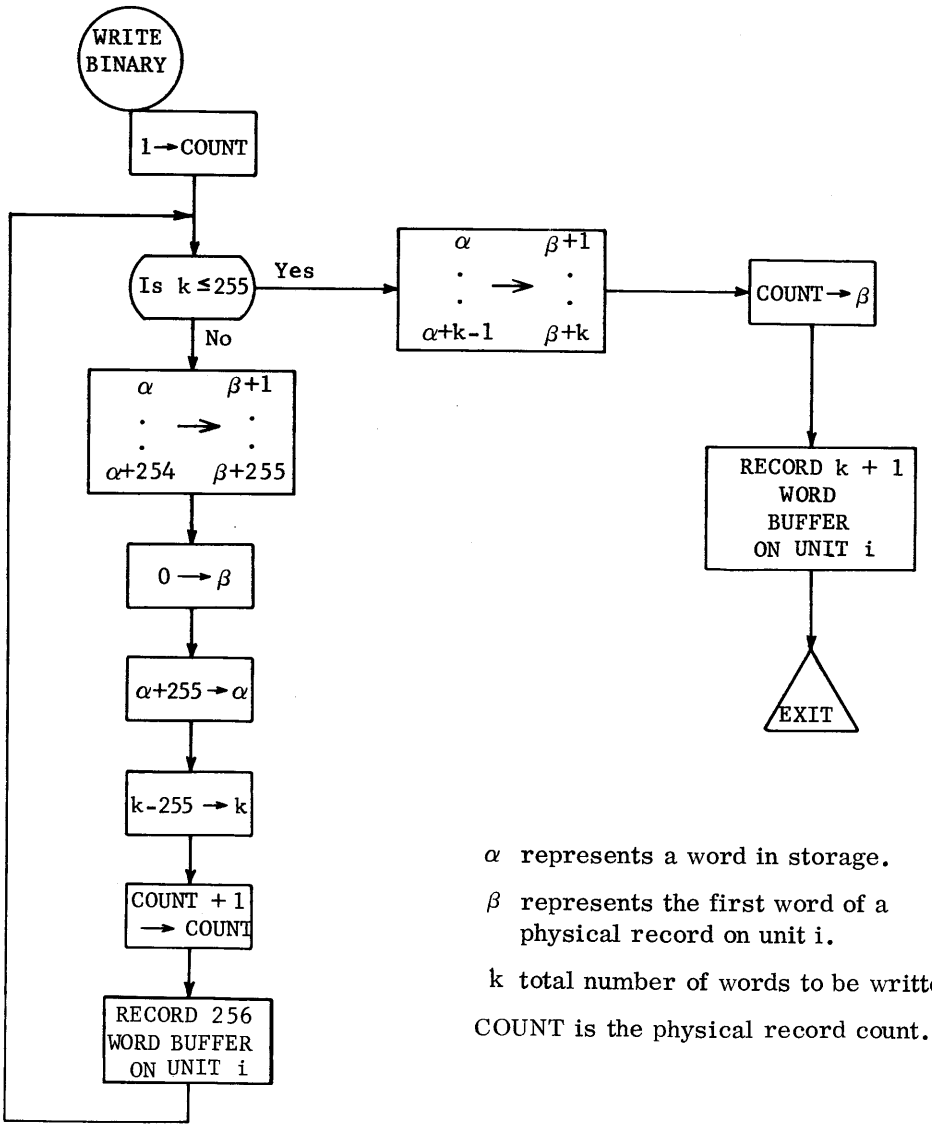
The number of words in the list (L) determines the number of physical records that will be written on that unit. A physical record contains a maximum of 256 words — the first word is a control word. The last physical record may contain from 2 to 256 words. The physical records written by one WRITE (i) L statement constitutes one logical record. The information is recorded in odd parity (binary mode); the method is illustrated in figures 10-1 and 10-2.

If there are n physical records in the logical record, the first word of the first n-1 physical records contain zero; the first word of the nth physical record contains the integer n. This first word indicates how many physical records exist in a logical record. If there is only one physical record in the logical record, the first word contains the integer 1.

Examples:

1) DIMENSION A(260), B(4)
WRITE(10)A, B
writes 1 logical record of 2 physical records

2) DO 5 I = 1, 10
5 WRITE TAPE 6, AMAX(I), (M(I, J), J = 1, 5)
writes 10 logical records each consisting of 1 physical record containing 6 words.



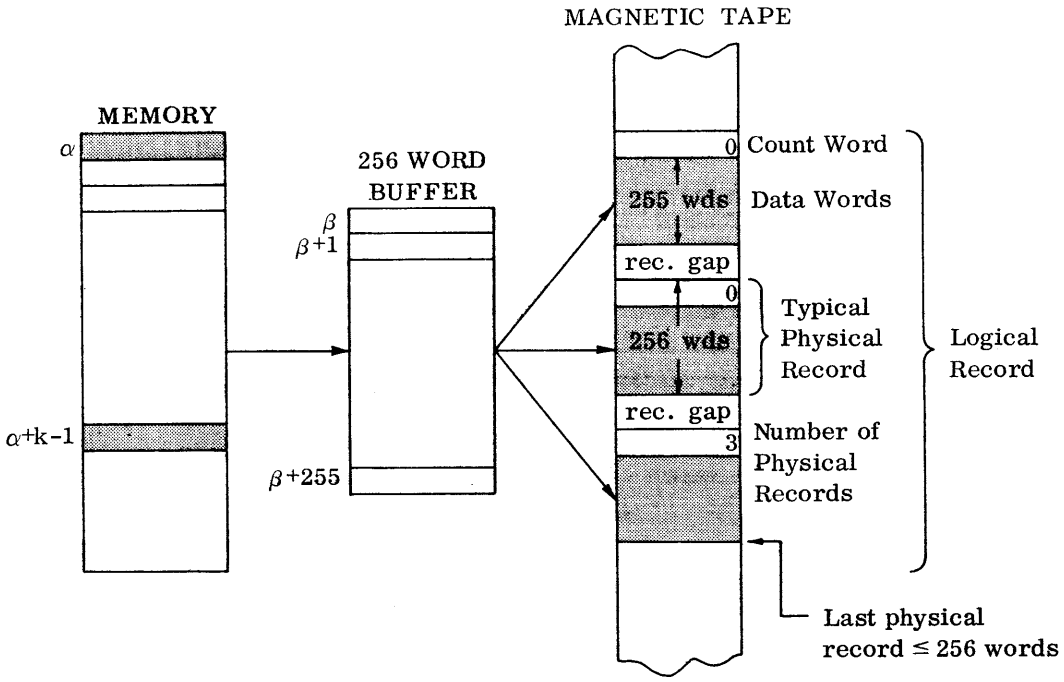
α represents a word in storage.

β represents the first word of a physical record on unit i.

k total number of words to be written.

COUNT is the physical record count.

Figure 10-1. WRITE: BINARY (ODD PARITY) k WORDS



EXAMPLE: Write 520 binary words on tape.

Set count to 1. First 255 words placed in buffer.
More words remain so first buffer word is 0.
Write 256 word physical record on tape.
Bump count 1.

Next 255 words to buffer. Same procedure as above.
Bump count 1.

10 words remain. Transfer to buffer.

Enter count (3) in first buffer word.
Write 11 word physical record on tape.
Exit.

Figure 10-2. WRITE: BINARY (ODD PARITY) k WORDS

10.2 READ STATEMENTS

If an input list and FORMAT specification list specifies more elements per record than can be filled by the input records, the job is terminated. A FORMAT EXCEEDS LINE LENGTH diagnostic is written. When the input unit is a magnetic tape, the unit is buffered ahead ten records (3400), or one record (3600). Buffering ahead stops if a parity error or EOF is detected on the previous record. A read on a bypass unit causes job termination and a diagnostic is given.

READ n,L

reads one or more card images from the standard input unit, converting the information from left to right, in accordance with FORMAT specification (n) and stores the converted data in the storage locations named by the list (L) identifiers. The images read may come from 80-column Hollerith cards, or from magnetic tapes, prepared off-line containing 80-character records in BCD mode. This statement is equivalent to READ (60,n) L.

Example:

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

READ(i,n)L

READ INPUT TAPE i,n,L

are equivalent forms which transfer one logical record of information from a specified logical unit (i), 1 through 60 and 63 to storage locations named by the list (L) identifiers according to FORMAT statement (n).

The number of words in the list and the FORMAT specifications must conform to the record structure on the logical unit (up to 136 characters in the BCD mode). A record read by READ (i,n)L should be the result of a BCD mode WRITE statement. A binary record read in BCD mode will produce a parity error.

Examples:

```
1)      READ INPUT TAPE 10,11,X,Y,Z
    11  FORMAT (3F10.6)

2)      TYPE DOUBLE D2
        DIMENSION D2(4)
        READ (10,12) D2
    12  FORMAT (4D25.16)

3)      READ (2,13) (Z(K),K=1,8)
    13  FORMAT (F10.4)
```

READ(i)L

READ TAPE i,L are equivalent forms which transfer one logical record of information from a specified logical unit (i), 1 through 59, to storage locations named by the list (L) identifiers.

A record read by READ (i) should have been written in binary mode. The count word is not transmitted to the input area, L. The number of elements in the list of READ (i) L must be equal to or less than the number of elements in the corresponding WRITE statement.

If the list (L) is omitted, READ (i) spaces over one logical record.

Caution

If the record read by READ (i) L was written with a BUFFER OUT statement, the first word of each physical record is not transmitted. If the first word is non-zero, the current record will be the last one read with this request; if the first word is zero, an attempt will be made to read another record.

Examples:

```
      DIMENSION C(264)
      READ (10)C

      DIMENSION BMAX (10), M2 (10, 5)
      DO 7 I=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)
```

10.3 BUFFER STATEMENTS

There are three primary differences between the buffer I/O statements and the read/write I/O statements.

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. (Section 10.5). A read/write control statement completes the operation indicated before returning control to the program.

   A magnetic tape written in odd parity must be buffered in odd parity; a tape written in BCD mode must be buffered in even parity. All tape input and output statements will be buffered if sufficient buffer storage is available.

In the descriptions that follow, these definitions apply.

i logical unit number: from 1 to 59 (integer constant or variable). The FORTRAN compiler will not check for reference to units 60-79; however, SCOPE will prevent any incorrect use of units greater than 59. If a unit is referenced by a BUFFER statement, it may not be referenced by a standard I/O statement unless a REWIND or SKIPFILE has occurred between the BUFFER and the standard I/O statement. The 3600 allows the programmer to reference a unit with a standard I/O statement and then a BUFFER statement.

p recording mode (integer constant or variable). For magnetic tapes 0 selects even parity; 1 selects odd parity; 2 selects reverse read even parity; 3 selects reverse read odd parity. †

A variable identifier: first word of data block to be transmitted (fwa).

B variable identifier: last word of data block to be transmitted (lwa).

In the BUFFER statements fwa must be in the same bank (3600) and less than or equal to lwa. If not, the job will be terminated.

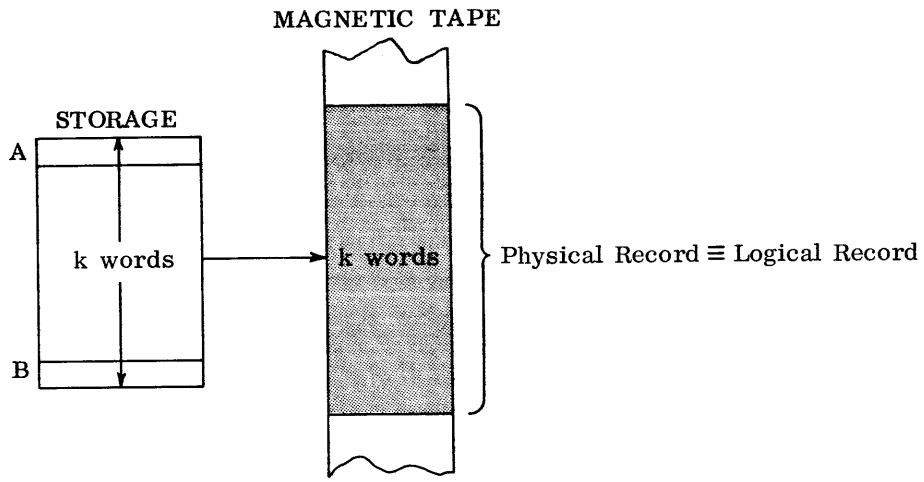
BUFFER IN (i,p) (A,B) transmits information from unit i in mode p to storage locations A through B. The record structure is shown in figure 2. If a unit containing BCD records written by WRITE (i,n) in a previous program is used by BUFFER IN, only one physical record (17 words or less), will be read. When a unit written by WRITE (i) in a previous program is read by BUFFER IN, provision must be made for the count word which is buffered in with the transmitted data. Only one physical record is read for each BUFFER IN statement.

---

† Options 2 and 3 may not be used for standard or non-tape units. Backspacing in the reverse mode is not allowed.

In generating a magnetic tape to be used in the reverse read mode, one word records on output should not be requested since the output routine will always pad the record to two words. Therefore, when a read in reverse mode with a one word request is executed, only the padded word will be transmitted.

BUFFER OUT (i,p) (A,B) transmits information from storage locations A through B, and writes one physical record on logical unit i in mode p. The physical record contains all the words from A to B inclusive (figure 10-3).



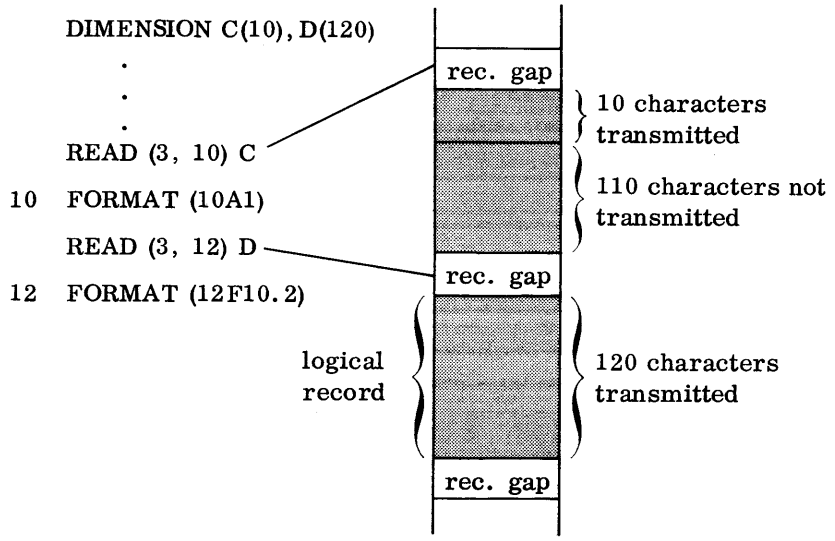
A first word address
B last word address
k total number of words to be written (B-A+1)

Figure 10-3. BUFFERED WRITE: BINARY OR BCD BUFFER OUT (i, p) (A, B)

PARTIAL RECORD

The unit always moves to the next logical record after a READ(i,n) L, READ(i)L, or to the next physical record after a BUFFER IN statement, even if the entire record is not transmitted. Consequently, the remainder of the record will not be read with the next READ or BUFFER IN statement.

Example:



10.4 UNIT HANDLING STATEMENTS

When REWIND or BACKSPACE follows a write operation, an end of file is written and backspaced over before the command is executed. If a read operation is specified on a unit for which the last operation was a write, the job is terminated with the diagnostic W-R SEQ ERROR (unless the console typewriter is referenced). To read from the unit, BACKSPACE or REWIND must be specified after the write operation.

Since programmer and scratch units assigned to the drum simulate magnetic tapes, the following statements operate the same way in Tape SCOPE and Drum SCOPE. However, only one backspace is allowed on unit 60, the standard input unit. The logical unit number, i, may be a simple integer variable or unsigned non-zero constant.

REWIND i — rewinds unit i, 1 to 59, to beginning of information. If unit is already rewound, the statement acts as a do-nothing.

BACKSPACE i — backspaces unit i one logical record. (A logical record is a physical record, except for units written by a WRITE (i) L statement.) If the unit is rewound, this statement acts as a do-nothing. When backspacing on standard units 61 or 62, no more records may be backspaced than have been written.

On standard unit 60, no more records may be backspaced than have been read. i may be 1 through 62. SKIPFILE restrictions are given under Unit Handling Routines. In Drum SCOPE, only one backspace is allowed on unit 60.

ENDFILE i — writes an end-of-file on unit i, 1 through 59.

UNIT HANDLING ROUTINES

BACKFILE and SKIPFILE are FORTRAN library routines which may be called by the user; i may be 1 - 59. BACKSPACE requests are not recommended following the sequence SKIPFILE, BACKSPACE, or SKIPFILE, BACKFILE, if any of the records to be backspaced were written by BUFFER OUT in binary mode (3600) or WRITE(i)L (3400).

CALL BACKFILE (i) — backspaces one file on logical unit i.

CALL SKIPFILE (i) — skips ahead one file on logical unit i.

