Systems Engineering Laboratories

SEL 810A/840A FORTRAN IV
Reference Manual

Systems Engineering Laboratories

Reference Manual

SEL 810A/840A FORTRAN IV

This publication supersedes the SEL 810A and 840A FORTRAN IV Reference Manual Dated December 15, 1966.

SEL 95054A
Price: $5.00

6901 West Sunrise Blvd., Fort Lauderdale, Florida 33310
Area Code 305 587-2900

TABLE OF CONTENTS

Section			Page
1	INTRODUCTION		1-1
2	ELEMENTS OF 810A/840A FORTRAN		2-1
	2.1	Quantities	2-1
	2.2	Constants	2-1
	2.3	Variables	2-3
	2.4	Statements	2-6
	2.5	Expressions	2-6
3	SPECIFICATION STATEMENTS		3-1
	3.1	Type	3-1
	3.2	Dimension	3-2
	3.3	Common	3-3
	3.4	Equivalence	3-4
	3.5	Data	3-6
4	EXPRESSIONS		4-1
	4.1	Arithmetic Expressions	4-1
	4.2	Logical Expressions	4-3
5	CONTROL STATEMENTS		5-1
	5.1	Unconditional Go To Statement	5-1
	5.2	Computed Go To Statement	5-1
	5.3	Assign Statement	5-2
	5.4	Assigned Go To Statement	5-2
	5.5	Arithmetic If Statement	5-3
	5.6	Logical If Statement	5-3
	5.7	Do Statement	5-4
	5.8	Continue Statement	5-5
	5.9	Pause Statement	5-5
	5.10	End Statement	5-5
	5.11	Stop Statement	5-6
	5.12	Return Statement	5-6
	5.13	Sense Light Subroutine	5-6
	5.14	Sense Light Test Subroutine	5-7
	5.15	Sense Switch Test Subroutine	5-7
	5.16	Accumulator Overflow Test Subroutine	5-7

TABLE OF CONTENTS (Continued)

Section			Page
6	SUBROUTINES, FUNCTIONS AND SUBPROGRAM STATEMENTS		6-1
	6.1	Naming Subroutines	6-1
	6.2	Arithmetic Statement Functions	6-1
	6.3	Library Functions	6-3
	6.4	Function Subprogram	6-3
	6.5	Subroutine Subprogram	6-6
	6.6	The Call Statement	6-6
	6.7	Block Data Subprogram	6-7
7	INPUT/OUTPUT		7-1
	7.1	Input/Output Lists	7-1
	7.1.1	Short-List Notation	7-1
	7.2	General Input/Output Statements	7-2
	7.3	Format	7-2
	7.4	Conversion Specifications	7-3
	7.4.1	I (Integer) Conversion	7-4
	7.4.2	F (Fixed - Point Decimal) Conversion	7-5
	7.4.3	E (Explicit Exponent) Conversion	7-6
	7.4.4	G (Generalized E or F) Conversion	7-7
	7.4.5	L (Logical) Conversion	7-7
	7.4.6	A (Alphanumeric) Conversion	7-7
	7.4.7	D (Double-Precision) Conversion	7-8
	7.4.8	Complex Conversion	7-8
	7.5	Editing Specifications	7-9
	7.5.1	H (Hollerith) Conversion	7-9
	7.5.2	X (Blank) Conversion	7-10
	7.5.3	New Record	7-10
	7.6	nP Scale Factor	7-10
	7.7	Repeated Format Specifications	7-11
	7.7.1	Multiple-Record Formats	7-12
	7.8	Variable Format	7-12
	7.9	Carriage Control	7-13
	7.10	Manipulative Statements	7-14
	7.10.1	End File Statement	7-14
	7.10.2	Rewind Statement	7-14
	7.10.3	Backspace Statement	7-14

TABLE OF CONTENTS (Continued)

APPENDIX A	SEL 810A and 840A Character Codes and Word Structure
APPENDIX B	Statement Index
APPENDIX C	Library Functions
APPENDIX D	Trace
APPENDIX E	Chaining
APPENDIX F	Operator Communications
APPENDIX G	In-Line Coding

LIST OF FIGURES

Figure		Page
2-1	Two-Dimensional Array, Storage Sequence of Elements	2-4
2-2	Three-Dimensional Array, Storage Sequence of Elements	2-5

LIST OF TABLES

Table		Page
4-1	Arithmetic Operators (except exponentiation)	4-1
4-2	Exponentiation Combinations	4-2
6-1	Library Functions	6-2
6-2	Mathematical Subroutines	6-5

SECTION 1

INTRODUCTION

This manual describes the combined features of 810A FORTRAN and 840A FORTRAN. The FORTRAN compiler on both these machines meets the ASA FORTRAN IV specifications.

This manual is written as a reference manual for programmers using the 810A/840A FORTRAN compiler. The manual assumes a basic knowledge of the FORTRAN language; it is not written as a text for beginning FORTRAN programmers.

SECTION 2

ELEMENTS OF 810A/840A FORTRAN

The elements of 810A/840A FORTRAN discussed in this section are Quantities, Constants, Variables, Statements and Expressions. Word formats may be found in Appendix A.

2.1 QUANTITIES

810A/840A FORTRAN manipulates floating point and integer quantities. Logical and alphanumeric words are treated as integer. Floating point quantities have an exponent and a fractional part. The following classes of numbers are floating point quantities:

REAL	810A	Exponent and sign 9 bits; fraction and sign 22 bits; range of number: 10^{77} with 6 significant digits.
	840A	Exponent and sign 9 bits; fraction and sign 38 bits; range of number: 10^{77} with 11 significant digits.
DOUBLE PRECISION	810A	Exponent and sign 9 bits; fraction and sign 37 bits; 11 significant digits.
	840A	Exponent and sign 9 bits; fraction and sign 61 bits; 18 significant digits.
COMPLEX	Two reals as defined above.	

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

INTEGER	810A	Represented by 16 bits; first bit is the sign; magnitude of constant must be less than 32768; 5 digits.
	840A	Represented by 24 bits; first bit is the sign; magnitude of constant must be less than 8388608; 7 decimal digits.
LOGICAL	Requires full word of storage. 1 bit represents TRUE; 0 bit represents FALSE.	
HOLLERITH	Alphanumeric information treated as an integer number.	

2.2 CONSTANTS

Five basic types of constants can be used in 810A/840A FORTRAN: integer, floating point, Hollerith, logical and alphanumeric. Complex and double precision constants can be formed from floating point constants. The type of constant is determined by its form and context.

INTEGER

Integer constants may contain up to 7 decimal digits on the 840A or 4 digits on the 810A (single precision). Double precision may be used to give a maximum of 10 digits.

Examples:		
	17432	-0
	53	8388670
	-24739	2

FLOATING POINT

REAL

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

Examples:		
	3.1415768	31.415768E-01
	314.15768E-2	0.0314E2
	31452E-4	

If the decimal point is omitted when expressing a real number in exponential form, the decimal point is assumed to be at the right side of the number.

DOUBLE PRECISION

Double precision constants assume the same characteristics as real data with the advantage of increased significance. Double precision constants contain a decimal exponent.

