3100
3200
3300
3500

COMPUTER SYSTEMS
FORTRAN
REFERENCE MANUAL

CONTROL DATA
CORPORATION

Additional copies of this manual may be obtained from the nearest Control Data Corporation Sales office listed on the back cover.

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

August, 1966
Pub. No. 60057600, Rev. B

PREFACE

This reference manual was written for programmers using the FORTRAN system. The manual assumes a basic knowledge of the FORTRAN language although it includes a section on general properties for users, less familiar with the construction of the FORTRAN language.

The manual contains information required to produce and run a FORTRAN job.

CONTENTS

3100/3200/3300/3500 FORTRAN SYSTEM 1

The FORTRAN system provides a convenient language for expressing mathematical and scientific problems in familiar notation.

A set of FORTRAN statements, presented as a source program to the FORTRAN compiler, produces an object program that contains the machine language commands for solving a problem. Compilation progresses sequentially, from one subprogram to the next; each subprogram is independently compiled. Once compiled, a program may be repeatedly loaded and run on the computer with varying sets of data. The compiler operates in conjunction with the SCOPE monitor system of a Control Data® 3100, 3200, 3300 or 3500 computer. It generates programs to be executed under SCOPE control. Source programs require little modification to be accepted by FORTRAN compilers for larger CONTROL DATA computers.

1.1 MACHINE CONFIGURATION

Basic configuration for compiling a source program:

- 8K memory unit
- Magnetic tape library unit
- Input device (card reader or magnetic tape unit)
- Output device (printer or magnetic tape unit)
- Punch output unit (card punch or magnetic tape unit)
- Two magnetic tape scratch units

1.2 OPTIONAL OUTPUTS

Outputs that may be selected by the programmer include:

- Relocatable binary cards or card images
- Source program listing
- Assembly language listing of machine instructions
- Load-and-go object program for immediate execution

Diagnostic messages are printed when the compiler detects coding errors.

1.3 EXPRESSIONS

An expression is a constant, variable (simple or subscripted), function, or any combination of these separated by operators and parentheses, written in compliance with the rules for constructing a particular type of expression.

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

1.4 STATEMENTS

The FORTRAN elements - expressions, operators, and operands - may be combined to form two types of statements, executable and non-executable. An executable statement performs a calculation or directs control of the program; a non-executable statement provides the compiler with information regarding variable structure, array allocation, and storage-sharing requirements. A group of FORTRAN statements make up a source program.

Statements can be divided into four classes:

- Declarative
- Replacement
- Control
- Input/Output

Declarative statements permit a programmer to (a) define the mode of a variable as character, real, integer, or other; (b) enter data; (c) reserve storage common to more than one subprogram or main program; and (d) overlay the same storage locations with variables and arrays during program execution.

Arithmetic replacement statements incorporate expressions for addition, subtraction, multiplication, division, and exponentiation. Logical replacement statements may include relational and logical operators.

Control statements alter the sequence of program execution conditionally or unconditionally.

Input/Output and encode/decode statements permit transfer of data from one storage location to another, or between computer storage and external equipments. Conversion and editing specifications provide diversity in input/output formats.

Masking operations are available through the FORTRAN library routines; internal and external machine conditions may be tested by FORTRAN library functions.

1.5 SOURCE PROGRAMS

A source program consists of one main program and/or several subprograms; they may be compiled individually and run as single programs.

The following chapters review characteristics of constants, variables, operators, operands, and expressions; set forth rules and conditions for use of FORTRAN statements; and describe organization of source decks.

1.6 CODING

Each FORTRAN program and subprogram is constructed of symbolic characters, identifiers, and operators arranged in statements to be punched on standard 80-column cards. Coding uses the following character set:

Alphanumeric characters:

letters A through Z
numbers 0 through 9

Special characters:

∧	blank
=	equals
+	plus
-	minus
*	multiplication
**	exponentiation
/	division or inter-record spacing
(	left parenthesis
)	right parenthesis
,	comma or separator
.	decimal point
$	delimits multiple statements written on same line

Blanks may appear anywhere in a source statement. They are significant only in Hollerith constants or FORMAT specifications.

1.7 CODING FORMS

Coding is written on forms with the following format:

Columns	Content
1-5	Statement label
6	Continuation designator (non-zero character)
7-72	Statements
73-80	Identification and sequencing
or	
1	C - Comment designator
2-72	Comments
73-80	Identification and sequencing

A line contains a string of up to 72 FORTRAN characters. The character positions in a line, columns, are numbered consecutively, 1 through 72.

A C in column 1 identifies the line as a comment; comments are for the convenience of the programmer and permit him to describe the program steps; they do not influence the program. A comment may be inserted at any point in the program. Comment cards are listed along with the source statements when the source list option is selected.

Columns 1 through 5 may be blank or may contain a label that identifies the line for reference elsewhere in the program. A statement number (label) must be unique within a subprogram, and in the range 1 through 32767.

A statement that is labeled and never referenced (a null) causes an informative diagnostic during compilation but does not inhibit execution of the compiled program.

The compiler ignores blanks and leading zeros in statement numbers.

Statements in the formats outlined in this manual appear in columns 7 through 72. A statement that exceeds the 66 characters allowed on a single card may be continued on successive cards.

The size of a source statement and, consequently, the number of allowable cards per statement are limited according to the equation:

$2n + m \leq 500$ characters

n = number of identifiers

m = number of symbols and constants

This permits at least five cards (4 continuation cards) per statement with the worst case being 167 single-character identifiers and 164 operators. Continuation cards may not be labeled; columns 1 through 5 must be blank. A character other than zero in column 6 designates continuation.

More than one statement may be written on a line (card) by using the $ to separate the statements, subject to the following rules:

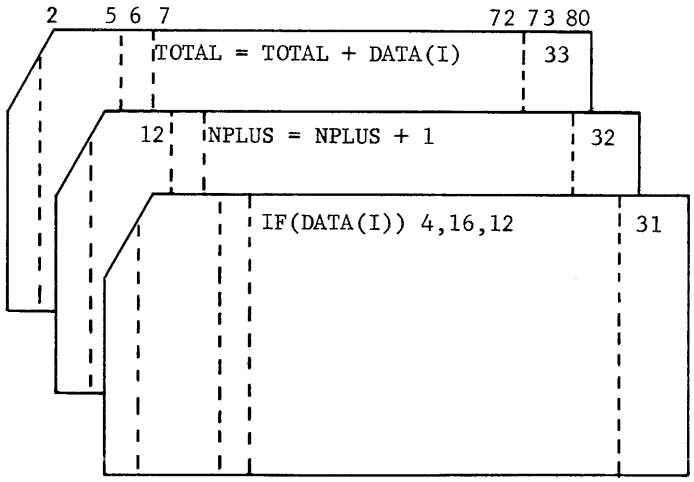
The statement following $ may not be labeled.

The $ may not be used with FORMAT statements or continuations of FORMAT statements.

A statement ends when the compiler reads a $ or a new record having column 6 blank.

The compiler does not interpret column 73 through 80. These columns are for sequencing or program identification.

SAMPLE FORTRAN CARDS



Three statements are shown, one on each card. One is labeled; all three are sequenced. Because no statement is continued, column 6 is blank.

SAMPLE FORTRAN CODING FORM

1 5	7 72	80
	PROGRAM BIGJOB	24
	DIMENSION DATA(800)	25
	SQUARE = 0.	26
	TOTAL = 0.	27
	NPLUS = 0	28
	NZERO = 0	29
	DO 20 I = 1, 800	30
	IF (DATA(I)) 4,16,12	31
12	NPLUS = NPLUS + 1	32
	TOTAL = TOTAL + DATA(I)	33
4	SQUARE = SQUARE+DATA(I)*DATA(I)	34
	GO TO 20	35
16	NZERO = NZERO + 1	36
20	CONTINUE	37
	B = NPLUS	38
	.	
	.	
	.	

ELEMENTS OF FORTRAN 2

2.1 CONSTANTS

FORTRAN accepts four basic types of constants: integer, octal, real, and Hollerith. The type of a constant is determined by its form. Each real constant occupies two consecutive computer words (24 bits per computer word). Each of the other three types occupies only one computer word.

If a constant exceeds the allowed range, the statement in which it is used is rejected during compilation and a diagnostic is provided.

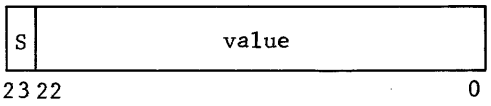
2.1.1 INTEGER

An integer constant consists of up to 7 decimal digits in the range $-2^{23} < n < 2^{23}$ (from -8,388,607 to 8,388,607).

Examples:

63	-3141592	8388607
247	3674631	-464646

The structure of the translated constant is:



S = 1; value is complemented (negative)
0; value is uncomplemented (positive)

2.1.2 OCTAL

An octal constant consists of up to 8 octal digits terminated by a B in the form

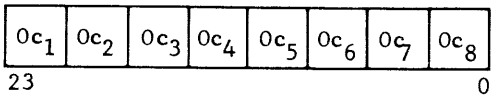
$n_1 \ldots n_i B$

A minus sign before the octal number designates the seven's complement.

Examples:

00B	23232323B	77777700B
00077777B	77B	-31314672B

The structure of the translated constant is:



Oc_i is a 3-bit octal digit.

2.1.3 REAL

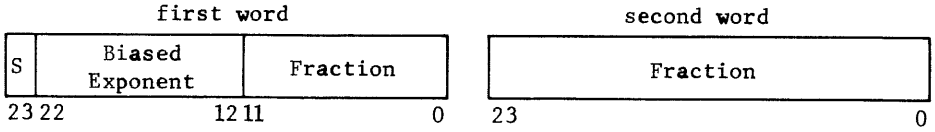
A real constant is represented by a string of up to 11 digits in the range $0 \leq n \leq 2^{36}-1$. It may be expressed with a decimal point or with an exponent representing a power of ten or both in the forms:

nE	nE±s	n.	n. E±s
n.n	n. nE±s	.n	.nE±s

s is the exponent to the base 10. The range of s is 0 through 308. The constant may be signed. The maximum value is .68719476735E308.

Examples: 3.1415768 -314. .0749162
.31415E1 3459E05 31.41592E-01
-.31415E+01

Structure of translated constant:



S = 1; fraction and biased exponent are complemented
0; fraction and biased exponent are uncomplemented

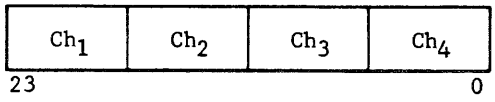
2.1.4 HOLLERITH

A Hollerith constant is a string of characters of the form nHw; n is an unsigned decimal integer, 1 through 4, that represents the length of the field, w. Spaces are significant in the field. When n is less than 4, the word is left-justified with BCD blanks (60_8) filling the remainder of the word. When n is greater than 4, the statement is rejected and a diagnostic is provided.

An alternate form of a Hollerith constant is nRw. When n is less than 4, the computer word is right-justified with zero fill. When n is greater than 4, the statement is rejected and a diagnostic is provided.

Examples: 2HOK 3HSUM 3ROUT
4HERGO 4R3600 1H)

Structure of translated word:



Ch_i is a 6-bit BCD character.

A Hollerith constant occupies one computer word and is represented by internal BCD codes treated as integers.

2.2 VARIABLES

Variable names are alphanumeric identifiers that represent specific storage locations.

The FORTRAN compiler recognizes simple and subscripted variable names. A simple variable name represents a single quantity; a subscripted variable name represents a single quantity within an array of quantities. The variable type is designated explicitly in a type declaration (Chapter 5) or implicitly by the first letter of the variable name. A first letter of I, J, K, L, M, or N indicates a fixed point (integer) variable; any other first letter indicates a floating point (real) variable.

2.2.1 SIMPLE VARIABLES

A simple variable name identifies the location in which a variable value can be stored. Any integer value in the range $|n| \leq 2^{23}-1$ may be referenced by a simple integer variable name.

Examples: N LOX M58
K2SO4 NOODGE M 58

Because spaces are ignored in variable names, M58 and M 58 are identical.

Any signed value from 10^{-308} to 10^{308} may be referenced by a simple floating point variable.

Examples: VECTOR A65302
BAGELS BATMAN

2.2.2 SUBSCRIPTED VARIABLES

An array is a block of successive memory locations comprising the elements of the array. Each element of an array is referenced by the array name plus a set of subscripts. The type of an array is determined by the array name or a type declaration. Arrays may have one, two, or three dimensions; the maximum number of array elements is the product of the dimensions. The maximum number of words used in an array cannot exceed 32767. The array name and its dimensions must be declared at the beginning of the program in a DIMENSION or COMMON statement.

2.2.3 SUBSCRIPT FORMS

A subscript has one of the following forms; c and d are integer constants and I is a simple integer variable.

(c*I±d) (I±d) (c*I)
(I) (c)

More than three subscripts cause a compiler diagnostic. Program errors may result if subscripts are larger than the dimensions initially declared for the array. A single subscript notation may also be used for a two- or three-dimensional array if it is the structural location of the variable. However, the elements of a single-dimensioned array $A(d_1)$ may not be referred to as A(I, J, K) or A(I, J). Diagnostics will occur if this is attempted.

Examples: A(I, J) B(I+2, J+3, 2*K+1) Q(14)
P(KLIM, J, LIM+5) SAM(J-6) A(133)
B(1, 2, 3) A(233)

At no time during program execution can a simple integer variable used as an index variable take on a value greater than 32767.

2.2.4 ARRAY STRUCTURE

Elements of an array are stored by column in ascending storage locations. Type real arrays require two storage locations for each array element; type character arrays, however, are stored 4 array elements per location.

The location of an array element with respect to the first element is determined by the maximum array dimensions and the type of the array.

The first element of array A(I, J, K) is (1, 1, 1). The location of element A(i, j, k) with respect to A(1, 1, 1) is

$$\text{Loc } A(i,j,k) = \text{Loc } A(1,1,1) + \left[(i-1)+(j-1)*I+(k-1)*I*J\right]*E$$

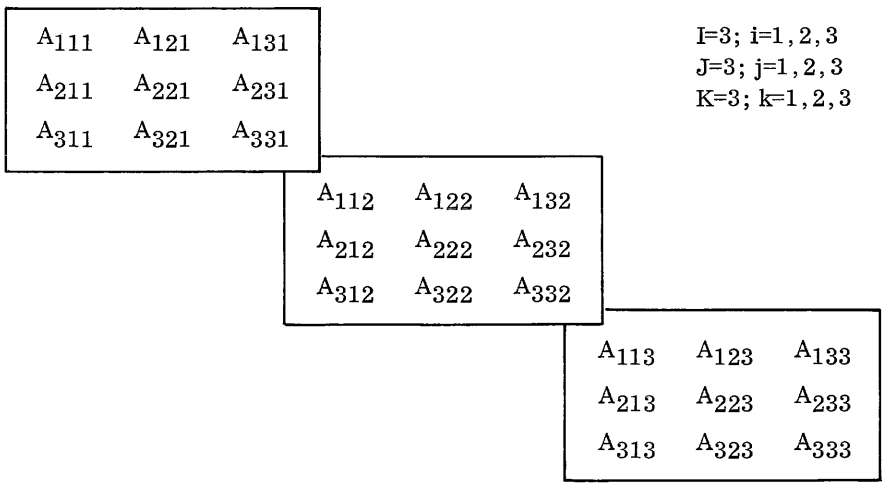
The quantity in braces is the subscript expression. E is the element length (the number of storage locations required for each element of the array). For integer arrays, E = 1; for real arrays, E = 2; for character arrays, E = 1/4. Subscripts i, j, k may be any of the allowed subscript forms.

Factoring the expression produces

a base address:	S, the first word of A(1, 1, 1)
a constant addend:	-(1-I+I*J)*E
an index function:	(i+I*j+I*J*k)*E

When i, j, k are other than simple variables, for example, c*I±d, constants such as d appear in the constant addend.

Example: In the array declared as A(3, 3, 3):



The elements of this real array are stored two words per element starting with A(1,1,1) in S, the lowest location reserved for the array.

locations	array element Ai,j,k
S, S+1	A_{111}
S+2, S+3	A_{211}
S+4, S+5	A_{311}
S+6, S+7	A_{121}
S+8, S+9	A_{221}
S+10, S+11	A_{321}
S+12, S+13	A_{131}
.	.
.	.
.	.
S+48, S+49	A_{133}
S+50, S+51	A_{233}
S+52, S+53	A_{333}

Referring to the example, if Loc A (1,1,1)=S: the locations of A(2,2,3) with respect to A(1,1,1) are:

Loc A(2,2,3) = Loc A(1,1,1) + [(2-1)+(2-1)*3+(3-1)*3*3] *2 = S+44, S+45

The following relaxation on the representation of subscripted variables is permissible:

Given $A(d_1, d_2, d_3)$ d_i are integer constants

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

similarly, for $A(d_1, d_2)$

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

and for $A(d_1)$

A implies A(1)

EXPRESSIONS 3

An expression can be a constant, a simple or subscripted variable, a function, or any combination of these separated by operators and parentheses. It can contain non-standard and mixed modes of arithmetic; however, when non-standard or mixed modes are used, special rules apply to operators. Non-standard arithmetic requires that the programmer add special COMPASS routines to the FORTRAN library.

3.1 ARITHMETIC EXPRESSIONS

The following operators are used in arithmetic expressions:

Symbol	Function
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Arithmetic elements are:

Constants

Variables (simple or subscripted)

Functions

Examples:

```
A
3.141592
B+16.8946
(A-B(I, J+2))
G*C(J)+4.1/(Z(J, 3*K))*SINF(V)
```

Rules:

1. In an arithmetic expression do not use adjacent arithmetic operators; X op op Y; or adjacent arithmetic elements; A(B+C)D.

2. If X is an expression, then (X), ((X)), et cetera, are expressions.

3. If X, Y are expressions, the following are expressions:

   X + Y X/Y

   X - Y X * Y

4. Expressions of the form X**Y and X**(-Y) are legitimate, subject to the restrictions in section 4.2, rule 4.

5. There is no implied multiplication.

   X(Y) does not imply X*(Y)

   Constant (X) does not imply constant *(X).

3.1.1 ORDER OF EVALUATION

The hierarchy of arithmetic operations is:

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

In an expression with no parentheses or within a pair of parentheses, in which unlike classes of operators appear, evaluation proceeds from left to right. The first operator of such an expression is compared against the second. If the first operator takes precedence over the second, the operation is scheduled for execution. If the second operator takes precedence over the first, or is equal to the first, the first operation is delayed and the second operator is compared against the third.

Example:

(A+B*C**D)

+ < * A+B delayed

* < ** B*C delayed

** last operator

C**D scheduled for execution

C**D → R_1

R indicates an intermediate result, not necessarily a temporary storage.

The operations that were delayed are then scheduled for execution in reverse order:

$B*R_1 \rightarrow R_2$

$A+R_2 \rightarrow R_3$

If an expression contains consecutive operations involving operators of like class (A+B+C, A*D*F, A-B+C), the left to right evaluation is modified. (This does not apply for A*B/C, A/B*C, A/B/C, or A**B**C.) The first operation, plus subsequent operations, will be delayed until the end of the statement is reached or an operator of unlike class is encountered.

Examples:

(A+B+C+D+E+F)

(R+S+T-U-V-W)

(A+B-C-D+E*R)

When the end of the statement is reached, the delayed operations are scheduled for execution in reverse order and the final expression is the last to be executed.

Examples:

$(A+_1B+_2C+_3D+_4E+_5F)$

order established by scan		evaluation
$+_1 = +_2$	A + B delayed	D + E $\rightarrow R_1$
$+_2 = +_3$	B + C delayed	C + $R_1 \rightarrow R_2$
$+_3 = +_4$	C + D delayed	B + $R_2 \rightarrow R_3$
$+_4 = +_5$	D + E delayed	A + $R_3 \rightarrow R_4$
$+_5$		R_4 + F $\rightarrow R_5$

If an unlike operator of greater precedence is encountered within a sequence of like operators ($A+_1B+_2C*D+_3E+_4F$), the evaluation will proceed as follows:

order established by scan		evaluation
$+_1 = +_2$	A + B delayed	C * D $\rightarrow R_1$
$+_2 < *$	B + C delayed	B + $R_1 \rightarrow R_2$
$* > +_3$	C * D scheduled for execution	A + $R_2 \rightarrow R_3$
$+_3 = +_2$	B + C scheduled for execution	R_3 + E $\rightarrow R_4$
$+_3 = +_1$	A + B scheduled for execution	R_4 + F $\rightarrow R_5$
$+_3 = +_4$	D + E delayed	
$+_4$ last operator	D + E scheduled	
	E + F scheduled	

If the unlike operator is of less precedence the evaluation will be as follows:

($A*_1B*_2C+D*_3E*_4F$)

order established by scan		evaluation
$*_1 = *_2$	A * B delayed	A * B $\rightarrow R_1$
$*_2 > +$	A * B scheduled for execution	R_1 * C $\rightarrow R_2$
$*_1 > +$	B * C scheduled for execution	D * E $\rightarrow R_3$
$+ < *_3$	C + D delayed	R_3 * F $\rightarrow R_4$
$*_3 = *_4$	D * E delayed	R_4 + $R_2 \rightarrow R_5$
$*_4$ last operator	D * E scheduled for execution	
	C + D remains delayed	
	E * F scheduled for execution	
	C + D scheduled for execution	

An expression of the form (A+B+C*D*E**F+G+H) would be evaluated in the following manner:

order established by scan		evaluation
$+_1 = +_2$	A + B delayed	E ** F $\rightarrow R_1$
$+_2 < *_1$	B + C delayed	D * $R_1 \rightarrow R_2$
$*_1 = *_2$	C * D delayed	C * $R_2 \rightarrow R_3$
$*_2 <$ **	D * E delayed	B + $R_3 \rightarrow R_4$
** $> +_3$	E ** F scheduled for execution	A + $R_4 \rightarrow R_5$
$*_2 > +_3$	D * E scheduled for execution	R_5 + G $\rightarrow R_6$
$*_1 > +_3$	C * D scheduled for execution	R_6 + H $\rightarrow R_7$
$+_2 = +_3$	B + C scheduled for execution	
$+_1 = +_3$	A + B scheduled for execution	
$+_3 = +_4$	F + G delayed	
$+_4$ last operator		
$+_4 = +_3$	F + G scheduled for execution	
	G + H scheduled for execution	

The use of additional parentheses will alter the order of evaluation and will greatly affect the results when working with mixed mode.

In parenthetical expressions within parenthetical expressions, evaluation begins with the innermost expression; they are evaluated as encountered in the left to right scanning process.

Examples:

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

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

   A ** B $\rightarrow R_1$

   R_1/C $\rightarrow R_2$

   D * E $\rightarrow R_3$

   R_3 * F $\rightarrow R_4$

   $R_4 + R_2 \rightarrow R_5$

   R_5 - G $\rightarrow R_6$

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

$A ** B \rightarrow R_1$

$C + D \rightarrow R_2$

$R_1/R_2 \rightarrow R_3$

$E * F \rightarrow R_4$

$R_4 - G \rightarrow R_5$

$R_3 * R_5 \rightarrow R_6$

3. When the expression contains a function, the function is treated as a parenthetical expression.

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

$COSF(Z) \rightarrow R_1$

$R_1 ** 2 \rightarrow R_2$

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

$R_3 + H(13) \rightarrow R_4$

Examples 4 and 5 are examples of expressions with embedded parentheses.

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

$C/D \rightarrow R_1$

$R_1 - E \rightarrow R_2$

$R_2 + B \rightarrow R_3$

$R_3 * A \rightarrow R_4$

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

$SINF(X) \rightarrow R_1$

$R_1 + 1. \rightarrow R_2$

$R_2 * A \rightarrow R_3$

$E + F \rightarrow R_4$

$-R_4 \rightarrow R_4$

$R_4 + D \rightarrow R_5$

$R_5 * C \rightarrow R_6$

$-Z/R_6 \rightarrow R_7$

$R_7 + R_3 \rightarrow R_8$

3.1.2 TYPE REAL ARITHMETIC

Expressions containing only type real constants and variables accept all standard operators and require no special rules.

3.1.3 TYPE INTEGER ARITHMETIC

Integer expressions are processed from left to right and modified by delayed operations. Also, dividing an integer quantity by an integer quantity yields a truncated result; thus 11/3 = 3. The expression I*J/K may yield a result different from the expression I*(J/K).

$$I*J/K = \frac{4*3}{2} = 6 \qquad I*(J/K) = \frac{4*3}{2} = 4$$

3.1.4 TYPE CHARACTER ARITHMETIC

FORTRAN includes routines that load, store, complement, add, and subtract CHARACTER constants and variables. Routines in COMPASS language may be supplied by the user. Characters dimensioned in an array are assigned four to a computer word. Simple variables are assigned one storage location per variable in the upper 6 bits (18 to 23) of the computer word. The compiler generates code which performs CHARACTER arithmetic in the lower 6 bits of the arithmetic registers and stores as the result the lower 6 bits of the A register.