10.5 STATUS CHECKING STATEMENTS

IF (EOF) and IF (IOCHECK) are the status checking statements to be used with the read/write I/O control statements. If they reference buffered units, the job will terminate abnormally. IF (UNIT) is the only status checking statement that may be used with BUFFER OUT/IN statements.

IF(EOF,i)n_1,n_2 — checks the previous read operation to determine if an end-of-file has been encountered on unit i. If it has, control is transferred to statement n_1; if not, control is transferred to statement n_2.

IF(IOCHECK,i)n_1,n_2 — checks the previous read operation to determine if a parity error has occurred on unit i. If it has, 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 — checks the status of units used in buffered operation. For input, the transfer points are interpreted as follows:

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 operation

n_4 irrecoverable parity error sensed on last operation

IF (UNIT, i)n_1, n_2, n_3

n_1 not ready
n_2 ready and no previous error
n_3 EOF or parity error

IF (UNIT, i)n_1, n_2

n_1 not ready
n_2 ready, EOF or parity error

For output, only n_1 (not ready) and n_2 (ready and no previous error) are used for write operations.

LENGTHF(i) is used with an integer variable, for example I = LENGTHF(i), to find the number of 48-bit words read during the last BUFFER operation on unit i. It should be preceded by an IF(UNIT, i) statement to insure that input/output is completed and there were no errors. LENGTHF (i) will force completion of an operation, but if a unit is referenced on which there were unchecked errors on the last operation, the job will be terminated.

Example:

	PROGRAM	REMARKS
	J=1	Set flag = 1.
	BUFFER IN (10, 0) (A, Z)	Initiate buffered read in even (BCD) parity.
4	IF (UNIT, 10)5, 6, 7, 8	Check status of buffered transfer. Not
5	GO TO (50, 4), J	finished. Do calculations at 50.
50	Computation not involving locations A - Z	
	J=J+1	Calculations complete; increase flag by 1.
	GO TO 4	Go to 4.
7	PRINT 70	
70	FORMAT (12H EOF UNIT 10)	End of file error
	GO TO 200	
8	PRINT 80	
80	FORMAT (21H PARITY ERROR UNIT 10)	
200	REWIND 10	Rewind tape and stop
	STOP	Stop
6	CONTINUE	Buffer transmission complete
	.	Continue program
	.	
	.	

TAPE ERRORS

EOF and parity errors are sensed on input operations:
An EOF error precludes the possibility of a parity error. The actual number of words read may be obtained with the LENGTHF (i) function for buffered I/O. The I/O routine will retry a read five times if errors occur in read operation.

If an input error occurs, the remainder of the record is not read. If an error occurs on a unit and another read or write request is made to that unit before an error check is made, the job will be terminated after issuing the message: HANGING PARITY or EOF ON UNIT N.

On write operations, the I/O routine will attempt to rewrite the record a maximum of five times (3400) or until it succeeds (3600).

10.6 ENCODE/DECODE STATEMENTS

The ENCODE/DECODE statements are comparable to the BCD WRITE/READ statements, with the difference being that no peripheral equipment is involved. Information is transferred under FORMAT specifications from one area of storage to another. Symbols are defined below:

c — Unsigned integer constant or an integer variable (simple or subscripted) specifying the number of characters in the record. c may be an arbitrary number of BCD characters.

The first record within an encoded (decoded) area starts with the leftmost character position specified by V and continues $\frac{c+7}{8}*8$ BCD characters, 8 BCD characters per computer word.
If the specification list (n) translates less than a full record, the remaining characters of the record are ignored for DECODE and blank-filled for ENCODE. The record length is $\frac{c+7}{8}*8$ characters.

Since each succeeding record begins with a new computer word, an integral number of computer words is allocated for each record $\frac{c+7}{8}$ words. The total words allocated for the combined records in one encoded (decoded) area must not exceed program bounds.

n — A statement number, a variable identifier or a formal parameter representing the FORMAT statement.

V — Variable identifier or an array identifier which supplies the starting location of the BCD record. The identifier may have standard or non-standard subscripts.

L — Input/output list.

ENCODE (c,n,V)L

Transmits machine-language elements in a manner similar to PRINT n, L and PUNCH n, L. The information of the list variables, L, is transmitted according to the FORMAT (n) and stored in locations starting at V, $\frac{c+7}{8}$ *8 BCD characters per record. If the I/O list (L) and specification list (n) translate more than $\frac{c+7}{8}$ *8 characters per record, an execution diagnostic occurs: FORMAT EXCEEDS LINE LENGTH (3600) or OUTPUT RECORD OVERFLOW (3400). If the number of characters converted is less than the record length, the remainder of the record is filled with blanks.

DECODE (c,n,V)L

Transmits and edits BCD characters in a manner similar to READ n, L. The information in consecutive records ($\frac{c+7}{8}$*8 BCD characters/record) starting at address V is translated according to the FORMAT (n) and stored in the list variables (L). If the number of characters specified by the I/O list and the specification list (n) is greater than $\frac{c+7}{8}$ *8 characters per record, an execution diagnostic occurs. If DECODE attempts to process a character illegal under a given conversion specification, an execution diagnostic occurs: INVALID CHARACTER ON INPUT (3600) or ILLEGAL CHARACTER, BCD INPUT (3400).

Examples:

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

```
LOC(1) SSSSxxxx
        .
        .
        .
LOC(6) xxxxαααα
       8 bcd ch/wd
```

To form SSSS αααα in storage location NAME:

```
      DECODE(8,1,LOC(6))TEMP
1     FORMAT(4X,A4)
      ENCODE(8,2,NAME) LOC(1),TEMP
2     FORMAT(2A4)
```

The DECODE statement places the last 4 BCD characters of LOC(6) into the first 4 characters of TEMP. The ENCODE statement packs the first 4 characters of LOC(1) and TEMP into NAME.

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

```
      ENCODE (8,1,NAME) LOC(1), LOC(6)
1     FORMAT (A4,R4)
```

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, I). This packed FORMAT is stored in SPECMAT. SPECMAT contains (2A8, Im), a variable FORMAT.

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

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

```
      PROGRAM ENCO2
      I=7RBCDEFGH
      IA=1H1
      ENCODE (7, 10, I)I, IA, I
   10 FORMAT (A2, A1, R4)
      PRINT 11, I
   11 FORMAT (O20)
      END
```

PRINT OUT

22012526273060

The BCD equivalent is

B1EFGHblank

4) In this example, accounting information is to be read from a magnetic tape prepared off-line from 80-column Hollerith card input. Each record on this tape will be 10 words (80 characters) long. The program is to initiate a read, decode the information of this read and initiate a second read while decoding the information obtained from the first read. Two 10-word buffers are used (AIN and CIN). The FORMAT specification in DECODE is

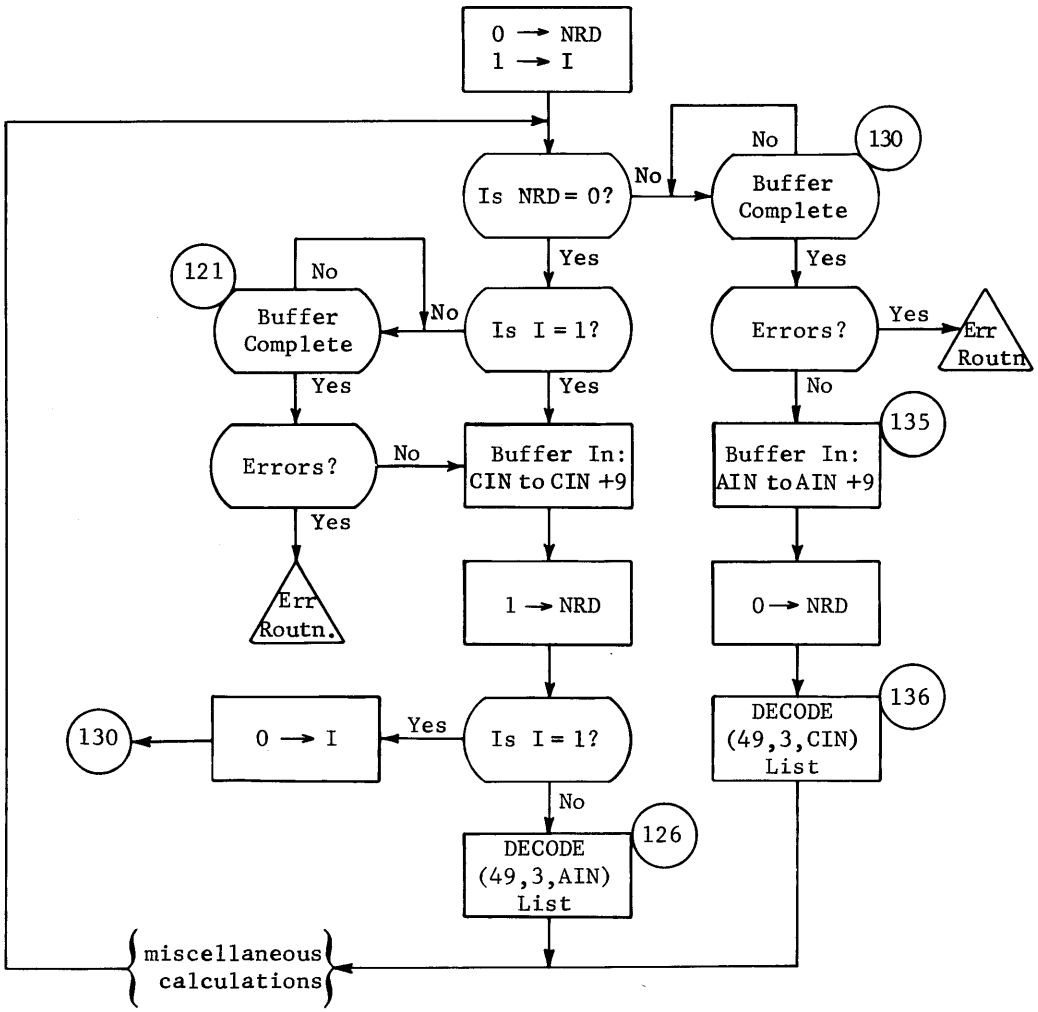
(6A1, A1, 8A1, A3, I2, A6, 4I2, 2A1, A8, A3, 2A1)

this specification breaks the first 49 characters of each BCD record read from magnetic tape. Let the list be the string of identifiers:

LIST: DT, CC, CN, PR, X, XM, N1, M1, N2, M2, CR, ADJ, PER, RUN, ATT

DT is an array of length 6; CN is an array of length 8; the remaining identifiers name simple variables.

Flow chart of the basic procedure:



Examples:

A(1) = ABCDEFGH

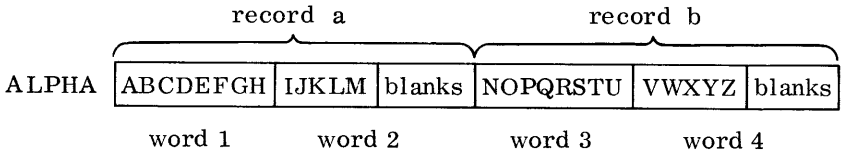
A(2) = IJKLM

B(1) = NOPQRSTU

B(2) = VWXYZ

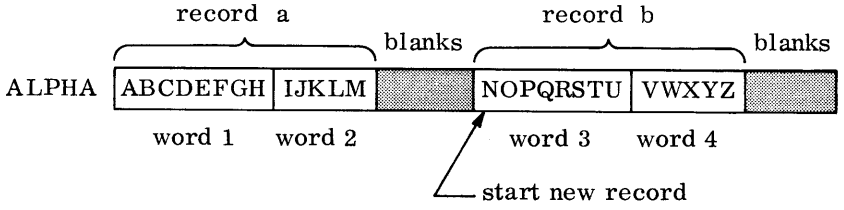
1) c=multiple of 8

```
      ENCODE (16, 10, ALPHA) A, B
   10 FORMAT (A8, A5)
```



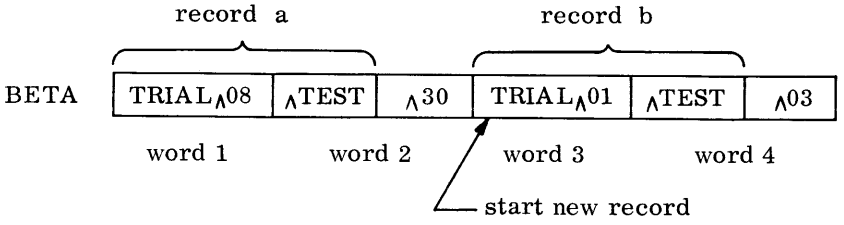
2) c≠multiple of 8

```
      ENCODE (13, 10, ALPHA)A, B
   10 FORMAT (A8, A5)
```



3) c≠multiple of 8

```
      DECODE (13, 10, BETA)A1, B1
   10 FORMAT (A8, A5)
```



A1(1) = TRIAL∧08
A1(2) = ∧TEST
B1(1) = TRIAL∧01
B1(2) = ∧TEST

COMPILATION AND DECK STRUCTURE 11

The SCOPE† monitor system performs the following functions:

loads and links subprograms and library subroutines

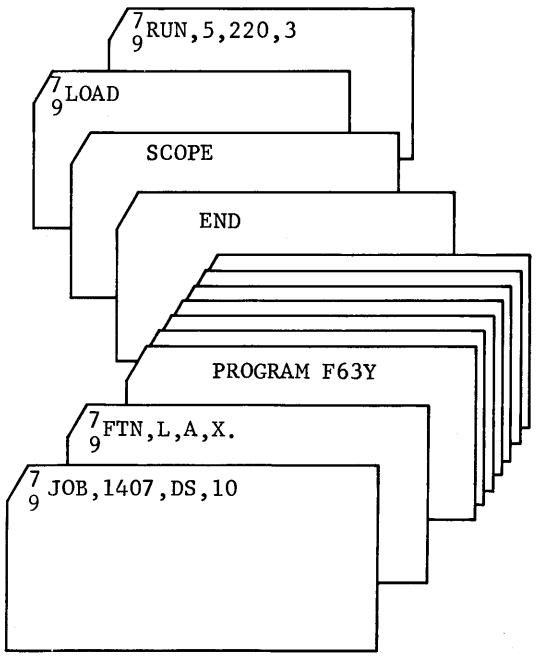
initiates compilation

initiates program execution

transfers records to a logical unit

The arrangement and content of control statements indicate the manner in which a job is to be processed. Jobs and control statements are submitted to SCOPE on the standard input unit; object subprograms and data may be contained on other logical units.

The deck structure for compilation and execution of a 3400/3600 FORTRAN program is:



Execute.

Load binary object program from unit 69.
SCOPE card indicates end of compilation deck; SCOPE begins in column ten.

FORTRAN program terminated by an END card.

Compile, list source language and assembled program and write binary program on logical unit 69.

JOB card.

†More detailed information is contained in SCOPE/Reference Manual.

11.1 CONTROL CARDS

SCOPE control statements contain 7, 9 punches in column 1, followed by statement name and parameters if required, separated by commas. Control statements are free field, but must be contained on a single 80-column card. No terminating character is required; omitted fields must be marked by commas. Blanks are ignored.

JOB CARD

A job consists of control statements and source or object programs. The JOB card defines the beginning of a job and provides installation accounting information, programmer identification, and a job processing time limit.

$^{7}_{9}$JOB, c, i, t

- c Charge number 1 to 6 (3600) or 12 (3400) alphanumeric characters.
- i Programmer identification 1 to 6 (3600) or 12 (3400) alphanumeric characters.
- t Time limit in minutes for the entire job including operator and computer operations. The maximum allowable time, 582 minutes (3400), or 2236 minutes (3600), will be assumed if no time limit is specified. This field may be omitted. In Drum SCOPE time may also be specified in minutes or in the form: minutes. seconds.

Examples:

$^{7}_{9}$JOB, 34 7-00, DS, 10

34 7-00 is charge number; DS is programmer's initials; 10 is job time limit in minutes.

Time limit omitted; maximum time assumed.

$^{7}_{9}$JOB, C234, DD

3600 Drum SCOPE

$^{7}_{9}$JOB(n), c, i, t

- (n) Optional decimal number that indicates how many extra logical units are required by the job in addition to the number allotted by the installation. n may not exceed 75. Each extra logical unit requested by the user requires five locations in bank O. Drum SCOPE uses this information when allotting memory.

FORTRAN CARD

The FORTRAN system is loaded and executed when the FORTRAN card is encountered. The 7, 9 punch in column 1 is followed by FTN which specifies 3400/3600 FORTRAN. All columns are free field.

$^{7}_{9}$FTN, options.

The options defined below may appear in any order separated by commas. The terminating period is optional; the field may also be terminated by the end of the card. Unrecognized options and extraneous characters are ignored. If no options are present, only error messages and the basic assembler headings are printed ($^{7}_{9}$FTN).

An option may be abbreviated to its first character only:

3400 $^{7}_{9}$FTN, L, P, X, A, I, C, R, F.

3600 $^{7}_{9}$FTN, L, P, X, A, I, C, B=n, *, R, F, Q.

An option may be followed by =n; n represents the logical unit number.