Examples:		
	-1.0D	3.141592654D
	314D-2	4863.792D05

A double precision symbolic data name must be declared in a DOUBLE PRECISION type statement.

COMPLEX

Complex constants are represented by pairs of real constants separated by a comma and enclosed in parentheses (R1, R2). R1 represents the real part of the complex number and R2, the imaginary part. Either constant may be preceded by a minus sign.

Examples:

FORTRAN Representation	Complex Number
(3.5E-2, -4.26E3)	0.035 - 4260.i
(-5.38E, 3.2)	-5.38 + 3.2i

HOLLERITH

A Hollerith constant is a string of alphanumeric characters. The form for a Hollerith constant is LHf where L is the length of field f including imbedded blanks. Any character of the FORTRAN character set may appear in a HOLLERITH field - letters, numbers, or special characters. The FORTRAN character set is listed in Appendix A. Blanks are valid and significant. The Hollerith word is left-justified with spaces (blanks)

	filling the remainder of the last word if the number of characters is not a multiple of 4 on the 840A or of 2 on the 810A.
LOGICAL	Logical constants have the value of TRUE or FALSE.
ALPHANUMERIC	Alphanumeric constants are strings of alphanumeric characters. The form for an alphanumeric constant is Aw, where w is the field width. Alphanumeric constants are typed in integer form and are left-justified in the word.

2.3 VARIABLES

Simple and subscripted variables are recognized. A simple variable represents a single quantity; a subscripted variable represents an array or a single element within an array. A variable is identified by an alphanumeric name of 6 or less characters, the first of which must be alphabetic. Variables are typed implicitly by name if the variable does not appear in a TYPE statement (see Section 3.1, Type Declarations). If the first letter of the variable name is an I, J, K, L, M, or N the variable is typed integer. Any other first letter indicates a floating point variable.

(1) Sample

General Form
1-6 alphanumeric characters, first of which must be alphabetic.

Examples:

SAM	VICTOR
J57B1	DOUGH
MONEYS	N
N1	M5

(2) Subscripts

A variable may be made to represent a one-, two-, or three-dimensional array by the use of subscripts enclosed in parentheses following the variable name. The variable then becomes a subscripted variable. The subscript may be integer constants, variables or expressions.

(3) Form of Subscripts

General Form: A subscript may take one of the following forms where C and K are any unsigned integer constants and V is an unsigned, non-subscripted integer variable:

C	2
V	I
V C	I 2
C * V	2 * I
C * V K	2 * I 1

(4) Subscripted Variables

General Form: A subscripted variable consists of a variable name followed by one, two or three subscripts enclosed in parentheses.

Examples: SAM (I, J) A(L + 2, J + 3, 5 * M)
VICTOR (I) B(I, 2 * K + 1)

(5) Array Structure

Elements of an array are stored by columns in ascending order of storage locations. The array may have one, two, or three dimensions. The storage for a two-dimensional array is shown in Figure 2-1.

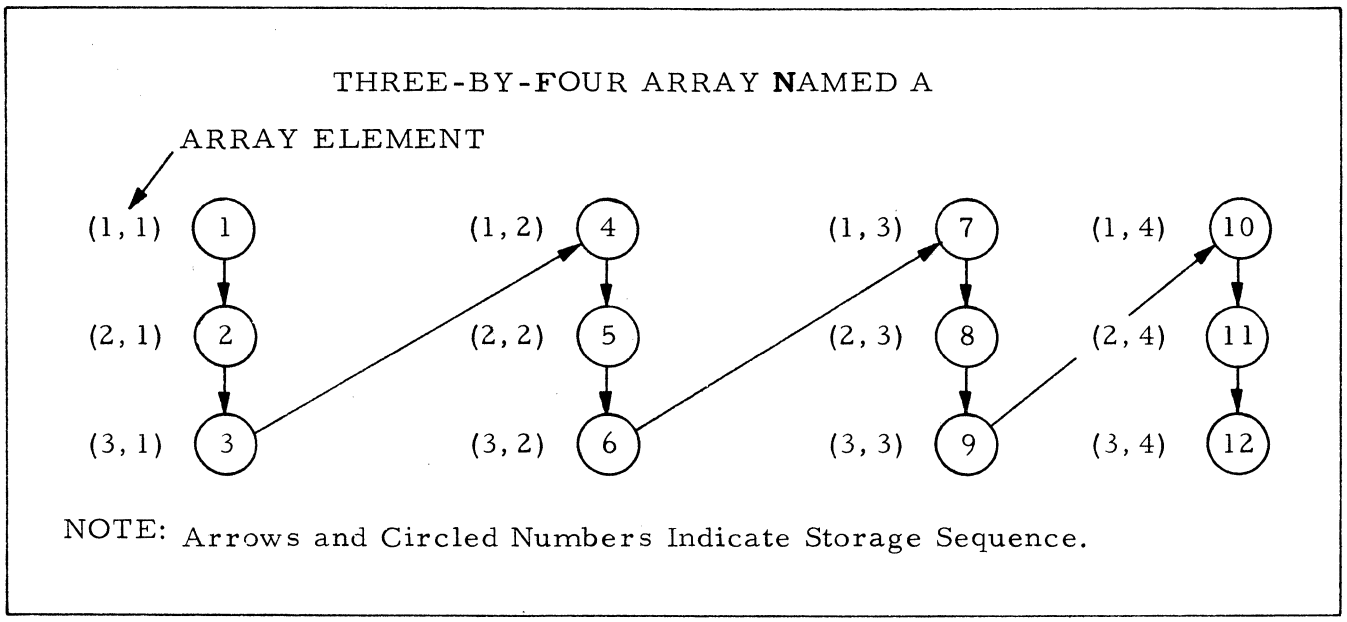


Figure 2-1. Two-Dimensional Array, Storage Sequence of Elements

Elements of an array are stored in sequential positions in memory. A two- or three-dimensional array is stored in consecutive locations such that its first subscript expression (i. e., the left most one) varies most rapidly and its last subscript expression varies least rapidly.

Figure 2-2 shows the storage sequence for a three-dimensional array. Planes are stored sequentially, with each plane stored in the same sequence as a two-dimensional array. As indicated by the numbering sequence in the diagram, the first column of the second plane follows the last column of the first plane, and the first column of the third plane follows the last column of the second plane.