Example:

```
1    5 7
      CHARACTER B, C, RESLT
      RESLT = B+C
```

B=8	10	00	00	00
C=9	11	00	00	00
RESLT=A	21	00	00	00

For character arithmetic, simple character values are used as 6-bit integer quantities, right justified. The result, the lower 6 bits of the A-register, is stored in the high order 6 bits of the computed variable.

3.2 MIXED-MODE ARITHMETIC

Arithmetic expressions can contain mixed types of constants and variables. Mixed-mode arithmetic is accomplished through the special library conversion subroutines (Appendix C).

The three standard operand types are real, integer, and character. The programmer may define one non-standard type per subprogram. Type declarations are covered in Chapter 5.

The following rules establish the relationship between the type of an evaluated expression and the types of the operands it contains.

Rules:

1. The order of dominance of the operand types within an expression from highest to lowest is:

   Non-standard

   Real

   Integer

   Character

2. The dominant operand type determines the type of an evaluated arithmetic expression.

3. A non-standard type may be mixed with any or all of the standard types.

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

   - When A or B is type other or character, the user must provide the conversion routines.
   - For the standard types the mode/type relationships are:

Type B \ Type A	Character	Integer	Real
Character	Character	Integer	Real
Integer	Integer	Integer	Real
Real	Real	Real	Real

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

In mixed mode arithmetic, the mode used to evaluate any portion of an expression is determined by the dominant type thus far encountered within the expression and the normal hierarchy of arithmetic operations; integer, mode will be used when an integer type is first encountered and will be converted to real mode when a real type is encountered.

Examples:

1. Given A, B type real; I, J type integer. The mode of evaluating the 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

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

   A * B $\rightarrow R_1$ real

   I - J $\rightarrow R_2$ integer

   Convert R_2 to real

   R_1 - R_2 $\rightarrow R_3$ real

3. Given C, D type character and I type integer, the mode of evaluating expression (C-D+I) is integer.

   C - D $\rightarrow R_1$ character

   Convert R_1 to integer

   R_1 + I $\rightarrow R_2$ integer

4. In the expression (I+C-D) variables C, D are character; I is integer. Evaluation proceeds:

   Convert C to integer

   I + C $\rightarrow R_1$ integer

   Convert D to integer

   R_1 - D $\rightarrow R_2$ integer

When an operation is to be performed on operands of different modes, conversion is implemented during the normal evaluation. When integer and real modes are mixed, FORTRAN will perform the conversion and the operation for:

add	divide
subtract	exponentiate
multiply	store

The generated calls and their descriptions are contained in Appendix C.

When integer and character, or real and character modes are mixed, FORTRAN generates code that performs the conversion and operation for:

add	load
subtract	complement
store	

If other operations (multiply, divide, exponentiate) are to be performed, the necessary conversion routines must be supplied by the user (Appendix C).

For mixed integer and character, the add and subtract routines are implemented during the evaluation, and no reference is made to external routines.

3.3 LOGICAL EXPRESSIONS

A logical expression has the general form

r_1 op r_2 op r_3

in which r_i are simple variables, arithmetic expressions, or relational expressions; and op is either the logical operator .AND. indicating conjunction or .OR. indicating disjunction.

The value of a logical expression is either true (1) or false (0).

Logical expressions are generally used in logical IF statements (section 6.2).

Rules:

1. Precede and follow logical operators .AND. and .OR. with either a relational or an arithmetic expression.

2. Precede .NOT. only with .OR., .AND. or the beginning of the expression.

   r_1 .AND. .NOT. r_2

   r_1 .AND. r_2

   .NOT. r_1

   r_1 .OR. r_2 .AND. r_3

3. If op is either .AND. or .OR. do not use a logical expression of the form r_1 op op r_2.

4. Do not enclose logical operators within parentheses.

5. Do not nest logical expressions.

6. Each logical operator applies to the relational expression up to the next logical operator or to the end of the logical expression, except in an expression of the form:

   r_1 .AND. .NOT. r_2 .OR. r_3

   .NOT. refers to r_2

   .AND. refers to .NOT. r_2

   .OR. refers to r_3

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

   .AND. .NOT. r_1

   .OR. .NOT. r_1

8. Logical statements are evaluated from left to right.

9. The logical operators are defined as follows:

.NOT. r_1	is false if r_1 is true
r_1 .AND. r_2	is true only if r_1 and r_2 are true
r_1 .OR. r_2	is false only if r_1 and r_2 are false

Examples:

1) R = A .OR. B .OR. C .OR. D .AND. E .AND. F .AND. G

   R = D .AND. E .AND. F .AND. G .OR. A .OR. B .OR. C .OR. D

These two statements provide the same results, but the first process may be faster since the truth value of A, B, or C eliminates the need for evaluating D, E, F, and G.

2) FORTRAN does not permit the form R = .NOT. (.NOT. (.NOT. (A.AND. B) .AND. C) .OR. .NOT. (B.AND. C. OR. A)). The same results may be obtained, however, by one of the following methods:

   The example may be reduced using the rules of Boolean logic, to:

   R = .NOT. A .AND. B .AND. C .OR. A .AND. .NOT. B .AND. C

The single statement may be replaced with a series of statements representing the nested groups:

```
X = A.AND.B
X = .NOT.X.AND.C
R = B.AND.C.OR.A
R = .NOT.X.OR..NOT.R
R = .NOT.R
```

3.4 RELATIONAL EXPRESSIONS

A relational expression has the form:

q_1 op q_2

The q's are arithmetic expressions; op is an 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. A relation is false if q_1 and q_2 do not satisfy the relation specified by op.

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. Relational expressions are converted internally to arithmetic expressions according to the rules of mixed mode arithmetic. These expressions are evaluated and compared with zero to determine the truth value of the corresponding relational expression.

Rules:

1. Use a relational operator between two arithmetic expressions.

2. In a relational expression, do not use more than two arithmetic expressions connected by a single relational operator; q_1 op q_2 op q_3 is not allowed.

3. Separate two relational expressions with a logical connector, .AND. or .OR., in the forms:

$$q_1 \text{ op } q_2 \left\{ \begin{matrix} .\text{AND.} \\ .\text{OR.} \end{matrix} \right\} q_2 \text{ op } q_3$$

$$q_1 \text{ op } q_2 \left\{ \begin{matrix} .\text{AND.} \\ .\text{OR.} \end{matrix} \right\} q_3 \text{ op } q_4$$

4. An arithmetic expression, by itself, is a relational expression. It is considered TRUE if the resultant value is non-zero.

5. The following relational expressions are equivalent.

q_1 op q_2

q_1 op (q_2)

(q_1) op (q_2)

(q_1) op q_2

6. Do not enclose relational operators with parentheses.

7. Do not nest relational expressions.

Examples:

Simple relational expressions

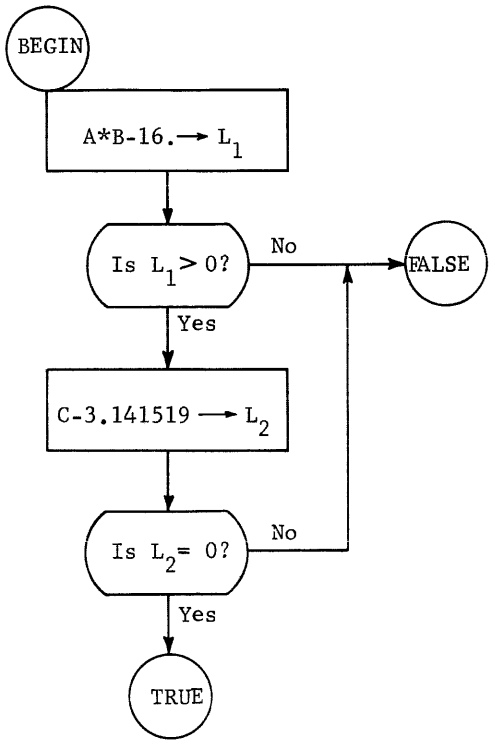
A .GT. 16.

(D-Q(I)*Z). LE. 3.141592

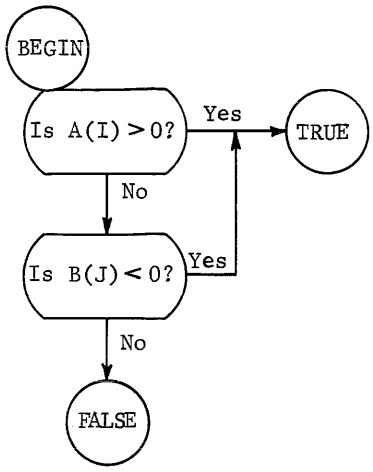
(B+C). LT. (A-D)

Evaluation of expressions

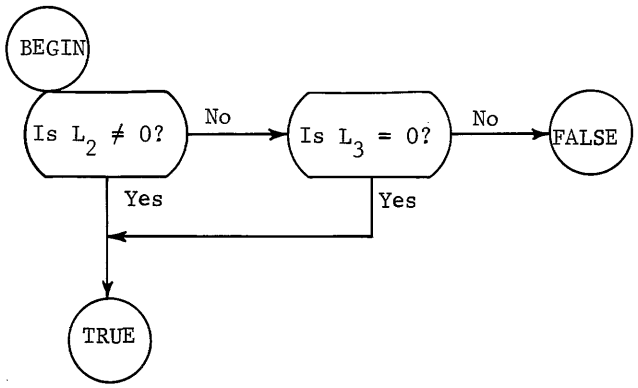
1) A*B .GT.16..AND. C.EQ. 3.141519



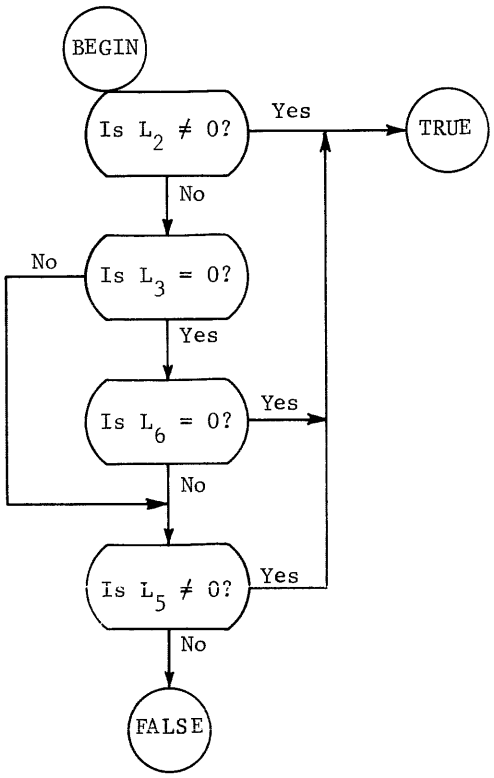
2) A(I) .GT. 0 .OR. B(J) .LT. 0



3) L2 .OR. .NOT. L3



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



3.5 MASKING FUNCTIONS

Masking expressions are formed with the aid of system library functions. Masking operations are performed bit-by-bit on the actual parameters of the function reference. The parameters may be constants, variables, functions, or expressions of integer type.

Although names of masking functions are nearly identical in appearance to logical operators, their meanings are different.

The masking functions listed below may be referred to in any expression. The resulting value is supplied in an expression wherever the masking function and arguments appear.

NOT (a)	Form in the accumulator the complement of the integer operand.
AND (a, b)	Form in the accumulator the bit-by-bit logical product of the integer operands.

OR (a,b) Form in the accumulator the inclusive OR of the integer operands.

EOR (a,b) Form in the accumulator the exclusive OR of the integer operands.

Scanning of the expression is left to right and masking functions in expressions are evaluated as they are encountered.

Operations performed by the functions:

a	b	NOT (a)	AND (a, b)	OR (a, b)	EOR (a, b)
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0

Examples:

a = 77770000
b = 77777777
c = 00001763
d = 20045000
} octal forms of integer values

NOT (a) is 00007777

AND (a,d) is 20040000

OR (c,d) is 20045763

EOR (b,c) is 77776014

REPLACEMENT STATEMENTS 4

4.1 REPLACEMENT STATEMENT

The general form of the arithmetic replacement statement is

v = e

e is an arithmetic, logical, or relational expression and v is any variable name, simple or subscripted. The operator = means that v is replaced by the value of expression e, with conversion for mode if necessary.

Examples:

```
RESLT = X+Y-2. *R
SUMX = X+Y+Z
ARG(LAB) = 2. *X+COMP**2
SCEL = BIG .LE. SMALL
TAB = .NOT. X .OR. Y .OR. Z .AND. A
EE = AA.GE.BB.OR.CC.GE.DD
```

4.2 MULTIPLE REPLACEMENT STATEMENT

The multiple replacement statement is an extension of the arithmetic replacement statement:

$$r_n = r_{n-1} = \ldots = r_2 = r_1 = a$$

a must be an arithmetic expression. r_i are simple or subscripted variables and may be any of the standard or non-standard types.

The multiple replacement statement indicates that each of the variables, $r_1 \ldots r_n$, will be replaced with the value of a in a manner analogous to that employed in mixed mode arithmetic statements.

Examples:

```
1   5  7
       I = R=4.6
             4.6 ⟶ R
             4   ⟶ I
```

```
1   5  7
       C= I = 123
              123 ⟶ I
               59 ⟶ C
```

```
1   5  7
       I= R= -14.6
             -14.6 ⟶ R
             -14   ⟶ I
```

I	integer	The numbers in the examples
R	real	represent the evaluations of
C	character	expressions

Example:

Problem — Convert radians to degrees and minutes

Solution

```
1   5  7
       DEGI=IDEG=DEG=BETA*57.296
       IMIN=(DEG-DEGI)*60
```

BETA	= 2.6 radians	(real)
DEG	= 148.9696 degrees	(real)
IDEG	= 148 degrees	(integer)
DEGI	= 148.0000 degrees	(real)
IMIN	= 58 minutes	(integer)

Result — 2.6 radians = 148 degrees 58 minutes

4.3 MIXED-MODE REPLACEMENT STATEMENT

Although the type of an evaluated expression is determined by the type of the dominant operand, this does not restrict the types that identifier a may assume. The following chart shows the a to e relationship for all of the standard modes.

Arithmetic Replacement Statement: a = e

a — is an identifier

e — is the evaluated arithmetic expression

Type of a \ Type of e	REAL	INTEGER	CHARACTER
REAL	Store e in a.	Convert e to REAL. Store in a.	Convert e to REAL and store in a.
INTEGER	Truncate e to an INTEGER. Store in a.	Store e in a.	e is stored as an INTEGER in a.
CHARACTER	Convert e to an INTEGER and store the 6 low-order bits in a.	Store the 6 low-order bits of e in a.	Store the 6 low-order bits of e in a.

Examples:

1)
```
1    5  7
       CHARACTER C,D
       DIMENSION C(2)
         D = C(1)+C(2)
```

C(1) and C(2) are added in character arithmetic. The low order 6 bits of the sum are stored as a character in D.

2)
```
1    5  7
       CHARACTER C
       DIMENSION C(2)
       I = C2-C1
```

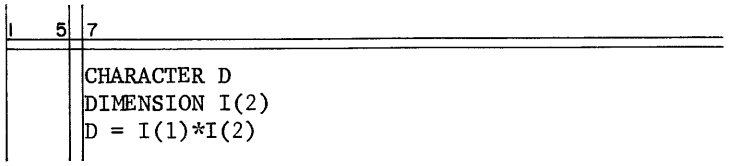
The expression (C_2-C_1) is evaluated in character arithmetic. The result is converted to integer and stored in I.

3)
```
1    5  7
       CHARACTER C
       DIMENSION C(3),I(3)
       J = I(1)-C(3)+I(3)
```

The evaluated expression I(1)-C(3)+I(3) is integer (Section 3.2).

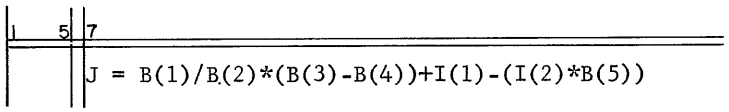
Convert C(3) to	integer
I(1)-C(3)	R_1 integer
R_1+I(3)	R_2 integer
J = R_2	integer

4)



The expression (I(1)*I(2)) is evaluated in integer arithmetic. The low-order 6 bits of the result are stored as a single character in D.

5)

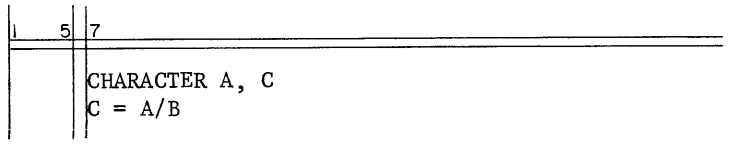


The expression is evaluated as follows

$B(3)-B(4) \rightarrow R_1$ real
$B(2)*R_1 \rightarrow R_2$ real
$B(1)/R_2 \rightarrow R_3$ real
Convert I(2) to real
$I(2)*B(5) \rightarrow R_4$ real
Convert I(1) to real
$I(1)-R_4 \rightarrow R_5$ real
$R_3+R_5 \rightarrow R_6$ real

The real value R_6 of the expression is converted to integer and stored as J.

6)



The compiler generates a call to an external subroutine (supplied by user) that divides character variable A, by real variable B.

$A/B \rightarrow R_1$ real

The result is converted to an integer and the lower 6 bits of the 24-bit word are stored in C.

DECLARATIVE STATEMENTS 5

Declarative statements are non-executable statements that:

Assign word structure to variables (TYPE)

Reserve storage for arrays and single variables (DIMENSION, COMMON)

Designate shared storage (COMMON, EQUIVALENCE)

Assign initial values to variables (DATA)

The declarative statements may be placed in any order prior to the first executable statement of a program or subprogram.

Unless prestored with a DATA statement, each word of reserved storage initially contains a return jump to the abnormal routine. The contents are non-zero.

5.1 TYPE STATEMENTS

A type statement designates the word structure of variable and function identifiers. FORTRAN recognizes three standard types with fixed word sizes and one non-standard type, the word size of which is defined by the programmer. Special compiler routines supplied by the user translate all expressions containing non-standard variables.

Type Declaration	Word Storage
REAL list	2 words/element
INTEGER list	1 word/element
CHARACTER list	6 bits/element
TYPE other (w) list	w words/element

List is a string of unsubscripted identifiers separated by commas. For example: A, B1, CAT, D36F, EUPHORIA

The designator, other, is any alphanumeric identifier that identifies the non-standard type. The number of words per element for each non-standard variable in the list is specified by the integer w.

Rules:

1. Type declarations must appear with other declarative statements prior to the first executable statement in a program or subprogram.

2. Unless declared, a variable is integer if the first character of its identifier is I, J, K, L, M, N and real if the first character is any other letter.

3. Only one TYPE other is allowed in a subprogram; more cause diagnostics.

4. An identifier declared more than once assumes the highest type declared.

TYPE other	(highest)
CHARACTER	
INTEGER	
REAL	(lowest)

5. An array identifier in list designates the entire array.

Examples:

```
1    5 7
      REAL EL, CAMINO, REAL, IDE63
      INTEGER QUID, PRO, QUO
      CHARACTER ALPHA, BETA, GAMMA
      TYPE COMPLEX (4) AI47, K156
```

5.2 DIMENSION STATEMENT

The non-executable statements DIMENSION and COMMON reserve storage for arrays. A subscripted variable in an expression represents an element of an array of variables.

DIMENSION v_1 (s_1, s_2, s_3), v_2(s_4, s_5, s_6),...

An array name, v_i, has up to three unsigned integer subscripts, s_i, separated by commas.

The number of storage locations reserved for a given array is determined by the product of the subscripts in the subscript string, and the number of words per type. An array may occupy a maximum of 32,767 words.

DIMENSION statements must appear with other declarative statements prior to the first executable statement in the program.

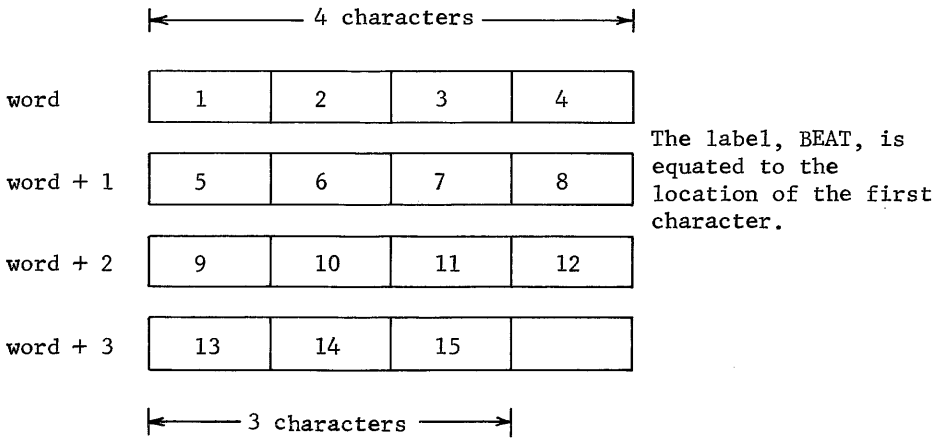
Example:

```
1    5  7
       REAL HERCULES
       CHARACTER BEAT
       DIMENSION HERCULES (10,20), BEAT (5,3)
```

The array HERCULES has 200 elements. Two locations are used to store each real element; the number of locations reserved is 400.

Character variables are dimensioned 4 characters per word in 6-bit elements, left to right, in each computer word.

The 15 elements in the array BEAT occupy 4 sequential words:



The number of locations for an integer array equals the number of elements specified by the subscripts.

5.3 COMMON STATEMENTS

A program may be divided into independently compiled subprograms that use the same data. The COMMON statements reserve storage areas - numbered or labeled - that can be referenced by more than one subprogram.

COMMON/DATA/list

Assigns labeled common storage locations to variables and arrays designated in the list. Values in labeled common may be preset with a DATA statement.

COMMON list

Assigns numbered (blank) common locations to variables and arrays designated in the list. These may not be preset with data.

Rules:

1. COMMON statements are placed with other declarative statements prior to the first executable statement in the program.

2. The identifier, DATA, for labeled common is fixed; any other identifier causes a diagnostic.

3. List is composed of subscripted or non-subscripted variable identifiers. If a non-subscripted array name appears in the list, the dimensions must be defined by a DIMENSION statement in the subprogram. Array names may be dimensioned by the COMMON statement. If dimensioned in both statements, those in the DIMENSION statement are used.

4. Attempting to list an identifier in both labeled and numbered common doubly defines the variable, and causes a diagnostic.

5. The order of identifiers in the COMMON statement determines their order in the common storage block.

6. At the beginning of program execution, the contents of numbered and labeled common (if not preset with a DATA statement) are undefined and non-zero.

7. The type and quantity of identifiers determine the length of the common block.

8. A subprogram may re-arrange the allocation of storage locations in common.

9. A subprogram may not increase the length of a labeled common block assigned by the first program. However, it may use less common than the first program.

10. When a subprogram does not need all of the locations reserved in common, dummy variables in the COMMON statement achieve correspondence of reserved areas.

Examples:

1) Labeled Common

```
1    5  7
       COMMON/DATA/D(15),F(3,3)
       COMMON/DATA/DATA(10)
```

The label DATA does not restrict use of "DATA" as an identifier.

```
1    5  7
       INTEGER Q, R
       COMMON/DATA/Q(4), R(4), S(2)
```

origin

Q(1)
Q(2)
Q(3)
Q(4)
R(1)
R(2)
R(3)
R(4)
S(1)
S(1)
S(2)
S(3)

Real variables each require two computer words.

2) Numbered Common

```
1    5  7
       COMMON A,B(2),K
       COMMON I(2),J(2)
```

origin

A(1)
A(1)
B(1)
B(1)
B(2)
B(2)
K(1)
I(1)
I(2)
J(1)
J(2)

3) Rearrangement of Common.

MAIN PROGRAM

```
1    5  7
       COMMON/DATA/C(20)
         .
         .
```

The labeled common occupies 40 storage locations.

SUBPROGRAM

```
1    5  7
       COMMON/DATA/A(10), I(10), K(10)
         .
         .
```

Labeled common is 40 storage locations. The first 20 locations (10 elements of array A) of the block are real elements. Array I occupies the next 10 locations and array K the last 10 locations.

4) Correspondence

MAIN PROGRAM

```
1    5  7
       COMMON/DATA/A, B, C
```

SUBPROGRAM

```
1    5  7
       COMMON/DATA/E, F, G
```

Only the values of E and G are used in the subprogram; F is a dummy variable that spaces over the area initially reserved for B.

5.4 EQUIVALENCE STATEMENT

The EQUIVALENCE statement permits storage locations to have several names.

EQUIVALENCE $(a_1, b_1, \ldots), (a_2, b_2, \ldots), \ldots$

$(a_i, b_i, \ldots)$ defines equivalent groups of two or more identifiers.