$^{7}_{9}$FTN, L, X=10

If n is 0 or not numerically defined, the option is ignored except for Q.

Options:		$n \neq 0$
LIST	List source language program on unit 61	List source language program on unit n, 1-59, 61
PUNCH	Punch relocatable binary deck on unit 62	Output relocatable binary deck on unit n, 1-59, 62
XECUTE	Write load-and-go on tape unit 69	Write load and go on unit n, 1-59, 69
ASSEMBLY	List assembled program on LIST unit. ASSEMBLY unit is always 61 if LIST option does not appear	List assembled program on LIST unit
INPUT	Input source from unit 60; same if option is not present	Input source from n, 1-59, 60
COSY	Punch a COSY deck on unit 62	Output on unit n, 1-59, 62
BCD	Output a COMPASS deck of the compiled assembly code. Unit n must be designated; 1-59, 62	Same as first column
	Compile code for one bank (all variables treated as local). If option not present, general compilation	Compile code for one bank

Options:		$n \neq 0$
REFERENCE	List COMPASS reference table on unit assigned for LIST.	List reference table on unit assigned for LIST.
FORMATS	Do not diagnose or crack FORMAT statements at compile time; if not present, FORMAT statements are cracked.	Do not diagnose or crack FORMAT statement at compile time.
Q-OPTION	Create object code using Q8QRESID for formal parameter substitution; if not present, in-line parameter substitution is used.	Create object code using Q8QRESID for formal parameter substitution.

Examples:

$^{7}_{9}$FTN, INPUT=49, LIST=30, ASSEM.

Source input is on logical unit 49.
Source language will be listed on unit 30.
Assembled program will be listed on unit 30.

$^{7}_{9}$FTN,A,L,X

Source input is on logical unit 60.
Listing of source and assembled program will be on unit 61.
Binary object program will be written on unit 69.

The FORTRAN card is followed by the source language subprograms to be compiled. The source deck may consist of the main program with its subroutine and function subprograms. The program may contain assembly (COMPASS) language subprograms and FORTRAN subprograms in any order.† COMPASS subprograms must not be COSY decks. For each source language subprogram, a binary object program may be produced and stored on the logical unit designated by a parameter on the FORTRAN card.

For the fastest execution of a 3600 FORTRAN program, the * option is specified. This eliminates the use of all augment instructions (ENO and RXT) in the object program, except for parameter references.

FORTRAN source subprograms must begin with a PROGRAM SUBROUTINE, or FUNCTION statement and terminate with an END card. Any COMPASS programs must begin with an IDENT card and end with an END card, both starting in column 10.

† For more detail see COMPASS/FORTRAN Mixed Deck, PSB no. 60137000.

SCOPE COMPILER CARD The SCOPE compiler card is required to indicate the end of the source subprograms. It is not a SCOPE control card, since it is recognized and interpreted by the FORTRAN compiler. The word SCOPE begins in column 10. No imbedded blanks are allowed.

LOAD CARD This statement loads binary object subprograms into storage from programmer and scratch units or the load-and-go unit.

$^{7}_{9}$LOAD, u

- u logical unit number from 1 to 59, 62 or 69. If u is blank, the load-and-go unit (69) is assumed (the comma is also omitted).

Example:

$^{7}_{9}$LOAD, 42 Subprograms are loaded from logical unit 42.

RUN CARD The RUN statement initiates execution by transferring control to the object program in storage. Execution time limit, print request limit, recovery indicator, and memory map suppression indicator are specified. The RUN statement is required to execute all object programs.

$^{7}_{9}$RUN, t, p, r, m

- t execution time limit in minutes. The entire job is terminated if the time limit is exceeded. If t is zero, a memory map is written but execution does not occur. The maximum limit is 582 minutes (3400) or 2236 minutes (3600). If the run time limit exceeds the remaining job time, execution terminates when the job time is depleted.
- p maximum number of print or write operations which may be requested on the standard output unit during the execution. This includes debugging dumps and any other execution output. The entire job is terminated if the print limit is exceeded. If blank, the print limit is determined by each installation.
- r recovery indicator which specifies an area to be dumped if the program does not run to normal termination.

r	dumped area, written on standard output unit
0 or blank	console
1	program
2	labeled common
3	program and labeled common
4	numbered common
5	console + program and numbered common
6	labeled and numbered common
7	all locations including SCOPE (3400); all locations in all banks except those occupied by SCOPE (3600)

m suppresses memory map. If m is blank, a listing of memory allocations after loading will be written on the standard output unit. No map is written if m is any other character.

Example:

$^{7}_{9}$RUN, 28,3000

Execution time limit is 28 minutes, and 3000 is the maximum number of print requests. The console dump, if the job is terminated abnormally, and the memory map will be written on the standard output unit.

11.2 EXAMPLES

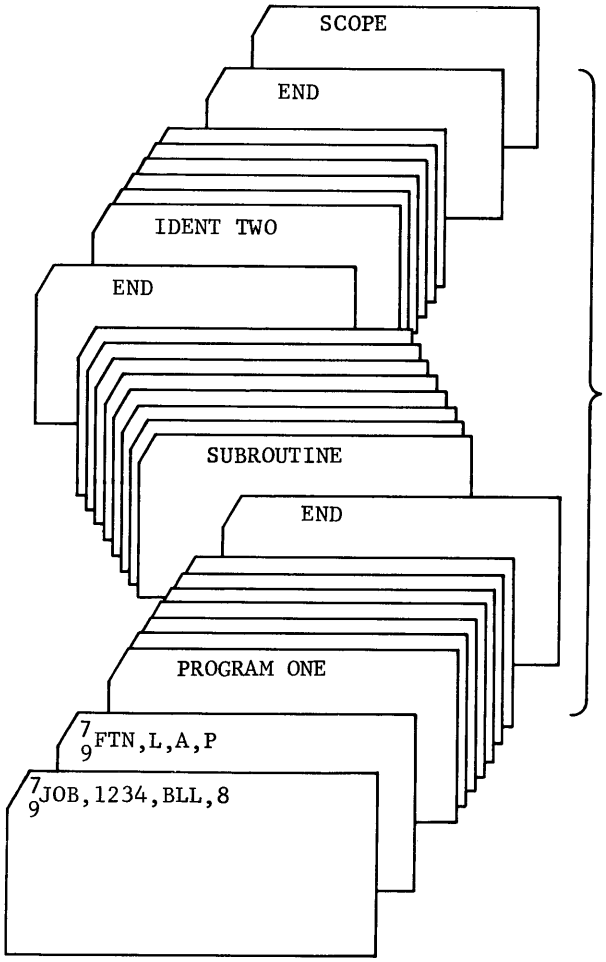
The examples illustrate the deck arrangement for compilation and execution of FORTRAN programs.

COMPILATION ONLY

Compile a FORTRAN program or subprogram.

```
7/9JOB,c,i,t
7/9FTN,options
source subprogram
    .
    .
    .
END
source subprogram
    .
    .
    .
END
SCOPE
```

A subprogram may be a program, subroutine, or function subprogram and must be terminated with an END card. Any number of COMPASS and/or FORTRAN subprograms may follow the FTN card, and they may appear in any order. A COMPASS card is not required to compile COMPASS subprograms. The END card of one subprogram is followed immediately by the first card of the next subprogram (PROGRAM, SUBROUTINE, FUNCTION or an IDENT (COMPASS) statement). Decks of subprograms to be compiled must terminate with a SCOPE card.



Signals end of compilation, begins in column 10.

IDENT and END cards begin in column 10.

Two FORTRAN source language subprograms and one COMPASS subprogram, each terminated by an END card.

Source input on unit 60. Listing of source and assembled program on unit 61. Punch relocatable binary deck on unit 62.

Charge number: 1234. Programmer: BLL. Time limit: 8 minutes.

EXECUTION ONLY

Execute directly from standard input unit.

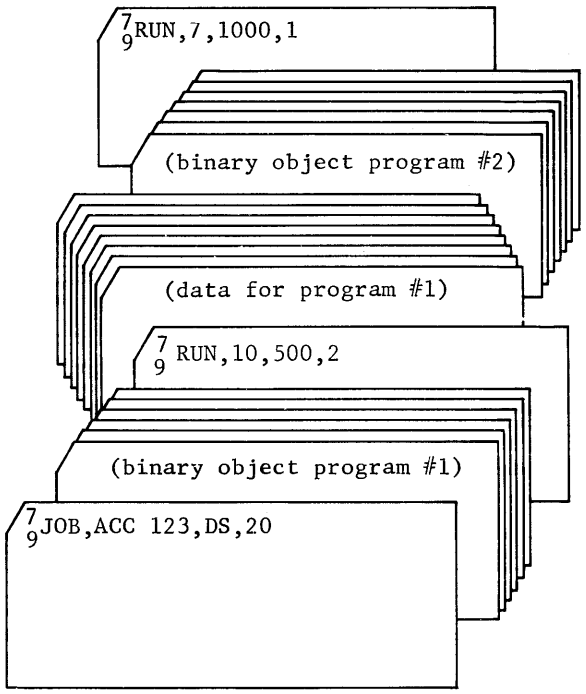
7/9JOB,c,i,t

binary object program

7/9RUN,t,p,r,m

data

Data to be included on the standard input unit must follow the RUN card. Several executions may be performed in the same job.



Time limit: 7 minutes. If terminated abnormally, dump is on standard output unit, map on standard output unit.

Data for object program follows RUN card on standard input unit.

Execute first object program. Time: 10 minutes. Print limit: 500. Dump labeled common on standard output unit for abnormal termination, memory map on standard output unit.

Charge number: ACC 123.
Programmer: DS
Job Time limit: 20 minutes.

Execute from a programmer unit.

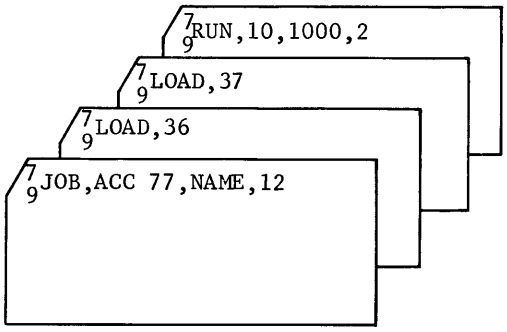
7/9JOB,c,i,t

7/9LOAD,u

7/9RUN,t,p,r,m

data

Subprograms may be loaded from different programmer units. If data is to be included on the standard input unit, it must follow the RUN card.



Execute time does not include load time.

Load binary subprograms from logical units 36 and 37. If there are two TRA cards, second must be last record on unit 37.

Job time: 12 minutes.

COMPILATION AND EXECUTION

Compile a FORTRAN program and execute it immediately.

7/9 JOB,c,i,t

7/9 FTN,options

source program (FORTRAN and/or COMPASS)

SCOPE

7/9 LOAD,u

7/9 RUN,t,p,r,m

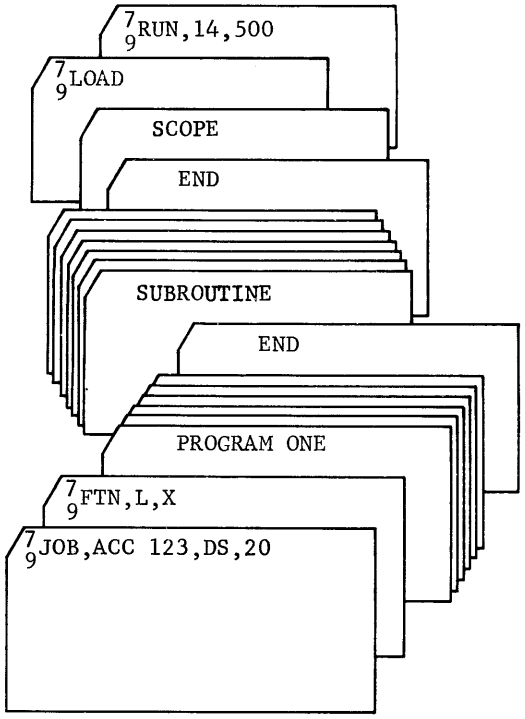
data

If the data is to be included on the standard input unit, it must follow the RUN card. For compilation only, any number of subprograms may follow the FTN card. The deck of subprograms to be compiled must terminate with a SCOPE card.

3600 Drum SCOPE only

Several runs can be included in one job, but they must be separated by EOF cards. Compilation errors prevent execution of succeeding run only, not the entire job.

7/9 EOF
7/9 RUN,t,p,r,m
data



Time: 14 minutes. Recovery, dump and map on standard output unit.
Load binary subprograms from unit 69 until end-of-file.
End of compilation; begins in column 10.

FORTRAN source language subprograms, each terminated by an END card.

Source input on unit 60; list source on 61; write object program on 69

Charge number: ACC 123.
Programmer: DS.
Time: 20 minutes.

Include binary decks.

7/9JOB,c,i,t

7/9FTN, options

source subprogram (3400/3600 FORTRAN and/or 3400/3600 COMPASS)

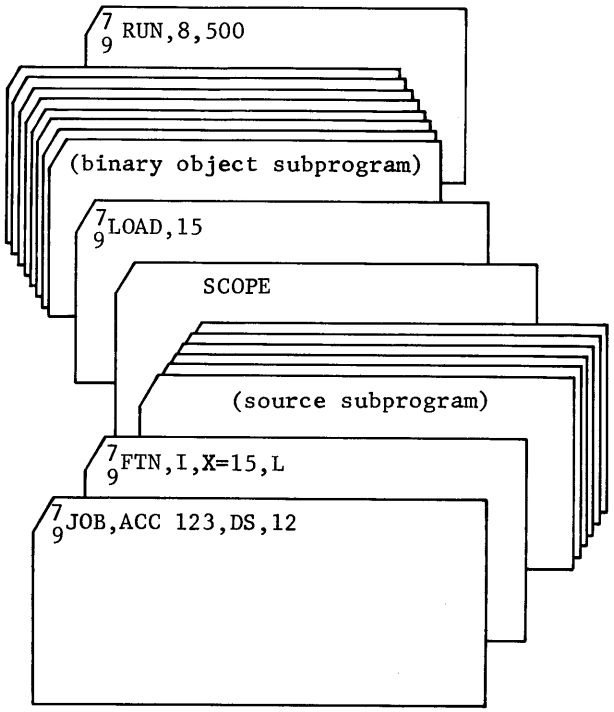
SCOPE

7/9LOAD,u

binary object subprogram

7/9RUN,t,p,r,m

After compilation is completed, the subprograms may be loaded in any order. Linkage does not take place until all subprograms are loaded.



11.3 EQUIPMENT ASSIGNMENT

SCOPE locates and assigns physical equipment at run time. All references to input/output units are by logical unit numbers. In Drum SCOPE, logical units are assigned to the drum unless the user specifies otherwise with an EQUIP statement. All programmer input tapes and all output units to be saved after job termination must be declared with EQUIP statements. For a complete description of the use of EQUIP statements, see the 3600 Computer System Drum SCOPE Reference Manual.

LOGICAL UNITS

PROGRAMMER UNITS (logical units 1-49) are retained throughout the job for reference by the program and are released at the end of the job. A programmer tape may be saved after a job is completed by an EQUIP statement. Otherwise, the tape will be available for reuse.

SCRATCH UNITS (logical units 50-59), for temporary use during the operation of a program, are released after each program execution and may not be saved.

SYSTEM UNITS (logical units 60-80), used by SCOPE and the programmer, are not released until the end of the monitor sequence, with the exception of a load-and-go unit and auxiliary library tape. They are defined by the monitor system, and the physical units to which they are assigned are determined by SCOPE. Following are the system units:

STANDARD INPUT (60)

All jobs to be processed by SCOPE are placed on this unit by the operator.

STANDARD OUTPUT (61)

Accounting information, diagnostics, dumps and job control statements are printed on this unit in BCD mode. The programmer may also use this unit for program output.

STANDARD PUNCH OUTPUT (62)

Output for punching is made to this unit. All records are written in binary mode unless specified otherwise.

INPUT COMMENT (63)

Comment from the operator to the monitor system is made on this unit. The programmer may also use it for input.

OUTPUT COMMENT (64)

Statements from the monitor system to the operator are made on this unit. The programmer may also use it for output information. The comment units are usually assigned to the console typewriter.

ACCOUNTING RECORD (65)

Job statements and total time used by the job are recorded on this unit. In Drum SCOPE the drum is used for storing accounting records and units 65-68 are reserved for use by the system.

LOAD-AND-GO (69)

Binary object programs transferred from the standard input unit or produced by compilation or assembly may be stored here prior to loading and executing. This unit may be saved by the programmer with an EQUIP statement. If the unit is not saved, it is released at execution time.

SATELLITES (66-68)

Satellite® units are assigned by SCOPE to computers which communicate with the 3600. SCOPE contains a Satellite control program which answers input/output requests from a Satellite and directs requests to it. Programmers cannot control Satellite units. In Drum SCOPE, Satellite units are replaced by background programs.

SCOPE LIBRARY (70)

The SCOPE library contains the monitor system and all programs and subroutines which operate under SCOPE, such as FORTRAN, COBOL, COMPASS, ALGOL and SORT.