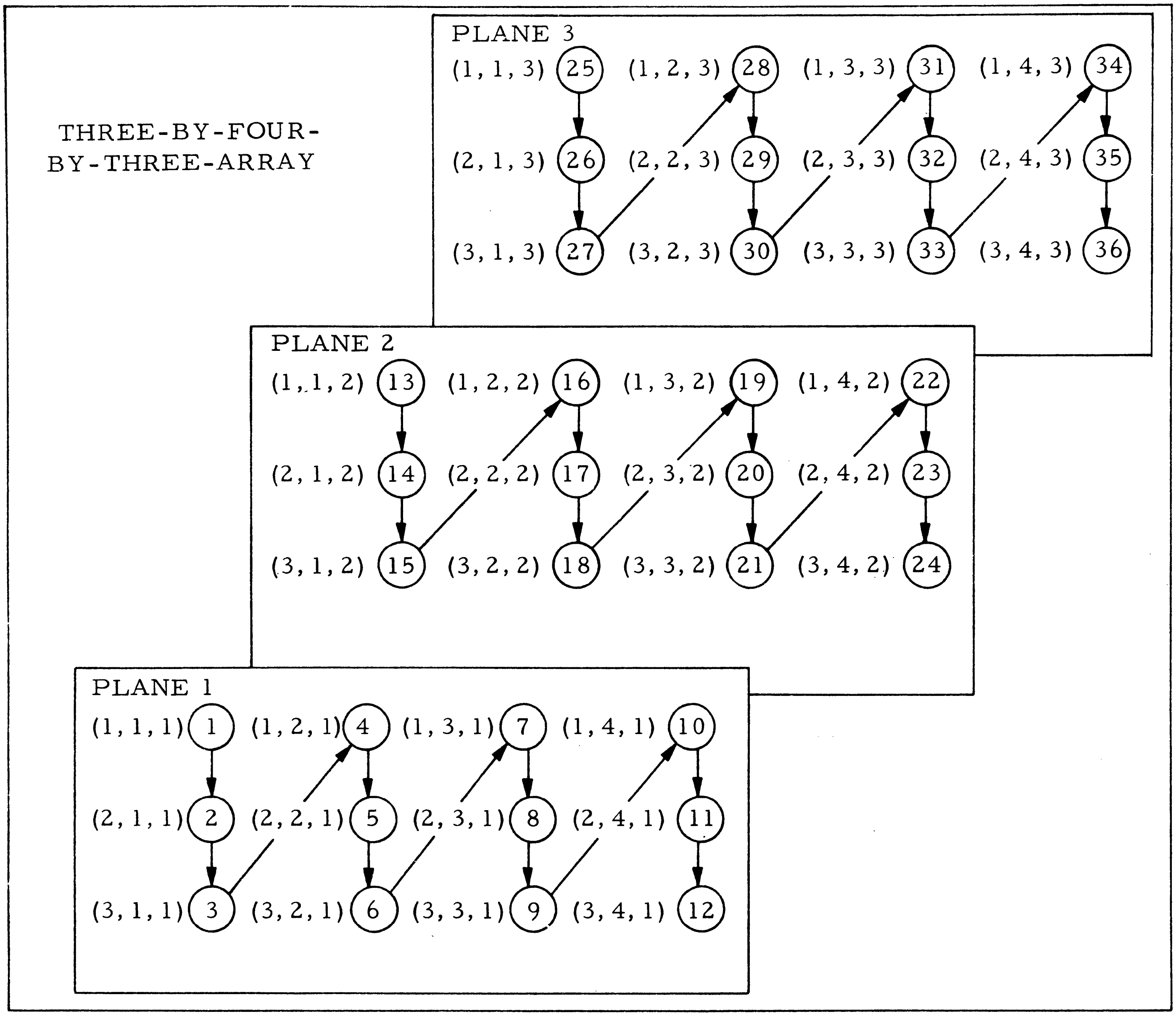


Figure 2-2. Three-Dimensional Array, Storage Sequence of Elements

2.4 STATEMENTS

Two types of statements are recognized by the FORTRAN compiler, executable and nonexecutable. An executable statement performs a calculation or directs the flow of the program. The executable statements are divided into the following groups:

(1) Arithmetic Statements which specify a numerical or logical calculation.

(2) Control Statements which govern the flow of the control in the program.

(3) Input/Output Statements which provide the necessary input/output routines and the input/output formats.

(4) Subprogram Statements which enable the programmer to define and use subprograms.

A nonexecutable statement is used to communicate to the compiler information regarding storage location, variable structure and storage sharing requirements. The specification statements (Section 3) fall under this basic type of statement.

2.5 EXPRESSIONS

Expressions are computational sequences that determine a value, either numeric or logical.

SECTION 3

SPECIFICATION STATEMENTS

The specification statements are: DATA, EQUIVALENCE, COMMON, DIMENSION, and TYPE. These statements provide information to the compiler about the constants and variables used in the program and also provide information about storage allocation. All specification statements are nonexecutable and must appear before the first executable statement in the program.

3.1 TYPE

There are six TYPE statements. All but EXTERNAL may be used to override the implicit typing of the FORTRAN compiler.

General Form

REAL $V_1, V_2, \ldots . V_n$

INTEGER $V_1, V_2, \ldots . V_n$

LOGICAL $V_1, V_2, \ldots . V_n$

DOUBLE PRECISION $V_1, V_2, \ldots . V_n$

COMPLEX $V_1, V_2, \ldots . V_n$

EXTERNAL $V_1, V_2, \ldots . V_n$

where:

Each V may be a variable name, an array name, a function name or an array declaration.

Examples:

```
REAL       INDEX,   NAME
INTEGER    DOG,     XIV
EXTERNAL   SIN,     MATH,    LOGSI
```

Once declared, data types remain constant throughout the program and cannot be changed. Variables that are subprogram names must appear in an EXTERNAL type statement. A variable may appear in two type statements only if one is the EXTERNAL type. The type statement must precede the first use of a name in any executable or DATA statement in the program.

3.2 DIMENSION

General Form
DIMENSION V_1 (i_1), V_2 (i_2), V_3 (i_3) . . . . . . , V_n (i_n)
where:
each V_n is a subscripted variable, and each i_n is composed of 1, 2 or 3 unsigned integer constants and/or variables.

Examples:

```
DIMENSION A(10, 10), B(5, 15, 20), C(100)
```

The DIMENSION statement provides the compiler with the necessary information to allocate the correct number of computer words for storage of the named arrays. The DIMENSION statement defines the maximum size of the arrays. If an array element is addressed which is larger than the specified maximum, the computational results will be erroneous.

In many subprograms, such as matrix manipulation, it may be necessary to vary the dimensions of an array each time the subprogram is called. This is done by including the array name and its dimensions as formal parameters in the FUNCTION or SUBROUTINE statement. The maximum dimension given any array must not exceed the actual dimension of the calling program.

Examples:

```
MAIN PROGRAM:

    DIMENSION A(10, 10), B(10,10), C(10,10)
    1 D(5,5), E(5,5), F(5,5)
                .
                .
                .
                .
    CALL ADDER (A, B, C, 10, 10)
                .
                .
                .
                .
    CALL ADDER (D, E, F, 5, 5)
                .
                .
                .
                .
    CALL ADDER (B, C, A, 10, 10)
```

SUBROUTINE:

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

3.3 COMMON

> General Form
>
> COMMON a, b, c . . . /r/d, e, f, . . / / g, h . .
>
> where:
>
> a, b, . . . are variables or array names, and
> /r/ is a variable that is a block name.

Examples:

```
COMMON  A, B, C /T/D, E/SAM/ F, G, H
COMMON  /BLOCK/I, J, S, T// BIG, SMALL
```

There are two types of COMMON storage. When no block name is given, or two slashes appear together, the array names or variables are said to be in blank, or unlabeled, COMMON. All unlabeled COMMON is stored together in the order of its appearance in the COMMON statements. Block, or labeled, COMMON is stored as separate blocks of data. All blocks given the same name occupy the same space.