The first elements of arrays may be aligned by equivalencing the array names; elements of integer or real arrays may be aligned by equivalencing singly subscripted variables. Array lengths need not be equal.

When element A(i, j, k) of array A(I, J, K) is to be aligned with an element in another array, the subscript, s, is determined by:

```
s = i+(j-1)*I+(k-1)*I*J
```

Example:

```
i    5| |7
      |DIMENSION A(2,3,4),B(3)
      |EQUIVALENCE (A(7),B)
```

in which A(7) represents element A(1, 1, 2) of array A and aligns it with element B(1) of array B.

loc P
loc P+1 } A(1,1,1) = A(1)

.
.
.

loc P+12
loc P+13 } A(1,1,2) = A(7) = B(1)

loc P+14
loc P+15 } A(2,1,2) = A(8) = B(2)

loc P+16
loc P+17 } A(1,2,2) = A(9) = B(3)

Rules:

1. EQUIVALENCE statements must appear with other declarative statements prior to the first executable statement in the program or subprogram.

2. No more than one element of an EQUIVALENCE set may belong to common.

3. An identifier used as a formal parameter cannot also be used in an EQUIVALENCE statement.

4. For CHARACTER and TYPE other, use only array names and simple (non-subscripted) variable names in EQUIVALENCE sets.

5. EQUIVALENCE cannot re-arrange common. However, arrays may be equivalent so that they change the length of the common block. See example 2.

6. Attempting to change the origin of a common block causes a diagnostic. See example 3.

7. An identifier may appear more than once in an EQUIVALENCE statement. See example 4.

8. An identifier in a COMMON statement used in an EQUIVALENCE set is the base identifier for the EQUIVALENCE statement. When none in the set belongs to common, the identifier with the lowest address becomes the base identifier. All other elements in the set are referenced to the base identifier.

Examples:

1) Align first elements of two arrays.

```
|1   5|7
|     |
|     |DIMENSION A(10,10), I(200)
|     |EQUIVALENCE (A,I)
|     |    .
|     |    .
|     |    .
|    5|READ (K1,10)A
|    6|READ (K1,20)I
```

The EQUIVALENCE statement assigns the first half of the first element of real array A and the first element of integer array I to the same storage location. The READ request at statement 5 directs that the values of array A be stored in consecutive locations. Before statement 6 is executed all operations using the values of array A must be complete. The values of array I will be read by statement 6 into the storage locations previously occupied by A. (For READ statements, refer to Section 10.2.)

2) Change the length of common.

```
1    5  7
       COMMON A
       DIMENSION A(5), B(5)
       EQUIVALENCE (A(3),B(1)
       loc.P       A1
       loc.P+2     A2
       loc.P+4     A3        B1
          .        A4        B2
          .        A5        B3
          .                  B4
                             B5
```

3) Illegal attempt to change origin of common.

```
1    5  7
       COMMON I
       DIMENSION I(5), R(3)
       EQUIVALENCE I(2), R(2)
           :        :
           :        :
       loc. P       I1   } R1
       loc. P+1     I2   }
       loc. P+2     I3   } R2
       loc. P+3     I4   }
       loc. P+4     I5   } R3
```

4) Multiple use of identifiers.

```
1    5  7
       EQUIVALENCE (A,B), (C,D), (E,F), (A,F), (B,D)
```

interpreted as

```
1    5  7
       EQUIVALENCE (A,B,C,D,E,F)
```

5.5 DATA STATEMENT

DATA statements permit variables in labeled common to accept constant values prior to program execution.

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

DATA (i(j, k, 1)=list)

DATA ((((i(I, J, K), I=n_1, n_2), J=m_1, m_2) K=ℓ_1, ℓ_2)=list)

i is an identifier representing a simple variable, array name, or a variable with integer constant (i, j, k) or integer variable (I, J, K) subscripts.

For implied DO loops, subscript I ranges from integer n_1 to n_2; subscript J ranges from m_1 to m_2, K ranges from ℓ_1 to ℓ_2.

List contains constants in the form:

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

r is an integer constant repetition factor that causes the parenthetical list following it to be repeated r times. A non-integer r produces a compiler diagnostic.

Rules:

1. DATA statements must appear with other declarative statements prior to the first executable statement of the program or subprogram.

2. Because only identifiers in labeled common may be preset, each identifier appearing in a DATA statement must also be in a labeled common statement.

3. Implied DO-loop notation is permissible with the restriction that DO indexing parameter, m_3 (Section 6.3), cannot appear. Implied DO loops are useful for storing constants in arrays. See example 1.

   When the number of list elements exceeds the range of the implied DO, a diagnostic is given and compilation of the DATA statement terminates. If the list is too short, the compiler gives a diagnostic but continues processing.

4. A constant preceded by a minus sign is complemented during conversion.

5. The structure of the constant rather than the type of the identifier determines the type of the stored constant. See examples 2, 3, and 4.

6. When CHARACTER variables are to be preset, each constant in the list fills one entire storage location. See example 5.

7. Ideally, there should be a one-to-one correspondence between the identifier locations and list elements. An array name specifies the first element address. The entire array may be prestored.

Examples:

1) Implied DO loop

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

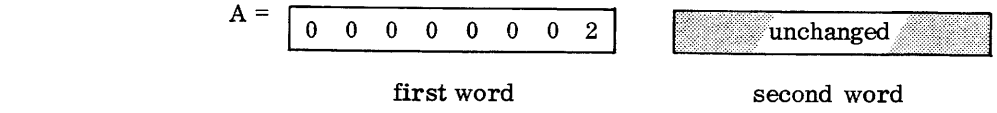
Result:

GIB	contains	1.
GIB+1	"	2.
GIB+2	"	3.
GIB+3	"	4.32
GIB+4	"	4.32
GIB+5	"	4.32
GIB+6	"	4.32
GIB+7	"	4.32
GIB+8	"	4.32
GIB+9	"	4.32

2) A storage location accepts any type of constant regardless of the type of the identifier.

```
1    5  7
       DATA (A=2)
```

Result: Contents of the first word of real element A becomes an integer 2 - not a real 2 as might be expected from the type of the identifier.



3) Store Hollerith constants in an integer array.

```
      COMMON/DATA/MESSAGE(3)
      DATA (MESSAGE = 3HWHO, 2HIS, 4HJOAN)
```

Result: MESSAGE contains WHO$\wedge$
MESSAGE+1 " IS$\wedge\wedge$
MESSAGE+2 " JOAN

4) An erroneous attempt to store real constants in an integer array produces the results shown below.

```
      COMMON/DATA/KILO(4)
      DATA(KILO = 2.6, 3.2E10)
```

Result: KILO contains the upper half of 2.6 in floating point
KILO+1 contains the upper half of 3.2E10 in floating point
KILO+2 contains the lower half of 3.2E10
KILO+3 is unchanged

5) DATA constants fill entire, not partial, storage locations. Thus, CHARACTER data may be pre-set in labeled common but the values are not necessarily packed. The examples show the wrong and right ways to pack CHARACTER data.

WRONG

```
      CHARACTER BOX
      COMMON/DATA/BOX(3,4)
      DATA(BOX = 3(1, 2, 3, 4))
```

Result: The constants 1, 2, 3, and 4 are repeated 3 times and stored in the low order 6 bits of 12 consecutive storage locations. Since BOX is dimensioned as 12 characters in 3 words, the results are erroneous.

RIGHT

```
1    5 7
      CHARACTER BOX
      COMMON/DATA/BOX (3,4)
            and
      DATA (BOX = 3(4H1234))
            or
      DATA (BOX = 3(01020304B)
```

Result: BOX contains 01 02 03 04
BOX+1 " 01 02 03 04
BOX+2 " 01 02 03 04

6) Use extra care when presetting arrays with data.

POOR

```
1    5 7
      COMMON/DATA/A(3), X
      DATA (A = 1., 2., 3., 4.)
```

Result: A contains 1.
A+1 " 2.
A+2 " 3.
X " 4. Contents of X are changed with or without the knowledge of the programmer.

POOR

```
1    5 7
      COMMON/DATA/C(3)
      DATA(C=1., 2.)
```

Result: C contains 1.
C+1 " 2.
C+2 is left undefined.

CONTROL STATEMENTS 6

Program execution normally proceeds from one statement to the next in the program. Control statements are used to alter the sequence or cause a number of iterations of a program section.

6.1 GO TO STATEMENTS

GO TO statements transfer control within a program or subprogram.

6.1.1 UNCONDITIONAL GO TO

GO TO n

Discontinues the current sequence of execution and resumes execution at the statement labeled n.

6.1.2 COMPUTED GO TO

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

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

Is a many-branch GO TO in which arithmetic expression e is evaluated prior to branching; n_i are statement numbers.

e is reduced to an integer value, j. If $j \leq 1$, n_1 is executed next; if $j \geq m$, n_m is executed next; otherwise, $j = i$, and n_i is executed next.

Example:

```
1    5  7
        A=1
        B=2
        C=1
         .
         .
         .
        GO TO (10,20,30), A*B-C
         .
         .
         .
   10   A=A+1
        GO TO (11,12,31), A*B-C
```

Control transfers to statement 10 and then to statement 31 (not shown).

6.2 IF STATEMENTS

Two- and three-branch IF statements conditionally transfer control.

6.2.1 THREE-WAY IF

IF (e) n_1, n_2, n_3

Control transfers according to the value of the arithmetic expression e.

e < 0	statement n_1 is executed next
e = 0	statement n_2 is executed next
e > 0	statement n_3 is executed next

In the test for zero, 0 = -0.

Examples:

```
1    5  7
        IF(A*B-C*SINF(X))10,10,20
        IF(I)5,6,7
        IF(A/B**2)3,6,6
```

6.2.2
LOGICAL IF

IF (ℓ) n_1, n_2

Arithmetic, logical, or relational expression ℓ is true (non-zero) or false (zero). If ℓ is true, statement n_1 is executed next; if ℓ is false, statement n_2 is executed next.

Examples:

```
|     5| 7
=============================================================
      IF(A .GT. 16. .OR. I .EQ. 0)5,10
      IF(A .AND. B)1,2
      IF(C-D .LE. D+E)4,6
      IF(X-Y)5,10
      IF (TCOUNT) 10,11
      IF (.NOT.A) 5,6
```

6.3 DO STATEMENT

The DO statement causes a predetermined sequence of instructions to be repeated a prescribed number of times, with the stepping of a simple integer variable after each iteration.

DO n i = m_1, m_2, m_3

Groups of statements are repeated according to the value of simple integer variable i which increases after each repetition. The DO loop terminates at statement number n. Indexing parameters m_k are unsigned integer constants or simple integer variables. i is initially set equal to m_1; after each execution of the DO loop, m_3 is added to i. (When omitted, m_3 assumes a value of 1.) When i becomes greater than m_2, the DO loop is satisfied.

A DO loop is composed of the DO statement, terminating statement n, and any intermediate statements. The range of a DO loop includes all of the statements following the DO statement down to and including the statement that terminates the loop. Statement n cannot be an IF, GO TO, or DO statement.

Rules:

1. Indexing parameters m_k may be unsigned integer constants or simple integer variables not greater than 32,767.

2. Constant parameters must be positive; when m_3 is zero an informative diagnostic is given.

3. When indexing parameters m_1 and m_2 are variables, they may be positive, negative, or zero. Indexing parameter m_3 may be a variable but must be positive.

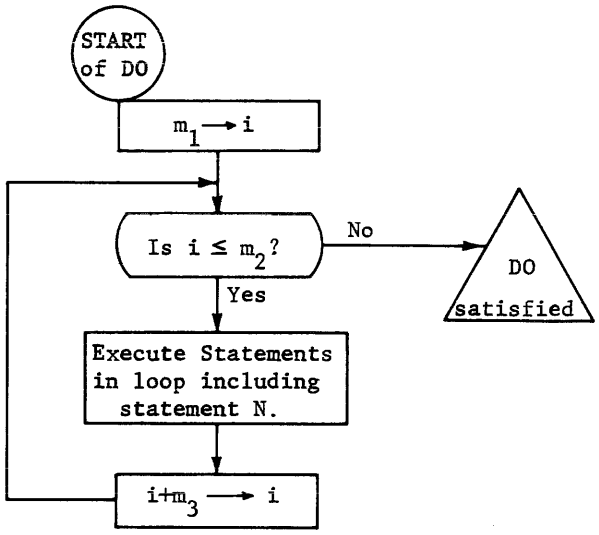
4. When the values of m_2 and m_3 are changed during the execution of the DO loop an informative diagnostic is provided.

5. i is initially equal to m_1; as soon as i exceeds m_2, looping terminates.

6. DO loops may be nested to a maximum of 10 deep.

6.3.1 DO LOOP EXECUTION

The initial value of i, m_1, is compared with m_2; if it does not exceed m_2, the loop is executed. i is increased by m_3 and again compared with m_2. The process continues until i exceeds m_2. Control then passes to the statement immediately following statement 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 statement after n.



When the DO loop is satisfied, the index variable i is no longer defined. If a transfer out of the DO loop occurs before the DO is satisfied, the value of i is preserved and may be used in subsequent statements.

6.3.2
DO NESTS

A DO loop containing another DO loop is 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. D_i represent DO statements; the subscripts indicate that D_1 appears before D_2 and D_2 appears before D_3, et cetera. n_i represent the corresponding limits of D_i; n_m must not appear after n_m-1; n_2 must not appear after n_1.

$$\begin{bmatrix} D_1 \\ \\ \\ n_1 \end{bmatrix} \qquad \begin{bmatrix} D_2 \\ \\ n_2 \end{bmatrix} \qquad \begin{bmatrix} D_3 \\ n_3 \end{bmatrix} \quad \ldots$$

Examples: Nested DO loops

```
      DO 1 I=1,10,2              D1
       :
      DO 2 J=1,5                 D2
       :
      DO 3 K=2,8                 D3
       :
    3 CONTINUE                   n3
       :
    2 CONTINUE                   n2
       :
      DO 4 L=1,3                 D4
       :
    4 CONTINUE                   n4
       :
    1 CONTINUE                   n1
       :
```

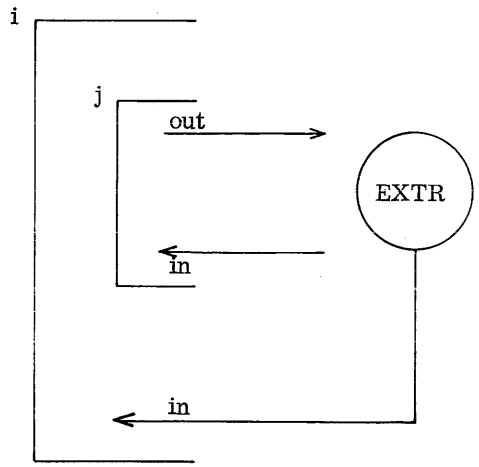
```
      DO 100 L=2, LIMIT          D1
       :
      DO 10 I=1,10               D2
       :
      DO 10 J=1,10               D3
       :
   10 CONTINUE                   n2,n3
       :
      DO 20 K=K1,K2              D4
       :
   20 CONTINUE                   n4
       :
  100 CONTINUE                   n1
       :
```

```
      DO 5 I=1,5                 D1
      DO 5 J=I,10                D2
      DO 5 K=J,15                D3
       :
       :
    5 CONTINUE                   n1,n2,n3
```

6.3.3 DO LOOP TRANSFER

In a DO nest, control transfers from one DO loop into a DO loop containing it, or out of a DO nest completely. Leaving a nested DO loop and then returning to the nest is a special case. In a DO nest, when the range of i includes the range of j, and a transfer out of j occurs, control may transfer back into the range of i or j.

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



6.4 CONTINUE STATEMENT

CONTINUE

Acts as a do-nothing instruction; control passes to the next sequential program statement. The CONTINUE statement is 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.

6.5 PAUSE STATEMENT

PAUSE

PAUSE n

The PAUSE statement transfers control to a system or user-supplied subroutine; PAUSE n halts the computer with n (1 to 5 octal digits) displayed in the A register on the console. When the START key on the console is pressed, program execution proceeds with the statement immediately following PAUSE. PAUSE (n omitted) halts the computer with zeros displayed in the A register on the console. An n greater than 5 octal digits causes an informative diagnostic.

6.6 STOP STATEMENT

STOP

STOP n

STOP n halts the computer with n (1 to 5 octal) displayed in the A register on the console. When the START key on the console is pressed, control transfers to the SCOPE monitor. STOP (n omitted) causes immediate exit to monitor. An n greater than 5 octal digits causes an informative diagnostic.

FUNCTION AND SUBROUTINE SUBPROGRAMS 7

7.1 MAIN PROGRAM AND SUBPROGRAMS

A main program may be written with or without references to subprograms. Subprograms (functions and subroutines) are sets of instructions that may be written and compiled separately from the main program and may be referred to by the main program.

The name of a function determines the type of the subprogram in the same way that names determine types of variables. Names of subroutine subprograms are not classified by type. A subroutine or function name must be unique within the subprogram.

A calling program is a main program or subprogram that refers to subroutines and functions.

7.1.1 PROGRAM STATEMENT

In each main program, the first statement must be of the following form where name is a 1- to 8-character alphanumeric identifier beginning with a letter.

```
PROGRAM name
```

PROGRAM name may be used only once in a main program, segment, or overlay. An overlay or segment requires use of PROGRAM name in an entry subprogram written in FORTRAN.

7.2 SUBROUTINE SUBPROGRAMS

A subroutine subprogram is composed of a set of FORTRAN statements, bounded by a SUBROUTINE statement and an END statement. A subroutine subprogram performs operations or calculations that may or may not return values to the calling program.

Subroutine subprograms are compiled independently of the main program and may be compiled in a separate run.

7.2.1 SUBROUTINE STATEMENT

A subroutine begins with the statement

SUBROUTINE name
or
SUBROUTINE name ($p_1, p_2, \ldots p_n$)

A subroutine name contains up to eight characters, the first of which is alphabetic. The name must not appear in a declarative statement or within the subroutine subprogram.

A subroutine statement can contain from 1 to 63 formal parameters, p_i; they may be array names, non-subscripted variables, or names of other function or subroutine subprograms. Formal parameters must not appear in any of the following declarative statements within the subroutine subprogram:

EXTERNAL
COMMON
DATA
EQUIVALENCE

A formal parameter representing an array, must be declared in a DIMENSION statement within the subroutine subprogram; otherwise, only the first element of the array is available to the subroutine subprogram.

7.2.2 CALL STATEMENT

A reference to a subroutine is a call upon a computational or operational procedure. No resultant value is identified or associated with the name of the subroutine. The subroutine subprogram returns values, if any, to the main program through formal parameters or common. The executable statement in the calling program for referring to a subroutine is:

CALL name
CALL name ($p_1, p_2, \ldots p_n$)

The CALL statement transfers control to the subroutine named. A RETURN or END statement in the subroutine subprogram returns control to the calling program. A called subroutine may not call the calling program or itself.

The actual parameters, p_i, of a subroutine call must agree in order, number 1 to 63, and type with the formal parameters of the subroutine subprogram. The following forms are acceptable for actual parameters:

arithmetic expression

constant

variable, simple or subscripted

array name

function reference

subroutine name

Logical expressions may not be actual parameters. A function reference, used as an actual parameter, must also be used in an EXTERNAL statement in the calling program.

When a subroutine is used with a parameter list, the subroutine name and its parameters must appear as separate actual parameters.

Examples:

1) Subroutine Subprogram

```
1    5 7
      SUBROUTINE ISHTAR (Y,Z)
      COMMON X(100)
      Z=0
      DO 5 I=1,100
    5 Z=Z+X(I)
      CALL Y
      RETURN
      END
```

Calling Program Reference

```
1    5 7
      COMMON A(100)
      EXTERNAL PRNTIT
              :
              :
      CALL ISHTAR (PRNTIT, SUM)
```

The formal parameters, Y and Z, in the subroutine subprogram, are replaced by PRNTIT and SUM. CALL Y is a call to subroutine PRNTIT; PRNTIT must appear in an EXTERNAL statement for the compiler to recognize it as a subroutine name.

2) Subroutine Subprogram (Matrix Multiply)

```
      SUBROUTINE MATMULT
      COMMON/DATA/X(20,20),Y(20,20),Z(20,20)
      DO10I=1,20
      DO10J=1,20
      Z(I,J)=0
      DO10K=1,20
   10 Z(I,J)=Z(I,J)+X(I,K)*Y(K,J)
      RETURN
      END
```

Calling Program Reference

```
      COMMON/DATA/A(20,20), B(20,20), C(20,20)
                :
                :
      CALL MATMULT
                :
                :
```

3) 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)
```

7.3 FUNCTION SUBPROGRAMS

A function subprogram is composed of a set of FORTRAN statements, bounded by a FUNCTION statement and an END statement. A function subprogram calculates a single value used in evaluating an expression.

Function subprograms are independently compiled of the main program and may be compiled in a separate run.

7.3.1 FUNCTION STATEMENT

A function subprogram begins with the statement:

FUNCTION name $(p_1, p_2, \ldots p_n)$ $\quad 1 \leq n \leq 63$

A function name contains up to eight characters, the first of which is alphabetic. The type (real, integer) of the result of a function is determined by the name of the function. The type may be implicitly defined as real or integer by its name (Section 2.2) or it may be explicitly defined by a TYPE statement (Section 5.1).

The function name may not appear in any of the following declarative statements:

DIMENSION
COMMON
EQUIVALENCE
EXTERNAL
DATA

A function name must appear at least once within the function subprogram as one of the following:

the lefthand identifier of a replacement statement

an element of an input list

an actual parameter of a subprogram call

Formal parameters, p_i, may be array names, non-subscripted variables, and names of other function or subroutine subprograms. Within function subprograms, formal parameters must not appear in any of the following declarative statements:

EXTERNAL
COMMON
DATA
EQUIVALENCE

A formal parameter representing an array must be declared in a DIMENSION statement within the function subprogram. Otherwise, only the first element of the array is available to the function subprogram.

A function must have at least one parameter.

7.3.2 FUNCTION REFERENCE

A reference to a function is a call upon a computational procedure for the return of a single value. The value returned is identified by and associated with the function identifier. The form of the function reference is:

name $(p_1, p_2, \ldots p_n)$

A function reference may be used in expressions in the same way as variable identifiers; name is the function name.

The actual parameters, p_i, in a function reference must agree in order, number (1 to 63), and type with the formal parameters of the function subprogram.

Rules:

1. The following forms for actual parameters are permissible:

   arithmetic expression

   constant

   variable, simple or subscripted

   array name

   function reference

   function or subroutine name

2. Logical expressions are not allowed as actual parameters.

3. When the name of a function subroutine appears as an actual parameter, the name must also appear in an EXTERNAL statement in the calling program.

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 sine of Q(K).

2) Function Subprogram

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

Function Subprogram Reference

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

In the function subprogram, TLIM and ULIM replace A and 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 with Subroutines as Parameters

The subprograms are defined by statements:

```
SUBROUTINE SQIRT(A, B)
FUNCTION ZEBRA(X, Y, Z, XX)
SUBROUTINE DONE(R, X, T, U, V)
```

```
1    5 7
      EXTERNAL SQIRT
           .
           .
           .
      QX=ZEBRA (SQIRT, ARG1, ARG2, OK)
           .
           .
           .
      CALL DONE(SQIRT,ONE,TWO,C,D)
```

ARG1 and ONE represent formal parameter A; ARG2 and TWO represent formal parameter B of SUBROUTINE SQIRT.

4) Function subprogram

```
1    5 7
      FUNCTION AL(W,X,Y,Z)
      CALL W(X,Y,Z)
      AL=Z**4.
      RETURN
      END
```

Function Subprogram Reference

```
1    5 7
      EXTERNAL SUM
           .
           .
           .
      G = AL(SUM, E, V, H)
```

7.4 EXTERNAL STATEMENT

When a CALL statement or function reference contains the name of a subroutine or function in its list of actual parameters, the name must be declared in an EXTERNAL statement in the form:

EXTERNAL $name_1$, $name_2$, ... $name_n$

$name_i$ is a function or subroutine name used as a parameter.

The EXTERNAL statement must precede the first executable statement of any program in which it appears.

Example:

To make a function reference PHI(p_1, p_2) in the statement C = D-PHI (Q (K), SINF): Function SINF is an actual parameter of the function PHI and must be declared in EXTERNAL statement.