AUXILIARY LIBRARIES (71-79)

Auxiliary libraries are used for library preparation and editing and are released at the end of the job.

SYSTEM SCRATCH RECORD (80)

Monitor equipment tables and accounting data for each job may be stored in the first record of this unit. If part of resident is destroyed by a running program, SCOPE uses the information on this tape for system recovery. The first scratch unit requested by a program will be assigned to this unit. The programmer cannot control this unit. In 3400 Drum SCOPE units 80-99 are reserved for use by the system in operating background programs; in 3600 Drum SCOPE unit 80 is reserved for system use.

The programmer may use system units 60-65 and 69-79 for input/output as long as he does not attempt any of the following operations:

rewind to load point (REWINDi)

backspace a record (BACKSPACEi)

write an end-of-file (END FILEi)

write on units 70, 60, 63 (except Drum SCOPE)

In Drum SCOPE 63 equivalent to 64.

APPENDIX SECTION

CODING PROCEDURES AND CHARACTER CODES A

STATEMENTS

FORTRAN coding forms contain 80 columns in which the characters of the language are written, one character per column. The statements are written in columns 7 through 72. Statements longer than 66 columns may be carried to the next line by using a continuation designator. More than one statement may be written on a line. Blanks may be used freely in FORTRAN statements to provide readability. Blanks are significant, however, in H fields.

The special character $ may be used to write more than one statement on a line. Statements so written may also use the continuation feature. A $ symbol may not be used as a statement separator with FORMAT statements or continuations of FORMAT 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
```

COMMENTS

Comment information is designated by a C in column 1 of a statement. Comment information will appear in the source program, but it is not translated into object code. Columns 2 through 80 may be used. Continuation is not permitted; that is, each line of comments must be preceded by the column 1 C designator.

All comment cards belonging to a specific program, or subprogram, should appear between the PROGRAM, SUBROUTINE, or FUNCTION statement and the END statement.

STATEMENT IDENTIFIERS

Any statement may have an identifier but only statements referred to elsewhere in the program require identifiers. A statement identifier is a string of from 1 to 5 digits, 1 to 99999, in columns 1 through 5. Leading zeros are ignored; 1, 01, 001, 0001 are equivalent forms. Zero is not a statement identifier. In any given program or subprogram each statement identifier must be unique. If the statement identifier is followed by a character other than zero in column 6 the statement identifier is ignored.

Statement identifiers of declarative statements (excepting FORMAT) are ignored by the compiler, except for diagnostic purposes.

CONTINUATION

In the first line of every statement, column 6 must be blank. If statements occupy more than one line, all subsequent lines must have a character other than blank or zero in column 6. A FORTRAN statement may contain up to 598 operators, delimiters (commas or parentheses) and identifiers; blanks are not included in this count. The number of continuations allowed is a function of the number of operators, delimiters and identifiers within it.

IDENTIFICATION FIELD

Columns 73 through 80 are always ignored in the translation 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.

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.

Blank cards within the input deck are treated as follows:

> If a blank card appears between a statement and its continuation, the continuation and other continuations following it are lost. Compilation continues.
>
> A blank card between two statements is ignored.

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

CARRIAGE CONTROL

When printing on-line, the first character of a record transmitted by a PRINT statement is a carriage control character for spacing on the printer. The carriage control characters are:

Character	Action
Blank or any character other than 0 or 1	Single space after printing
0	Double space before printing
1	Eject page before printing

The characters, 0 and 1, are not printed. Some printers provide additional codes which are given in the specific manuals.

When printing off-line, the printer control is determined by the installation.

CHARACTER CODES

Source Language Character	BCD (Internal only)	Punch position in a Hollerith Card Column	Source Language Character	BCD (Internal only)	Punch position in a Hollerith Card Column
A	21	12-1	Y	70	0-8
B	22	12-2	Z	71	0-9
C	23	12-3	0	00	0
D	24	12-4	1	01	1
E	25	12-5	2	02	2
F	26	12-6	3	03	3
G	27	12-7	4	04	4
H	30	12-8	5	05	5
I	31	12-9	6	06	6
J	41	11-1	7	07	7
K	42	11-2	8	10	8
L	43	11-3	9	11	9
M	44	11-4	/	61	0-1
N	45	11-5	+	20	12
O	46	11-6	-	40	11
P	47	11-7	blank	60	space
Q	50	11-8	.	33	12-8-3
R	51	11-9	)	34	12-8-4
S	62	0-2	$	53	11-8-3
T	63	0-3	*	54	11-8-4
U	64	0-4	,	73	0-8-3
V	65	0-5	(	74	0-8-4
W	66	0-6	=	13	8-3
X	67	0-7	≠	14	8-4

STATEMENT INDEX B

† N = Non-executable E = Executable

			Page
Storage Allocations	DIMENSION V_1, V_2, ..., V_n	N	4-2
	COMMON/I_i/List	N	4-4
	EQUIVALENCE (A, B, ...), (A1, B1, ...) ...	N	4-7
	DATA (I_1 = List), (I_2 = List), ...	N	4-11
3600 only	BANK, (b_1), $name_1$, ..., (b_2), $name_2$, ...	N	4-16

Arithmetic Statement Function

	Function (p_1, p_2, ... p_n) = Expression	E	7-8

Symbol Manipulation, Control and I/O

Replacement	V = E	Arithmetic	E	3-1
		Logical/Relational	E	3-4
		Masking	E	3-4
		Multiple	E	3-5
Intra-program Transfers		GO TO n	E	6-1
		GO TO m, (n_1, ... n_m)	E	6-1
		GO TO (n_1, ..., n_m) E	E	6-2
		GO TO (n_1, ..., n_m), E	E	6-2
		IF (A) 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 (SENSE LIGHT i) n_1, n_2	E	6-3
		IF (SENSE SWITCH i) n_1, n_2	E	6-4
		IF DIVIDE {FAULT / CHECK} n_1, n_2	E	6-4
		IF EXPONENT FAULT n_1, n_2	E	6-4
		IF OVERFLOW FAULT n_1, n_2	E	6-5
		IF (EOF, i) n_1, n_2	E	10-10
		IF (IOCHECK, i) n_1, n_2	E	10-10
		IF (UNIT, i) n_1, n_2, n_3, n_4	E	10-10
		IF (UNIT, i) n_1, n_2, n_3	E	10-11
		IF (UNIT, i) n_1, n_2	E	10-11
Loop Control		DO n i = m_1, m_2, m_3	E	6-5

Miscellaneous Program Controls			Page
	ASSIGN s TO m	E	6-2
	SENSE LIGHT i	E	6-3
	CONTINUE	E	6-8
	PAUSE; PAUSE n	E	6-8
	STOP; STOP n	E	6-9
I/O Format	FORMAT ($spec_1$, $spec_2$, ...)	N	9-4
I/O Control Statements	READ n, L	E	10-5
	PRINT n, L	E	10-1
	PUNCH n, L	E	10-1
	READ (i,n) L READ INPUT TAPE i,n,L	E	10-5
	WRITE (i,n) L WRITE OUTPUT TAPE i,n,L	E	10-2
	READ (i) L READ TAPE i, L	E	10-6
	WRITE (i) L WRITE TAPE i, L	E	10-2
	BUFFER IN (i,p) (A,B) BUFFER OUT (i,p) (A,B)	E	10-8
I/O Unit Handling	END FILE i	E	10-10
	REWIND i	E	10-9
	BACKSPACE i	E	10-9
	CALL BACKFILE (i)	E	10-10
	CALL SKIPFILE (i)	E	10-10
Internal Data Manipulation	ENCODE (c,n,V) L	E	10-13
	DECODE (c,n,V) L	E	10-13
Termination	END	N/E	6-9

Form	Definition	Actual Parameter Type	Mode of Result
ABS(X), ABSF(X)	Absolute value	Real	Real
ACOS(X), ACOSF(X)	Arccosine X radians	Real	Real
AIMAG(C)	Obtain imaginary part of complex number	Complex	Real
AINT(X), INTF(X)	Truncation, integer	Real	Real
ALOG(X), LOGF(X)	Natural log of X	Real	Real
ALOG10(X)	Log of X to base 10	Real	Real
AMAX0(I_1, I_2, ...), MAX0F(I_1, I_2, ...)	Determine maximum argument	Integer	Real
AMAX1(X_1, X_2, ...), MAX1F(X_1, X_2, ...)		Real	Real
AMIN0(I_1, I_2, ...), MIN0F(I_1, I_2, ...)	Determine minimum argument	Integer	Real
AMIN1(X_1, X_2, ...), MIN1F(I_1, I_2, ...)		Real	Real
AMOD(X_1, X_2), MODF (X_1, X_2)	X_1 modulo X_2	Real	Real
ASIN(X), ASINF(X)	Arcsine X radians	Real	Real
ATAN(X), ATANF(X)	Arctangent X radians	Real	Real
ATAN2(X_1, X_2)	Arctangent $\frac{X_1}{X_2}$	Real	Real
CABS(C)	Magnitude or modulus of C	Complex	Real
CANG(C)	Argument or angle associated with a complex number	Complex	Real
CATAN(C)	Complex arctangent of C in radians	Complex	Complex
CCOS(C)	Complex cosine of C in radians	Complex	Complex
CEXP(C)	Complex exponential of C	Complex	Complex
CLOG(C)	Complex natural log of C	Complex	Complex
CMPLX(X_1, X_1)	Form complex number	Real	Complex
CONJG(C)	Conjugate of C	Complex	Complex
COS(X), COSF(X)	Cosine X radians	Real	Real
COTF(X) (3600 only)	Co-tangent of X radians	Real	Real
CSIN(C)	Complex sine of C in radians	Complex	Complex

Form	Definition	Actual Parameter Type	Mode of Result
CSQRT(C)	Complex square root of C	Complex	Complex
CUBERTF(X)	Cube root of X	Real	Real
DABS(D)	Absolute value of double precision arguments	Double	Double
DATAN(D)	Double precision arctangent of D in radians	Double	Double
DATAN2(D_1, D_2)	Angle whose tangent is $\frac{D_1}{D_2}$	Double	Double
DBLE(X)	Convert single to double	Real	Double
DCOS(D)	Double precision cosine D in radians	Double	Double
DCUBRT(D)	Double precision cube root of D	Double	Double
DEXP(D)	Double precision exponential of D	Double	Double
DIM(X_1, X_2), DIMF(X_1, X_2)	If $X_1 > X_2$: $X_1 - X_2$ If $X_1 \leq X_2$: 0	Real	Real
DLOG(D)	Double precision natural log of D	Double	Double
DLOG10(D)	Log of double precision D to base 10	Double	Double
DMAX1(D_1, ..., D_n)	Determine maximum double precision argument	Double	Double
DMIN1(D_1, ..., D_n)	Determine minimum double precision argument	Double	Double
DMOD(D_1, D_2)	D_1 modulo D_2	Double	Double
DPOWER(D_1, D_2) (3400 only)	$D_1^{D_2}$	Double	Double
DSIGN(D_1, D_2)	Sign of D_2 time D_1	Double	Double
DSIN(D)	Double prec. sine of D in radians	Double	Double
DSQRT(D)	Double prec. square root of D	Double	Double
EXP(X), EXPF(X)	e to Xth power	Real	Real
FLOAT(I), FLOATF(I)	Integer to Real Conversion	Integer	Real
IABS(I), XABSF(I)	Absolute Value	Integer	Integer
IDIM(I_1, I_2), XDIMF(I_1, I_2)	If $I_1 > I_2$: $I_1 - I_2$ If $I_1 \leq I_2$: 0	Integer	Integer
IDINT(D)	Double to Integer Conversion	Double	Integer
IFIX(X), XFIXF(X)	Real to Integer Conversion	Real	Integer
INT(X), XINTF(X)	Real to Integer Conversion	Real	Integer
INTF(X), AINT(X)	Truncation to Integer	Real	Real

Form	Definition	Actual Parameter Type	Mode of Result
ISIGN(I_1, I_2), XSIGNF(I_1, I_2)	Sign of I_2 times $\lvert I_1 \rvert$	Integer	Integer
ITOJ(I, J)	I^J	Integer	Integer
ITOX(I, X)	I^X	Integer Real	Real
LENGTHF(I)	Number of words read/written on unit I	Integer	Integer
MAX0(I_1, I_2, ...), XMAX0F(I_1, I_2, ...)	Determine maximum argument	Integer	Integer
MAX1(X_1, X_2, ...), XMAX1F(X_1, X_2, ...)		Real	Integer
MIN0(I_1, I_2, ...), XMIN0F(I_1, I_2, ...)	Determine minimum argument	Integer	Integer
MIN1(X_1, X_2, ...), XMIN1F(X_1, X_2, ...)		Real	Integer
MOD(I_1, I_2), XMODF(I_1, I_2)	I_1 modulo I_2	Integer	Integer
POWRF(X_1, X_2)	$X_1^{X_2}$	Real	Real
RAN(N) (3400 only), RANF(N)†	Generate Random Number	Negative Positive	Real Integer
REAL(C)	Obtain real part of complex number	Complex	Real
SIGN(X_1, X_2), SIGNF(X_1, X_2)	Sign of X_2 times $\lvert X_1 \rvert$	Real	Real
SIN(X), SINF(X)	Sine X radians	Real	Real
SNGL(D)	Double to Real Conversion	Double	Real
SQRT(X), SQRTF(X)	Square root of X	Real	Real
TAN(X) (3400 only), TANF(X)	Tangent X radians	Real	Real
TANH(X), (3400 only) TANHF(X)	Hyperbolic tangent X radians	Real	Real
TIMEF(X) (3600 only)	Current time in floating point format	Real	Real
XTOI(X, I)	X^I	Real Integer	Real
LIBRARY SUBROUTINES			
Q8QERROR(k, m)	Error trace and message	Integer Any	BCD
Q8QERSET(n) (3600 only)	Set/clear error key	Integer (0 or 1)	BCD

† Any compiled calls to RANF(N), used as a function, treat the resulting value as real; to get a value treated as integer use: EQUIVALENCE(X, I) and use I to get an integer value.

X = RANF(+1)

The following 3600 functions will be coded in-line rather than called as closed routines. The closed function may be obtained by the appearance of the name in an EXTERNAL statement. If any of these function names appear as actual parameters, they must also appear in an EXTERNAL statement.

ABS or ABSF	IABS or XABSF	DBLE	REAL
SIGN or SIGNF	ISIGN or XSIGNF	AIMAG	DABS
DIM or DIMF	IDIM or XDIMF	CONJG	
FLOAT or FLOATF	INTF or AINT	CMPLX	

FORTRAN TABLE LIMITS

The phrase "literal appearance" means that if the same item appears more than once, it must be counted each time. The term "unique" means that an item is counted only once regardless of how many times it appears.

1. Total unique appearances of variables in EQUIVALENCE statements must not exceed 500.

2. Total unique appearances of variables in COMMON statements must not exceed 500.

3. Combined total unique appearances of variables in DIMENSION, COMMON, EQUIVALENCE, TYPE, EXTERNAL, PROGRAM, SUBROUTINE and FUNCTION statements must not exceed 699.

4. Total unique appearances of constants and statement numbers (literal appearances for Hollerith) during processing:

   declarative statements - 500 (3600); 600 (3400)

   arithmetic statements - 1000 + 3 (700-DV) - (FPR + V + ASF)

   DV number of unique variable names used in item 3.

   FPR number of formal parameter references in the program.

   V number of unique local variable names.

   ASF number of literal operators, operands and delimiters in each arithmetic statement function to the right of the replacement operator.

5. Not more than 50 branches in a computed GO TO statement.

6. Nested DO-loops must not exceed 50.

7. Literal appearances of operators, operands and delimiters within any one statement must not exceed 600. This includes the expansion of arithmetic statement functions within the executable statement.

8. Non-declared functions must not exceed 200. This includes two entries for each arithmetic statement function.

9. The number of parameters for any SUBROUTINE, FUNCTION or CALL may not exceed 63.

10. A Hollerith constant may not contain more than 136 characters.

11. Unique ENTRY statements may not exceed 20 in any one subprogram.

12. The limit on the number of non-declared (local) variable names in a subprogram is:
1000 + 3 (700-DV) - (FPR + V + ASF) - C

DV, FPR, V and ASF are defined in item 4.

C is the number of unique constants in executable statements (item 4).

13. The number of standard index functions in any one subprogram is limited to 95.

An index function is an expression representing the variable portion of the address calculation of an array element. Identical index functions will have only one entry in the index function table.

For example:
```
DIMENSION A(5,5), B(5,10)  C(5,20)
TYPE DOUBLE C
```

The references A(2*I, 2*J), B(2*I-3, 2*J+10) and C(I, J) will be represented by the same index function.

14. The number of non-standard index functions is 32.

15. The limit on dimensions for array $A(d_1, d_2, d_3)$ referenced as A(I, J, K) is:

$$e*(d_1*(K*d_2+J)+I) < 32767$$

d_1, d_2 are dimensions one and two respectively.

e is the element size (i.e., double e = 2, real e = 1)

I, J, K are the 1st, 2nd, and 3rd subscript values respectively.

CALLING SEQUENCES

E

The following material assumes familiarity with COMPASS instructions and coding procedures. The detailed discussion of calling sequences for standard arithmetic expressions should aid the user in writing additional functions and non-standard type arithmetic subroutines. All arithmetic subroutines for non-standard arithmetic must be provided by the user.

INSTRUCTION TYPES

During compilation of an expression, the translator generates the following instruction types to execute the operations indicated by the operators.

Instruction Types	Operators
Add operand	+
Subtract operand	-
Multiply operand	*
Divide operand	/
Complement accumulator	- (unary)
Power	**
Load operand Load negative operand Store operand	operand manipulations

Instructions are generated independently of the arithmetic mode and type of operand. The mode of the accumulator and operands as well as the element size are determined from the TYPE declarations or the variable name convention. They are fixed for standard types (real, integer, double, complex, logical).

The appropriate machine instruction or a jump to a routine which executes the intended operation then replaces the generated instruction type.

CALL IDENTIFIER

3400

Load and load complement instructions for all modes and arithmetic involving reals or integers, exclusively, generate 3400 COMPASS machine instructions; these operations are performed in line.

To perform double and complex operations (other than load, load complement), logical operations and conversions for mixed mode arithmetic, the compiler generates calls to library routines.

3600

Load and load complement instructions for all modes, and store instructions for all modes except logical, and arithmetic involving reals, integers, or double precision, exclusively, generate 3600 COMPASS machine instructions.

To perform complex operations (other than store, load and load complement), logical operations, and conversions for mixed mode arithmetic, the compilar generates library routine calls.

Library routine calls have the form:

QnQoomst

n indicates the number of operands to be treated.

n = 0 for operations on the accumulator only
n = 1 if the operand is a full or multiple word element
n = 2 for exponentiation; not defined for partial word operands
n = 3 if the operand is a partial word or byte-sized element

oo indicates the operation code. The operation is determined by the operator in the expression.

00 Load accumulator with operand
01 Load accumulator with complement of operand
02 Add operand to accumulator
03 Subtract operand from accumulator
04 Multiply accumulator by operand
05 Divide accumulator by operand
06 Complement accumulator
07 Raise $operand_1$ to the power $operand_2$
10 Store accumulator in operand

m indicates the mode of the accumulator before store operations and after all other operations.

0 mode is integer
1 mode is real
2 mode is double
3 mode is complex
4 mode is logical
5 mode is non-standard
6 mode is non-standard
7 mode is non-standard

s indicates the mode of the operand. The values of s are the same as those defined for m.

t indicates the mode of the exponent. It appears only with identifiers of the form Q2Q07mst; for other QnQ identifiers, it is always 0. Exponentiation involving a partial word operand is not permitted, except where the exponent is an integer constant 1-8.

Examples:

```
TYPE REAL A
TYPE INTEGER B
TYPE COMPLEX C
C = (A + B)
```

Translator Instructions	Conversions	Call Identifier
Load A	none	none
Add B	integer to real	Q1Q02100
Store C	real to complex	Q1Q10130

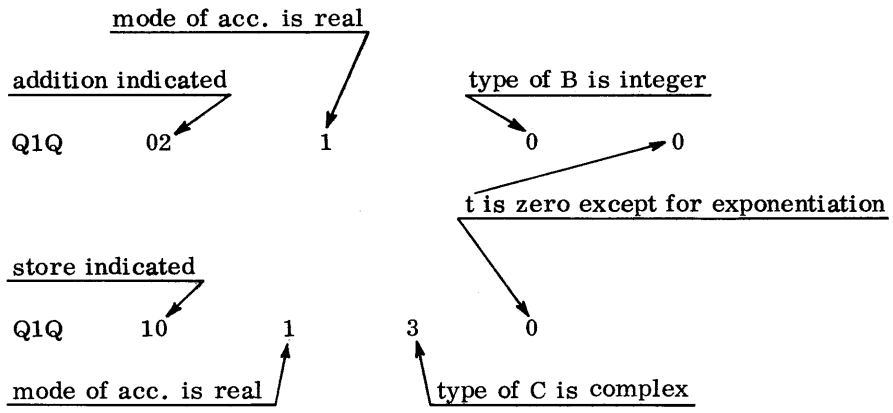
Resulting 3400 COMPASS object code:

```
                                    Interpretation
      LDA     A                     Transmit contents of location A to accumulator.
+     RTJ     Q1Q02100              Go to subroutine; convert B to real and add to accumulator.
      00      B
+     RTJ     Q1Q10130              Go to subroutine; convert accumulator to complex and
      00      C                     store in C.
```

Resulting 3600 COMPASS object code:

```
                                    Interpretation
      ENO     $A
      LDA     A                     Transmit contents of location A to accumulator.
      EXT     Q1Q02100              Generates external symbol naming subroutine.
+     BRTJ    ($)Q1Q02100           Go to subroutine.  Convert B to real and add to accumulator.
      ENO     $B
      LDA     B
      EXT     Q1Q10130              Go to subroutine, convert accumulator to complex.
+     BRTJ    ($)Q1Q10130
      DSTA    ($)C                  Store accumulator in C.
```

Breakdown of QnQ identifiers used in example:



CALLING SEQUENCES

Standard groups of COMPASS instructions are generated when jumps are made to QnQ subroutines, library functions, and subprograms.

QnQ Subroutines

Q0Q Subroutines

For operation 06, complement accumulator, the following code is generated:

```
3400    L       RTJ     Q0Q06mst
        L+1     Return

3600    L       BRTJ    ($)Q0Q06mst,,*
        L+1     Return
```

Q1Q Subroutines

For full word operand (1-7 words per operand) and all operations except 06 and 07, the following code is generated:

```
3400    L       RTJ     Q1Qoomst
                00      V, b
        L+1     Return
```

3600	L	BRTJ	($)Q1Qoomst, , *	
	L+1	xxx	($)V, B	xxx is DLDA, DLAC, DSTA
	L+2	Return		
	L	BRTJ	($)Q1Qoomst	
	L+1	ENO	$V	
		xxx	V, b	
	L+2	Return		xxx is LDA, LAC, STA

V is the operand + constant addend

b is the index designator; the content of b is an indexing quantity (index function) reflecting variable subscripts of the operand.

The effective operand address is (b) + operand + constant addend. b, (b) and/or the constant addend may be 0.

Constant addend is a bias on the base address to balance a portion of the index function contained in b, or simply a position relative to the base array address of a variable with constant subscripts. To calculate the constant addend and (b) for element A ($d_i * i \pm c_i, d_j * j \pm c_j, d_k * k \pm c_k$) in array A (I, J, K) the following formula is used.

Base Address	Constant Address	Index Function
Locn A+	$(-1 \pm c_i + I*(-1 \pm c_j + J*(-1 \pm c_k)))*f +$	$(d_i*i + I*(d_j*j + J*(d_k*k)))*f$

d_i, d_j, d_k, c_i, c_j, c_k are unsigned integer constants

f is the element length (1-7 words)

Q2Q Subroutines

For operation 07, exponentiation, the following code is generated.

3400	L	SLJ	* + 1
		00	V, b_1
	L+1	RTJ	Q2Q07mst
		00	U, b_2
	L+2	Return	

3600	L	BRTJ	($)Q2Q07mst, , *			
	L+1	DLDA	($)V, b_1	or	ENO LDA	$V V, b_1
	L+2	DLDA	($)U, b_2	or	ENO LDA	$U U, b_2
	L+3	Return				

V is the base operand + constant addend

U is the exponent operand + constant addend

b_1, b_2 are index designators

Q3Q Subroutines - Logical

The calling sequence is:

3400	L	SLJ	* + 1
		n	constant addend, b
	L+1	RTJ	Q3Qoomst
		pof	operand
	L+2	Return	
3600	L	BRTJ	($)Q3Qoomst, , *
	L+1	n	constant addend, b
		pof	($) operand
	L+2	Return	

n is the element length in bits.

b is an index register.

pof is the parameter offset which appears in the object code as 00. An offset is the number of bits between the left end of the word and the logical bit. The parameter offset is passed along with the operand address when the operand is a parameter in a subroutine call. During execution, it is transmitted with the parameter to all Q3Q calls within the subroutine.

constant addend is a position relative to the base array address of a variable with constant subscripts.

For logical arithmetic, the effective operand address is computed as follows by an object time routine:

a = ((n*((b)+ca))+pof)/p with remainder of d

a is first word address (FWA) addend (quotient)

d is actual offset (remainder)

n is element length in bits

(b) is content of index register

ca is constant addend

pof is parameter offset

p is packing number (32 bits per word for logical; 48 bits per word for byte)

The effective operand is the n bits of word FWA + a, d bits from left.

Q3Q Subroutines - Byte

The code generated for byte arithmetic is the same as for logical arithmetic. The offset for a byte is the number of bits between the left end of the word and the leftmost bit of the byte element. The programmer must include instructions in his Q3Q routine to compute the effective operand address and to locate the effective operand, according to the equation used for logical arithmetic.

p the packing number for bytes, is 48 bits per word.

Example:

FORTRAN	3400 COMPASS Calling Sequences	3600 COMPASS Calling Sequences
PROGRAM OFFSET		
DIMENSION A(20)		
TYPE OTHER5(/8)A		
.		
.		
.		
.		
CALL SAM (A(3)) ⟶	RTJ SAM 01 *+2 .Znum. 00 [20] ** [A]	BRTJ ($)SAM,,* SLJ *+2 01 DICT. .Znum. 00 [20] ($)A [A]
.		
.		
.		
END	The offset is calculated by the Q9QEVALB routine and stored with the parameter address at location .Znum. as indicated by the bracketed terms.	
SUBROUTINE SAM (B)		
DIMENSION B (15)		
TYPE OTHER5(/8)B		
I=23		
C=B(I-15) ⟶	SLJ *+1 10 -16, b RTJ Q3Q00550 00 [20] ** [B+1]	BRTJ ($)Q3Q00550,,* 10 -16, b 00 [20] ($)B [B+1]
.		
.		
.		
END	This Q3Q00500 routine must compute the effective operand address; it may call Q9QEVALB to do this. b is an index containing 23.	

Calculations performed in example:

1) for constant addend and index function

   Locn B - (1+15)+(8+15)
   Locn B - (16) + (23)

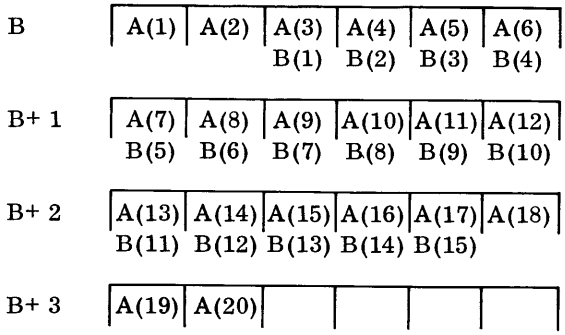
   ca = -16
   (b) = 23

2) effective operand address

   a.d = ((8*(23+(-16)) + 16)/48

   a = 1, FWA addend
   d = 24, actual offset

In memory:



SUBPROGRAMS

The subprograms (function or subroutine) are called by the following sequence.

FORTRAN:

CALL SUBNME (p_1, p_2, . . . , p_n)

or

V = FUNNME (p_1, p_2, . . . , p_n)

3400 COMPASS:

L	RTJ	name	
	np	*+m	
L+1	00	p_1	address of actual parameter
	00	p_2	
L+2	00	p_3	
	.	.	
	.	.	
	.	.	
	00	p_n	
L+m	Return		

3600 COMPASS

The routine always returns to L+1 which, in turn, causes a jump to L+(m+1).

```
L          BRTJ     ($)name,,*
L+1        SLJ      *+m
           np       DICT.
L+2        00       ($)p1      ┐
           00       ($)p2      │  address of actual parameter
           .           .       │
           .           .       │
           .           .       │
           .           .       │
           00       ($)pn      ┘
L+(m+1)    Return
```

m is $\frac{np+1}{2}$ +1

np is the number of parameters

DICT. contains the entry point into the subroutine called and is used by the standard error procedure.

The 00 operation code will be replaced by the offset for partial word parameters.

3400

```
           .
           .
           .
.Znum.     00 [offset]   base address and FWA addend    if actual parameter specifies a partial word element.

           00            effective address             if actual parameter specifies a multi-word element.
```

3600

```
           .
           .
           .
.Znum.     00 [offset]   base address + FWA addend      if actual parameter specifies a partial word element.

           00            effective address             if actual parameter specifies a multi-word element.
           .
           .
           .
```

When the call for a subprogram with a partial-word actual parameter is generated, the offset is calculated by special library routines, Q9QEVALL and Q9QEVALB. Q9QEVALL is called for logical elements and Q9QEVALB, for byte-size elements. The offset is made available to the subprogram at execution time by storing it with the parameter relative to the word tagged . Znum. .

Examples:

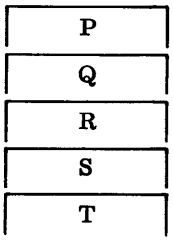
(1) Function Subprogram Reference

Z = QUAINT (P, Q, R, S, T)

Results in call:

```
3400    +       RTJ     QUAINT
        -       05      * + 4
        +       00      P       )
        -       00      Q       |
        +       00      R       }   non-subscripted multi-word elements
        -       00      S       |
        +       00      T       )
                Return

3600            BRTJ    ($)QUAINT
                SLJ     * + 4
                05      DICT.
        +       00      ($)P    )
        -       00      ($)Q    |
        +       00      ($)R    }   non-subscripted multi-word elements
        -       00      ($)S    |
        +       00      ($)T    )
                Return
```

In storage:



(2) Subroutine Subprogram

```
CALL    SAM (M, M(3), M(4))     M is one word per element
```

Results in call:

```
3400    +       RTJ     SAM
        -       03      * + 3
        +       00      M           address of operand
        -       00      M + 2       effective address is 3rd word
        +       00      M + 3       effective address is 4th word
        +       Return

3600            BRTJ    ($)SAM
                SLJ     * + 3
                03      DICT.
        +       00      ($)M
        -       00      ($)M+2
        +       00      ($)M+3
        +       Return
```

In storage:

M	M(1)
M + 1	M(2)
M + 2	M(3)
M + 3	M(4)

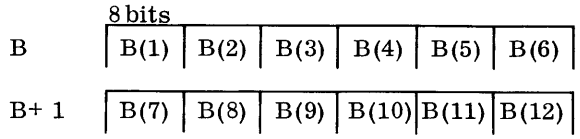
(3) DIMENSION B(12)
TYPE OTHER6 (/8) B
CALL SAM (B, B(2), B (11))

Results in call:

```
3400    +       RTJ     SAM
        -       03      * + 3
.Znum.          00      B           First element of B array is leftmost character of first
                                    word.
        -       00 [10] ** [B]      Second element of B array is offset from the left 8 bits
                                    (octal 10) but is still in first word.
        +       00 [40] ** [B + 1]  Eleventh element of B array is in second word and is
                                    offset 32 bits from left.

3600            BRTJ    ($)SAM
                SLJ     *+3
                03      DICT.

.Znum.          00      ($)B
        -       00 [10] ($)B [B]
        +       00 [40] ($)B [B + 1]
```

The values in the parentheses indicate the contents of the word at object time.

In storage:



Q9QEVALL subroutine calling sequence:

```
3400            ENA     ca, b
        L       RTJ     Q9QEVALL
                pof     operand
        L+1     xxx     .Znum.
```

ca is the constant addend

b is the index register

pof is the parameter offset

xxx is STA for storing in upper half word
STQ for storing in lower half word

.Znum. is .Z and number followed by a period (.Z00001.); this tags the location where the calculated address is stored.

```
3600    L       BRTJ    ($)Q9QEVALL,,*
        L+1     01      ca, b
                pof     ($)operand
        L+2     xxx     .Znum.
```

Q9QEVALB subroutine calling sequence:

```
3400    L       ENA     n
                ENQ     ca, b
        L+1     RTJ     Q9QEVALB
                pof     operand
        L+2     xxx     .Znum.
```

n is the element length in bits

ca, b, pof, xxx, .Znum. are defined above.