If dimension specification appears in a COMMON statement, it need not appear in a DIMENSION statement.

Example:

```
COMMON  A(4, 4, 4)
```

3.4 EQUIVALENCE

> <u>General Form</u>
>
> EQUIVALENCE (a, b, c, ...), (d, e, f, ...) ...
>
> where:
>
> a, b, c; d, e, f; ... are variables or array names that are to share the same storage location.

Example:

EQUIVALENCE (A (2, 3, 5), C(5), D)

An element of a multi-dimensional array may be expressed as the equivalent single dimensioned subscript.

Example:

Element A(2, 1, 2) of the three-dimensional array A(2, 2, 2) may be written as A(6) and equivalenced to variable C(5) as follows:

EQUIVALENCE (A(6), C(5))

The correspondence of a multiple subscripted variable to a single subscripted variable is:

A(i, j, k) = A (the value of (i + (j-1) * I + (k-1)*I*J))

where:

i, j, k are integer constants and I, J are the integer constants appearing in DIMENSION A(I, J, K)

Storage allocation is different for equivalenced arrays depending on whether the storage area is a COMMON block or not.

If two arrays, not in COMMON, are equivalenced:

DIMENSION A(3), B(5), C(4)

EQUIVALENCE (A(3), C(2))

Storage allocations are assigned as follows:

L	A(1)	
L+1	A(2)	C(1)
L+2	A(3)	C(2)
L+3		C(3)
L+4		C(4)
L+5	B(1)	
L+6	B(2)	
L+7	B(3)	
L+8	B(4)	
L+9	B(5)	

However, if the arrays are in COMMON

```
COMMON  A(3), B(5), C(4)

EQUIVALENCE (A(3), C(2))
```

Storage is assigned as follows:

L	A(1)	
L+1	A(2)	C(1)
L+2	A(3)	C(2)
L+3	B(1)	C(3)
L+4	B(2)	C(4)
L+5	B(3)	
L+6	B(4)	
L+7	B(5)	

Variables brought into a COMMON block through the use of an EQUIVALENCE statement may increase the size of the block. The COMMON block may only be increased beyond the last storage assignment for that block.

Example:

```
COMMON  A, B, C

DIMENSION  D (3)

EQUIVALENCE  (D(1), B)
```

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

Illegal equivalencing:

```
COMMON  A, B, C

DIMENSION  D (3)

EQUIVALENCE (B, D(3))

                                 D(1)
Origin ---   L        A          D(2)
             L+1      B          D(3)
             L+2      C
```

The above example is illegal as the COMMON block is increased upwards.

3.5 DATA

> General Form
>
> DATA list/literals/, $list_2$/$literals_2$/, ...
>
> where:
>
> list is a list of variables being defined and literals is a list of associated constants

Examples:

```
DATA  A, B, I/14.314, 7.2, 3HEND/, C, D/5.0, 3.2/
```

The literals in a data statement may be integer, real, double precision, complex, or alphanumeric. An alphanumeric field is written as nH followed by n alphanumeric characters. Each group of 4 characters forms a word (2 characters on the 810). If n is not a multiple of 4 (or 2), the characters are left-justified and the remainder of the word is filled with blanks.

Variables used in a DATA statement may not appear in a COMMON statement. To enter variables in a COMMON block, the BLOCK DATA subprogram must be used (see Section 6.7).

SECTION 4

EXPRESSIONS

This section details the two types of statements allowable in 810A/840A FORTRAN, arithmetic expressions and logical expressions.

4.1 ARITHMETIC EXPRESSIONS

An arithmetic expression may be a constant, a variable (simple or subscripted) or an evaluated function. Arithmetic operators may be combined with constants, variables and functions to form complex expressions.

Arithmetic operators are:

+ addition

- subtraction

/ division

* multiplication

** exponentiation

Tables 4-1 and 4-2 show which constants, variables and functions may be combined by the arithmetic operators to form valid expressions. Table 4-1 gives the valid combinations with respect to the arithmetic operators +, -, *, /. Table 4-2 gives valid combinations with respect to the arithmetic operator **. In these tables, Y indicates a valid combination and N indicates an invalid combination.

Table 4-1. Arithmetic Operators (except exponentiation)

+, -, *, /	Real	Integer	Complex	Double Precision	Logical
Real	Y	N	Y	Y	N
Integer	N	Y	N	N	N
Complex	Y	N	Y	N	N
Double Precision	Y	N	N	Y	N
Logical	N	N	N	N	N

Table 4-2. Exponentiation Combinations

**	Real	Integer	Complex	Double Precision	Logical
Real	Y	Y	N	Y	N
Integer	N	Y	N	N	N
Complex	N	Y	N	N	N
Double Precision	Y	Y	N	Y	N
Logical	N	N	N	N	N

Any real constant, variable or function name combined with a complex or double precision quantity will result in a complex or double precision quantity.

Certain operators may not appear in sequence. The expression A * * B * * C is not permitted; it must be written as either A * * (B * * C) or (A * * B) * * C, whichever is intended, whereas A * B * C is permissible. Basic rules of algebra are used.

The hierarchy of operations may be altered by the use of parentheses to specify operations to be done first. Where parentheses are omitted, the order of operations is:

(1)	**	exponentiation
(2)	* and /	multiplication and division
(3)	+ and -	addition and subtraction

Expressions are scanned from left to right.

In the following examples, R indicates an intermediate result.

Examples:

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

D + E --→ R_1

C/R_1 --→ R_2

B-R_2 --→ R_3

A*R_3 --→ R_4

evaluation complete

$$5. * (3. ** SPC + SQRT (A ** 2)) / 4. * (B ** ABS (X))$$

A**2	--→	R_1
SQRT(R_1)	--→	R_2
3.**SPC	--→	R_3
R_2+R_3	--→	R_4
B**ABS(X)	--→	R_5
5*R_4	--→	R_6
R_6/4.	--→	R_7
R_7*R_5	--→	R_8

evaluation complete

4.2 LOGICAL EXPRESSIONS

A logical expression consists of certain sequences of logical constants, logical variables, references to logical functions, and arithmetic expressions separated by logical operation symbols or relational operation symbols. When evaluated, a logical expression always has the value .TRUE. or .FALSE..

The logical operation symbols are as follows: (a and b represent logical expressions):

LOGICAL OPERATOR	DEFINITION
.NOT. a	Has the value .TRUE. if a has the value .FALSE., or has the value .FALSE. if a has the value .TRUE..
a .AND. b	Has the value .TRUE. if a and b both have the value .TRUE., or has the value .FALSE. if either a or b have the value .FALSE..
a .OR. b	Has the value .TRUE. if either a or b have the value .TRUE., or has the value .FALSE. if a and b both have the value .FALSE..

Two logical operators may not appear adjacent to each other unless the second logical operator is .NOT..

NOTE: The logical operators shown above and relational operators shown below must be preceded and followed by a period.

Logical comparison may be effected by use of the following relational operators:

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

The value of a logical relation is .TRUE. if satisfied, .FALSE. if not satisfied. In the absence of parentheses indicating a hierarchy of operations, logical expressions are evaluated as follows:

(1) Arithmetic expressions are evaluated

(2) Logical relations are determined:

.EQ., .GE., .GT., .LT., .LE., .NE.