```
1    5 7
      EXTERNAL SINF
```

PHI, the function originally referenced, begins with the following statements:

```
1    5 7
      FUNCTION PHI(ALFA,PHI2)
      PHI=PHI2 (ALFA)
```

Formal parameter ALFA takes the value Q(K); formal parameter PHI2 calls for SINF. Thus, the function subprogram PHI calculates the sine of Q(K).

7.5 ENTRY STATEMENT

ENTRY name

Identifies an alternate entry point in a subprogram to be entered if the name of the entry point rather than the normal function name is referenced in a statement. ENTRY may not be labeled nor be within a DO loop.

7.6 ENTRY CALL OR REFERENCE

To enter a subprogram at the ENTRY statement, the name of the entry point is called (subroutine) or referenced (function) in the same way as a subroutine or function.

ENTRY names must agree with the type of the function name when used in a function subprogram.

The actual parameters with the ENTRY statement must agree in type and mode with the formal parameters in the FUNCTION or SUBROUTINE statement for the subprogram.

Example:

```
1    5 7
=======================================================
                :
                :
   45 R=S+JAM(Q,2.*P)
```

Subprogram execution would begin at ENTRY JAM in the subprogram.

```
1    5 7
=======================================================
      FUNCTION JOE(X,Y)
   10 JOE=X+Y
      RETURN
      ENTRY JAM
      IF...
      :
      :
      END
```

7.7 RETURN STATEMENT

RETURN

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

In function references, control returns to the statement containing the function. In subroutine subprograms, control, in most cases, returns to the calling program. A RETURN statement in the main program causes an exit to the monitor.

7.8 END STATEMENT

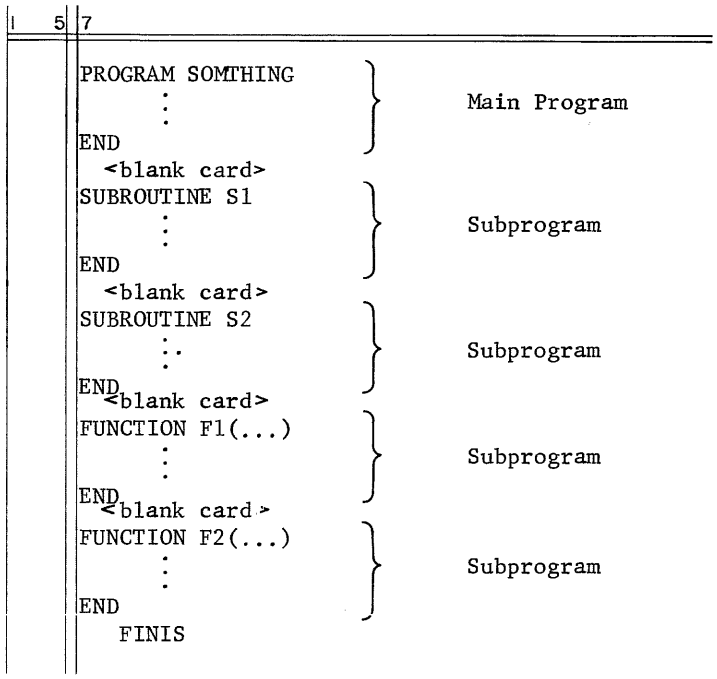
END

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

7.9 PROGRAM ARRANGEMENT

FORTRAN compilation assumes that all statements and comments inserted between a PROGRAM, SUBROUTINE, or FUNCTION statement and an END statement belong to one program. Comments inserted between END and a SUBROUTINE or FUNCTION statement are associated with the preceding program. A blank card is required between subprograms and a FINIS card follows the last subprogram.

Example:


1 5 7
PROGRAM SOMTHING
END
<blank card>
SUBROUTINE S1
END
<blank card>
SUBROUTINE S2
END
<blank card>
FUNCTION F1(...)
END
<blank card>
FUNCTION F2(...)
END
FINIS
Main Program
Subprogram
Subprogram
Subprogram
Subprogram

LIBRARY SUBPROGRAMS 8

8.1 SUBROUTINE LIBRARY

The FORTRAN library of subroutine subprograms includes:

Subroutine	Definition
SLITE (i)	set sense light i
SLITET (i,j)	test sense light i
SSWTCH (i,j)	test sense switch i
DVCHK (i)	check divide fault
EXFLT (i)	check exponent fault
OVERFL (i)	check overflow fault
EOFCK (i,j)	test for end-of-file on unit i
IOCHK (i,j)	test for parity error on unit i
UNITST (i,j)	test status of unit i
OVERLAY (o, s, i)	load and execute overlay
SEGMENT (o, s, i)	load and execute segment
FORTDUMP (b,e,m,d)	system dump routine

Parameter i specifies the unit or component number and j specifies the location of the result. See Chapter 9, Machine Condition Subprograms. For Overlay and Segment, o specifies the overlay identification, and s the segment.

8.2 FUNCTION LIBRARY

The following FORTRAN library functions are pre-defined and may be referenced by any program or subprogram. X represents real values; I represents integer values. F is optional as a final character in most function names. For machine conditions, i designates the component or unit number.

Function	Definition
ABS(X); ABSF(X)	absolute value
IABS(I); XABSF(I)	
ALOG(X); LOGF(X)	natural log of X
ATAN(X); ATANF(X)	arctangent of X radians
COS(X); COSF(X)	cosine of X radians
EXP(X); EXPF(X)	e to xth power
FLOAT(I); FLOATF(I)	integer to real conversion
IFIX(X); XFIXF(X); FIXF(X)	real to integer conversion
SIGN(X_1, X_2); SIGNF(X_1, X_2)	sign of X_2 times X_1
ISIGN(I_1, I_2); XSIGNF(I_1, I_2)	sign of I_2 times I_1
SIN (X); SINF(X)	sine of X radians
SQRT (X); SQRTF (X)	square root of X
SLITEF (i)	set sense light i
SLITETF (i)	test sense light i
SSWTCHF (i)	test sense switch i
DVCHKF (i)	check divide fault
EXFLTF (i)	check exponent fault
OVERFLF (i)	check overflow fault
EOFCKF (i)	test for end-of-file on unit i
IOCHKF (i)	test for parity error on unit i
UNITSTF (i)	test status of unit i
LENGTHF (i)	words in last BUFFER IN on unit i
NOT (a) AND (a,b) OR (a,b) EOR (a,b)	integer masking functions

The functions XABSF, LOGF, FIXF and XSIGNF must appear in a TYPE declaration to indicate the correct mode of the result.

8.3 OVERLAY AND SEGMENT

With the library subroutines OVERLAY and SEGMENT, a FORTRAN source program can call from an overlay tape portions of a program too large for available storage. OVERLAY and SEGMENT do not partition a program or prepare an overlay tape. Each subprogram loads the called overlay or segment into storage and transfers control to the entry point address.

For preparation of overlay tapes, refer to the 3200 SCOPE/COMPASS Reference Manual, Publication No. 60057700.

The overlay structure consists of a MAIN program and associated overlays and segments. The MAIN program can call OVERLAY to load a particular overlay; the overlay can call SEGMENT to load its associated segments. The main program resides in memory throughout the entire execution.

A segment may reference subprograms (entry points) in its associated overlay or main program. An overlay may reference subprograms (entry points) in the main program.

When an error occurs during calling or loading overlays and segments, a diagnostic message is written and the job terminates abnormally.

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

```
CALL OVERLAY (o,s,i)
        or
CALL SEGMENT (o,s,i)
```

o, s, and i are integer constants or variables. All must be present. Their order is fixed. o and s may be 0 through 99.

- o — overlay identification number for overlay and its segments
- s — segment identification number; s = 0 for CALL OVERLAY
- i — number of the logical unit on which overlay tape is mounted

An overlay or segment is entered and left through a return jump instruction. Other exits may be taken to the calling program or main program but bookkeeping in the loader prevents loading of a new overlay or segment until an exit is made through the entry point of the overlay or segment. Therefore, an alternative exit from an overlay or segment should be a CALL name statement (FORTRAN) or a return jump (COMPASS) - either of which returns control to the calling segment or overlay. The called subprogram, if called by an overlay may not call an overlay; if called by a segment, it may not call a segment.

CALL OVERLAY appears only in the main program. In an overlay, CALL SEGMENT calls segments belonging to the overlay. In the main program, use of CALL SEGMENT applies only in the special case in which an overlay has called the main program. Execution may then proceed from main program to overlay to segment. The segment called from the main program must belong to the overlay that called the main program.

Example:

1) Main Program

```
      PROGRAM MAIN
          :
      CALL OVERLAY (3,0,25)
          :
```

FORTRAN OVERLAY 3 on Logical Unit 25

```
      PROGRAM OVRLY
          :
      CALL SEGMENT (3,16,25)
          :
      CALL SEGMENT (3,12,25)
          :
      RETURN
```

Program MAIN calls OVRLY (Overlay 3 of logical unit 25). OVERLAY loads OVRLY and transfers control to it. In OVRLY are calls for segments of Overlay 3. Segment 16 is called, loaded and executed. Not until after it returns control to OVRLY can Segment 12 be called, loaded and executed. When execution of OVRLY is completed, control returns to program MAIN. Until OVRLY returns control to MAIN, a second overlay cannot be called.

2) Special case in which main program can properly call a segment.

MAIN PROGRAM

```
      PROGRAM MAIN
           :
           :
      CALL OVERLAY (3,0,25)
           :
           :
      SUBROUTINE PLUS (A,X,K)
           :
           :
      CALL SEGMENT (3,4,25)
      RETURN
```

FORTRAN OVERLAY 3 on Logical Unit 25

```
      PROGRAM OVRLY
           :
           :
      CALL PLUS (ALPHAI, BABYI, LINKI)
           :
           :
      CALL SEGMENT (3,16,25)
           :
           :
      RETURN
```

Program MAIN calls OVRLY which is loaded and executed. In OVRLY is a call to PLUS, a subprogram of MAIN. (Overlays and segments can reference entry points of the main program.) Because it was called by Overlay 3, PLUS can call any segments of Overlay 3. In this example, PLUS calls Segment 4 of Overlay 3. After Segment 4 is executed, control returns to PLUS which may then call another segment or return control to OVERLAY 3 (OVRLY). OVRLY, when it resumes control may call segments of Overlay 3, subprograms, or may return control to MAIN. When MAIN resumes control, it may call another overlay.

3) A segment incorrectly attempting to call another segment.

In example 2, OVRLY called Segment 16 of Overlay 3. Segment 16 cannot appear as shown below:

FORTRAN SEGMENT 16 of OVERLAY 3

```
1    5  7
       PROGRAM SGMNT
            :
            :
       CALL PLUS (ALPHA, BABY, LINK)
            :
            :
       RETURN
```

In SGMNT is a call to subprogram PLUS. In PLUS, the call for Segment 4 of Overlay 3 cannot be honored because a segment is calling another segment. A diagnostic occurs.

8.4 FORTRAN DUMP

The FORTDUMP subroutine permits a programmer to request a printout of the contents of variables in octal, character, or decimal floating point.

Each time it is called, the routine prints, on the standard output unit, a line containing (1) dump identification, (2) the COMPASS address of the calling sequence, and (3) contents of the A and Q registers, the three index registers, and the interrupt mask register. When the register file option is elected, FORTDUMP prints REGISTER FILE followed by the contents of all 64 high-speed registers.

The memory dump consists of 8-word lines of data printed in the designated mode and preceded by the absolute octal address of the first word on the line. When FORTDUMP detects a line that contains words all identical to the last word of the preceding line, the line is suppressed. The suppressed line or lines are noted with the word GAP on the listing.

To call FORTDUMP, use the statement

```
CALL FORTDUMP(b, e, m, d)
```

b = simple or subscripted variable identifier of first word to be dumped

e = simple or subscripted variable identifier of last word to be dumped

m = mode; an octal constant or a variable identifier for location of the octal constant

Octal Constant	Mode
1	Octal
2	Character
3	Floating point
4	Register file
5	Octal; register file
6	Character; register file
7	Floating point; register file

d = Hollerith or octal (internal BCD) constant, or the variable identifier giving the location of 4 BCD characters that identify the dump

Rules:

1. To prevent excessive printout, avoid calling FORTDUMP within a loop.

2. A first word address (b) greater than the last word address (e) produces a diagnostic message on the printout.

3. When character or octal modes are used for type REAL variables, only the upper half of the last word will be printed.

4. Floating point mode for REAL variables converts two words of memory at a time to decimal output in the form

-.xxxxxxxxxx-eee

Examples:

1)

```
1    5  7
          :
          :
      MODE = 1
      IDENT = 3HK+1
      CALL FORTDUMP (MATRIX,MATRIX(16),MODE,IDENT)
```

MATRIX	contains	41	MATRIX+6	contains	47
MATRIX+1	"	42	MATRIX+7	"	0
MATRIX+2	"	43	MATRIX+8	"	0
MATRIX+3	"	44	:	:	:
MATRIX+4	"	45			
MATRIX+5	"	46	MATRIX+15	"	0

Result:

```
K+1  LOC 77625 A 42200160 Q 00000000 B1 00000 B2 77562 B3 00144 IMR 0017
OCTAL MEMORY
77324  00000051  00000052  00000053  00000054  00000055  00000056  00000057  00000000
GAP
                                   *END*
```

2)

```
1   5 7
       COMMON (A,B,C,D)
          :
       CALL FORTDUMP (A,D,7,4HK+12)
```

A contains -6.75432
B " 354.0000
C " .01
D " 6.754 E1

Result:

```
K+12  LOC 77663 A 42017600 Q 00000023  B1 06734 B2 00144 B3 00000 IMR 2020
REGISTER FILE
00000  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
  :       :         :         :         :         :         :         :         :
00070  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
FLOATING POINT MEMORY
77470  -.6754320000 001    .3540000000 003    .1000000000-001    .6754000000 002
                                   *END*
```

8.5 MACHINE CONDITION SUBPROGRAMS

Library functions and subroutines exist in FORTRAN that will test and modify sense lights, switches, and fault conditions. Most machine conditions may be tested or modified by either a subroutine or a function subprogram. The two methods differ primarily in the way a resulting value (if any) is stored for use by the program. The parameters must be of type integer. A subroutine is referenced with a CALL statement:

```
1    5  7
       CALL SSWTCH (I,J)
```

A function is referenced the same as other library functions.

```
1    5  7
       GO TO (10, 20) SSWTCHF(I)
```

Each function reference has at least one parameter even though it is not always used; DVCHKF, EXFLTF, and OVERFLF do not use input parameters.

8.5.1 SENSE LIGHT CONTROL

subroutine SLITE (i)
function SLITEF (i)

For i equals 1 through 24, either subprogram turns on sense light i. For i equals zero, the subprogram turns off all sense lights. For i other than 0 through 24, the subprogram provides a diagnostic. Neither subprogram returns results to the caller.

8.5.2 SENSE LIGHT TEST

subroutine SLITET (i, j)
function SLITETF (i)

For i equals 1 through 24, either subprogram tests sense light i. If sense light i is on, the subroutine or function turns it off. The subroutine then stores a one in variable j; the function returns a one (in the A register) to the referencing statement. If sense light i is off, a two is stored or returned.

For i other than 1 through 24, the subprogram provides a diagnostic.

8.5.3 SENSE SWITCH TEST

subroutine SSWTCH (i, j)
function SSWTCHF (i)

i may be 1 through 6 representing one of the 6 console sense switches. Either subprogram tests sense switch i. If sense switch i is on, the subroutine stores a one in variable j; the function returns a one (in the A register) to the referencing statement. If sense switch i is off, a two is stored or returned.

If i is other than 1 through 6, the subprogram provides a diagnostic.

8.5.4 DIVIDE FAULT CHECK

subroutine DVCHK (j)
function DVCHKF (i)

Either subprogram tests and turns off the divide fault indicator. If the indicator is on, the subroutine stores a one in variable j; the function returns a one (in the A register) to the referencing statement. If the indicator is already off, a two is stored or returned. (The i in the function reference is a dummy parameter that is required but not used.)

8.5.5 EXPONENT FAULT TEST

subroutine EXFLT (j)
function EXFLTF (i)

Either subprogram tests and turns off the exponent fault indicator. If the indicator is on, the subroutine stores a one in variable j; the function returns a one (in the A register) to the referencing statement. If the indicator is already off, a two is stored or returned. (The i in the function reference is a dummy parameter that is required but not used.)

8.5.6 OVERFLOW TEST

subroutine OVERFL (j)
function OVERFLF (i)

Either subprogram tests and turns off the overflow indicator. If the indicator is on, the subroutine stores a one in variable j; the function returns a one (in the A register) to the referencing statement. If the indicator is already off, a two is stored or returned. (The i in the function reference is a dummy parameter that is required but not used.)

INPUT/OUTPUT FORMAT SPECIFICATIONS 9

Data transmission and conversion between storage and external units require format lists and I/O control statements (Chapter 10). The I/O statements specify the input/output device and process, and list the data to be moved. For binary information, no format list is required. Format lists are provided in FORMAT statements or in specially prepared variable arrays.

9.1 I/O LIST

The list portion of an I/O control statement indicates the data elements and the order of transmission from left to right. Elements are simple variables or array names (subscripted or non-subscripted). All variables in the list must be standard types. A type other variable produces a compiler diagnostic. List elements are separated by commas; their order must correspond to the order of the format list.

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)

9.1.1 DO-IMPLYING SEGMENTS

A DO-implying segment consists of one or more list elements and indexing values. Dimensioned arrays may appear in the list with values specified for the range of the subscripts in an implied DO loop.

The general form for a DO-implying segment is:

$(\ldots((\text{list}, \gamma_1=m_1,m_2,m_3), \gamma_2=n_1,n_2,n_3), \ldots, \gamma_i=zz_1,zz_2,zz_3)$

$m_k, n_k, \ldots zz_k$ are unsigned integer constants or predefined positive integer variables. When the third indexing parameter ($m_3, n_3, \ldots zz_3$) is omitted, a one is used for incrementing.

γ_i are index variables.

A list element may be a simple variable, a dimensioned variable, or an array name.

The first index variable (γ_i) defined in the list is incremented first. Data named in the implied DO loops is transmitted by increments of m_3 until m_2 is exceeded. (When m_3 is omitted, the increment value is 1.) When the first index variable reaches m_2, it is reset; the next index variable to the right (γ_2) is then incremented and the process is repeated until the last index variable (γ_i) has been incremented.

The general form for arrays is:

$(((A(I,J,K),\ \gamma_1=m_1,m_2,m_3),\ \gamma_2=n_1,n_2,n_3),\ \gamma_3=p_1,p_2,p_3)$

I, J, K are subscripts of A and must be of the standard form.

$\gamma_1, \gamma_2, \gamma_3$ may represent I, J, K. $\gamma_1 \neq \gamma_2 \neq \gamma_3$

A DO-implying segment for an array may replace a nest of DO loops of the form:

```
1    5  7
        DO 10 γ3=p1, p2, p3
        DO 10 γ2=n1, n2, n3
        DO 10 γ1=m1, m2, m3
              .
              .
              .
        Transmit list elements
              .
              .
              .
   10   CONTINUE
```

An implied DO loop may also be used to transmit a simple variable, a sequence of variables, or an array a number of times. In the segment (A, K=1,10) A will be transmitted 10 times.

The limit to which implied DO loops may be nested is determined by the length of the statement.

Examples:

1) Example of a DO-implying segment nested 5 deep:

   (((((A(I,J,K), B(M), C(N), N=1,10,1), M=1,5), K=KK(1), KK(2), KK(3)), J=1,60,15), I=1,10,1)

   During execution each subscript (index variable) is set to the initial index value: I=1, J=1, K=KK(1), M=1, N=1. The segment replaces a DO loop nest of the form:

```
1    5  7
=========================================
     |  DO 15 I=1,10,1
     |  DO 15 J=1,60,15
     |  DO 15 K=KK(1), KK(2), KK(3)
     |  DO 15 M=1,5
     |  DO 15 N=1,10,1
     |     .
     |     .
     |  Transmit A(I,J,K), B(M), C(N)
     |     .
     |     .
  15 |  CONTINUE
```

2) Elements of A, a 3 by 3 matrix, will be transmitted by columns using:

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

3) Elements of A will be transmitted by rows using:

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

4) In the list segment (B(J), L, (A(I,L), I=1,L), J=3,9,3), L must have a value before it can be used as an index variable. The segment replaces a DO loop nest of the form:

```
1    5  7
=========================================
     |  DO 11 J=3,9,3
     |     .
     |     .
     |  Transmit B(J) and L
     |     .
     |     .
     |  DO 11 I=1,L
     |     .
     |     .
     |  Transmit A(I,L)
     |     .
     |     .
  11 |  CONTINUE
```

5) CAT, DOG, and RAT will each be transmitted 10 times with the segment

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

9.1.2 TRANSMISSION OF ARRAYS

In an I/O list, an array name without subscripts causes the entire array to be transmitted.

Examples:

```
      .
      .
      .
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

$\ldots, ((SPECS(I, J, K), I=1, 7), J=1, 5), K=1, 3), \ldots$

9.2 FORMAT STATEMENT

The binary coded decimal (BCD) I/O control statements require a format list for internal/external conversion of the I/O list elements. The list is usually given with a FORMAT statement:

FORMAT ($spec_1, \ldots k_x(spec_m, \ldots), k_y spec_n, \ldots$)

$spec_i$ are format specifications. The repetition factors k_i must be unsigned integer constants. When k is outside the parenthesis, the specifications within are all repeated k times. (Section 9.5)

The FORMAT statement is nonexecutable and can appear anywhere in the program. The word FORMAT is reserved and cannot be used as an identifier.

9.3 CONVERSION SPECIFICATIONS

Data elements in I/O lists are converted according to format lists that contain conversion specifications and editing codes.

FORTRAN conversion specifications:

Ew.d	Single precision floating point with exponent
Fw.d	Single precision floating point without exponent
Iw	Decimal integer conversion
Ow	Octal integer conversion
Aw	Alphanumeric conversion (left justification with blank fill)
Rw	Alphanumeric conversion (right justification with zero fill)

FORTRAN editing codes:

wX	Intra-line spacing
wH	Heading and labeling
/	Begin new record

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

9.3.1 Ew.d OUTPUT

The Ew.d specification converts floating point numbers in storage to the BCD character form for output. The field occupies w positions in the output record; the corresponding rounded floating point number appears right justified in the field as

$\underline{\wedge}\ \alpha.\alpha\ldots\alpha E\underline{\wedge}ee$	when $\lvert ee\rvert \leq 99$
$\underline{\wedge}\ \alpha.\alpha\ldots\alpha Eeee$	when $99 < eee \leq 308$
$\underline{\wedge}\ \alpha.\alpha\ldots\alpha$-eee	when $-308 \leq eee < -99$

$\alpha.\alpha\ldots\alpha$ are the most significant digits of the integer and fractional part; eee are the digits in the exponent. If d is zero or blank, the decimal point and digits to the right of the decimal do not appear. The fractional part contains a maximum of 11 digits. Field w must be wide enough to contain the integer portion, signs, decimal point, E and the exponent.

When the field is not wide enough to contain the output value, digits are dropped from the right of the fraction and the fraction sign may be lost. An asterisk is inserted immediately before exponent designator E if the negative sign is lost. A w field width less than 5 causes a format error. If the field is longer than the output value, the quantity is right justified with blanks in the excess positions to the left. For negative exponents that occupy 3 digits, E is suppressed.

Examples (TAPE1 and OUT are integer variables):

1) Proper use of Ew.d specification

RIGHT

```
1    5 7
      WRITE (TAPE1, 10)A
   10 FORMAT (E10.3)
```

A contains -67.32 or +67.32

Result: -6.732E^01 or ^6.732E^01

2) Minus sign not provided for

WRONG

```
1    5 7
      WRITE (TAPE1, 10)A
   10 FORMAT (E8.3)
```

A contains +67.32 or -67.32

Result: 6.73E^01 or 6.7*E^01

3) w is larger than required

```
1    5 7
      WRITE (OUT, 25) A
   25 FORMAT (E14.4)
```

A contains 412.679

Result: ^^^^ 4.1268E^02

4) E suppressed

```
1    5 7
      WRITE (OUT, 16)X
   16 FORMAT (E11.4)
```

X contains 4.12673×10^{-200}

Result: ^4.1267-200

9.3.2 Ew.d INPUT

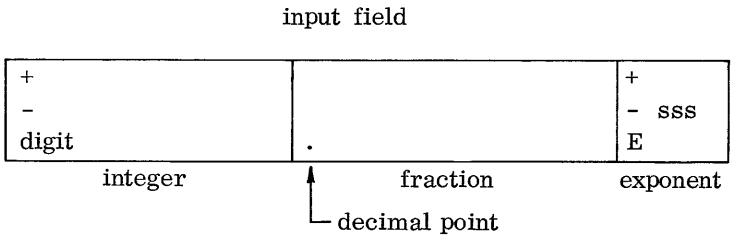
The Ew.d input specification converts the number in the input field to real and stores it in the appropriate location in memory.

w specifies the total number of characters in the input field. In the left-to-right scanning process, blanks in the field are interpreted as zeros.

The subfields for an input value may include integer, fraction, and exponent in the forms:

n.mEsss	n.m sss
n.m	.m sss
.mEsss	
n	
.m	
Esss	

Subfield structure of the input field:



For an integer (n) input value with no decimal point indicated, the E conversion specification scales the value.