```
3600    L       BRTJ    ($)Q9QEVALB,,*
        L+1     n       ca, b
                pof     ($) operand
        L+2     xxx     .Znum.
```

Example:

FORTRAN:

```
    PROGRAM OTHER58
    TYPE OTHER5 (/3) A
    TYPE OTHER6 (/8) C, SUZY
    TYPE OTHER7 (3) E
    DIMENSION A(20), C(10), E(10)
    EXTERNAL SUZY
    C(1)=SUZY (A(5), C(2), E(3))
```

3600 COMPASS

```
                EXT     Q9QEVALB
                BRTJ    ($)Q9QEVALB,,*  This routine calculates parameter offset, num-
                03      4               ber of lists in element A, and constant addend
                00      ($)A            to base.
                STA     .Z00002.        Parameter offset and A are stored in upper
                                        portion of .Z00002.
                EXT     Q9QEVALB
                BRTJ    ($)Q9QEVALB,,*
                10      1
                00      ($)C
                STQ     .Z00002.        Parameter offset and C are stored in lower
                                        portion of .Z00002.
                ENA     E+6
                SAU     .Z00002.+1
                BRTJ    ($)SUZY,,*
                SLJ     *+3
                03      DICT.
.Z00002.        00      ($)A
        -       00      ($)C
        +       00      ($)E+6          The parameter E is a multi-word element.
```

LIBRARY FUNCTIONS

Library functions have two entry points as they may be called by value or by name. Some are also called for expression evaluation; these are named with the conventions for mixed mode arithmetic.

CALL-BY-NAME

Included in this calling sequence are the MAX and MIN library functions and all library functions with Q8Q name entry.

FORTRAN:

$$V = \text{LIBNME } (p_1, p_2, \ldots, p_n)$$

COMPASS

3400

	L	RTJ	LIBNAME
		np	* + m
	L+1	00	p_1
		00	p_2
	L+2	00	p_3
		.	.
		.	.
		.	.
	L+m	Return	p_n

addresses of actual parameters

In the special case of a function with a variable number of parameters (MIN and MAX functions), the 00 op-code in the upper half of L + 1 will be replaced by np, the number of parameters.

3600

The routine always returns to L + 1 which, in turn, causes a jump to L+(m = 1).

L	BRTJ	($) LIBNME
L+1	SLJ	* + m
	np	DICT.
L+2	00	($)$p_1$
	00	($)$p_2$
	.	.
	.	.
	.	.
	.	.
	00	($)$p_n$
L+(m + 1)	Return	

addresses of actual parameters

m is $\frac{np+1}{2}$ + 1

np is the number of parameters

DICT. contains the entry point into the subroutine called and is used by standard error procedure.

CALL-BY-VALUE

The call by value for most library subroutines which have one or two parameters generates the following sequence. The actual parameter is passed to the A or Q register or both.

FORTRAN:

$V = LIB(p_1, p_2)$

COMPASS:

3400

The class of routines included in this calling sequence includes all trigonometric functions:

LOGF	INTF	XDIMF	XTOI
EXPF	XINTF	MODF	ITOX
SQRTF	FLOATF	XMODF	ITOJ
CUBERTF	XFIXF	SIGNF	
ABSF	RANF	XSIGNF	
XABSF	DIMF	POWRF	

```
          LDA     p1
          LDQ     p2
    L     RTJ     LIB
    L+1   Return
```

3600

```
          LDA     p1
          LDQ     p2
    L     BRTJ    ($)LIB,,*
    L+1   SLJ     *+1
          00      DICT.
    L+2   Return
```

This calling sequence includes all the trigonometric functions. These routines also have a call by name calling sequence to the entry Q8Qname.

ABSF	TANF	INTF	LOGF	POWRF	also have a Q2Q07mst entry point.
XABSF	ATANF	XINTF	SIGNF	XTOI	
COSF	TANHF	SQRTF	XSIGNF	ITOX	
ACOSF	RANF	CUBERTF	DIMF	ITOJ	
SINF	MODF	FLOATF	XDIMF		
ASINF	XMODF	EXPF	XFIXF		

The typical 3600 library function entry points are:

```
        .               .               .
        .               .               .
        .               .               .
Q8QNAME         UBJP            (*)**
L+1             XMIT            (*)*-1, ($)Q8QDICT.
L+2             XMIT            (*)*-2, ($)DICT.
L+3             BRTJ            ($)Q8QLOADA
NAME            UBJP            (*)**
L+5             XMIT            (*)*-1, ($)Q8QDICT.
L+6             XMIT            (*)*-2, ($)DICT.
L+7             Normal return from Q8QLOADA
        .               .               .
        .               .               .
        .               .               .
```

The call by name transfers to the special routine Q8QLOADA which analyzes the call by name and makes it a call by value; the routine is then executed as if it had been called by value.

COMPILATION DIAGNOSTICS F

Diagnostics prepared by the compiler during compilation are output with the program listing and immediately follow the source program.

3400 general form is:

ERROR IN STATEMENT NUMBER n

xx yyyy zzz ..... z

n is the statement number in which the error occurred or the number of statements beyond the last numbered statement.

xx is the error type and may be:

- NC no compilation
- NX no execution
- WN warning - informative

yyyy is the octal error code number

zzz ...z is the error message

3400

Error Messages

1002 A PREVIOUS DO TERMINATES ON THIS DO STATEMENT

1003 A RUNNING INDEX USED PREVIOUSLY IN THIS NEST

1004 NESTING CAPACITY OF THE COMPILER IS EXCEEDED

1005 THE CONSTANT PARAMETER OF A DO OR DO-IMPLYING LOOP EXCEEDS 32767

1006 THE PARAMETERS OF A DO OR DO-IMPLYING LOOP MUST BE UNSIGNED INTEGER

1007 THE INITIAL VALUE OF A DO OR DO-IMPLYING LOOP MUST NOT EXCEED UPPER BOUND IF BOTH ARE CONSTANT

1010 THE RUNNING SUBSCRIPT IN A DO OR DO-IMPLYING LOOP MUST BE A SIMPLE INTEGER VARIABLE

1011 INCORRECT FORM FOR ENTRY STATEMENT

1012 ENTRY STATEMENT LABELED

1013 ENTRY STATEMENT IN MAIN PROGRAM

1014 ALL DECLARATIVE STATEMENTS MUST PRECEDE THE FIRST EXECUTABLE STATEMENT

1015 THE NUMBER OF INDEX VARIABLES EXCEEDS THE CAPACITY

1016 NO PATH TO THIS STATEMENT

1020 A DO LOOP MAY NOT TERMINATE AT THIS STATEMENT

3400

1021 A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO
1022 THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES
1023 ILLEGAL STATEMENT LABEL
1024 NON-STANDARD INDEXING IS NOT PERMITTED IN DO STATEMENTS
1025 THE TERMINAL LABEL OF A DO MUST BE AN INTEGER CONSTANT
1026 PREVIOUSLY USED ENTRY NAME
1027 NUMBER OF ENTRY STATEMENTS EXCEEDS 20
1030 UNLABELED FORMAT STATEMENT
1031 IF THIS IS AN ARITHMETIC STATEMENT IT HAS NO LEFT HAND SIDE
1032 OBJECT OF ASSIGN OR ASSIGNED GO TO NOT A SIMPLE INTEGER VARIABLE
1033 STATEMENT LABEL IN GO TO STATEMENT NOT INTEGER CONSTANT
1035 PARAMETER TO THIS STATEMENT NOT INTEGER CONSTANT OR VARIABLE
1036 ILLEGAL SUBROUTINE NAME
1037 PARAMETER STRING IS NOT WELL-FORMED
1040 ASSIGNED STATEMENT LABEL IS NOT INTEGER
1042 SUBPROGRAM OR VARIABLE NAME USED AS ENTRY
1050 NO END CARD
1051 ENTRY STATEMENT INSIDE A DO LOOP
1053 THE INCREMENT IN A DO OR DO-IMPLYING LOOP IS 0
1501 REAL CONSTANT EXCEEDS 2**1023
1502 ILLEGAL CHARACTER IN NUMERIC FIELD
1503 MORE THAN 16 OCTAL DIGITS
1504 NUMBER TOO LARGE
1505 ILLEGAL CHARACTER IN ALPHANUMERIC FIELD
1506 ILLEGAL CHARACTER IN EXPONENT FIELD OR REAL NUMB
1507 EXPONENT EXCEEDS 309
1510 INTEGERS MAY NOT EXCEED 2**47-1
1700 INDEX VARIABLE NOT IN TABLE
1777 MORE THAN 100 ERRORS, END COMPILATION
2000 MISSING PROGRAM NAME
2001 PROGRAM, SUBROUTINE OR FUNCTION CARD NOT FIRST CARD OF DECK
2002 IMPROPER FORMAT OF PROGRAM STATEMENT, PROBABLY MORE THAN 8 CHARACTERS
2003 IMPROPER SUBROUTINE OR FUNCTION STATEMENT TERMINATION OR PARAMETER ERROR
2004 ALPHABETIC CHARACTER DOESNT START NAME
2005 DUPLICATE VARIABLE NAME IN DIMENSION STATEMENT
2006 NO LEFT PARENS AFTER VARIABLE NAME

3400

2007 VARIABLE DIMENSION IDENTIFIER NOT IN PARAMETER LIST

2010 MORE THAN 3 DIMENSIONS IN DECLARATION OF ARRAY

2011 NO RIGHT PARENTHESIS DELIMITER IN SUBSCRIPT DECLARATION

2012 VARIABLE DIMENSIONED ARRAY USED IN COMMON

2013 MORE THAN 63 PARAMETERS

2015 NO SLASH (/) SEPARATOR IN BLOCK DESIGNATION

2016 UNDEFINED SEPARATOR IN COMMON STATEMENT

2017 NON-CONSTANT SUBSCRIPT IN COMMON DIMENSIONING

2020 SUFFIX 5, 6 OR 7 NOT ON TYPE-OTHER NAME

2021 TYPE OTHER DOUBLY DEFINED

2022 ELEMENT LENGTH DESIGNATOR NOT (W) OR (/B)

2023 LEFT, RIGHT PARENTHESIS OR COMMA MISSING IN EQUIVALENCE

2024 TYPE OTHER APPEARING WITH SUBSCRIPTS

2025 EQUIVALENCE CAUSES REORIGIN OF COMMON

2026 FORMAL PARAMETER OR ADJUSTABLE DIMENSION IN EQUIVALENCE

2027 NON-CONSTANT SUBSCRIPT IN EQUIVALENCE

2030 DECLARED VARIABLE IN EXTERNAL STATEMENT

2031 COMMON/EQUIVALENCE ERROR

2032 LEFT/RIGHT PARENS NOT MATCHING

2033 IMPLIED-DO ERROR IN DATA STATEMENT, NO = AFTER DO VARIABLE, NON-CONSTANT DO LIMITS, ETC.

2034 NO = AFTER IDENTIFIER

2035 SUBSCRIPTED VARIABLE NOT PREVIOUSLY DIMENSIONED

2036 DATA TO ADJUSTABLE DIMENSIONED OR PARTIAL WORD ARRAY

2037 MULTIPLE DATA TO NON-DIMENSIONED VARIABLE

2040 DUPLICATE BLOCK NAME

2041 EQUIVALENCE OVERLAPS COMMON BLOCKS

2042 FORMAL PARAMETER IN COMMON DECLARATION

2043 VARIABLE NAME GREATER THAN 8 CHARACTERS OR NO COMMA SEPARATOR

2044 NON-CONSTANT DATA IN LIST

2045 DOUBLY DEFINED VARIABLE IN COMMON

2046 REPEAT COUNT MUST BE AN INTEGER CONSTANT 1-32767

2047 VARIABLE EQUATED TO ITSELF + N

2050 (W) IS NOT AN INTEGER 1 THRU 7 OR (/B) IS NOT A DIVISOR OF 48

2051 VARIABLE DEFINED IN PREVIOUS TYPE STATEMENT

2052 DOUBLY DEFINED FORMAL PARAMETER

3400

2053 PROGRAM CARD CONTAINS PARAMETER LIST

2144 COMPILER NON-EXECUTABLE STATEMENT TABLE EXCEEDED

2146 COMPILER COMMON OR BLOCK TABLE EXCEEDED

2147 COMPILER EQUIVALENCE TABLE EXCEEDED

2150 MACHINE OR TABLE ERROR, VARIABLE NOT IN DIMENLIST

2201 COMMA MISSING IN PARAMETER LIST OR VARIABLE MORE THAN 8 CHARACTERS

2203 ILLEGAL SEQUENCE OR USE OF OPERATORS

2205 POSSIBLE MACHINE ERROR IN PROCESSING COMMON EXPRESSIONS

2206 ILLEGAL OR MISSING OPERATOR

2207 ILLEGAL REPLACEMENT IN ARITHMETIC STATEMENT

2210 AN * HAS BEEN INSERTED FOR THE APPEARANCE OF N (,) (,) V OR) N

2212 LOGICAL OR MASKING OPERATOR IN CALL PARAMETER

2213 ILLEGAL REPLACEMENT APPEARS IN AN EXPRESSION

2701 POSSIBLE MACHINE ERROR, BAD SCRATCH TAPE

2702 COMPASS DID NOT SEE END CARD

4001 FIRST WORD OF ASF IS NOT AN IDENTIFIER

4002 NO REFERENCE TO ONE OF THE PARAMETERS IN STATEMENT OF AN ARITHMETIC STATEMENT FUNCTION

4003 ERROR IN ASF SET-UP, NO END IN STRING

4004 ARITHMETIC STATEMENT FUNCTION DOUBLY DEFINED

4021 POSSIBLE MACHINE ERROR. ARITHMETIC FAULT TYPE NOT RECOGNIZED

4022 POSSIBLE MACHINE ERROR. MACHINE CONDITION TEST NOT RECOGNIZED

4023 PARAMETER NOT TYPE INTEGER

4024 I IS OUTSIDE THE PERMITTED RANGE

4026 UNIT NUMBER MUST BE A SIMPLE INTEGER VARIABLE OR AN INTEGER CONSTANT

4030 UNIT NUMBER NOT FOLLOWED BY)

4031 AN IF UNIT STATEMENT WITHOUT 2-4 BRANCH POINTS

4101 BRANCH POINT ERROR IN IF STATEMENT

4102 LOGICAL IF IS FORMED INCORRECTLY

4103 TWO OR MORE RELATIONAL OPERATORS IN THE SAME RELATIONAL SUB-EXPRESSION

4104 LOGICAL EXPRESSION INCORRECTLY FORMED

4105 RELATIONAL SUB-EXPRESSION FORMED INCORRECTLY

4106 THE .NOT. OPERATION MUST BE FOLLOWED BY EITHER (OR AN OPERAND

4107 POSSIBLE MACHINE ERROR. LOGICAL OPERATOR NOT RECOGNIZED

4110 POSSIBLE MACHINE ERROR IN EVALUATING LOGICAL EXPRESSION

4112 LOGICAL CONNECTIVE MUST BE FOLLOWED BY (OR AN OPERAND

3400

4113 LOGICAL SUBEXPRESSION BEGINS WITH AN OPERATOR
4114 EXCESS LEFT PARENTHESIS IN LOGICAL EXPRESSION
4200 MASKING ARITHMETIC EXPRESSION TOO LONG
4201 ARITHMETIC SUB-EXPRESSION IN MASKING STATEMENT NOT FULLY PARENTHESIZED
4202 FUNCTION CALLED INCORRECTLY
4210 MASKING EXPRESSION INCORRECTLY FORMED
4212 THE FIRST ELEMENT OF A BOOLEAN EXPRESSION NOT AN OPERAND, (OR .NOT.
4213 NOT FOLLOWED ONLY BY .AND., .OR.,)
4214 OPERATORS .AND., .OR. NOT FOLLOWED BY EITHER (, .NOT., OR AN OPERAND
4215 MASKING OPERANDS MUST BE REAL OR INTEGER
4220 REPLACEMENT VARIABLE FOR AN EXPRESSION USING LOGICAL OPERATORS NOT LOGICAL
4402 DIMENSION OF VARIABLE GREATER THAN 32767
5001 ILLEGAL MARK IN COLUMN 6
5002 UN-RECOGNIZED STATEMENT
5003 ASSUMED DIMENSION STATEMENT
5004 ASSUMED BACKSPACE STATEMENT
5005 ASSUMED WRITE - TAPE STATEMENT
5006 ASSUMED SUBROUTINE STATEMENT
5007 ASSUMED READ-INPUT-TAPE STATEMENT
5010 ASSUMED WRITE-OUTPUT-TAPE STATEMENT
5011 TOO MANY CHARACTERS IN IDENT
5012 ASSUMED SENSE-LIGHT STATEMENT
5013 ASSUMED IF-DIVIDE-FAULT STATEMENT
5014 ASSUMED IF-OVERFLOW-FAULT STATEMENT
5015 ASSUMED IF-EXPONENT-FAULT STATEMENT
5016 STATEMENT TOO LONG
5017 UN-MATCHED PARENTHESIS
5020 ILLEGAL USE OF BOOLEAN OR RELATIONAL OPERATOR
5021 ASSUMED IF-SENSE-LIGHT STATEMENT
5022 ASSUMED IF-SENSE-SWITCH STATEMENT
5023 ASSUMED BUFFER OUT STATEMENT
5024 ASSUMED EQUIVALENCE STATEMENT
5025 IMPROPER LENGTH FOR HOLLERITH CONSTANT
5026 ILLEGAL USE OF PERIOD
5027 ILLEGAL CONSTANT TYPE
5030 STATEMENT ENDS WITH *

3400

5032 LABEL AND MARK IN COLUMN 6
5040 TOO MANY SUBSCRIPT INDICES
5041 ADJACENT COMMAS
5042 RIGHT PAREN PRECEDED BY COMMA
5043 LEFT PAREN FOLLOWED BY COMMA
5044 EMPTY PARENTHETICAL EXPRESSION
5045 LIMIT FOR NON-STANDARD SUBSCRIPT EXPRESSIONS EXCEEDED
5046 NUMBER OF CONSTANTS EXCEEDS COMPILER LIMIT
5050 NUMBER OF FUNCTIONS EXCEED COMPILER LIMIT
5051 LABELED BLANK STATEMENT--CONTINUE ASSUMED
5052 NUMBER OF IDENTIFIERS EXCEEDS COMPILER LIMIT
5053 LIMIT FOR STANDARD INDEX FUNCTIONS EXCEEDED
5060 ASF PARAMETERS DO NOT AGREE IN NUMBER
5061 TOO MANY ASF ENTRIES, POSSIBLY DUE TO RECURSIVE CALL
5062 ASF PARAMETER LIST NOT ENDED
5063 ILLEGAL USE OF PROGRAM, SUBROUTINE OR FUNCTION NAME
5201 FORMAT CONTAINS ZERO MULTIPLIER
5202 FORMAT CONTAINS ILLEGAL MULTIPLIER FOR , + -) OR /
5203 FORMAT CONTAINS MISSING OR ZERO FIELD WIDTH
5204 FORMAT SPECIFICATION IS NOT FOLLOWED BY A DELIMITER
5205 FORMAT CONTAINS A NUMBER GREATER THAN 262143
5206 FORMAT CONTAINS ONE OF FOUR IMPROPER ADJACENT SEPARATOR PAIRS ( ) . , (, OR ,)
5207 FORMAT CONTAINS ILLEGAL PLUS OR MINUS SIGN
5210 FORMAT CONTAINS AN IMPROPER C PATTERN
5211 FORMAT CONTAINS ILLEGAL CHARACTER
5212 FORMAT CONTAINS MORE THAN 10 PARENTHESIS LEVELS
5213 FORMAT HAS NO MATCHING CLOSING PARENTHESIS
5214 NUMBER LARGER THAN 999
5215 FORMAT CONTAINS CHARACTERS OUTSIDE THE LAST CLOSING PARENTHESIS
5216 NO OPENING PARENTHESIS ON FORMAT STATEMENT
5217 * TYPE FORMAT HAS NO CLOSING *
5225 FORMAT HAS A D FIELD GREATER THAN 26
5400 NO. OF BRANCHES IN COMPUTED GO TO EXCEEDS 50
5401 ASSIGNED STATEMENT LABEL EXCEEDS FIVE DIGITS
6001 PARENTHESIS USAGE OR DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL IN I/O DATA LIST
6002 WRONG FORMAT OF I/O STATEMENT DATA LIST WAS NOT YET PROCESSED

3400

6003 I/O TAPE NUMBER GREATER THAN 80

6005 ILLEGAL SUBSCRIPT IN I/O DATA LIST

6006 INPUT OF DATA INTO A CONSTANT IS ILLEGAL

6007 TRANSMISSION OF BYTE SIZED DATA IN BINARY MODE IS ILLEGAL

6010 I/O TYPE 5, 6 OR 7 PROHIBITED

7001 TYPE OTHER OPERAND DOES NOT APPEAR IN DEVVARLIST. POSSIBLE MACHINE ERROR

7002 ERASABLE STORAGE REQUIRED IS TOO LARGE

7003 TYPE OTHER INTERMIXED IN ARITHMETIC

7004 LOGICAL OR BYTE SIZED OPERAND(S) USED IN EXPONENTIATION

7005 IMPROPER OPERAND

3600 general form is:

```
***FORTRAN SOURCE CODE ERRORS****
txxxx TYPE ERROR IN STATEMENT NUMBER nn
     (error message)
```

t type of error

0 = informative

1, 2 = destructive - errors which prevent execution

4 = fatal - errors which terminate compilation

xxxx error number

nn statement number in which the error occurred or followed by PLUS n where n is the number of statements beyond the last numbered statement.

Error Messages

1001 FUNCTION NAME NOT USED AS REPLACEMENT IN FUNCTION

1002 A PREVIOUS DO TERMINATES ON THIS DO STATEMENT

1003 A RUNNING INDEX USED IN THIS STATEMENT HAS BEEN USED PREVIOUSLY IN THIS NEST

1004 THE NESTING CAPACITY OF THE COMPILER HAS BEEN EXCEEDED

1005 THE CONSTANT PARAMETERS OF A DO OR DO-IMPLYING LOOP CANNOT EXCEED 32767

1006 THE PARAMETERS OF A DO OR DO-IMPLYING LOOP MUST BE UNSIGNED INTEGER

1007 THE INITIAL VALUE OF A DO OR DO-IMPLYING LOOP MUST NOT EXCEED THE UPPER BOUND IF BOTH ARE CONSTANT

1010 THE RUNNING SUBSCRIPT IN A DO OR DO-IMPLYING LOOP MUST BE A SIMPLE INTEGER VARIABLE

1011 THE CORRECT FORM FOR THE ENTRY STATEMENT IS ENTRY NAME

1012 ENTRY STATEMENTS SHOULD NOT BE LABELED

1013 MAIN PROGRAMS SHOULD NOT CONTAIN ENTRY STATEMENTS

1014 ALL DECLARATIVE STATEMENTS MUST PRECEDE THE FIRST EXECUTABLE STATEMENT

1015 THE NUMBER OF INDEX VARIABLES EXCEEDS THE CAPACITY OF THE COMPILER

1016 THERE IS NO PATH TO THIS STATEMENT

1017 A DO LOOP TERMINATES AT THIS STATEMENT

1020 A DO LOOP MAY NOT TERMINATE AT AN END STATEMENT

1021 DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO

1022 THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES

1023 STATEMENT LABELS MUST BE BETWEEN 1 AND 99999

1024 NON-STANDARD INDEXING IS NOT PERMITTED IN DO STATEMENTS

3600

1025	THE TERMINAL LABEL OF A DO MUST BE AN INTEGER CONSTANT
1026	THIS ENTRY NAME HAS BEEN USED PREVIOUSLY
1027	THE MAXIMUM PERMISSIBLE NUMBER OF ENTRY STATEMENTS IS 20
1030	THIS FORMAT STATEMENT IS UNLABELED
1031	IF THIS IS AN ARITHMETIC STATEMENT IT HAS NO LEFT HAND SIDE
1032	THE OBJECT OF AN ASSIGN OR ASSIGNED GO TO MUST BE A SIMPLE INTEGER VARIABLE
1033	STATEMENT LABELS IN GO-TO STATEMENTS MUST BE INTEGER CONSTANTS
1034	THE OBJECT OF A COMPUTED GO TO MUST BE A SIMPLE INTEGER VARIABLE
1035	THE PARAMETER TO THIS STATEMENT MUST BE AN INTEGER CONSTANT OR VARIABLE
1036	THE SUBROUTINE NAME IS NOT LEGITIMATE
1037	THE PARAMETER STRING IS NOT WELL-FORMED
1040	THE ASSIGNED STATEMENT LABEL IS NOT AN INTEGER
1042	SUBPROGRAM OR VARIABLE NAME USED AS ENTRY
1050	NO END CARD TERMINATES THIS ROUTINE
1051	THE ENTRY STATEMENT MAY NOT OCCUR INSIDE A DO LOOP
1053	THE INCREMENT IN A DO OR DO-IMPLYING LOOP MUST NOT BE ZERO
1502	ILLEGAL CHARACTER IN NUMERIC FIELD
1503	GREATER THAN 16 CHARACTERS IN OCTAL
1504	ILLEGAL CONVERSION-NUMBER TOO LARGE OR ILLEGAL CHARACTER
1700	INDEX VARIABLE NOT IN PROPER TABLE
1777	MORE THAN 25 ERRORS WERE DETECTED DURING COMPILATION. THE FIRST 25 ARE RECORDED ABOVE
2001	PROGRAM, SUBROUTINE OR FUNCTION CARD NOT FIRST CARD OF DECK
2002	IMPROPER FORMAT OF PROGRAM, SUBROUTINE, OR FUNCTION STATEMENT
2003	IMPROPER SUBROUTINE OR FUNCTION STATEMENT TERMINATION OR PARAMETER ERROR
2004	NAME NOT STARTING WITH ALPHABETIC CHARACTER
2005	DIMENSIONED VARIABLE ALREADY DIMENSIONED OR DECLARED EXTERNAL
2006	NO LEFT PARENS AFTER VARIABLE NAME
2007	VARIABLE DIMENSION IDENTIFIER NOT IN PARAMETER LIST
2010	MORE THAN 3 DIMENSIONS IN DECLARATION OF ARRAY
2011	NO RIGHT PARENTHESIS DELIMITER IN SUBSCRIPT DECLARATION

3600

2012 VARIABLE DIMENSIONED ARRAY USED IN COMMON

2013 MORE THAN 63 FORMAL PARAMETERS

2014 VARIABLE DIMENSION IDENTIFIER NOT INTEGER VARIABLE

2015 NO SLASH (/) SEPARATOR IN BLOCK DESIGNATION

2016 UNDEFINED SEPARATOR IN COMMON STATEMENT

2017 NON-CONSTANT SUBSCRIPT IN COMMON DIMENSIONING

2020 SUFFIX 5, 6 OR 7 NOT ON-TYPE OTHER-NAME

2021 TYPE OTHER 5, 6 OR 7 DOUBLY DEFINED

2022 ELEMENT LENGTH DESIGNATOR NOT (S) OR (/F)

2023 LEFT, RIGHT PARENTHESIS OR COMMA MISSING IN EQUIVALENCE

2024 TYPE OTHER 5, 6 OR 7 APPEARING WITH SUBSCRIPTS

2025 THIS EQUIVALENCE CAUSES A REORIGIN OF THE COMMON BLOCK

2026 FORMAL PARAMETER OR ADJUSTABLE DIMENSION IN EQUIVALENCE

2027 NON-CONSTANT SUBSCRIPT IN EQUIVALENCE

2030 DECLARED VARIABLE APPEARING IN EXTERNAL STATEMENT

2031 COMMON/EQUIVALENCE ERROR

2032 LEFT/RIGHT PARENS NOT MATCHING OR COMMA MISSING

2033 IMPLIED-DO ERROR IN DATA STATEMENT, NO = AFTER DO VARIABLE, OR NON-CONSTANT DO LIMITS, OR DO VARIABLE DOES NOT AGREE WITH SUBSCRIPT

2034 NO = AFTER IDENTIFIER

2035 A VARIABLE APPEARS WITH SUBSCRIPTS BUT HAS NOT BEEN DIMENSIONED

2036 DATA TO ADJUSTABLE DIMENSIONED OR PARTIAL WORD ARRAY

2037 MULTIPLE DATA TO NON-DIMENSIONED VARIABLE

2040 DUPLICATE BLOCK NAME

2041 EQUIVALENCE OVERLAPS COMMON BLOCKS

2042 FORMAL PARAMETER OR PROGRAM NAME APPEARS IN COMMON DECLARATION

2043 VARIABLE NAME GREATER THAN 8 CHARACTER OR NO COMMA SEPARATOR

2044 NON-CONSTANT DATA IN LIST

2045 DOUBLY DEFINED VARIABLE IN COMMON

2046 REPEAT COUNT MUST BE AN INTEGER CONSTANT 1-32767

2047 VARIABLE EQUATED TO ITSELF+N

2050 (S) IS NOT AN INTEGER 1 THRU 7, OR (/F) IS NOT A DIVISOR OF 48

3600

2051	ONE OF THE VARIABLES HAS BEEN DEFINED IN A PREVIOUS TYPE STATEMENT
2052	DOUBLY DEFINED FORMAL PARAMETER
2144	DECLARED VARIABLE TABLE LIMIT EXCEEDED
2146	COMPILER COMMON OR BLOCK TABLE EXCEEDED
2147	EQUIVALENCE TABLE LIMIT EXCEEDED
2150	MACHINE OR TABLE ERROR, VARIABLE NOT IN DIMENLIST
2201	COMMA MISSING IN PARAMETER LIST OR VARIABLE MORE THAN 8 CHARACTERS
2202	IMPROPER USE OF FUNCTION NAME
2203	ILLEGAL SEQUENCE OR USE OF OPERATORS
2204	MIXED MODE-TYPE 5 AND/OR 6 AND/OR 7
2205	ERROR IN PROCESSING COMMON EXPRESSIONS IN ARITHMETIC STATEMENT
2206	ILLEGAL OPERATOR OR MISSING OPERATOR
2207	ILLEGAL REPLACEMENT IN ARITHMETIC STATEMENT
2210	AN * HAS BEEN INSERTED FOR THE APPEARANCE OF N (,) (,) V . OR) N
2211	MORE THAN 63 PARAMETERS IN CALL OR FUNCTION
2212	LOGICAL OR MASKING OPERATOR IN CALL PARAMETER
2213	ILLEGAL REPLACEMENT APPEARS IN AN EXPRESSION
3700	MISSING INDEX FUNCTION
3702	FORMAT STATEMENT CONTAINS A NUMBER GREATER THAN 377 OCTAL
3703	FORMAT STATEMENT CONTAINS UNNECESSARY PLUS OR MINUS SIGN
3704	FORMAT STATEMENT CONTAINS A ZERO OR ILLEGAL MULTIPLIER
3705	FORMAT STATEMENT CONTAINS AN ILLEGAL CHARACTER
3706	FORMAT STATEMENT CONTAINS A MISPLACED PERIOD, SIGN, OR DELIMITER
3707	FORMAT STATEMENT NOT SEPARATED BY LEGAL DELIMITER
3710	FORMAT STATEMENT CONTAINS CHARACTERS OUTSIDE LAST CLOSE PARENTHESIS
3711	FORMAT STATEMENT CONTAINS AN IMPROPER ADJACENT SEPARATOR PAIR
3712	FORMAT STATEMENT CONTAINS MORE THAN 136 HOLLERITH CHARACTERS
3713	FORMAT STATEMENT IS MISSING A CLOSING * OR THERE ARE MORE THAN 136 CHARACTERS BEFORE CLOSING *
3714	FORMAT STATEMENT CONTAINS SCALING FACTOR GREATER THAN 13
3715	FORMAT STATEMENT CONTAINS AN ILLEGAL C-PATTERN
3716	FORMAT STATEMENT CONTAINS AN ILLEGAL D FIELD

3600

3717	FORMAT STATEMENT IS MISSING A CLOSING PARENTHESIS
3720	FORMAT STATEMENT CONTAINS MORE THAN TEN PARENTHESIS LEVELS
3721	FORMAT STATEMENT CONTAINS A MISSING OR ZERO FIELD WIDTH OR MISSING D FIELD FOLLOWING DECIMAL
3722	FORMAT STATEMENT CONTAINS AN ELEMENT WHICH WILL NEVER BE REACHED
3723	FORMAT STATEMENT CONTAINS ILLEGAL ( ) DELIMITER COMBINATION
4001	FIRST WORD OF ASF IS NOT AN IDENTIFIER
4002	ASF INVOLVES ITSELF
4003	ERROR IN ASF SET-UP -NO END IN STRING
4004	ARITHMETIC STATEMENT FUNCTION DOUBLY DEFINED
4005	ASF PARAMETER ERROR
4021	POSSIBLE MACHINE ERROR. ARITHMETIC FAULT TYPE NOT RECOGNIZED
4022	POSSIBLE MACHINE ERROR. MACHINE CONDITION TEST NOT RECOGNIZED
4023	THE PARAMETER OF THIS STATEMENT MUST BE TYPE INTEGER
4024	I IS OUTSIDE THE PERMITTED RANGE
4025	STATEMENT NUMBER IS OUT OF RANGE
4026	UNIT NUMBER MUST BE A SIMPLE INTEGER VARIABLE OR AN INTEGER CONSTANT
4030	UNIT NUMBER MUST BE FOLLOWED BY)
4031	AN IF UNIT STATEMENT MUST HAVE 2-4 BRANCH POINTS
4100	STATEMENT NUMBER IS OUT OF RANGE
4101	BRANCH POINT ERROR IN IF STATEMENT
4102	LOGICAL IF IS FORMED INCORRECTLY
4103	TWO OR MORE RELATIONAL OPERATORS IN THE SAME RELATIONAL SUB-EXPRESSION
4104	LOGICAL EXPRESSION INCORRECTLY FORMED
4105	RELATIONAL SUB-EXPRESSION FORMED INCORRECTLY
4106	THE .NOT. OPERATION MUST BE FOLLOWED BY EITHER (OR AN OPERAND
4107	POSSIBLE MACHINE ERROR. LOGICAL OPERATOR NOT RECOGNIZED
4110	POSSIBLE MACHINE ERROR IN EVALUATING LOGICAL EXPRESSION
4112	LOGICAL CONNECTIVE MUST BE FOLLOWED BY (OR AN OPERAND
4113	A LOGICAL SUBEXPRESSION MAY NOT BEGIN WITH AN OPERATOR
4114	EXCESS LEFT PARENTHESIS IN LOGICAL EXPRESSION
4200	MASKING ARITHMETIC EXPRESSION TOO LONG

3600

4201	ARITHMETIC SUB-EXPRESSION IN MASKING STATEMENT NOT FULLY PARENTHESIZED
4202	FUNCTION CALLED INCORRECTLY
4210	MASKING EXPRESSION FORMED INCORRECTLY
4212	THE FIRST ELEMENT OF A BOOLEAN EXPRESSION MUST BE AN OPERAND,) , OR . NOT.
4213	) MAY BE FOLLOWED ONLY BY . AND. , . OR. , OR)
4214	THE OPERATORS . AND. , . OR. MUST BE FOLLOWED BY EITHER (, . NOT. , OR AN OPERAND
4215	MASKING OPERANDS MUST BE REAL OR INTEGER
4220	THE REPLACEMENT VARIABLE FOR AN EXPRESSION USING LOGICAL OPERATORS MUST BE LOGICAL IF THE STATEMENT IS LOGICAL, OR REAL OR INTEGER IF IT IS MASKING
4401	EQUIVALENCE ATTEMPTS TO REORDER COMMON
4402	DIMENSION OF VARIABLE GREATER THAN 32767 OR TOTAL DIMENSIONED AREA EXCEEDS 32767
5001	ILLEGAL MARK IN COLUMN SIX
5002	UN-RECOGNIZED STATEMENT
5003	ASSUMED DIMENSION STATEMENT
5004	ASSUMED BACKSPACE STATEMENT
5005	ASSUMED WRITE-TAPE STATEMENT
5006	ASSUMED SUBROUTINE STATEMENT
5010	ASSUMED WRITE-OUTPUT-TAPE STATEMENT
5011	TOO MANY CHARACTERS IN IDENTIFIER -MAX 8
5012	ASSUMED SENSE-LIGHT STATEMENT
5013	ASSUMED IF-DIVIDE-FAULT STATEMENT
5014	ASSUMED IF-OVERFLOW-FAULT STATEMENT
5015	ASSUMED IF-EXPONENT-FAULT STATEMENT
5016	STATEMENT TOO LONG. TABLE OVERFLFOW IN ARITHMETIC PROCESSING
5017	UN-MATCHED PARENTHESES
5020	ILLEGAL USE OF BOOLEAN OR RELATIONAL OPERATOR
5021	ASSUMED IF-SENSE-LIGHT STATEMENT
5022	ASSUMED IF-SENSE-SWITCH STATEMENT
5023	ASSUMED BUFFER OUT STATEMENT
5024	ASSUMED EQUIVALENCE STATEMENT
5025	IMPROPER LENGTH FOR HOLLERITH CONSTANT

3600

5026	ILLEGAL USE OF PERIOD
5027	ILLEGAL CONSTANT TYPE
5030	STATEMENT ENDS WITH ASTERISK
5031	ILLEGAL CHARACTER IN LABEL FIELD OR ZERO USED AS STATEMENT LABEL (MAY NOT INHIBIT EXECUTION)
5032	CARD HAS LABEL AND MARK IN COLUMN 6
5040	TOO MANY SUBSCRIPT INDICES
5044	EMPTY PARENTHETICAL EXPRESSION
5045	LIMIT FOR NON-STANDARD SUBSCRIPT EXPRESSIONS EXCEEDED
5046	NUMBER OF CONSTANTS EXCEEDS COMPILER LIMIT
5047	SUBSCRIPT ON NON-DIMENSIONED VARIABLE OR IMPROPER USE OF DECLARED VARIABLE
5050	NUMBER OF FUNCTIONS EXCEED COMPILER LIMIT
5051	LABELED OR IMBEDDED BLANK STATEMENT--CONTINUE ASSUMED
5052	NUMBER OF IDENTIFIERS EXCEEDS COMPILER LIMIT
5053	LIMIT FOR STANDARD INDEX FUNCTIONS EXCEEDED
5060	ASF PARAMETERS DO NOT AGREE IN NUMBER
5061	TOO MANY ASF ENTRIES, POSSIBLY DUE TO RECURSIVE CALL
5062	ASF PARAMETER LIST NOT ENDED
5063	ILLEGAL USE OF PROGRAM NAME
5074	S OF IF (L)S CANNOT BE ANOTHER IF(L)S OR DO-STATEMENT
5076	TYPE DOUBLE ASSUMED
5077	MACHINE MALFUNCTION (DETECTED BY SCANNER)
5400	TOO MANY BRANCHES IN COMPUTED GO TO
5402	EXTRANEOUS COMMAS IN COMPUTED-GO TO STATEMENT
6001	PARENTHESIS USAGE OR DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL IN I/O DATA LIST
6002	WRONG FORMAT OF I/O STATEMENT. DATA LIST WAS NOT YET PROCESSED
6003	TAPE NUMBER IN I/O STATEMENT IS GREATER THAN 64
6004	PARITY IN I/O STATEMENT IS NOT EQUAL TO 0 OR 1
6005	INVALID SUBSCRIPTING OCCURS IN DATA LIST BECAUSE OF IMPLIED DO-LOOP
6006	INPUT OF DATA INTO A CONSTANT IS ILLEGAL
6010	I/O TYPE 5, 6 OR 7 PROHIBITED

3600

7001 TYPE OTHER OPERAND DOES NOT APPEAR IN DEVARLIST. POSSIBLE MACHINE ERROR

7002 ERASABLE STORAGE REQUIRED IS TOO LARGE

7003 TYPE OTHER INTERMIXED IN ARITHMETIC

7004 LOGICAL OR BYTE SIZED OPERAND(S) USED IN EXPONENTIATION

7005 IMPROPER OPERAND

7007 ERROR IN GENERATING IN-LINE FUNCTION CODE

Assembly Errors

The following errors are detected in FORTRAN programs during the assembly phase of compilation:

U The rightmost identifier on the line is not defined.

If the first character of the identifier is a period (eg., .n or.-n), it indicates that statement number n is missing.

If the first character is alphabetic, it indicates that the identifier did not appear to the left of an = (replacement) operator, or in a READ or subroutine argument list.

D The identifier to the left of the line is multiply defined. This usually indicates that the same statement number n (translated from .n or ..n) appears more than once in a subprogram.

C Data statement attempted to initialize variables in blank or numbered common block.

STANDARD ERROR PROCEDURE

Execution diagnostics are output during object time; they may have one of three formats:

A. ERROR DETECTED DURING {INPUT / OUTPUT} CONVERSION ON UNIT n

(message)
{FORMAT (locn) pointer to faulty format specification ↑ / DATA (locn) pointer to faulty data ↑}

A= (value of A) Q= (value of Q)

ERROR DETECTED IN ROUTINE {Name}
(program sequence trace with relative location of the call)

The program sequence trace begins with the subprogram which called the Q8QERROR routine. The calling programs are listed in sequence until the main program is encountered or until 60 lines have been printed. The relative location from which the call was made is indicated with each subprogram name.

Example:

Assume that a program has executed the following set of calls:

PROGRAM MAIN	at MAIN +00005	-	calls SUB1
SUBROUTINE SUB1	at SUB1 +02561	-	calls SUB2
SUBROUTINE SUB2	at SUB2 +12003	-	calls FUN1
FUNCTION FUN1	at FUN1 +00205	-	calls TSH.

TSH. detects an error.

The program sequence trace is as follows:

ERROR DETECTED IN ROUTINE I/O TSH.
CALLED FROM FUN1 +00205
CALLED FROM SUB2 +12003
CALLED FROM SUB1 +02561
CALLED FROM MAIN +00005

Formatted I/O Messages	Description
IMPROPER FORMAT SPECIFICATION	Error in format specification; such as 4FE12.4.
PARENTHESIS NESTING TOO DEEP	Nesting in a format statement has exceeded 10 deep.
EXTRA RIGHT PARENTHESIS	Unmatched right parenthesis encountered.
FORMAT EXCEEDS LINE LENGTH	The format statement is either calling for the translation from, or the insertion of, data into the record area outside the limits. Normally the limits are 10 words on card input, 17 words on output, and 10 words on punching.
EXPONENT OVERFLOW DURING CONVERSION	A calculation during conversion has overflowed the limits of the floating point format.
ZERO FIELD WIDTH	A format specification such as E0.4 has been encountered.
INVALID CHARACTER ON INPUT	An invalid character in the input field, such as an 8 or 9 in octal input, or a letter in numeric input.
F P NUMBER IN I-FORMAT	A floating point number has been read under I format.
INTEGER INPUT TOO BIG	A number in I-input field exceeded machine capacity. On output, a format field which is too small to contain the number will be filled with asterisks.
LIST EXCEEDS DATA	A list in binary read called for more data than the logical record contained.

B. (message)
A = (value of A) Q = (value of Q)
ERROR OCCURRED ON UNIT NO n.
ERROR DETECTED IN ROUTINE (name of routine)
(program sequence trace)

Message	Description
UNCHECKED END OF FILE UNCHECKED PARITY ERROR	On last read operation, end-of-file or parity error encountered was not checked for. End of file checked by: IF (EOF, i) n1, n2 or by IF(UNIT, i) n1, n2, n3 which takes the n3 branch. Parity errors are checked by IF (IOCHECK, i) n1, n2.
STANDARD REFERENCE TO NON-STANDARD TAPE	Standard operation requested on a unit used for buffer operations.
IMPROPER BUFFER IN/OUT PARAMETERS	LWA may be greater than or equal to FWA and both must be in the same bank or mode parameter is improper.
SYNC ERROR	End of file encountered in middle of logical record during binary read.
EOF ATTEMPTED ON LUN GT 59	Attempt to write an end-of-file on SCOPE system units.

C. (message)

Message	Description
W-R SEQ	Read operation attempted on unit which has been written, but not backspaced or rewound; unit number is listed.
BIG UNIT	Unit number reference exceeds 79.
UN REQ.	IOP entered with function code larger than 37_8 (undefined request).
BYPASS	Read on a bypass unit is illegal.

Q8QERROR

The standard error procedure routine is also available to the programmer to print execution error messages not provided by the compiler. This routine is included in the library and may be called as follows:

```
CALL Q8QERROR(k,m)
```

k the error return key, may be an integer expression (arithmetic).

m the location of first word of programmer's error message, may be a simple or subscripted variable.

If k= 0, the job will terminate after printing the message followed by EXECUTION DELETED.

If k≠ 0, control will return to the program calling Q8QERROR after printing the message.

The error message must be in BCD and terminated with a period; there may be no imbedded periods. The message will be printed on the standard output unit similar to standard execution errors.

A call to the Q8QERROR routine generates the following COMPASS calling sequence:

```
BRTJ    ($)Q8QERROR
SLJ     *+2
02      DICT.
00      ($)K
00      ($)M
```

Form of the output from Q8QERROR:

(message)
A = (Value of A) Q = (Value of Q)
ERROR DETECTED IN ROUTINE (name)
(program sequence trace)

Below is a list of the standard diagnostics and the name of the routine which issues the diagnostic:

ROUTINE NAME	MESSAGES
SQRTF	NEG ARG.
SINF	X >2 **36.
EXPF	ARG GR THAN 709.
LOGF	ARG = 0/NEG.
POWRF	EXP = 0/NEG. ; B * LN (A) GT 709.; BASE LSTHN ZERO.
ASINF	X GT 1.
XTOI	EXP OVERFLOW.; X = 0, I = 0 OR NEG.

ROUTINE NAME	MESSAGES
XFIXF	INTEGER TOO BIG.
ITOJ	OVERFLOWED INTG.; J GT 47.; I = 0, J = 0/NEG.
TANF	ARG GT 2**36.
COTF	ARG = 0.
Q8QMODF	Q = ZERO.
Q8QXMODF	DIVISOR IS ZERO.
DSQRT	NEG ARG.
DLOG	ARG = 0/NEG.
DSIN	X GR THAN 2**83.
DEXP	ARG GR THAN 709.
ATAN2	ATAN (0/0) UNDEFINED.
DATAN2	ARCTAN (0/0) UNDEFINED.
IDINT	INTEGER TOO BIG.
DMOD	ARG2=0, ARG1/0 UNDEFINED.
DTOI	D=0, EXP= 0/NEG.
DIMF	OVERFLOW ERROR.
XDIMF	OVERFLOW ERROR.
Q2Q07331	ILLEGAL POWER.
Q8QSENLT	SENSE LIGHT OUT OF RANGE.
Q8QIFSSW	SENSE SWITCH OUT OF RANGE.
SLI.	TYPE OTHER NOT ALLOWED IN I/O LISTS.

3400 EXECUTION DIAGNOSTICS

Math Library Error Messages (three forms)

number of arguments = 0

ILLEGAL INPUT TO name, message.

CALLED FROM xxxxx

number of arguments = 1

ILLEGAL INPUT TO name, message.

CALLED FROM xxxxx ARG = yyyyyyyyyyyyyyyy

number of arguments = 2

ILLEGAL INPUT TO name, message.

CALLED FROM xxxxx ARG1=yyyyyyyyyyyyyyyy ARG2=zzzzzzzzzzzzzzzz

xxxxx is the address from which the library subroutine was called.

zzzzzzzzzzzzzzzz
yyyyyyyyyyyyyyyy } are arguments (in octal) with which the subroutine was entered.

Messages

SQRTF, NEG ARG.

EXPF, ARG GT 709.

LOGF, ARG IS NEG OR ZERO

SIN/COS, ABS (ARG) GE 2**36.

TANF, ABS (ARG) GE 2**36.

XDIMF, I1-I2 OVERFLOWED.

DIMF, A1-A2 OVERFLOWED.

MODF, ARG2=0, ARG1/0 IS UNDEFINED.

XMODF, ARG2=0, ARG 1/0 UNDEFINED.

XFIXF, ARG GT MAX.

XINTF, ARG GT MAX.

POWRF, BASE=0, EXP LE ZERO.

POWRF, NEGATIVE BASE.

POWRF, A**B OVERFLOWED.

XTOI, BASE=0, EXP LE ZERO.

ITOJ, EXP GT 47.

ITOJ, BASE=0, EXP LE ZERO.

ITOJ, I**J OVERFLOWED.

ASIN/ACOS, ABS(ARG) GT 1.

COMPLEX**REAL, DOUBLE, OR COMPLEX, RESULT IS MULTIVALUED.

Q2Q07220, BASE=0, EXP=NEG OR ZERO.

DSQRT, NEG ARG.

DSIN/DCOS, ABS (ARG) GE 2**83.

DLOG, ARG IS NEG OR ZERO.

DEXP, ARG GT 709.

CONVERSION OF REAL, DOUBLE, OR COMPLEX TO INTEGER, ARG TOO LARGE.

CANG/ATAN2, ATAN (0/0) UNDEFINED.

DATAN2, ARCTAN (0/0) IS UNDEFINED.

IDINT, ARG TOO LARGE TO FIX.

DMOD, ARG2=0, ARG1/0 IS UNDEFINED.

I/O Messages

The general form is:

message

CALLED FROM xxxxx

xxxxx is the address from which the I/O routine was called.

Message	Description
MODE OTHER NOT ALLOWED IN BUFFER IN-OUT	Mode specification in buffer statement is not 0 or 1.
LIMITS ON BUFFER IN-OUT INCONSISTENT	Last word address is greater than first word address in buffer statement.
LIST EXCEEDS DATA	End of logical record reached in a binary read before all list elements are processed.
SYNC. ERROR	In a binary read: end-of-file was encountered or more than 127 records were read within a logical record; or the tape is positioned on a backspace request.
TYPE OTHER NOT ALLOWED IN I/O LISTS	A list element is specified which is not 1 or 2 words in length.
IF (EOF/IOCHECK) ON NON-STD UNIT	IF (IOCHECK) or IF(EOF) statements are used on a non-standard tape unit.
HANGING PARITY OR EOF ON UNIT n	Unchecked parity or end of file condition remains on unit n when another operation is requested.
STAND. REF. TO NON-STAND. UNIT n	A standard request is given to a tape on which a buffered operation was performed.

Message	Description
NON-STAND. REF. TO STAND. UNIT n	A buffered request is given to a tape on which a standard operation was performed.
ERROR IN IOPACK BIG UNIT UNIT n	A logical unit greater than 80 is specified.
ERROR IN IOPACK UNIT REQ UNIT n	The request to IOP was not a recognizable code.
ERROR IN IOPACK W/R SEQ UNIT n	A read request follows a write request.
ERROR IN IOPACK WRITE CK UNIT n	An attempt was made to write on unit n five times with no success. This comment is also printed on the typewriter.
ERROR IN IOPACK BSP PAR UNIT n	Sufficient core was not available to permit backspacing by a bookkeeping means. A record was read in both binary and BCD mode to determine the backspacing, and parity errors resulted.
ERROR IN IOPACK BAD OPER UNIT n	An I/O request is rejected by SCOPE for reasons other than availability.

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET

Pub. No. 60132900

THIS FORM IS NOT INTENDED TO BE USED AS AN ORDER BLANK. YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

CUT ALONG LINE

PRINTED IN U.S.A.

FROM NAME: _____

BUSINESS ADDRESS: _____

_____

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

CUT ALONG LINE

MD248

FOLD

FOLD

STAPLE

STAPLE

► ►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

3400/3600/3800 FORTRAN REFERENCE MANUAL

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60132900

Litho in U.S.A.