(3) .NOT.

(4) .AND.

(5) .OR.

Logical Statements have the general form:

General Form

a = b

where:

a is a logical variable or analog element
b is a logical expression

The logical expression is evaluated and the previous value of the logical variable on the left of the equals sign is replaced with .TRUE. or .FALSE..

In the following examples it is assumed that all variables on the left of the equals sign are typed logical and all other variables are typed real.

LOGICAL STATEMENT	INTERPRETATION
A = .FALSE.	The previous value of A is replaced by the logical constant .FALSE..
B = X.LE.5.	If X is less than or equal to 5, B has the value .TRUE., otherwise B is set equal to .FALSE..
C = X.GT.5. .OR.Y.LT.Z	Determine a value of .TRUE. or .FALSE. for X.GT.5. (it is .TRUE. if X is .GT.5 and .FALSE. if X is less than or equal to 5). Determine a value of .TRUE. or .FALSE. for Y.LT.Z. If the value of either relation is .TRUE., replace the previous value of C with .TRUE.; otherwise replace the previous value of C with .FALSE..
LOGIC = .TRUE. .AND. 400.GE.X	If 400 is greater than or equal to X, store .TRUE. in LOGIC; otherwise store .FALSE..
A(1) = .NOT. (X.EQ.50./Y**2)	If X equals 50. divided by Y^2, store .FALSE. in logical array element A(1); otherwise store .TRUE. in A(1).
B = X .AND. .NOT. Y	If Y is .FALSE. and X is .TRUE. store the value .TRUE. in B; otherwise store .FALSE. in B.
D = X .GT. (50.*Y*W(X-2.))	The arithmetic expression is evaluated in the conventional manner (innermost parenthesis is evaluated first). If X is greater than the final result, .TRUE. is stored in D; otherwise .FALSE. is stored in D.

SECTION 5

CONTROL STATEMENTS

The control statements are used to alter the normal flow of control of statements from the sequential mode. Control may be transferred to an executable statement only.

5.1 UNCONDITIONAL GO TO STATEMENT

> <u>General Form</u>
>
> GO TO n
>
> where:
>
> n is a statement number

Example:

GO TO 25

This statement causes control to be transferred to statement number 25.

5.2 COMPUTED GO TO STATEMENT

> <u>General Form</u>
>
> GO TO (n_1, n_2, n_3 ...... n_m), i
>
> where:
>
> n_1, n_2, n_3 ...... n_m are statement numbers, and i is a nonsubscripted integer variable.

Example:

GO TO (40, 20, 30, 45), K

This statement causes control to be transferred to statement number 40, 20, 30, 45, depending on whether the value of K is 1, 2, 3, or 4, respectively, at the time of execution of the statement. In the example, if K is 3 at the time of execution, control would transfer to the third statement number in the list; statement number 30.

5.3 ASSIGN STATEMENT

General Form
ASSIGN n to i
where:
n is a statement number, and i is an integer variable

Examples:

ASSIGN 17 to J

ASSIGN 9 to JA

This statement causes a subsequent GO TO n_1 (m_1, m_2, m_3, .., m_j) to transfer control to statement number i, where i is one of the statement numbers included in the series m_1, m_2, m_3, ..., m_j.

5.4 ASSIGNED GO TO STATEMENT

General Form
GO TO n, (m_1, m_2, m_3, ..., m_j)
where:
n is a non-subscripted integer variable appearing in a previously executed ASSIGN statement, and m_1, m_2, m_3, ..., m_j are statement numbers.

Example:

GO TO J,(5, 17, 3, 9, 24)

This statement causes control to be transferred to the statement number last assigned to n by an ASSIGN statement. 5, 17, 3, 9, 24 are a list of statement numbers that J may assume.

5.5 ARITHMETIC IF STATEMENT

General Form

IF (A) n_1, n_2, n_3

where:

A is an arithmetic expression, and n_1, n_2, n_3 are statement numbers.

Examples:

```
IF (A (I, J) - B) 10, 15, 2

IF (X** 2 + 3.) 3, 3, 10
```

This statement causes a branch to statement number n_1, n_2 or n_3 if the value of the arithmetic expression is less than, equal to, or greater than zero, respectively.

5.6 LOGICAL IF STATEMENT

General Form

IF (L) S

where:

L is a logical expression, and

S is any executable statement, except a DO statement or another logical IF statement.

Examples:

```
IF (A .LE. B) GO TO 7

IF (A .AND. B) CALL BOTH

IF (L) X = SIN Y
```

If the value of the logical expression L is .TRUE., statement S is executed. Control is then transferred to the next statement following the logical IF unless S is a GO TO statement or an arithmetic IF, in which case control is transferred as indicated.

If the value of the logical expression L is .FALSE., the statement following the logical IF is executed.

5.7 DO STATEMENT

<u>General Form</u>

DO m i = n_1, n_2, n_3

where:

m is a statement number

i is a non-subscripted integer variable

n_1, n_2, n_3 are unsigned integer constants or integer variables.

If n_3 is not stated, it is assumed to be 1.

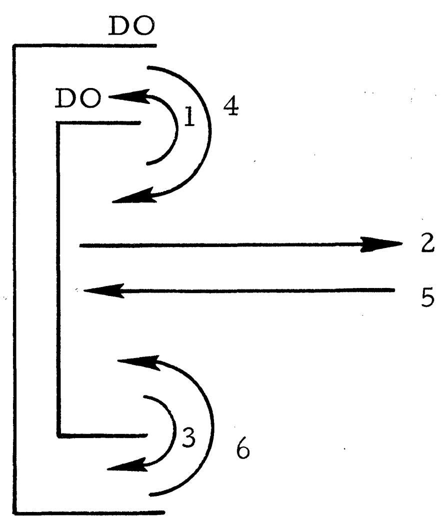
Examples:

```
DO 30 I = 5, 20, 5

DO 25 I = 1, K
```

The DO statement is a command to execute repeatedly the statements between the DO statement and m. As shown in the general form, n_1 is the initial setting of i; n_2 is the terminal value of i and n_3 is the increment by which i is raised on each pass through the DO loop. A DO loop will be executed at least once.

The <u>range</u> of a DO is that set of statements between the DO statement and statement m. After the last pass through the DO loop, the DO is said to be <u>satisfied</u>. The <u>index</u> of the DO, i, is available throughout the range of the DO for use in computations but it can never be altered in any way. The DO parameters, n_1, n_2, and n_3 also may not be altered in any way while in the range of DO loop.



<u>DO LOOPS</u>

1, 2, 3 are permitted transfers.

4, 5, 6 are illegal transfers.

The DO loop may contain within it other DO loops, provided that each DO loop is completely contained within the range of the outer loop. Such a configuration is called nested DOs. Control may be transferred freely while inside the range of a DO; it is also permissable to transfer control out of the range of a DO or to another DO statement. It is illegal to transfer into the range of a DO from outside.

A DO loop cannot end on an IF or GO TO type statement.