An integer subfield begins with a sign (+ or -) or a digit followed by a string of digits and ends with a decimal point, E, sign, or end of w.

A fraction subfield begins with the decimal point, includes a string of digits, and ends with a sign, E, or the end of w.

An exponent subfield may begin with E or a sign. When it begins with E, the sign may appear between E and the digits in the exponent. The digits in the exponent must be less than or equal to 308; the entire input quantity must be in the range of $\pm 10^{308}$.

When no decimal point is present, d in the Ew.d specification is a negative power factor of ten. The internal representation of the input quantity becomes:

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

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

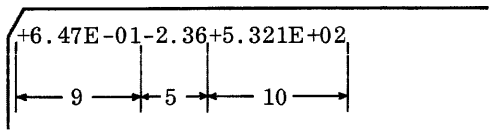
When E conversion is specified, and a decimal point occurs in the input constant, the decimal point overrides d. The input quantity 3.67294+5 may be read by any specification allotting necessary field length but will always be stored as 3.67294×10^{5}.

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

w field length must be the same as the length of the input field. When w is too long, incorrect numbers may be read, converted, and stored as shown below. When w is too short, a portion of the input field may be left unread. The w field includes significant digits (maximum of 11), signs, decimal point, E, and exponent.

Example:

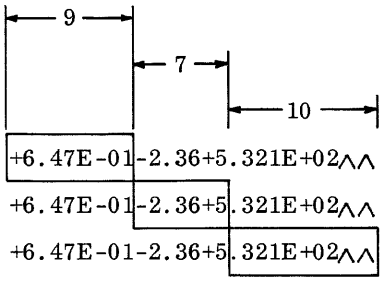
INPUT CARD



INCORRECT SPECIFICATION

```
|    5| 7
|     |READ 20, A,B,C
|  20 |FORMAT (E9.3, E7.2, E10.3)
```

Reading proceeds as follows:



First, +6.47E-01 is read, converted, and placed in location A.

Next, -2.36+5 is read, converted, and placed in location B. The number 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 legitimate under the definitions and restrictions.

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

In this example, numbers are incorrectly read, converted, and stored, yet there is no immediate indication that an error has occurred.

Examples:

Input Field	Ew.d Input Specification	Converted Value	Remarks
+143.26E-03	E11.2	.14326	Subfields all present
-12.437629E+1	E13.6	-124.37629	Subfields all present
8936E+004	E9.10	.008936	Input number converted as $8936 \times 10^{-10+4}$
327.625	E7.3	327.625	No exponent subfield
4.376	E5	4.376	No d in specification
-.0003627+5	E11.7	-36.27	Integer subfield contains only minus
-.0003627E5	E11.7	-36.27	Integer subfield contains only minus
1E1	E3.0	10.	Input number converted as $1. \times 10^{1}$
E+06	E10.6	0.	No integer or fraction subfield. Zero stored regardless of exponent
1.E 1	E6.3	10.	Blanks interpreted as zeros

9.3.3 Fw.d OUTPUT

The Fw.d specification converts floating point numbers in storage to BCD form for output. The field occupies w positions in the output record; the corresponding list element must be a floating point quantity that is converted and rounded to a decimal number, right justified in the w field, as:

$\wedge \delta \cdots \delta . \delta \cdots \delta$

δ represents the most significant digits of the number (maximum 11). The number of decimal places to the right of the decimal is specified by d. If d is zero or omitted, the decimal point and digits to the right do not appear. If the number is positive, the + sign is suppressed.

If the field is too short to accommodate the number, high-order digits are discarded from the left, the sign is suppressed, and an asterisk appears in the rightmost character position to indicate the deletion.

If the field w is longer than required to accommodate the number, it is right justified with blanks occupying the excess field positions to the left.

Examples (TAPE1 and OUT are integer variables):

1) Proper Specification

```
1    5 7
       WRITE (TAPE1, 10)A
   10  FORMAT (F10.5)
```

A contains +123.45678 or -123.45678

Result: ^123.45678 or -123.45678

2) w too small to accommodate integer portion

```
1    5 7
       WRITE (TAPE1, 10)A
   10  FORMAT (F8.5)
```

A contains +123.45678

Result: 23.4567*

3) w too small to accommodate sign

```
1    5 7
       WRITE (42, 12)A
   12  FORMAT (F6.3)
```

A contains -67.460

Result: 67.46*

4) w larger than required

```
1    5 7
       WRITE (OUT, 25)A
   25  FORMAT (F10.3)
```

A contains 412.6727

Result: ^^^412.673

9.3.4 Fw.d INPUT

The Fw.d specification converts a number in an input field (specified by w) to real and stores it in memory. The input field consists of an integer and a fraction subfield. An omitted subfield is assumed to be zero.

Permissible subfield combinations are:

Integer fraction

Integer by itself

Fraction by itself

An integer subfield begins with a digit, + or -; blanks in the field are interpreted as zeros. The integer field is terminated by a period, or by the end of the input field.

A fraction subfield begins with a decimal point; it is terminated by the end of the input field.

In the Fw.d specification, d acts as a negative power factor of ten when the fraction subfield is not present. The internal representation is: (integer subfield) x 10^{-d}. For example, the specification F4.4 causes the input quantity 3267 to be converted and stored as 3267 x 10^{-4} = .3267.

A decimal point in the input quantity causes d to be ignored. For example, 3.6789 may be read under any F6.d specification but will always be stored as 3.6789.

When d does not appear it is assumed to be zero. For example, the input quantity +14.62 is read into memory as 14.62 by the specification F6.

The maximum number of significant digits that may appear in the combined integer-fraction field is 11. Excess digits are discarded from the right during the conversion process.

The field length specified by w in Fw.d should always be the same as the actual length of the input field containing the input number. When it is too long, incorrect numbers may be read, converted and stored. When it is too short, significant digits may be lost.

Examples:

Input Field	Fw.d Input Specification	Converted Value	Remarks
367.2593	F8.4	367.2593	Integer and fraction field
37925	F5.7	.0037925	No fraction subfield. Input number converted as 37925 x 10^{-7}
-4.7366	F7	-4.7366	No d in specification
.62543	F6.5	.62543	No integer subfield
.62543	F6.d	.62543	Decimal point overrides d of specification
+144.15E-03	F11.2	.14415	Exponents are legitimate in F input

9.3.5 iw OUTPUT

The Iw specification converts decimal integer values in the output list to BCD character form for output. The output quantity occupies w output record positions; it will appear right justified in the field w, as:

$$\underline{\wedge}\,\delta_1 \cdots \delta_n$$

δ_i represent decimal digits (maximum 7) of the integer. When the integer is positive the + sign is suppressed.

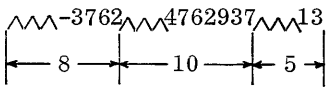
When the field w is larger than required, the output quantity is right justified with blanks occupying excess positions to the left. When the field is too short, characters are discarded from the left; an asterisk inserted at the right indicates deletion.

Example:

```
1    5 7
      WRITE (K1, 10) J,K,L
   10 FORMAT (I8, I10, I5)
```

J contains -3762
K contains +4762937
L contains +13

Result:



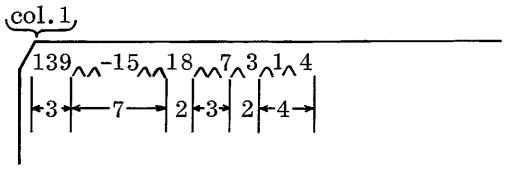
9.3.6 Iw INPUT

The Iw input specification converts the input field to a decimal integer. The field is w characters in length and the corresponding list element must be a decimal integer quantity.

Input field w consists of an integer subfield that contains only a plus or minus, 0 through 9, or blank, interpreted as zero. When a sign appears, it must precede the first digit in the field. The value is stored right-justified in the specified variable.

Example:

Input Card




```
1    5 7
      READ (K3, 10) I,J,K,L,M,N
   10 FORMAT (I3,I7,I2,I3,I2,I4)
```

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

9.3.7
Ow OUTPUT

The Ow output specification converts internal binary to BCD octal integers. The output quantity occupies w output record positions and appears as:

$\delta_1 \cdots \delta_n$

δ_i represent octal digits (maximum 8 for types integer and character, 16 for type real). No sign appears; a negative octal number is represented as it appears in storage in one's complement form. If w is greater than required, the number is right justified. If w is too small, the rightmost octal digits in storage occupy the output field; the left portion of the word is lost.

Example:

```
1    5 7
       WRITE (K1, 12) I,J,K,A
   12  FORMAT (03,08,04,018)
```

I contains 00000144_8 $= 100_{10}$
J " 00002450_8 $= 1320_{10}$
K " 00000044_8 $= 36_{10}$
A " $2012452275000000 = 1124.572_8 = 596.739_{10}$

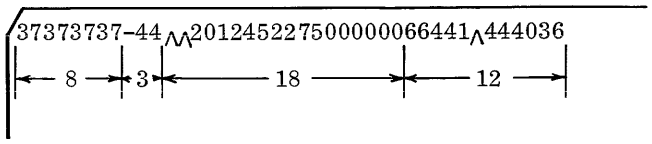
Result: 144^^^^2450^^44^^2012452275000000

9.3.8
Ow INPUT

The Ow input specification provides a method of entering octal quantities into storage. The input field w has a maximum of 8 octal digits for character or integer and 16 for real. The string of octal digits may be preceded by a sign; a negative sign causes the one's complement of the quantity to be stored. Blanks in the field are interpreted as zeros. Only octal digits (0-7) may appear.

Example:

Input Card




```
1    5 7
          READ (K1, 10) K,L,P,M
   10 FORMAT (08,03,018,012)
```

Result: K contains 37373737
L " 77777733 (complement form)
P " 20124522 | 75000000
M " 10444036 (6644 lost)

9.3.9 Aw OUTPUT

With the Aw output specification internal BCD (Appendix A) is converted to external alphanumeric characters.

$\alpha_i \ldots \alpha_n$

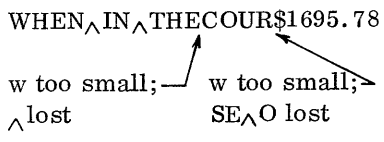
α_i represent alphanumeric characters. Maximum is 4 characters for types integer; 8 for type real. When w is larger than required, the character string is right justified with blank fill to the left. When the field is too small, the leftmost characters appear in the output field; any other characters are lost.

Example:

```
1    5 7
          INTEGER A,B,C
          WRITE(K5,14) A,B,C,R,Q
   14 FORMAT (2A4, A3, A4, 2A8)
```

A contains 66302545 = WHEN
B " 60314560 =^IN^
C " 63662560 = THE^
R " 6346245122656046 = COURSE^O
Q " 5301061105330710 = $1695.78

Result:

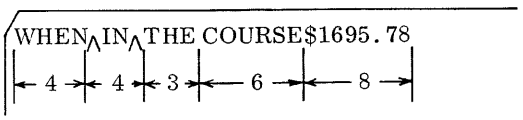


9.3.10 Aw INPUT

The Aw input specification accepts up to four 6-bit characters for types integer and character variables, and eight 6-bit characters for type real variables. A blank in the input field is converted to the 6-bit equivalent BCD code for blanks (60). If w exceeds the allowed number of characters (4 or 8), only the rightmost characters are stored in the variable defined in the I/O list. If w is less than the allowed number, the characters in the input field are stored left justified in the variable with blank fill to the right.

Example:

Input Card




```
1    5 7
      INTEGER A,B,C
      READ (K4,10) A,B,C,R,Q
   10 FORMAT (2A4,A3,A6,A8)
                        incorrect specification
```

Result:

A contains	66302545	=	WHEN
B "	60314560	=	∧IN∧
C "	63662560	=	THE∧ (left justified with blank fill)
R "	64516225	=	URSE (CO lost)
Q "	5301061105330710	=	$1695.78

9.3.11 Rw OUTPUT

The Rw output specification is similar to the Aw specification for converting internal BCD words (Appendix A) to external alphanumeric characters. The w output field is:

$\alpha_1 \ldots \alpha_n$

α_i represent alphanumeric characters (Maximum is 4 characters for types integer; 8 characters for type real.) When w is larger than required to represent the characters, the character string is right justified with zero fill at the left. When the field is too small, the rightmost characters appear in the output field; any other characters are lost.

Example:

```
1   5  7
       INTEGER A,B,C
       WRITE (K5,14) A,B,C,R,Q
   14  FORMAT (2R5,R3, R4, R8)
```

A	contains	66302545	= WHEN
B	"	60314560	= ∧IN ∧
C	"	63662560	= THE∧
R	"	6346245122656046	= COURSE∧ O
Q	"	5301061105330710	= $1695. 78

Result:

```
|← 5 →|← 5 →|← 3 →|← 4 →|←  8  →|
 0WHEN0∧IN∧HE∧SE∧O$1695.78
```

w too large; zero fill (0WHEN, 0∧IN∧)

w too small; T lost; COUR lost (HE∧, SE∧O)

9.3.12 Rw INPUT

The Rw input specification, like the Aw specification, accepts up to four 6-bit characters for types integer and character variables and eight 6-bit characters for type real variables. A blank in the input field is converted to the 6-bit equivalent BCD code for blanks (60). If w exceeds the allowed number of characters (4 or 8) only the rightmost characters are stored in the variable defined in the I/O list. If w is less than the allowed number, the characters in the input field are stored right justified with zero fill at the left.

Example:

Input Card

```
WHEN^IN^THECOURSE$1695.78
|<-4->|<-4->|<-3->|<--6-->|<--8-->|
```

```
1    5  7
        INTEGER A,B,C
        READ (K4,10) A,B,C,K,Q
     10 FORMAT (2R4,R3,R6,R8)
```

incorrect specifications (R3, R6)

Result:

A	contains	66302545	= WHEN
B	"	60314560	= ^IN^
C	"	00636625	= 0THE (right justified with zero fill)
K	"	64516225	= URSE (CO lost)
Q	"	5301061105330710	= $1695.78

9.4 EDITING SPECIFICATIONS

Editing specifications define spacing between characters and lines, skip records, begin new records, and provide a method of adding headings and comments.

9.4.1 SPACE (wX)

The wX specification produces blanks in an output record or skips w characters on input to permit spacing of input/output quantities. If w is zero or blank, a value of 1 is assumed. The comma after X is optional; see example 2.

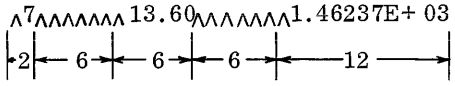
Examples:

1) Output Spacing

```
1    5 7
       INTEGER A
       WRITE (K1,10)A,B,C
    10 FORMAT (I2,6X,F6.2,6X,E12.5)
```

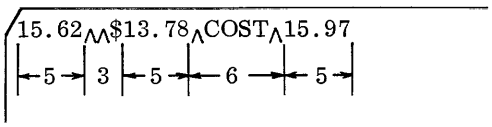
A contains 7
B " 13.6
C " 1462.37

Result: ^7^^^^^^13.60^^^^^^1.46237E+03



2) Skipping on Input

Input Card




```
1    5 7
       READ (K3,11)R,S,T
    11 FORMAT (F5.2,3X,F5.2,6X,F5.2)
             or
    11 FORMAT (F5.2,3XF5.2,6XF5.2)
```

Result: R contains 15.62
S " 13.78 (^^$ spaced over)
T " 15.97 (^COST^ spaced over)

9.4.2 wH OUTPUT

The wH output specification provides a method of including a set of w 6-bit Hollerith characters (Appendix A) in the output record in the form of comments, titles, headings, and carriage control characters.

An unsigned integer w specifies the number of characters to the right of H to be included in the output record. If w = 0, or is missing, a value of 1 is assumed. The comma following the wH specification is optional.

Examples:

1) No I/O List

```
|1    5|7
=======================================================
      |WRITE (K7,20)
   20 |FORMAT (28H BLANKS COUNT IN AN H FIELD.)
```

Output record: ∧ BLANKS ∧ COUNT ∧ IN ∧ AN ∧ H ∧ FIELD.

2) Mixed Specifications

```
|1    5|7
=======================================================
      |WRITE (K4,30)A
   30 |FORMAT (6H LMAX=,F5.2)
```

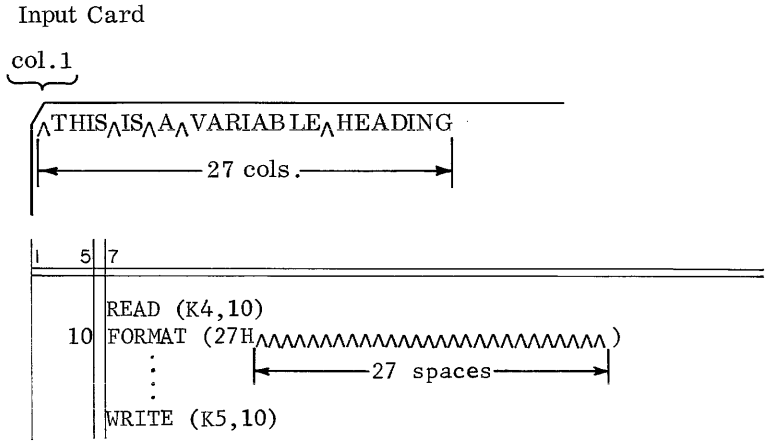
A contains 1.5

Output record: ∧LMAX=∧1.50

9.4.3 wH INPUT

With the H specification, a new heading is read into an existing H field. When the new characters on an input record are read, the corresponding characters are placed into the format list designated in the I/O statement. A subsequent output statement puts the new characters in the output record. w specifies the number of characters in the input field.

Example: Input Card:



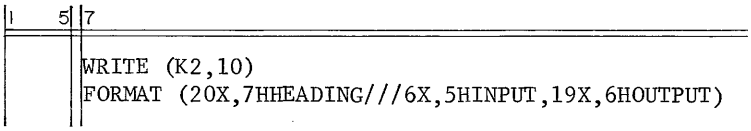
Result: ^THIS^IS^A^VARIABLE^HEADING

9.4.4 NEW RECORD (/)

A slash, signaling the end of a BCD record, may appear anywhere in the specifications list. It need not be separated from other list elements by commas; consecutive slashes may appear. During output, it is used to start new records, cards, or lines. During input, / specifies the beginning of the next card or record.

Example:

1)



Output Record:

^^^^^^^^^^^^^^^^^^^^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.

2) A contains -11.6
B " .325
C " 46.327
D " -14.261

```
      WRITE (K1,11) A,B,C,D
   11 FORMAT (2E10.2/2F7.3)
```

Result: ^-1.16E^01^^3.25E-01 Line 1
^46.327-14.261 Line 2

```
      WRITE (K2,11) A,B,C,D
   11 FORMAT (2E10.2/ /2F7.3)
```

Result: ^-1.16E^01^^3.25E-01 line 1
line 2
^46.327-14.261 line 3

3) AMAX contains 3.62
AMAX+1 " -4.03
AMAX+2 " -9.78

```
      WRITE (K1,15) (AMAX(I),I=1,3)
   15 FORMAT (8H^ANSWERS2 (/) 3F8.2)
```

Result: ^ANSWERS line 1
line 2
^^^^3.62^^^-4.03^^^-9.78 line 3

9.5 REPEATED SPECIFICATIONS

A Format specification may be repeated by using an unsigned integer constant k as a repetition factor as follows:

k (spec)

spec may be any conversion specification.

For example, if two quantities K, L are to be printed, the program could be written:

```
|1    5|7
=========================================
      |WRITE (K1,10)K,L
   10 |FORMAT (I2,I2)
```

Since the specifications for K, L are identical, the FORMAT statement may be written:

```
|1    5|7
=========================================
   10 |FORMAT (2I2)
```

When a group of format specifications repeats itself, as in

FORMAT (E15.3, F6.1, I4, I4, E15.3, F6.1, I4, I4)

the use of k produces:

FORMAT (2(E15.3, F6.1, 2I4))

The parenthetical grouping of the format specifications is called a repeated group.

Repeated groups may not be nested; nor may parentheses be used within repeated groups.

Examples:

1) ILLEGAL:

```
|1    5|7
=========================================
   10 |FORMAT (1H1, 5(25X, (F6.2)))
   11 |FORMAT (E5.4,3(4X, F5.1, 2(E6.2)))
```

2) LEGAL:

```
|1    5|7
=========================================
   12 |FORMAT (1H1(25X,5(F6.2)))
```

9.6 VARIABLE FORMAT

Format lists need not be provided with FORMAT statements; instead, they can be placed in arrays. Placing format lists in arrays and referencing the arrays instead of the FORMAT statement permits the programmer to change, index, and specify formats at the time of execution.

Format arrays are prepared by storing a format list, including left and right parentheses, as it would otherwise appear with a FORMAT statement. Variable specifications can be read in from cards, changed with replacement statements, or preset in labeled common with a DATA statement.

Examples:

1) Prepare an array for format list

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

```
1    5  7
      DIMENSION IVAR (8)
      READ (K1,1) (IVAR (I), I = 1,8)
    1 FORMAT (8A4)
```

Result:

IVAR	contains	(E12	IVAR+4	contains	E20.
IVAR+1	"	.2,F	IVAR+5	"	3,F9
IVAR+2	"	8.2,	IVAR+6	"	.3,I
IVAR+3	"	I7,2	IVAR+7	"	4)^^

When using the specifications, reference the array:

```
1    5  7
      WRITE (K2, IVAR (1)) A,B,I,C,D,E,J
        or
      WRITE (K2, IVAR) A,B,I,C,D,E,J
```

Specifications can be changed with replacement statements:

```
1    5 7
      IVAR (4) = 4HΛΛΛ2
```

removes I7 from the format list, permitting

```
1    5 7
      WRITE (K2, IVAR) A,B,C,D,E,J
```

2) Prepare two lists that can be selected by indexing at time of execution.

Lists: (E12.2, F8.2, 2I7)
(F8.2, E12.2, 2I7)

```
7
COMMON/DATA/LAIS (8)
DATA (LAIS = 4H(E12,4H.2,F,4H8.2,4H2I7),4H(F8.,4H2,E1,4H2.2,4H2I7)
```

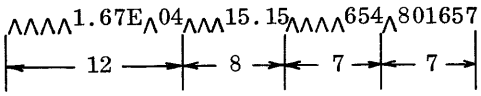
Result:

LAIS	contains	(E12	LAIS+4	contains	(F8.
LAIS+1	"	.2,F	LAIS+5	"	2,E1
LAIS+2	"	8.2,	LAIS+6	"	2.2,
LAIS+3	"	2I7)	LAIS+7	"	2I7)

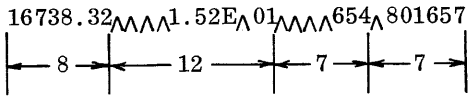
These could be referenced as:

```
1    5 7
          .
          .
          .
      I = 1
      Go to 16
          .
          .
          .
      I = 5
   16 WRITE (K3, LAIS (I)) A,B,I,J
          .
          .
          .
```

Result: When I = 1



When I = 5



9.7 CARRIAGE CONTROL

The first character of a listable output record is used for printer carriage control, and is not printed. Usually, this character is an H format in a FORMAT specification which is used by a PRINT or WRITE on OUT statement.

Character	Action Before Print	Action After Print
1	Skip to format level 8	Space one line
2	Skip to format level 7	Space one line
3	Skip to format level 6	Space one line
4	Skip to format level 5	Space one line
5	Skip to format level 4	Space one line
6	Skip to format level 3	Space one line
7	Skip to format level 2	Space one line
8	Skip to format level 1	Space one line
A	No space	Skip to format level 8
B	No space	Skip to format level 7
C	No space	Skip to format level 6
D	No space	Skip to format level 5
E	No space	Skip to format level 4
F	No space	Skip to format level 3
G	No space	Skip to format level 2
H	No space	Skip to format level 1
blank	No space	Space one line
0 (zero)	Space one line	Space one line
-	Space two lines	Space one line
*	No space	No space
other	No space	Skip to format level 1

INPUT/OUTPUT STATEMENTS 10

Input/output statements control the transfer of information between the storage unit and an external device.

In the I/O control statements:

- i indicates the logical unit number; must be a simple integer variable or an integer constant.
- n identifies the format list. It is either a FORMAT statement number or the name of the variable containing the format list. Binary data transmission does not require n.
- list indicates the I/O list (Section 9.1)

Binary data is transmitted with odd parity-checking bits; BCD with even parity-checking bits.

10.1 OUTPUT STATEMENTS

10.1.1 PRINT RECORD

PRINT n, list

Transfers information from the storage locations in list to the standard output unit. This information is transferred as line printer images, 136 characters or less per line, in accordance with format list 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.

Example:

```
1    5  7
       PRINT 16, A
    16 FORMAT (10H^RESULT^=^, F7.3)
          .
          .
          .
       PRINT MESS, ADDR
          .
          .
          .
```

Array MESS is a variable format list. It currently is:

(10H^ERROR^AT^, 08)

MESS	contains	(10H
MESS+1	"	^ERR
MESS+2	"	OR^A
MESS+3	"	T^, O
MESS+4	"	8)^^

10.1.2

PUNCH CARD RECORD PUNCH n, list

Transfers information from the memory locations given by the list to the standard punch unit. This information is transferred as Hollerith images, 80 characters or less per card in accordance with the format list n. A maximum of 80 characters is permitted on one card.

10.1.3

WRITE BINARY RECORD WRITE (i) list

WRITE TAPE i, list

Transfer binary information from the storage locations given by the list identifiers to the specified logical unit (i may be 1-55). If the list is omitted, the statement acts as a no operation.

The number of elements in the list determines the number of physical records to be written on the unit. A physical record contains a maximum of 128 words; the first word is a count word, the remaining 127 words contain the transmitted data. The last physical record may contain from 2 to 128 words. The physical records written by one write binary record statement constitute one logical record. Information is recorded in odd parity (binary mode), as illustrated in figures 1a and 1b.

For k physical records in the logical record, the first word of all records except the kth contains zero; the first word of the kth record contains the integer k. If there is only one physical record, the first word contains the integer 1.

Examples:

```
1    5 7
      DIMENSION A(260), B(4)
      WRITE (10) A, B
             or
      WRITE TAPE 10, A, B
          :
          :
      DO 5 I = 1, 10
    5 WRITE (6) AMAX(I),(M(I,J), J=1,5)
             or
    5 WRITE TAPE 6, AMAX(I),(M(I,J), J=1,5)
```

10.1.4 WRITE BCD RECORD

WRITE (i,n) list

WRITE OUTPUT TAPE i,n,list

Transfer information from storage locations given by identifiers in the list to tape unit (i) according to the format list n. i may be 1 to 55, 59, 61, or 62.

A logical record containing up to 136 characters is written on magnetic tape in even parity (BCD mode). Each logical record is one physical record. The number of elements in the I/O list and the format list (n) determines the number of records to be written on a unit. If the logical record is less than 136 characters, the remainder of the record is filled with blanks to the nearest multiple of 4 characters.

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

WRITE: BINARY (ODD PARITY) k WORDS

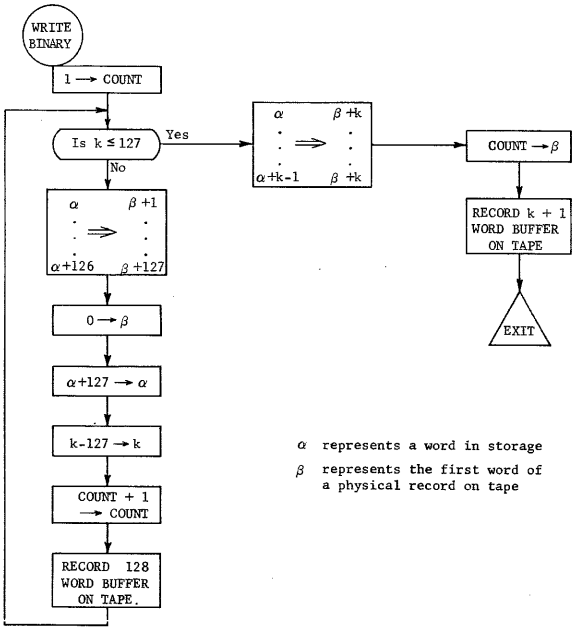


Figure 1a

MEMORY TAPE SCHEMATIC

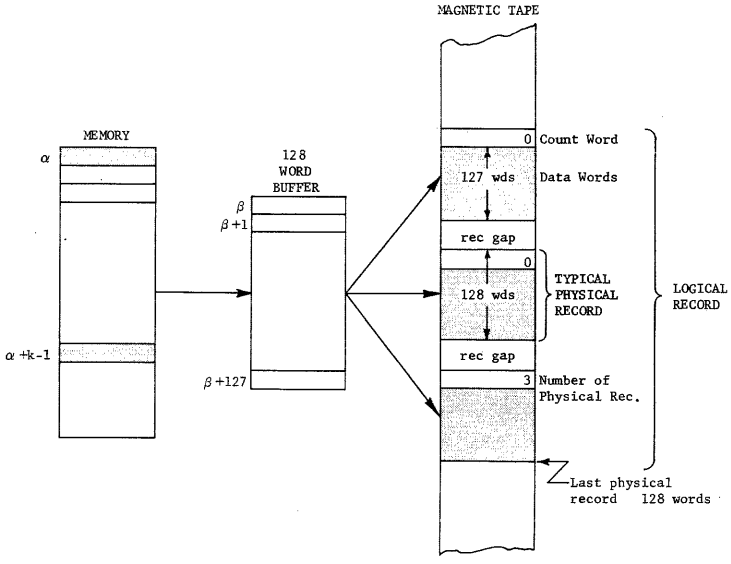


Figure 1b

Examples:

```
|    5|7
|     |WRITE (10,20) A,B,C
|     |                                        or
|     |                                  WRITE OUTPUT TAPE 10,20,A,B,C
|  20 |FORMAT (3F10.6)
|     |WRITE (4,21)
|     |                                        or
|     |                                  WRITE OUTPUT TAPE 4,21
|  21 |FORMAT (33H THIS STATEMENT HAS NO DATA LIST.)
```

10.2
INPUT STATEMENTS

10.2.1
READ CARD RECORD

READ n, list

Reads one or more card images from the standard input unit, converting the information, from left to right, in accordance with format list n, and stores the converted data in the locations named in the I/O list. The images read may come from 80-column Hollerith cards or from magnetic tapes prepared off-line containing 80-character records in BCD mode.

Example:

```
|    5|7
|     |READ 10, A,B,C
|  10 |FORMAT (3F10.4)
```

10.2.2
READ BINARY RECORD

READ (i) list

READ TAPE i, list

Transfer one logical record of information from logical unit i (1 through 55) to storage locations named by the list identifiers.

The record being read must have been written in binary mode by a WRITE (i) list or WRITE TAPE i, list statement. The word count is not transmitted to the input area. The number of words in the list must be equal to or less than the number of words in the corresponding write statement.

When the list is omitted, the binary read statement spaces over one logical record.

Examples:

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

10.2.3

READ BCD RECORD

READ (i, n) list

READ INPUT TAPE i, n, list

Transfer one logical record of information from logical unit i (1 through 55, 58, 60) to storage locations specified in the list according to format list 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 BCD mode). The record being read must have been written in BCD mode; reading a binary record with a BCD read statement causes a parity error.

Examples:

```
      READ INPUT TAPE 10, 11, X, Y, Z
          or
      READ (10,11) X, Y, Z
         .
         .
         .
   11 FORMAT (3F10.6)
         .
         .
         .
      I = 5
      READ (2, MB(I)) (Z(K), K = 1,8)
```

Array MB contains indexed variable specifications

```
MB (1) contains (F10
MB (2)     "     .4)^
MB (3)     "     (F10
MB (4)     "     .3)^
MB (5)     "     (F7.
MB (6)     "     2)^^
```

Because I = 5, the input format of array Z is F7.2

10.3 BUFFER STATEMENTS

Primary differences between the buffer and read/write I/O statements are:

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

2. The buffer control statements are not associated with I/O or format lists; data transmission occurs (without conversion) to or from one area in storage. Read/write control statements, however, are associated with I/O lists and, in BCD transmission, with format lists.

3. Only one physical record is transferred per buffer request.

4. 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.

5. Buffer statements permit reverse reading.

10.3.1 BUFFER IN

BUFFER IN (i,p) (a,b)

- i logical unit (1 to 55)
- p direction and mode;
  - p = 0 forward read, BCD mode
  - 1 forward read, binary mode
  - 2 reverse read, BCD mode
  - 3 reverse read, binary mode
- a first variable of the block to be transmitted
- b last variable of the block to be transmitted

This statement transmits one physical record of information in mode p from unit i to storage locations a through b. If the tape being read was written by a BCD write statement, only one physical record (136 characters or less) is read. Provision must be made for the count word which is buffered in with the data written with binary write statements. When p = 2 or 3, the record is read in reverse and the words stored from b to a.

When BUFFER IN requests transmission of more words than are on the record, the words are stored from a to k, where k is less than b and contains the last word on the record, regardless of the direction of the read.

A magnetic tape written in binary mode must be read in odd parity; a tape written in BCD mode must be read in even parity.

The first word address (a) must be less than or equal to the last word address (b) or the job is terminated.

10.3.2 BUFFER OUT

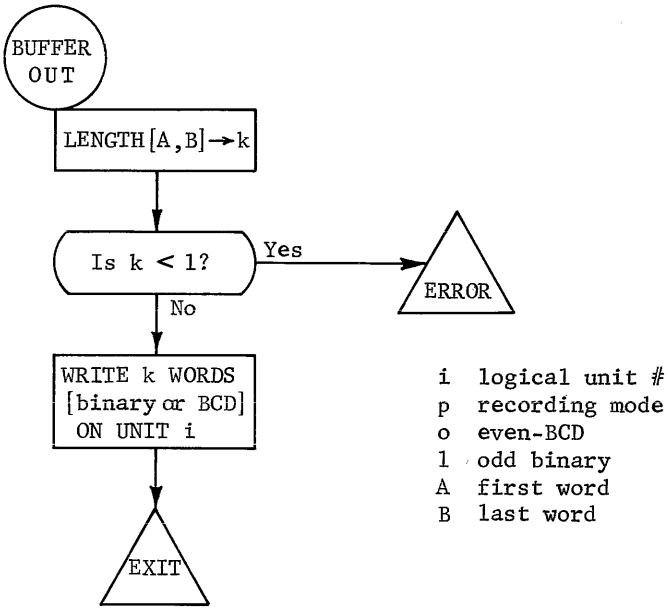
BUFFER OUT (i,p) (a,b)

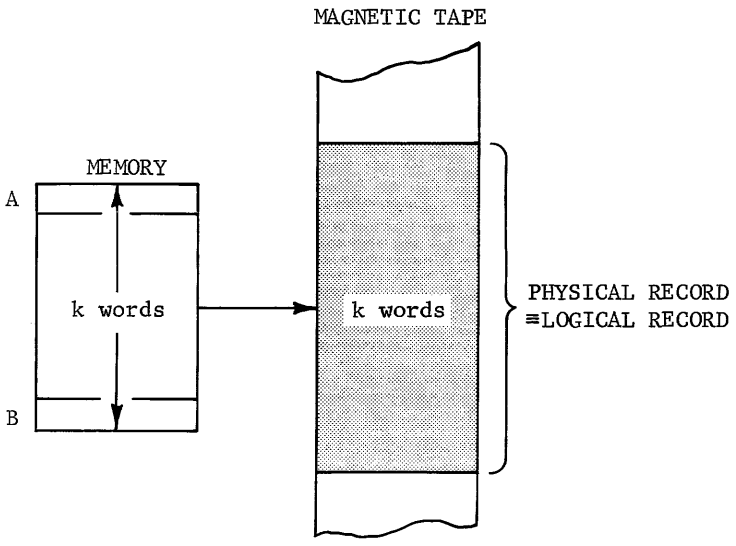
- i logical unit (1 to 55)
- p 0 BCD mode
  1 binary mode
- a first variable of block to be transmitted
- b last variable of block to be transmitted

The statement 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. Attempting to reverse BUFFER OUT by using p = 2 or 3 causes a diagnostic.

BUFFERED WRITE: BINARY OR BCD

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



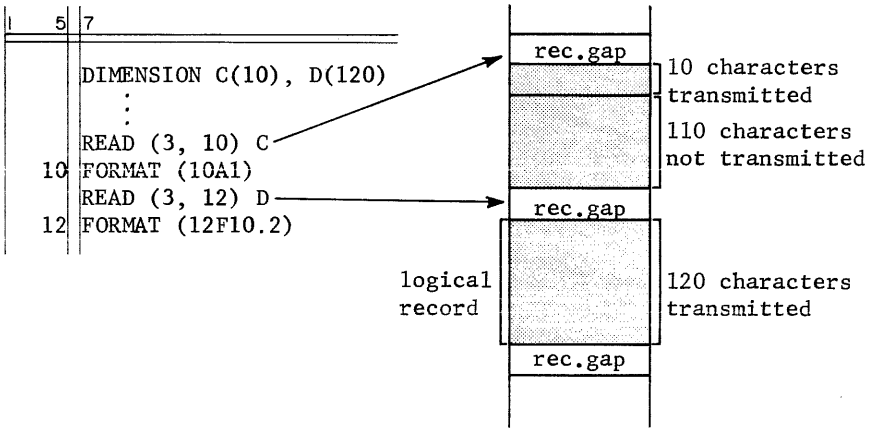


The first word address (a) must be less than or equal to the last word address (b) or the job is terminated.

10.4 PARTIAL RECORD

After a BCD or binary read, the tape always moves to the next logical record; after a BUFFER IN statement, the tape moves to the next physical record even though the entire record may not have been transmitted. Consequently, the next READ or BUFFER IN statement does not read the remainder of the record. For a reverse BUFFER IN, the tape moves to the beginning of the same physical record.

Example:



10.5 TAPE HANDLING STATEMENTS

When a REWIND or a BACKSPACE is the next I/O operation after a write operation, the FORTRAN I/O routine writes an end-of-file, backspaces over it, and then executes the command.

The logical unit number, i, is an integer variable or an integer constant.

10.5.1
REWIND

REWIND i

Rewinds the magnetic tape mounted on unit i (1 to 55) to the load point. When the tape is already rewound, the statement acts as a do-nothing statement.

10.5.2
BACKSPACE

BACKSPACE i

Backspaces the magnetic tape mounted on unit i (1 to 55, 60, 61, or 62) one logical record. (A logical record is a physical record except for tapes written by a WRITE (i) list or WRITE tape i, list statement.) When the tape is already at the load point (rewound) BACKSPACE i acts as a do-nothing statement.

10.5.3
ENDFILE

ENDFILE i

Writes an end-of-file on the magnetic tape mounted on unit i (1-55). A diagnostic is provided if i is not a magnetic tape unit.

10.6
STATUS CHECKING

The FORTRAN library contains functions and subroutines that check status after I/O operations.

All parameters must be type integer.

EOFCK and IOCHK are used with read/write I/O control statements. When they reference buffered units, the job terminates abnormally.

10.6.1
END-OF-FILE CHECK

subroutine EOFCK (i, j)
function EOFCKF (i)

Either subprogram checks the status of the previous I/O request on logical unit i. If an end-of-file was encountered while trying to read, the subroutine stores a one in variable j; the function returns a one (in the A register) to the calling statement. A two is stored or returned if the condition did not occur.

10.6.2 PARITY CHECK

subroutine IOCHK (i, j)
function IOCHKF (i)

Either subprogram checks the status of the previous I/O request on logical unit i to determine if a parity error has occurred. When an error has occurred, the subroutine stores a one in variable j; the function returns a one (in the A register) to the calling statement. A two is returned when no parity error has occurred.

10.6.3 UNIT STATUS TEST

subroutine UNITST (i, j)
function UNITSTF (i)

Either subprogram checks the status of the last buffer operation on logical unit i. For the subroutine, a value is stored in j; for the function, a value is returned in the A register where it is used by the calling statement. The value is determined accordingly:

j = 1 if the previous buffer operation is not complete.

2 if the buffer operation is complete with no errors.

3 if the buffer operation is complete, but an end-of-file has been sensed.

4 if the buffer operation is complete, but a parity error occurred.

10.6.4 LENGTH TEST

function LENGTHF (i)

Determines the number of words transferred during the last BUFFER IN operation on unit i. A unit status test must precede a length test. Call UNITST or UNITSTF to insure that the last operation on the unit is complete prior to calling LENGTHF. The number of words is returned in the A register for use by the calling statement.

Example:

```
1    5  7
                .
                .
                .
       J = 1                              Set flag 1
       BUFFER IN (10,0) (A,Z)             Initiate buffered read for
                                          logical unit 10, even parity.
                                          The first word of the block
                                          is A; the last Z
     4 GO TO (5, 6, 7, 8), UNITSTF (10)   Check transmission status
     5 GO TO (50, 4), J                   Read not finished: perform
                .                         other calculations at
                .                         statement 50
                .
    50  computation not involving
        information in locations A-Z
                .
                .
                .
       J = 2                              Calculations complete;
                                          increase flag by 1
       GO TO 4
     7 KERR = LENGTHF (10)                KERR will contain number
                                          of words read
       WRITE (21, 70)                     Error message
    70 FORMAT (12H EOF UNIT 10)           End-of-file error
       GO TO 200
     8 KERR = LENGTHF (10)                KERR will contain number
                                          of words read
       WRITE (21,80)
    80 FORMAT (12HPARITY ERROR)
   200 REWIND 10
       STOP
     6 CONTINUE                           BUFFERED transmission
                .                         complete; continue
                .                         program
                .
```

10.6.5 ERROR PROCEDURES FOR TAPES

Attempting to read past an EOF without checking for EOF causes job termination. For BCD or binary operations, EOFCK or EOFCKF sense the condition; for BUFFER IN, UNITST or UNITSTF sense the condition.

EOF should be checked before parity. For read parity errors, the input routine retries the read three times. If the error persists, a diagnostic is provided and control returns to the program.

On write parity errors, the output routine attempts to rewrite the record by backspacing over the last physical record, erasing 6 inches of tape, and writing. For BCD or binary operations, IOCHK or IOCHKF sense for parity errors; for BUFFER IN, UNITST or UNITSTF check parity.

10.7 INTERNAL TRANSMISSION

The ENCODE/DECODE statements are comparable to the BCD write/read statements with the essential difference that no peripheral equipment is used in the data transfer. Information is transferred under format specifications from one area of storage to another.

In the following descriptions:

- n is a FORMAT statement number, a variable identifier or a formal parameter representing the associated format list.
- list is the input/output list.
- v is a variable identifier or an array identifier that supplies the starting location of the records. The identifier may be subscripted.
- c is an unsigned integer or an integer variable (simple or subscripted) specifying the length of a record. c may be an arbitrary number of BCD characters. The first record starts with the leftmost character of the location specified by v and contains c BCD characters.

10.7.1 ENCODE

ENCODE (c, n, v) list

Converts the information in the list according to format list n and stores it in locations starting at v, c BCD characters per record. If the format list attempts to convert more than c characters per record, a diagnostic occurs. If the number of characters converted by the format list is less than c, the remainder of the record is filled with blanks.

When c is not a multiple of 4, the last record does not fill a computer word; the remainder of the word is blank-filled.

10.7.2 DECODE

DECODE (c, n, v) list

Converts and edits information from records consisting of c consecutive BCD characters (starting at address v) according to format list n and stores it in the I/O list. When the format list specifies more than c characters per record, a diagnostic is provided. If DECODE attempts to process a character illegal under a given conversion specification, a diagnostic occurs. When fewer than c characters are specified, the remainder of the word is ignored.

Examples:

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

   LOC(1) contains SSxx

   ⋮

   LOC(6) " xxNN

   To form SSNN in NAME:

```
|1   5|7
=======================================================
      |DECODE (4,1,LOC(6)) TEMP
    1 |FORMAT (2X,A2)
      |ENCODE (4,2,NAME) LOC(1),TEMP
    2 |FORMAT (2A2)
```

   The DECODE statement places NN, the last, two BCD characters of LOC(6) into the two leftmost character positions of TEMP, the temporary storage. The ENCODE statement packs SS, the first two characters of LOC(1) into the left of NAME and packs NN to the right of SS.

By using the R specification, the program can be shortened to:

```
      ENCODE (4,1,NAME) LOC(1),LOC(6)
    1 FORMAT (A2,R2)
```

2) DECODE can be used to calculate a field definition in a format specification during program execution. At some point in the program, m is to be specified in the statement FORMAT (2A8,Im). m is subject to the restriction $2 \leq m \geq 9$.

```
      DIMENSION JSPEC (2)
          :
          :
      IF (M .LT. 10 .AND. M .GT. 1) 1, 2
    1 ENCODE (8,100,JSPEC) M
  100 FORMAT (6H(2A8,I,I1,1H))
          :
          :
      WRITE (10,JSPEC),A,B,J
```

The field definition, M, is tested to insure that it is within limits. If not, control goes to statement 2 which may be an error routine. When M is within limits, ENCODE prepares a format list including m in JSPEC. The 6H format transmits (2A8,I to the left of JSPEC; I1 packs m in the next character position; 1H) closes the list with a right parenthesis. After ENCODE, JSPEC contains (2A8,Im).

A write statement referring to format JSPEC writes A and B under specification A8, and the quantity J under specification Im, where m may vary from 2 to 9.

3) ENCODE can be used to re-arrange and change information in a record.

```
      :
      :
      I = 3HBCD
      IA = 1H1
      ENCODE (8,10,B)I,IA,I
   10 FORMAT (A2,A1,R2)
      WRITE (10,11)B
   11 FORMAT (020)
      :
      :
```

I contains 22232460
IA " 01606060

A2 selects 2223 from I and packs it in the left of B.

A1 selects 01 from IA and packs it to the right of 2223.

R2 selects 2460 from I and packs it to the right of 222301.

Because the format list converts fewer than c characters per record, 3 blanks are placed in the remainder of the record.

After the ENCODE, B contains 2223012460606060 = $BC1D_{\Lambda\Lambda\Lambda\Lambda}$

Printout:

$\Lambda\Lambda\Lambda\Lambda$2223012460606060 (the octal equivalent of $BC1D_{\Lambda\Lambda\Lambda\Lambda}$)

PROGRAM OPERATION 11

The SCOPE-32 loader is called into storage by the executive routine whenever programs are to be loaded. The loader:

Loads relocatable binary information

Links independently compiled subprograms

Loads and links library routines referenced by a loaded program

Selects and loads required I/O drivers from the library tape

Assigns equipment to logical units

A main program may be written with or without references to subprograms. In all cases, the first statement of the main program must be a PROGRAM statement.

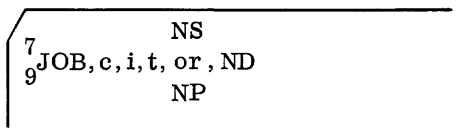
11.1 CONTROL CARDS

Compilation and execution of FORTRAN programs under SCOPE control requires SCOPE control cards, identifiable by a 7,9 punch in column 1. The name of the control statement followed by any necessary parameters begins in column 2. Control card name and parameters must be contained on a single 80-column card.

Control statements may be read in sequence from the standard input unit, or may be serially presented by the operator. The operator enters them through the comment-from-operator unit - usually the console typewriter.

11.1.1 JOB CARD

FORTRAN jobs submitted for processing under SCOPE require a JOB statement that supplies information to the installation accounting routine, identifies the programmer, and sets a job-processing time limit. The JOB card must be immediately preceded by a SEQUENCE statement. JOB-card format is:



c charge number, 0-8 characters

i programmer identification; any number of characters although the accounting routine uses only the first four.

t time limit in minutes for the entire job, including idle time for setup as well as running time. The time is interpreted by the installation accounting routine.

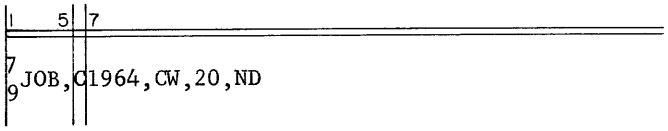
NS indicates a single, non-stacked job. Use of NS implies NP as well. Prior to execution of the job, all system tape units are rewound and unloaded making all I/O units available to the programmer---with no system unit protection.

NP suppresses system I/O protection. To prevent unintentionally writing on system units, do not use NP when executing FORTRAN programs.

ND inhibits post-execution dump. If this field is blank or omitted, a dump will be taken.

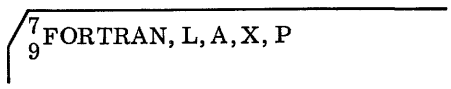
The c, i, and t fields are fixed; they can be blank except for commas. The NS/NP field and the ND field are interchangeable (free field), and may be omitted.

Example:



11.1.2 FORTRAN CONTROL CARD

FORTRAN is a SCOPE library program; to be called, loaded, and executed, it requires a SCOPE library control card.



L list source language on standard output unit

A list assembly language on standard output unit

X write load-and-go on standard load-and-go unit

P punch relocatable binary deck on standard punch unit

Parameters can be interchanged or omitted. A parameter can be one or more characters in length; only the first character - L, A, X, or P - is significant.

Example:

```
1    5 7
7
9FORTRAN,LIST,ASSEMBLY,XECUT,PUNCH
```

The assembly listing and source language listing will be on the standard output unit. The load-and-go and the punch output will be processed on standard units.

11.1.3 LOAD CARD

This statement loads binary object subprograms into storage.

```
7
9LOAD,u1,u2,u3
```

u_i identify the logical unit numbers, 1-56, previously defined by an EQUIP statement (section 11.2) for this run. If logical units are not specified, the loader attempts to load from logical unit 60, standard input (INP). No more than three units may be specified.