5.8 CONTINUE STATEMENT

General Form
CONTINUE

The CONTINUE statement is a dummy statement and does not alter the normal sequencing of the program. It is classified as an executable statement. The main use of the CONTINUE statement is as a reference point and as a last statement in the range of a DO.

5.9 PAUSE STATEMENT

General Form
PAUSE or
PAUSE n
where:
n is an unsigned octal integer constant of 1 to 5 octal digits

The PAUSE statement is used to halt the program at some time during execution to allow some external set-up, such as the changing of tapes. Operator action is necessary to restart the program. Once restarted, the program resumes at the first executable statement after the PAUSE. The identification constant, n, indicates the particular PAUSE statement which caused the delay, since the identification is displayed. The identification constant may be 1 to 5 octal digits.

5.10 END STATEMENT

General Form
END

The END statement terminates compilation of a program. The END statement must be the last physical statement in a source program.

5.11 STOP STATEMENT

General Form
STOP or STOP n

The STOP statement causes a final termination of the program. The constant, n, when included indicates the particular STOP statement that ended execution of the program.

5.12 RETURN STATEMENT

General Form
RETURN

RETURN is the normal exit from any subprogram to the calling program.

5.13 SENSE LIGHT SUBROUTINE

General Form
CALL SLITE (i)
where:
i is an integer constant corresponding to sense light numbers.

This subroutine turns on the sense light designated by the argument, i. If the argument is zero, all sense lights are set to OFF.

The sense lights are simulated by reserving a word in memory in which all bits are set to zero for OFF, or the bits are set to one, simulating ON.

5.14 SENSE LIGHT TEST SUBROUTINE

General Form
CALL SLITET (i, K)
where:
i is the sense light number, and
k = 1 if light is ON, or
k = 2 if light is OFF

This subroutine checks the status of the sense lights indicated by the first argument, i. If the sense light is ON, the second argument, k, is equal to one; if the sense light is OFF, k is equal to two.

5.15 SENSE SWITCH TEST SUBROUTINE

General Form
CALL SSWTCH (i, k)
where:
i is the sense switch number, and
k is the status of the sense switch

This subroutine is used to check the status of the sense switches designated by the first argument, i. If the sense switch is ON, k returns with a value of one; if the sense switch is OFF, k returns with the value of two.

5.16 ACCUMULATOR OVERFLOW TEST SUBROUTINE

General Form
CALL OVERFL (j)
where:
j is the status of the overflow indicator

If an overflow has occurred in the accumulator register, a call to this subroutine turns OFF the overflow indicator and returns with a value of one for the argument, j; otherwise j will return with a value of two.

SECTION 6

SUBROUTINES, FUNCTIONS AND SUBPROGRAM STATEMENTS

There are four classes of subroutines in FORTRAN: arithmetic statement functions, built-in functions, FUNCTION subprograms and SUBROUTINE subprograms. The first three classes are grouped as functions. Functions differ from subprograms as they are always single-valued (they return a single result) and they are referenced by an arithmetic expression. Subprograms are referenced by a CALL statement and can return more than one value.

6.1 NAMING SUBROUTINES

All four types of subroutines are named in the same manner. A subroutine name consists of 1-6 alphanumeric characters, the first of which must be alphabetic.

The type of a function, which determines the type of the result, may be implicitly typed by the function name. In the case of an arithmetic statement function, the name may be placed in a type statement to override the implicit type; a FUNCTION subprogram type may be written preceding the word FUNCTION to override the type implied by the function name. The type of a built-in, or library, function is indicated within the FORTRAN processor and does not have to appear in a type statement (see Table 6-1).

The type of a SUBROUTINE subprogram is immaterial, as the type of the results are dependent only upon the type of the variable names appearing in the calling sequence.

6.2 ARITHMETIC STATEMENT FUNCTIONS

Arithmetic statement functions are defined by a single arithmetic expression and are applicable only to the source program in which they are defined.

General Form

$a(ARG_1, ARG_2 \ldots.) = b$

where:

a is a function name,

ARG_1, ARG_2 .... are
the dummy arguments of the function,
and b is an arithmetic expression

TABLE 6-1

LIBRARY FUNCTIONS

FUNCTION	DEFINITION	NUMBER OF ARGUMENTS	NAME	TYPE	
				ARGUMENT	FUNCTION
ABSOLUTE VALUE	$\|ARG\|$	1	ABS IABS DABS	REAL INTEGER DOUBLE	REAL INTEGER DOUBLE
TRUNCATION	SIGN OF ARG TIMES LARGEST INTEGER $\leq \|ARG\|$	1	AINT INT IDINT	REAL REAL DOUBLE	REAL INTEGER INTEGER
REMAINDERING*	ARG_1 (MOD ARG_2)	2	AMOD MOD	REAL INTEGER	REAL INTEGER
CHOOSING LARGEST VALUE	MAX (ARG_1, ARG_2 ···)	$\geq$ 2	AMAX0 AMAX1 MAX0 MAX1 DMAX1	INTEGER REAL INTEGER REAL DOUBLE	REAL REAL INTEGER INTEGER DOUBLE
CHOOSING SMALLEST VALUE	MIN (ARG_1, ARG_2 ···)	$\geq$ 2	AMIN0 AMIN1 MIN0 MIN1 DMIN1	INTEGER REAL INTEGER REAL DOUBLE	REAL REAL INTEGER INTEGER DOUBLE
FLOAT	CONVERSION FROM INTEGER TO REAL	1	FLOAT	INTEGER	REAL
FIX	CONVERSION FROM REAL TO INTEGER	1	IFIX	REAL	INTEGER
TRANSFER OF SIGN	SIGN OF ARG_2 TIMES $\|ARG_1\|$	2	SIGN ISIGN DSIGN	REAL INTEGER DOUBLE	REAL INTEGER DOUBLE
POSITIVE DIFFERENCE	ARG_1 − MIN (ARG_1, ARG_2)	2	DIM IDIM	REAL INTEGER	REAL INTEGER
OBTAIN MOST SIGNIFICANT PART OF DOUBLE-PRECISION ARG.		1	SNGLE	DOUBLE	REAL
OBTAIN REAL PART OF COMPLEX ARG.		1	REAL	COMPLEX	REAL
OBTAIN IMAGINARY PART OF COMPLEX ARG.		1	AIMAG	COMPLEX	REAL
EXPRESS SINGLE-PRECISION ARG. IN DOUBLE-PRECISION FORM		1	DBLE	REAL	DOUBLE
EXPRESS TWO REAL ARGS. IN COMPLEX FORM	$ARG_1 + ARG_2\sqrt{-1}$	2	CMPLX	REAL	COMPLEX
OBTAIN CONJUGATE OF A COMPLEX ARG	FOR ARG = X + iY, C = X − iY	1	CONJG	COMPLEX	COMPLEX

* The function MOD (ARG_1, ARG_2) is defined as $ARG_1 - [ARG_1/ARG_2]\ ARG_2$, where [x] is the integral part of x.