An end-of-file terminates loading from each unit. SCOPE loads from the units in the order specified and then loads binary information from INP if any exists.

With or without a LOAD statement, unless preceded by a control statement such as XFER (3200 SCOPE COMPASS Reference Manual), a binary object program on the standard input unit is loaded into memory when encountered by the monitor system.

Example:

```
1    5 7
7
9LOAD,42
```

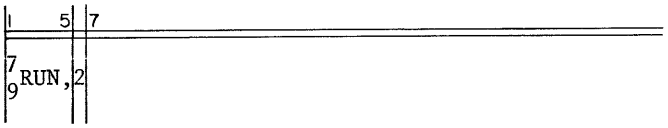
11.1.4
RUN CARD

The RUN statement initiates program execution by transferring control to the object program in memory. It is required for all program runs and follows the relocatable binary deck if the program is on INP or the LOAD statement if the program is on an input tape other than INP.

t — is the execution time limit in minutes (0 to 999) used by the accounting routine. If t is omitted, the accounting routine assumes blanks.

NM — suppresses the memory map that would otherwise be written on the standard output unit (OUT). The map is a list of memory allocations of a loaded program prior to execution. When NM is omitted, a map is prepared.

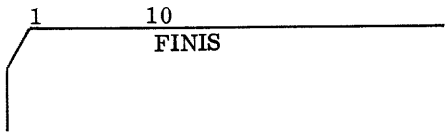
Example:



The estimated execution time is 2 minutes; a memory map will be prepared.

11.1.5
FINIS CARD

The FINIS card notifies the compiler that there are no more decks to be compiled. The word FINIS must begin in column 10:



11.2 EQUIPMENT ASSIGNMENT

SCOPE assigns physical equipment at run time. All references to I/O units are by logical unit numbers. The programmer must declare logical unit numbers on EQUIP cards placed after the JOB card or before the RUN card.

Hardware Definition

```
7/9EQUIP,xx=hh
```

This statement assigns logical unit xx to available equipment of hardware type hh. A diagnostic is provided if specified hardware is unavailable.

hh =	MT	Magnetic tape
	CR	Card Reader
	PR	Printer
	CP	Card Punch
	TY	Console Typewriter
	TS	Channel Typewriter

Equating Units

```
7/9EQUIP,xx1=xx2
```

Once defined, a logical unit may be equated with others. The second statement equates logical units xx_i. Specifying a system unit (56-63) on the left side terminates a job. xx_2 may be a system unit.

Example:

```
1    5 7
7/9EQUIP,40=CR
7/9EQUIP,10=40
```

Physical Unit Assignment

```
7
9EQUIP,xx=hhCcEeUuu
```

xx	logical unit number
hh	hardware type mnemonic
c	channel number 0-7, prefixed by C
e	equipment number (controller) prefixed by E
uu	unit number (device), prefixed by U

The table illustrates the variety of formats permitted for this statement. An x indicates use of a parameter; a blank indicates its omission.

hh	Cc	Ee	Uuu
x			
x	x		
x	x	x	
x	x	x	x
	x	x	x
	x	x	

Specification of a non-existent channel, equipment, or unit number causes job termination accompanied by a diagnostic.

Example:

```
1    5 7
7
9EQUIP,16=MTC0E0U03
```

11.2.2 LOGICAL UNITS

PROGRAMMER UNITS:

Logical units 1 through 49, retained throughout each job for reference by the programmer.

SCRATCH UNITS:

Logical units 50 through 55, held for temporary use during execution of each program. These units are released after execution of each program.

SYSTEM UNITS:

System units (logical units 56-63) are standard units defined by SCOPE. The physical assignments of units differ according to the installation.

unit 56 Load-and-go (LGO)

Binary information may be stored here prior to loading and executing.

unit 57 Accounting Record (ACC)

Job statements and total time used by the jobs are recorded on this unit. Each installation must provide the accounting routine.

unit 58 Input Comment (CFO)

Comments from the operator to the monitor system are made on this unit.

unit 59 Output Comment (CTO)

Statements from the monitor system to the operator are made on this unit. This unit may be used by the programmer for output.

unit 60 Standard Input (INP)

Jobs to be processed by SCOPE are placed on this unit by the operator.

unit 61 Standard Output (OUT)

Accounting information, diagnostics, dumps, and job control statements are recorded on this unit in BCD mode. The programmer normally uses this unit for off-line listable output.

unit 62 Standard Punch Output (PUN)

Output for punching is recorded on this unit. All records are written in binary mode unless otherwise specified.

unit 63 SCOPE Library (LIB)

The SCOPE library contains the monitor system and all of the programs and subprograms which operate under the control of SCOPE; such as, FORTRAN, COMPASS, COBOL.

11.2.3 EQUIPMENT REQUIREMENTS

The FORTRAN jobs require a minimum system consisting of:

1 input unit (card reader)

1 output unit (printer)

3 magnetic tapes

This configuration will compile and execute a job containing single or multiple subprograms. During compilation, the system employs:

LU55 Scratch

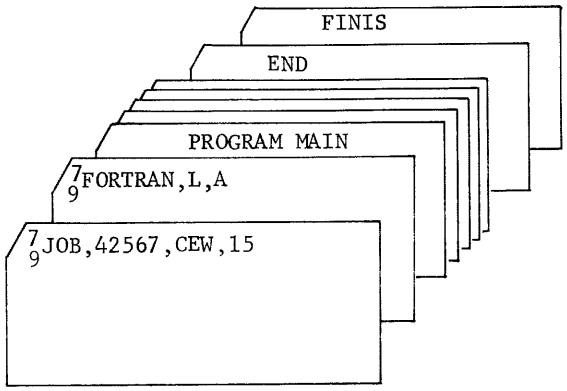
LU56 Load-and-go and scratch

LU63 SCOPE Library

11.3 DECK STRUCTURE

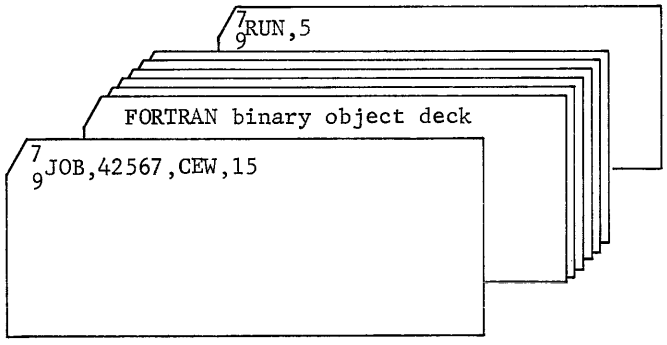
The examples illustrate deck arrangements for compilation and execution of FORTRAN programs and subprograms.

COMPILATION ONLY



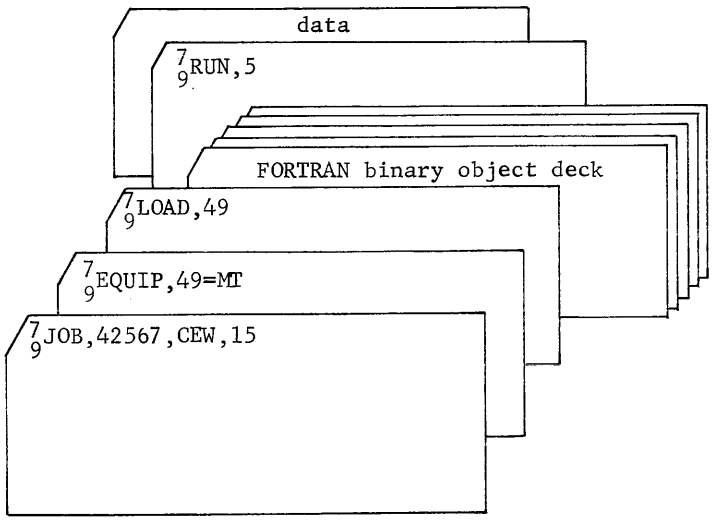
EXECUTION ONLY

Execute directly from the standard input unit:

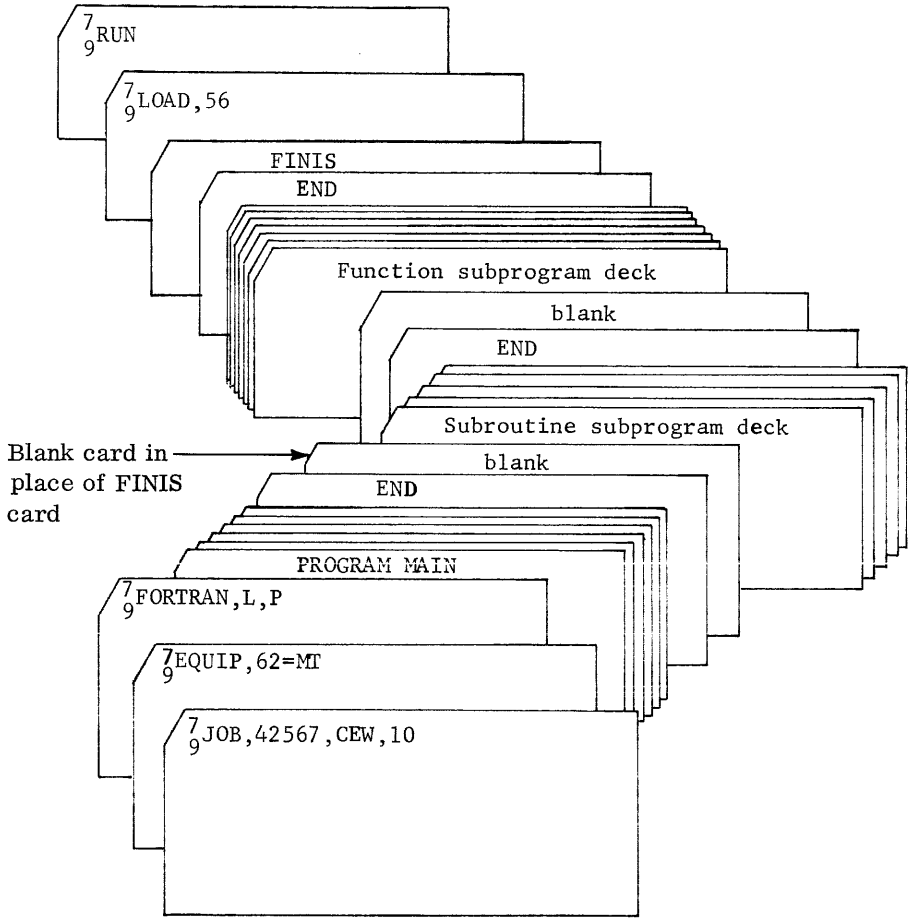


Execute time does not include load time. For this example, execution time limit is 5 minutes; job limit is 15 minutes.

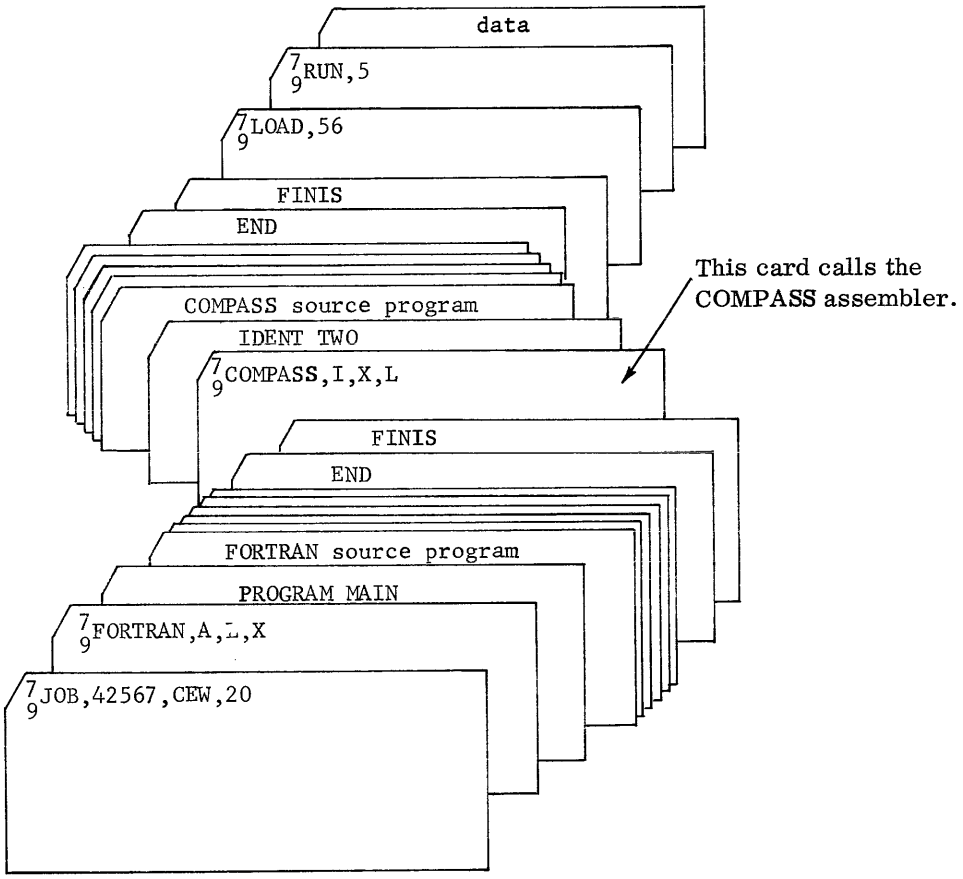
Execute from programmer unit number 49 in addition to the standard input unit:



COMPILATION OF FORTRAN MAIN PROGRAM WITH SUBPROGRAMS



FORTRAN PROGRAM WITH COMPASS SUBPROGRAM



APPENDIX SECTION

CHARACTER CODES

Source Language Character	BCD (Internal only)*	Punch position in a Hollerith Card Column
A	21	12-1
B	22	12-2
C	23	12-3
D	24	12-4
E	25	12-5
F	26	12-6
G	27	12-7
H	30	12-8
I	31	12-9
J	41	11-1
K	42	11-2
L	43	11-3
M	44	11-4
N	45	11-5
O	46	11-6
P	47	11-7
Q	50	11-8
R	51	11-9
S	62	0-2
T	63	0-3
U	64	0-4
V	65	0-5
W	66	0-6
X	67	0-7
Y	70	0-8
Z	71	0-9
0	00	0
1	01	1
2	02	2
3	03	3
4	04	4
5	05	5
6	06	6
7	07	7
8	10	8
9	11	9
/	61	0-1
+	20	12
-	40	11
blank	60	space
.	33	12-8-3
)	34	12-8-4
$	53	11-8-3
*	54	11-8-4
,	73	0-8-3
(	74	0-8-4
=	13	8-3
≠	14	8-4

*Magnetic Tape Codes same as 1604.

Subprogram Statements

Entry Points	PROGRAM name	N*
	SUBROUTINE name	N
	SUBROUTINE name ($p_1, p_2, \ldots, p_n$)	N
	FUNCTION name ($p_1, p_2, \ldots, p_n$)	N
	ENTRY name	N
Inter-subroutine	EXTERNAL $name_1, name_2, \ldots, name_n$	N
Transfer Statements	CALL name ($c_1, c_2, \ldots, c_n$)	E
	RETURN	E

Data Declaration and Storage Allocation

Type Declaration	REAL list	N
	INTEGER list	N
	CHARACTER list	N
	TYPE other (w) list	N
Storage Allocations	DIMENSION $v_1(s_1, s_2, s_3), v_2(s_4, s_5, s_6), \ldots$	N
	COMMON list	N
	COMMON/DATA/list	N
	EQUIVALENCE $(a_1, b_1, \ldots), (a_2, b_2, \ldots), \ldots$	N
	DATA (i_1=list), (i_2=list)...	N
	DATA (i(j,k,1,)=list)	N
	DATA (((i(I, J), I=i_1, i_2), J=m_1, m_2)=list)	N

*N Non-executable; E Executable

Symbol Manipulation, Control and I/O

Replacement	v = e	E
	$r_n=r_{n-1}=\ldots r_2=r_1=a$	E
Intra-program Transfers	GO TO n	E
	GO TO $(n_1,\ldots,n_m),e$	E
	IF (e) n_1,n_2,n_3	E
	IF (ℓ) n_1,n_2	E
Loop Control	DO n i=m_1,m_2,m_3	E
Miscellaneous Program Controls	CONTINUE	E
	PAUSE; PAUSE n	E
	STOP: STOP n	E
I/O Format	FORMAT ($spec_1$, $spec_2\ldots$)	N
I/O Control Statements	READ (i, n) list	E
	WRITE (i, n) list	E
	READ n, list	E
	READ (i) list	E
	READ INPUT TAPE i, n, list	E
	READ TAPE i, list	E
	WRITE OUTPUT TAPE i, n, list	E
	WRITE TAPE i, list	E
	WRITE (i) list	E
	BUFFER IN (i, p) (a, b)	E
	BUFFER OUT (i, p) (a, b)	E
	PRINT n, list	E
	PUNCH n, list	E

I/O Tape Handling	ENDFILE i	E
	REWIND i	E
	BACKSPACE i	E
Internal Data Manipulation	ENCODE (c,n,w) list	E
	DECODE (c,n,w) list	E
Program and Subroutine Termination		
	END	N/E

MIXED MODE

The form of the external call is:

```
RTJ   QIQ   op   t1t2

77    loc              parameter word
```

op two BCD characters representing operation to be performed:

AD	add	MU	multiply
SB	subtract	DV	divide
ST	store	EX	exponentiate

t_1 single BCD character representing type of value in accumulator

t_2 single BCD character representing type of operand value

t_1 and t_2 may be:

I	integer
R	real
X	character

loc address of operand

If the operand is an element of an array, the parameter word indicates the index register that contains the increment value to obtain the proper element. If the array is non-character, any index register may be used. For a character array, the index register is:

B2 for add, subtract, and store operations
B1 for all other operations

The following routines are implemented for mixed mode:

Q1QEXRR	real	**	real
Q1QEXRI	real	**	integer
Q1QEXIR	integer	**	real
Q1QEXII	integer	**	integer
Q1QADIR	integer	+	real
Q1QADXR	character	+	real
Q1QADRI	real	+	integer
Q1QADRX	real	+	character
Q1QSBIR	integer	-	real
Q1QSBXR	character	-	real
Q1QSBRI	real	-	integer
Q1QSBRX	real	-	character

Q1QMUIR	integer	*	real
Q1QMURI	real	*	integer
Q1QDVIR	integer	/	real
Q1QDVRI	real	/	integer
Q1QSTXR	convert character to real and store		
Q1QSTRI	convert real to integer and store		
Q1QSTRX	convert real to character and store		
Q1QSTIR	convert integer to real and store		

TYPE CHARACTER The form of the external call is the same as for mixed mode arithmetic. The operations represented by op are:

MU	multiply
DV	divide
EX	exponentiate

The address of the operand value may be a 17-bit address.

Example:

```
1    5  7
        CHARACTER L, M, N, P
        L = M**N+P
```

The compiler generates a call to an external routine that performs the multiplication. (Operand M is in the A register when the return jump is executed.)

```
RTJ   Q1QMUXX
77    N
```

To multiply character variables, the user must provide a COMPASS routine that has Q1QMUXX as its entry point.

Example:

```
1    5 7
      CHARACTER A, C
      C = A/B
```

The compiler generates a call to an external routine that performs the division. (Operand A is in the A register when the return jump is executed.)

```
RTJ   Q1QDVXR
77    B
```

To divide character variables, the user must supply the external routine.

TYPE OTHER

The form of the external call is:

RTJ nnn op t_1 t_2

77 loc parameter word

nnn — three BCD characters representing the type of operation. The first character is the first letter of the non-standard name (C for complex). The second character is either the second letter of the non-standard name or a period. The third character is always a period.

op — two BCD characters representing operation to be performed.

LD	load	AD	add
LN	load negative	SB	subtract
ST	store	MU	multiply
CM	complement	DV	divide
		EX	exponentiate

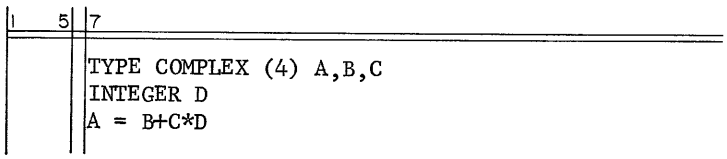
t_1 and t_2 — represent the accumulator and operand values:

R	real
I	integer
X	character
O	other

The parameter word serves the same purpose as it does in mixed mode.

The external call for complement is not followed by a parameter word. The return is to the instruction following the RTJ instruction.

Example:



```
      TYPE COMPLEX (4) A,B,C
      INTEGER D
      A = B+C*D
```

The compiler generates the following calls to external routines, which must be supplied by the user.

```
RTJ     CO. LDOO     routine to load C
77      C
RTJ     CO. MUOI     routine to multiply
77      D
RTJ     CO. ADOO     routine to add B
77      B
RTJ     CO. STOO     routine to store
77      A            result in A
```

STANDARD SUBPROGRAM CALLING SEQUENCE

The standard calling sequence recognized by FORTRAN subroutines and functions is:

```
RTJ     name
77      L(p1)
.       .
.       .
.       .
77      L(pn)
```

The subprogram being referenced is identified by name. $L(p_i)$ is the address of the i^{th} parameter. If name is a FORTRAN subprogram, it must appear in a SUBROUTINE or FUNCTION statement as

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

A FORTRAN program may reference a COMPASS subprogram by a function reference or subroutine call which produces the above calling sequence.

Example:

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

compiles to:

RTJ	NAME
77	$L(p_1)$
.	.
.	.
.	.
77	$L(p_n)$

FORTRAN produces code that saves and restores contents of index registers for subroutines and functions.

DIAGNOSTICS D

COMPILATION

At the completion of compilation of each subprogram, any errors encountered during compilation are indicated on the standard output unit. A diagnostic in one of the following forms is printed with the error message.

ERROR TYPE xxxx DETECTED AT n STATEMENTS BEYOND NO. m

ERROR TYPE xxxx DETECTED AT STATEMENT m

xxxx	4-digit error code
m	a source statement label, or 0 - the first statement of the subprogram

Example:

ERROR TYPE 0156 DETECTED AT 13 STATEMENTS BEYOND STATEMENT NO. 0
COMMON NAME DOES NOT START WITH AN ALPHABETIC CHARACTER

Error messages are informative (I), destructive (D), or fatal (F).

A type I error produces an informative message, only. The erroneous statement is compiled and the program is executed.

A destructive error prevents the statement from being compiled. Compilation resumes with the next statement but execution is inhibited.

Detection of a fatal error in a statement causes immediate termination of compilation of the program or subprogram. Compilation resumes with the next subprogram but the compiled programs are not executed.