Examples:

```
ROOT (A, B, C) = (-B * SQRT (B** 2-4. * A * C)/(2. *A)

FIRST (X) = A * X + B

Z (X, Y) = 7.3*SIN(X) + 4.7 * COS (Y)
```

During compilation, the statement function definition is inserted in the code wherever the statement function reference appears as an operand in an expression. The statement function name must not appear in a COMMON, DIMENSION, EQUIVALENCE or EXTERNAL statement. All statement functions must appear before the first executable statement in the program or subprogram, but they must follow all specification statements. The arguments of the function are symbolic names that are replaced by the actual call arguments when the function is used.

Example:

```
DIMENSION   X(5), Y (5), Z (5)

ROOT        (A, B, C) = (-B*SQRT (B** 2-4.0*A*C) /2.0*A)
      .
      .
      .
      .
APPLE      = ROOT (X(I),  Y(I),  Z(I) ) * BIG/AVG
```

The dummy arguments assume the values of X(I), Y(I), and Z(I), respectively and the statement function expression is executed.

```
APPLE = (-Y(I) * SQRT (Y(I) ** 2-4.0 * X(I) * Z(I) ) / (2.0 ** X(I) * BIG/ AVG
```

6.3 LIBRARY FUNCTIONS

Library functions are pre-defined subroutines within the FORTRAN processor. A list of available library functions is given in Table 6-1.

6.4 FUNCTION SUBPROGRAM

<u>General Form</u>

FUNCTION name (ARG_1, ARG_2, ARG_3, ... ARG n) or
type FUNCTION name (ARG_1, ARG_2..... ARG n)

where:

Name is the symbolic name of a single-valued function, and ARG_1, ARG_2, ... ARG n are variable names or the dummy name of a SUBROUTINE or FUNCTION subprogram, and type is a data type name, i.e., INTEGER, REAL, etc.

Examples:

```
FUNCTION SAM (X, Y, A)

REAL FUNCTION IBAR (TEMP, ALT)

INTEGER FUNCTION JOE (IX, JOKE, SAT)

DOUBLE PRECISION FUNCTION DP (A, C, X)

COMPLEX FUNCTION ABLE (BMIX, AMIX)

LOGICAL FUNCTION TRFAL (S, T, U)
```

The first statement of a FUNCTION subprogram must be a FUNCTION statement; the last statement must be an END statement. There must be at least one RETURN statement. The name of the function must appear on the left side of an arithmetic statement or in an input statement.

Example:

```
FUNCTION SAM (X, Y, A)
      .
      .
      .
      .
      .
      .
SAM = A ** 2 + B
      .
      .
      .
      .
      .
      .
RETURN
END
```

The FUNCTION subprogram may contain any FORTRAN statement, except SUBROUTINE or another FUNCTION statement.

There must be at least one argument in the FUNCTION statement. Dummy arguments may not appear in an EQUIVALENCE statement in the FUNCTION subprogram. The arguments of a FUNCTION statement may be considered dummy names which are replaced at execution time by the actual arguments in the calling program. The actual arguments must agree in number, order, and type with the dummy arguments.

When the dummy argument is an array name, a DIMENSION or COMMON (with dimensions) statement must appear in the FUNCTION subprogram; also the corresponding actual argument must be a dimensioned array name.

A FUNCTION subprogram is referenced by using its name as an operand in an arithmetic or logical expression.

When the name of a FUNCTION subprogram is used as an actual argument, the name must appear in an EXTERNAL statement.

The FUNCTION subprograms that are available with FORTRAN are given in Table 6-2.

TABLE 6-2

MATHEMATICAL SUBROUTINES

FUNCTION	DEFINITION	NUMBER OF ARGUMENTS	NAME	TYPE ARGUMENT	TYPE FUNCTION
EXPONENTIAL	e^{ARG}	1	EXP DEXP CEXP	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
NATURAL LOGARITHM	LOG_e (ARG)	1	ALOG DLOG CLOG	REAL DOUBLE COMPLEX	REAL DOUBLE COMP
COMMON LOG	LOG_{10} (ARG)	1	ALOG10 DLOG10	REAL DOUBLE	REAL DOUBLE
TRIGONOMETRIC SINE	SIN(ARG)	1	SIN DSIN CSIN	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
TRIGONOMETRIC COSINE	COS(ARG)	1	COS DCOS CCOS	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
HYPERBOLIC TANGENT	TANH (ARG)	1	TANH	REAL	REAL
SQUARE ROOT	$(ARG)^{1/2}$	1	SQRT DSQRT CSQRT	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
ARCTANGENT	ARCTAN(ARG) $ARCTAN(ARG_1/ARG_2)$	1 1 2 2	ATAN DTAN ATAN2 DATAN2	REAL DOUBLE REAL DOUBLE	REAL DOUBLE REAL DOUBLE
REMAINDERING*	ARG_1 (MOD ARG_2)	2	DMOD	DOUBLE	DOUBLE
MODULUS		1	CABS	COMPLEX	REAL

* The function MOD (ARG_1, ARG_2) is defined as $ARG_1 - [ARG_1/ARG_2]\ ARG_2$, where $[x]$ is the integral part of x.

6.5 SUBROUTINE SUBPROGRAM

> General Form
>
> SUBROUTINE name (ARG_1, ARG_2. . . . ARG n)
>
> where:
>
> Name is the symbolic name of a subprogram, and, each argument, if any, is a variable name or the dummy name of a SUBROUTINE or FUNCTION subprogram.

Examples:

SUBROUTINE MAYBE (A, B, X, BIG)

SUBROUTINE SORT

The SUBROUTINE statement must be the first statement of a SUBROUTINE program. The SUBROUTINE returns values, if any, either through one or more of the arguments listed or through COMMON storage.

The actual arguments in the calling program must agree in order, type and number with the dummy arguments in the subroutine. If a dummy argument is an array name, a DIMENSION or COMMON (with dimensions) statement must appear in the SUBROUTINE subprogram; also the corresponding actual argument in the CALL statement must be a dimensioned array name. No dummy argument may appear in an EQUIVALENCE statement in the SUBROUTINE subprogram.

The SUBROUTINE subprogram must have at least one RETURN statement. It may contain any FORTRAN statement except FUNCTION or another SUBROUTINE statement.

If the name of a FUNCTION or another SUBROUTINE subprogram is used as an argument, the name must first appear in an EXTERNAL statement.

Example:

EXTERNAL SIN

CALL ANGLE (A, X, SIN)

6.6 THE CALL STATEMENT

The CALL statement transfers control to the subprogram and gives it the actual arguments. The arguments may be:

(1) Any type of constant

(2) Any type of subscripted or non-subscripted variable.

(3) An arithmetic or logical expression.

(4) Alphanumeric characters, written as Hollerith fields, nH.

The arguments of the subroutine in the CALL statement must agree in order, number and type with the corresponding dummy arguments in the SUBROUTINE statement in the called program.

6.7 BLOCK DATA SUBPROGRAM

To enter data from a DATA statement into a COMMON block during compilation, the BLOCK DATA subprogram must be used. This subprogram contains only the DATA, COMMON, DIMENSION and TYPE statements associated with the data being entered. The BLOCK DATA subprogram may not contain any executable statements. The first statement must be the BLOCK DATA statement.