Error Code	Message	Type
0000	PROGRAM TOO LARGE FOR COMPILER TABLES. SEGMENT AND RE-COMPILE	F
0001	FIRST STATEMENT NOT PROGRAM, SUBROUTINE, OR FUNCTION--ASSUME PROGRAM JOB	I
0002	PROGRAM FORMAT ERROR--ASSUME PROGRAM JOB	I
0003	PROGRAM NAME DOES NOT START WITH AN ALPHABETIC CHARACTER--ASSUME PROGRAM JOB	I
0004	PROGRAM STATEMENT OUT OF ORDER	I
0005	LABELED DECLARATIVE STATEMENT	I
0010	FUNCTION OR SUBROUTINE FORMAT ERROR	D
0011	NO FORMAL PARAMETERS IN FUNCTION STATEMENT	D
0012	FUNCTION NAME DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0013	FORMAL PARAMETER DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0014	SUBROUTINE STATEMENT OUT OF ORDER	D
0015	MORE THAN 63 FORMAL PARAMETERS	D
0020	SUBROUTINE FORMAT ERROR	D
0021	SUBROUTINE NAME DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0022	FUNCTION STATEMENT OUT OF ORDER	D
0030	DIMENSION FORMAT ERROR	F
0031	ILLEGAL USE OF NAME IN DIMENSION STATEMENT	F
0032	DIMENSION NAME DOES NOT START WITH AN ALPHABETIC CHARACTER	F
0033	SUBSCRIPT IS NOT AN INTEGER CONSTANT LESS THAN 32768	F
0034	FORMAT ERROR IN DIMENSIONING	F
0035	MORE THAN THREE SUBSCRIPTS	F

0130	EXTERNAL FORMAT ERROR	D
0131	EXTERNAL NAME DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0132	ILLEGAL USE OF NAME IN EXTERNAL	D
0150	LABELED COMMON FORMAT ERROR	D
0151	COMMON FORMAT ERROR	D
0152	ILLEGAL USE OF NAME IN COMMON STATEMENT	D
0153	MORE THAN THREE SUBSCRIPTS IN COMMON STATEMENT	D
0154	IDENTIFIER APPEARS MORE THAN ONCE IN COMMON	D
0156	COMMON NAME DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0157	TWO ELEMENTS IN A SET ARE IN COMMON	D
0160	BLOCK NAME OTHER THAN DATA	D
0161	MORE THAN 253 IDENTIFIERS IN LABELED COMMON OR IN BLANK COMMON	D
0240	IDENTIFIER IN TYPE STATEMENT IS PROGRAM OR SUBROUTINE NAME	D
0241	TYPE FORMAT ERROR	D
0242	MORE THAN ONE TYPE OTHER	D
0243	TYPE OTHER FORMAT ERROR	D
0244	TYPE OTHER SIZE IS NOT AN INTEGER CONSTANT LESS THAN 32768	D
0245	CONFLICT IN SIZE FOR TYPE OTHER STATEMENTS	D
0246	ZERO WORD LENGTH FOR TYPE OTHER	D
0247	TYPE OTHER NAME DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0250	NAME TO BE TYPED DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0300	EQUIVALENCE FORMAT ERROR	D
0301	SUBSCRIPT FOR TYPE OTHER OR CHARACTER	D

0302	ILLEGAL USE OF NAME IN EQUIVALENCE	D
0304	EQUIVALENCE RELATION ERROR	D
0306	NAME IN EQUIVALENCE DOES NOT START WITH AN ALPHABETIC CHARACTER	D
0400	DATA FORMAT ERROR	D
0401	SUBSCRIPT NOT INTEGER VARIABLE OR INTEGER CONSTANT LESS THAN 32768	D
0402	TOO MANY SUBSCRIPTS FOR DIMENSIONED VARIABLE	D
0403	FIRST USE OF IDENTIFIER	D
0404	NAME NOT SIMPLE OF DIMENSIONED VARIABLE AND IN LABELED COMMON	D
0405	MORE THAN ONE CONSTANT IN DATA LIST FOR SIMPLE VARIABLE	D
0406	SUBSCRIPT NOT INTEGER CONSTANT LESS THAN 32768	D
0407	SUBSCRIPT EQUALS ZERO	D
0411	DATA LIST TOO LONG	D
0412	CONFLICT IN USE OF NAMES OR VARIABLES IN DO IMPLYING LOOP	D
0413	N2 LESS THAN N1 -- I = N1, N2	D
0414	ILLEGAL USE OF NAME IN DATA STATEMENT	D
0415	FORMAT ERROR IN DATA LIST	D
0416	ATTEMPTED NEST OR REPEATS IN DATA LIST	D
0420	K = 0 USED FOR REPEAT	D
0421	DATA LIST TOO SHORT	I
0422	--APPEARS IN DATA LIST	D
0423	-(APPEARS IN DATA LIST	D
0461	MORE THAN 64 DECLARATION STATEMENTS	F
0477	NO LABELED COMMON FOR DATA	D
0501	PARENTHESES DO NOT MATCH.	D

0502	LABEL ON CONTINUATION CARD OR COLUMN SIX OF LABELED STATEMENT MISPUNCHED	D
0503	STATEMENT LABEL NOT IN SPECIFIED FORMAT. ALPHABETIC CHARACTER, NON-INTEGER NUMBER, OR THE LABEL IS GREATER THAN 32767	D
0504	LABEL ON THIS STATEMENT HAS BEEN USED PREVIOUSLY.	D
0505	IDENTIFIER OF MORE THAN EIGHT CHARACTERS.	D
0506	IDENTIFIER NAME FOLLOWED BY PERIOD. POSSIBLE ERROR IN LOGICAL OPERATOR OR OMISSION OF SEPARATOR OR ARITHMETIC OPERATOR BEFORE A CONSTANT.	D
0507	STATEMENT TOO LONG FOR COMPILER TABLES.	D
0510	MORE THAN 20 CHARACTERS HAVE BEEN USED IN A NUMERIC CONSTANT FIELD.	D
0511	THE FIRST CONSTANT IN THIS STATEMENT IS INCORRECT	D
0512	I/O STATEMENT IS NOT CORRECT IN FORM.	D
0513	R OR H FIELD OF HOLLERITH CONSTANT NOT SPECIFIED CORRECTLY OR FIELD TOO LONG.	D
0514	CANNOT IDENTIFY STATEMENT TYPE. STATEMENT NAME MIS-SPELLED OR MISUSED OR AN EQUAL SIGN IS MISPLACED.	D
0515	SYNTAX ERROR IN FORMAT SPECIFICATION.	D
0516	CONSTANT IN STOP OR PAUSE STATEMENT SHOULD BE 5 OR LESS DIGITS.	I
0520	COMPUTED GOTO EXPRESSION CONTAINS A LOGICAL OPERATOR	D
0521	MULTIPLE REPLACEMENT STATEMENT CONTAINS LOGICAL OPERATOR	D
0522	FORMAT STATEMENT NOT LABELED	D
1000	COMPUTED GO TO LABEL LIST NOT IN CORRECT FORMAT.	D
1001	LABEL REFERENCED IN NOT AN INTEGER NUMBER.	D
1002	IF STATEMENT WITHOUT RIGHT PARENTHESIS AFTER EXPRESSION.	D

1003	INCORRECT NUMBER OF LABELS IN IF STATEMENT.	D
1004	LABEL LIST IN INCORRECT FORMAT IN IF STATEMENT.	D
1005	LABEL REFERENCE NOT INTEGER NUMBER OR NOT LESS THAN OR EQUAL TO 32767.	D
1006	THIS STATEMENT IS TOO LONG TO FIT IN PI LIST.	D
1007	LEFT SIDE OF REPLACEMENT STATEMENT NOT IN CORRECT FORMAT.	D
1010	AN ILLEGAL FORM OF SUBSCRIPTING HAS OCCURRED IN THIS STATEMENT.	D
1011	SUBSCRIPT IS NOT A SIMPLE INTEGER VARIABLE.	D
1012	SUBSCRIPT MODIFIER IS NOT AN INTEGER CONSTANT OR IS GREATER THAN 32767.	D
1013	DIMENSIONED VARIABLE WITH TOO MANY SUBSCRIPTS APPEARS IN THIS STATEMENT.	D
1014	AN ILLEGAL FORM OF SUBSCRIPTING HAS OCCURRED IN THIS STATEMENT.	D
1015	CONSTANT ADDEND OF A DIMENSIONED VARIABLE IS GREATER THAN 32767.	D
1016	MORE THAN 63 INDEX FUNCTION GENERATED.	D
1017	POSSIBLE MACHINE OR COMPILER ERROR. I-BIT SET IN FORMAT PARAMETER IDLIST ENTRY BUT NO DIMENSIONED VARIABLE IN IDLIST.	D
1020	PROGRAM OR SUBROUTINE NAME MAY NOT APPEAR	D
1022	INDEX FUNCTION MULTIPLIER GREATER THAN 32767.	D
1024	SUBROUTINE NAME IN CALL STATEMENT USED BEFORE AS A SIMPLE OR DIMENSIONED VARIABLE.	D
1025	INCORRECT FORM OF PARAMETER LIST IN A CALL STATEMENT.	D
1026	ENTRY STATEMENT USED BEFORE AS A SIMPLE OR DIMENSIONED VARIABLE.	D
1027	ENTRY STATEMENT CANNOT APPEAR IN A MAIN PROGRAM.	D

1030	DO STATEMENTS NESTED MORE THAN 10 DEEP.	D
1031	RUNNING INDEX OF A DO NOT A SIMPLE INTEGER VARIABLE.	D
1032	EQUAL SIGN DOES NOT FOLLOW THE RUNNING INDEX OF A DO.	D
1033	FIRST QUANTIFIER IN A DO NOT FOLLOWED BY A COMMA.	D
1034	DO STATEMENT DOES NOT END AFTER THIRD QUANTIFIER.	D
1035	QUANTIFIER IN DO STATEMENT IS NOT AN INTEGER CONSTANT OR SIMPLE INTEGER VARIABLE.	D
1036	SECOND QUANTIFIER IN A DO STATEMENT NOT FOLLOWED BY A COMMA OR END OF STATEMENT.	D
1037	DO STATEMENT WHICH TERMINATES AT THIS STATEMENT NUMBER IS OUT OF ORDER.	D
1040	ENTRY NAME USED ELSEWHERE IN THE PROGRAM.	D
1041	AN ENTRY STATEMENT CANNOT BE LABELED.	D
1042	ENTRY NAME USED ELSEWHERE IN PROGRAM.	D
1043	AN ENTRY STATEMENT CANNOT APPEAR WITHIN A DO LOOP.	D
1044	REFERENCE TO A STATEMENT LABEL OF ZERO IS ILLEGAL.	D
1046	THE RUNNING INDEX IN A DO MAY BE CHANGED WITHIN THE LOOP.	I
1047	THE UPPER LIMIT VARIABLE (M2) OF THE DO MAY BE CHANGED WITHIN THE LOOP.	I
1050	DO INCREMENT VARIABLE (M3) MAY BE CHANGED WITHIN THE LOOP.	I
1052	DO QUANTIFIER GREATER THAN 32767	D
1053	DO INCREMENT (M3) IS ZERO.	I
1054	ALL DECLARATIVE STATEMENTS MUST APPEAR BEFORE THE FIRST EXECUTIVE STATEMENT.	D
1055	DO STATEMENTS MAY NOT TERMINATE ON A TRANSFER STATEMENT.	D

1056	FUNCTION SUBPROGRAM NAME NOT DEFINED.	I
1057	THE DO STATEMENT THAT TERMINATES ON THIS LABEL IS NOT TERMINATED BEFORE END CARD.	D
1060	LOGICAL OPERATORS MAY NOT APPEAR IN ACTUAL PARAMETERS.	D
1500	INTEGER NUMBER GREATER THAN 2**23-1.	D
1510	NO DECIMAL DIGITS PRECEDING OR FOLLOWING THE DECIMAL POINT.	D
1520	ILLEGAL OCTAL CHARACTER	D
1530	MORE THAN ONE DECIMAL POINT	D
1540	THE REAL NUMBER HAS MORE THAN ELEVEN DIGITS OR HAS EXCEEDED 2**36-1 IRRESPECTIVE OF THE DECIMAL POINT.	D
1550	ILLEGAL CHARACTER IN EXPONENT OR EXPONENT TOO LARGE.	D
1560	ILLEGAL DECIMAL CHARACTER	D
1570	MORE THAN 8 OCTAL CHARACTERS IN STRING	D
2001	NO REPLACEMENT VARIABLE IN LOGICAL REPLACEMENT STATEMENT.	D
2002	NO REPLACEMENT OPERATOR IN LOGICAL REPLACEMENT STATEMENT.	D
2003	LEFT PAREN MUST ENCLOSE LOGICAL IF	D
2004	RIGHT PAREN MUST CLOSE LOGICAL IF	D
2005	LOGICAL OR RELATIONAL OPERATOR OCCURRED WITHIN PARENTHESES	D
2006	AN EXPRESSION MUST PRECEDE .AND., .OR., OR END OF STATEMENT	D
2007	TWO CONSECUTIVE RELATIONAL OPERATORS	D
2010	A LOGICAL OPERATOR MUST APPEAR BETWEEN TWO NOTS	D
2011	AN EXPRESSION MAY NOT PRECEDE .NOT.	D
2012	AN EXPRESSION MUST FOLLOW A RELATIONAL OPERATOR	D

2013	STATEMENT LABEL USED IN LOGICAL IF IS UNDEFINED	D
2014	MORE THAN 4096 LA BELS HAVE BEEN GENERATED IN A LOGICAL EXPRESSION	D
2500	WRONG FORMAT OF I/O STATEMENT, I/O LIST NOT YET PROCESSED.	D
2501	PARITY DESIGNATOR IN I/O STATEMENT MUST BE A SIMPLE INTEGER VARIABLE OR NUMBER EQUAL TO 0 OR 1	D
2502	FORMAT ERROR OF DO-IMPLYING INDEX.	D
2503	THE PARAMETERS OF A DO-IMPLYING LOOP MUST BE UNSIGNED INTEGER CONSTANTS OR SIMPLE INTEGER VARIABLES.	D
2504	THE CONSTANT PARAMETERS OF A DO-IMPLYING LOOP MUST BE LESS THAN 32768.	D
2505	UNIT NUMBER MUST BE A SIMPLE INTEGER VARIABLE OR INTEGER CONSTANT LESS THAN 64.	D
2506	MORE THAN 145 LEFT PARENTHESES HAVE BEEN ENCOUNTERED IN AN I/O DATA LIST. REMOVE REDUNDANT PARENTHESES AND RESUBMIT.	D
2507	TYPE OTHER VARIABLES MAY NOT APPEAR IN AN I/O DATA LIST.	D
2510	A VARIABLE MUST PRECEDE THE DO-IMPLYING INDEX IN AN I/O DATA LIST.	D
2511	THE RUNNING SUBSCRIPT IN A DO-IMPLYING LOOP MUST BE A SIMPLE INTEGER VARIABLE.	D
2512	WRONG FORMAT OF I/O DATA LIST OR ILLEGAL ENTRY IN I/O DATA LIST.	D
2513	ILLEGAL MODE OF I/O PARAMETER.	D
2700	THE FORMAT STATEMENT REFERENCED DOES NOT APPEAR IN THE SOURCE PROGRAM.	D
3000	FUNCTION NAME USED AS REPLACEMENT VARIABLE.	D
3001	COMMA APPEARS ON PARENTHESES LEVEL ZERO	D
3002	ILLEGAL USE OF ACTUAL PARAMETER	D
3021	OPERATOR MISSING IN ARITHMETIC EXPRESSION	D

3026	/ - APPEARS IN ARITHMETIC EXPRESSION	D
3060	FUNCTION TERMINATES ARITHMETIC EXPRESSION	D
3061	ILLEGAL USE OF FUNCTION IN ARITHMETIC EXPRESSION	D
3065	FUNCTION + APPEARS IN ARITHMETIC EXPRESSION	D
3066	FUNCTION - APPEARS IN ARITHMETIC EXPRESSION	D
3067	FUNCTION * APPEARS IN ARITHMETIC EXPRESSION	D
3070	FUNCTION / APPEARS IN ARITHMETIC EXPRESSION	D
3071	FUNCTION ** APPEARS IN ARITHMETIC EXPRESSION	D
3072	FUNCTION = APPEARS IN ARITHMETIC EXPRESSION	D
3112	CONSTANT = APPEARS IN ARITHMETIC EXPRESSION	D
3114	NUMBER (APPEARS IN ARITHMETIC EXPRESSION	D
3120	+ TERMINATES ARITHMETIC EXPRESSION	D
3125	+ + APPEARS IN ARITHMETIC EXPRESSION	D
3126	+ - APPEARS IN ARITHMETIC EXPRESSION	D
3127	+ * APPEARS IN ARITHMETIC EXPRESSION	D
3130	+ / APPEARS IN ARITHMETIC EXPRESSION	D
3131	+ ** APPEARS IN ARITHMETIC EXPRESSION	D
3133	+ , APPEARS IN ARITHMETIC EXPRESSION	D
3135	+) APPEARS IN ARITHMETIC EXPRESSION	D
3140	- TERMINATES ARITHMETIC EXPRESSION	D
3145	- + APPEARS IN ARITHMETIC EXPRESSION	D
3146	- - APPEARS IN ARITHMETIC EXPRESSION	D
3147	= * APPEARS IN ARITHMETIC EXPRESSION	D
3150	- / APPEARS IN ARITHMETIC EXPRESSION	D
3151	- ** APPEARS IN ARITHMETIC EXPRESSION	D
3153	- , APPEARS IN ARITHMETIC EXPRESSION	D
3155	-) APPEARS IN ARITHMETIC EXPRESSION	D

```
3160        * TERMINATES ARITHMETIC EXPRESSION                          D
3165        * + APPEARS IN ARITHMETIC EXPRESSION                        D
3166        * - APPEARS IN ARITHMETIC EXPRESSION                        D
3170        * / APPEARS IN ARITHMETIC EXPRESSION                        D
3173        * , APPEARS IN ARITHMETIC EXPRESSION                        D
3175        * ) APPEARS IN ARITHMETIC EXPRESSION                        D
3205        / + APPEARS IN ARITHMETIC EXPRESSION                        D
3207        / - APPEARS IN ARITHMETIC EXPRESSION                        D
3210        / / APPEARS IN ARITHMETIC EXPRESSION                        D
3211        / ** APPEARS IN ARITHMETIC EXPRESSION                       D
3213        / , APPEARS IN ARITHMETIC EXPRESSION                        D
3215        / ) APPEARS IN ARITHMETIC EXPRESSION                        D
3225        ** + APPEARS IN ARITHMETIC EXPRESSION                       D
3226        ** - APPEARS IN ARITHMETIC EXPRESSION                       D
3227        ** * APPEARS IN ARITHMETIC EXPRESSION                       D
3230        ** / APPEARS IN ARITHMETIC EXPRESSION                       D
3231        ** ** APPEARS IN ARITHMETIC EXPRESSION                      D
3233        ** , APPEARS IN ARITHMETIC EXPRESSION                       D
3235        ** ) APPEARS IN ARITHMETIC EXPRESSION                       D
3240        = TERMINATES ARITHMETIC EXPRESSION                          D
3247        = * APPEARS IN ARITHMETIC EXPRESSION                        D
3250        = / APPEARS IN ARITHMETIC EXPRESSION                        D
3251        = ** APPEARS IN ARITHMETIC EXPRESSION                       D
3252        = = APPEARS IN ARITHMETIC EXPRESSION                        D
3253        = , APPEARS IN ARITHMETIC EXPRESSION                        D
3255        = ) APPEARS IN ARITHMETIC EXPRESSION                        D
3260        , TERMINATES ARITHMETIC EXPRESSION                          D
```

3267	, * APPEARS IN ARITHMETIC EXPRESSION	D
3270	, / APPEARS IN ARITHMETIC EXPRESSION	D
3271	, ** APPEARS IN ARITHMETIC EXPRESSION	D
3272	, = APPEARS IN ARITHMETIC EXPRESSION	D
3273	, , APPEARS IN ARITHMETIC EXPRESSION	D
3275	,) APPEARS IN ARITHMETIC EXPRESSION	D
3300	(TERMINATES ARITHMETIC EXPRESSION	D
3307	(* APPEARS IN ARITHMETIC EXPRESSION	D
3310	(/ APPEARS IN ARITHMETIC EXPRESSION	D
3311	(** APPEARS IN ARITHMETIC EXPRESSION	D
3312	(= APPEARS IN ARITHMETIC EXPRESSION	D
3313	(, APPEARS IN ARITHMETIC EXPRESSION	D
3315	( ) APPEARS IN ARITHMETIC EXPRESSION	D
3334	) (APPEARS IN ARITHMETIC EXPRESSION	D
3500	STATEMENT NOT LABELED AND IS PRECEDED BY A NON-RETURN TRANSFER. STATEMENT CAN NOT BE EXECUTED.	I
3501	INCORRECT STATEMENT FORMAT	D
3502	N IS NOT AN INTEGER NUMBER IN A STOP OR PAUSE	D
3503	ENTRY STATEMENT CAN NOT BE LABELED	D
3504	STATEMENT LABEL REFERENCED IN UNDEFINED	D
3505	NO LEFT PAREN IN ARITHMETIC IF	D
3506	CALL STATEMENT MUST TERMINATE WITH A RIGHT PAREN	D
3507	NO EXPRESSION IN ARITHMETIC IF	D
3600	POSSIBLE MACHINE ERROR. RECOMPILE JOB.	F
5001	FORTRAN I/O ERROR REWIND REJECT LUN 55	F
5002	FORTRAN I/O ERROR READ REJECT LUN 60	F

5004	FORTRAN I/O ERROR WRITE REJECT LUN 55	F
5005	FORTRAN I/O ERROR WRITE REJECT LUN 56	F
5006	FORTRAN I/O ERROR BINARY CARD LUN 60	F
5007	FORTRAN I/O ERROR END OF FILE LUN 60	F
5010	FORTRAN I/O ERROR END OF TAPE LUN 55	F
5011	FORTRAN I/O ERROR WRITE ERROR LUN 55	F
5012	FORTRAN I/O ERROR END OF TAPE LUN 56	F
5013	FORTRAN I/O ERROR WRITE ERROR LUN 56	F
5014	FORTRAN I/O ERROR READ ERROR LUN 60	D
5015	FORTRAN I/O ERROR READ ERROR LUN 55	F
5016	FORTRAN I/O ERROR READ ERROR LUN 56	F
5017	POSSIBLE MACHINE ERROR--ENDPI	F
5030	LABELED COMMON REQUIRES MORE THAN 32768 WORDS OF STORAGE	F
5031	NUMBERED COMMON REQUIRES MORE THAN 32768 WORDS OF STORAGE	F
5032	PROGRAM LENGTH EXCEEDS 32768. SEGMENT AND RECOMPILE	F

I/O DIAGNOSTICS

The standard format used in printing the majority of diagnostics from the object time I/O routines is the following:

ERROR IN rrrrrrrr CALLED FROM xxxxx z ... z_n

rrrrrrrr	name of the routine in which the error occurred
xxxxx	address from which the routine was last called
$z_1 \ldots z_n$	error message

Message	Description
PARITY ERROR ON LU xx	A persistent parity error occurred on logical unit xx; execution continues. This condition can be sensed by using IOCHK or IOCHKF.
END OF TAPE ON LU xx	An end of tape condition was sensed on logical unit xx. The unit is unloaded and the next READ or WRITE tape is mounted. Execution continues.
LOST DATA ON LU xx	A lost data condition was encountered on logical unit xx; the job will be terminated abnormally.
ILLEGAL I/O ON LU xx	An illegal I/O operation was requested on logical unit xx. a) Logical unit is not in the defined range for the particular I/O operation. b) SCOPE rejected the I/O request, such as reading from the printer or writing on the card reader, or attempting to write a tape which does not have a write enable ring.
UNCHECKED EOF ON LU xx	On the last read operation, an end of file was encountered on logical unit. xx and no check has been made.
STANDARD REF TO BUFFER TAPE xx	A standard data transmission operation has been requested on logical unit xx where the previous operation was a buffer.
RECORD OVERFLOW LU xx	A BCD read or write request calls for input or output of data outside of the defined record area. For BCD read or write, the record limit is 136 characters; for ENCODE/DECODE, the record limit is c characters.
LIMITS ON BUFFER I/O INCONSISTENT	The first word and last word addresses of the BUFFER IN/BUFFER OUT statement do not comply with the rule that first word address must be less than or equal to last word address.
LIST EXCEEDS DATA	The list of a binary read request called for more data than was in the logical record.
SYNC ERROR	During a binary tape read, an end of file was encountered in the middle of a logical record. Since this cannot be generated by the binary write routine, an error has occurred.

Message	Description
ILLEGAL CODE ON INPUT	An illegal character was encountered in the input field; for example, an 8 or 9 in an octal field or an alphanumeric character in a numeric field.
INTEGER INPUT GE 2**23	An Iw input field contained a number exceeding the range of integer constants (greater than $2^{23}-1$).
EXPONENT OVERFLOW	A calculation during conversion overflowed the limits of the floating point format.
TOO MANY DECIMAL PTS	More than one decimal point encountered in an Ew.d or Fw.d input field.

FORMAT STATEMENT DIAGNOSTICS

Errors encountered during interpretation of format lists result in the following diagnostic:

FORMAT ERROR x - zzzzz

x numeric error code
zzzzz first word address of the format statement in which the error occurred.

x	Description
1	Format list does not begin with a left parenthesis.
2	Illegal repeat factor was encountered. a) repeat factor equals zero b) repeat factor is not an integer
3	Unrecognizable format conversion; the format conversion is designated by a symbol other than E, F, I, A, R, O, H, X.
4	Illegal field width or missing field width. a) field width equals zero b) no field width present c) illegal character in field width specification
5	A number precedes a slash, comma, or right parenthesis.
6	Parenthesis error a) repeat groups may not be nested b) a parenthetical grouping may not appear within a repeat group.
7	Improper format of numeric field within format statement. a) more than one decimal point appears in the numeric field b) numeric value exceeds $2^{15}-1$

OVERLAY – SEGMENT DIAGNOSTICS

An error detected while calling or loading a segment or overlay causes the job to terminate abnormally. A message is written in the form:

xx ERROR IN OVERLAY-SEGMENT

Error Code xx	Meaning
EX	Exit made from main program.
MR	Read error; or overlay or segment cannot be found.
MT	Logical unit (parameter i) is not a magnetic tape unit.
OI	Parameter o in CALL OVERLAY is not 1 through 99.
OM	Overlay which called segment is not in core.
OO	Overlay called by overlay or segment.
OS	Segment called out of order.
RD	Read error from tape unit.
SI	Parameter s in CALL SEGMENT is not 1 through 99.
SZ	Parameter s in CALL OVERLAY is not zero.

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET

3100/3200/3300/3500 COMPUTER SYSTEMS
FORTRAN Reference Manual

Pub. No. 60057600, Rev. B August, 1966

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.

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
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

FOLD FOLD

STAPLE STAPLE

► ►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN, 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60057600 Rev. B.

Litho in U.S.A.