Example:

```
BLOCK DATA

DIMENSION B(5), X(10)

COMMON / ALPHA/ A, B, C / RLM/ X, Y, Z

INTEGER B

DATA (A, B(1) / 5.3, 7 /, X (1), X (3) / 3.14, 10.93 /
```

Note in the example above that all elements in the COMMON block must be listed even though they do not appear in the DATA statement. Data may be entered into more than one COMMON block in a single BLOCK DATA statement.

SECTION 7

INPUT/OUTPUT

The statements that control the input and output of information to or from the computer may be grouped as follows:

General I/O statements: The READ and WRITE statements that transmit data between core storage and I/O devices.

Manipulative I/O statements: Statement such as END FILE, BACKSPACE, and REWIND which manipulate I/O devices.

Format specifications: The FORMAT statement, which is non-executable, gives a description of the incoming or outgoing data.

7.1 INPUT/OUTPUT LISTS

An I/O list is a list of items, separated by commas, which contains the names of the variables or arrays to be transmitted by a general I/O statement. The list may contain subscripted variables or an implied DO, single or nested. The following example shows the use of nested DO's in an I/O list.

A, (B(I), I = 1, 3), (C(J), D(J), J = 1, 3), ((E (I, J), I = 1, 10, 2,), J = 1, 2)

This list implies that the information in the external I/O device is arranged as follows:

A, B(1), B(2), B(3), C(1), D(1), C(2), D(2), C(3), D(3), E(1, 1), E(3, 1), E(5, 1), ... E (9, 1), E(1, 2), E(3, 2)..... E(9, 2)

The input list K, A(K), or K, (A (I), I = 1, K) is valid, as K is read in prior to its use as an index or as an indexing parameter.

Any number of quantities may appear in a single list, however, each quantity must have the correct format corresponding to it. The list controls the quantity of data read.

7.1.1 Short-List Notation

An array that has been previously dimensioned in a COMMON, DIMENSION, or data type statement may be transmitted without subscripts.

Example:

```
DIMENSION A (5)

READ (5, 10) A
```

The entire array, A, is read in, that is, 5 quantities of A are read and stored.

7.2 GENERAL INPUT/OUTPUT STATEMENTS

There are two types of input statement used by 810A/840A FORTRAN. The basic forms are:

READ (i) List (Non-Formatted)

READ (i, n) List (Formatted)

where:

- i is a code number identifying the input device (logical unit number). It may be an unsigned integer constant or integer variable.
- n is a statement number of a FORMAT statement or the name of an array in which the necessary format information is stored.
- LIST is an optional list of the names of variables, array, and/or array elements that are to receive input values at execution time by this particular READ statement.

A non-formatted READ statement causes the information or logical unit i to be read in as binary information. A formatted READ is executed under control of a FORMAT statement (n). The decimal and/or alphanumeric data read is then converted into internal form.

Output statements are identical to the input statements:

WRITE (i) List (Non-Formatted)

WRITE (i, n) List (Formatted)

If no list is included any Hollerith information and/or line spacing instruction in the FORMAT statement is executed.

7.3 FORMAT

<u>General Form</u>

n FORMAT (S_1, S_2, ... S_m) or

n FORMAT (S_1, S_2, ... S_m/S'_1,
S'_2, ... S'_m/S''_1, ... S''_m)

where:

each S_i is a format specification, and n is a statement number.

Examples:

FORMAT (1H1, 5X, I2, 2X, F10.3/5X, I4, F7.3)

FORMAT (6F10.2)

FORMAT (I5, 3(2X, F7.2))

FORMAT statements are non-executable and may be placed anywhere in the source program. Each FORMAT statement must have a unique statement number, by which it is referenced.

Slashes (/) in the FORMAT statement are used to signify record terminators. When writing on an off-line printer, the maximum record length corresponds to the length of one printed line. When punching a card, the maximum record length is 80 characters if the card is to be read on-line.

During input/output of data, the program scans the FORMAT statement specified by the READ or WRITE statement. When a format specification is found for numeric field and there are still items in the list to be transmitted, the input/output of the numeric data takes place according to the format specification, and the program continues the scan of the FORMAT statement. If no items remain to be transmitted, execution of that particular I/O statement ceases. Thus, a FORMAT statement is repeated until the list associated with it is exhausted.

7.4 CONVERSION SPECIFICATIONS

The data elements in an I/O list are converted from external to internal or from internal to external representations. The FORMAT specifications may also contain editing codes.

Conversion Codes

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

Editing Codes

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

Both w and d are unsigned integer constants; w indicates the field width and d specifies the number of digits to the right of the decimal point within the field.

7.4.1 I (Integer) Conversion

Form: Iw

(1) Input: The input field w may contain only the characters +, -, 0 through 9, or blank. When a sign is included, it must precede the first digit in the field. Blanks are interpreted as zeros.

Input Examples:

```
    READ (1, 10) I, J, K, L, M
 10 FORMAT (I3, I4, 2I3, I2)
```

Input Card:

1 2 3	∧ -1 5	1 ∧ 1	∧ ∧ 2	3 ∧
← 3 →	← 4 →	← 3 →	← 3 →	← 2 →

I contains 123
J contains -15
K contains 101
L contains 2
M contains 30

(2) Output: The I conversion is used to output decimal integer values. In the output data field, digits are right-justified.

Output Examples:

```
    WRITE (2, 10) I, J, K,        I = -1 2 3
 10 FORMAT (3I5)                  J = +5
                                  K = +7942
```

Output Record:

∧ -1 2 3	∧ ∧ ∧ ∧ 5	∧ 7 9 4 2
← 5 →	← 5 →	← 5 →

7.4.2 F (Fixed - Point Decimal) Conversion

Form: Fw.d

(1) Input: The input field consists of an integer and a fraction subfield. A decimal point supplied by the input data overrides the decimal point indicated by d.

Examples:

Input Field	Format	Converted Values	Remarks
127.394	F7.3	127.394	Integer and fraction field
127394	F7.3	127.394	No fraction subfield. Input converted as 127394×10^{-3}
1.27394	F7.3	1.27394	Decimal point overrides d.

(2) Output: On output, the corresponding list element must be in floating point. The number is output as a right-justified decimal number in the field w. The field, w, must be large enough to allow for a sign, even if the number is positive. For numbers less than 1.0 the field must allow room for a leading zero and a decimal point.

Examples:

```
   WRITE (2, 10) A        A contains 127.394
10 FORMAT (F10.3)
   RESULT:  127.394

   WRITE (2, 11) A
11 FORMAT (F8.3)
   RESULT:  127.394
```

WRITE: (2, 12) A	A contains 127.394
12 FORMAT (F7.3)	
RESULT: $27.394	No provision made for the sign of the number, so $ is printed followed by as many digits as possible. If number is negative, an equal sign precedes the digits printed.

7.4.3 E (Explicit Exponent) Conversion

Form Ew.d

(1) Input: An E field specification consists of a decimal number, with or without a fractional subfield, and an exponent. The general form of the exponent is E ± XX, where XX is the numeric exponent. Blanks appearing in the exponent field are ignored, therefore blanks may be deleted. If the sign of the exponent is omitted, it is considered to be positive; if the sign of the exponent is used, the E may be omitted.

Examples:

INPUT:

127394-3

12.7394E1

1273.94E+01

As in F conversion, if the data field being read contains a decimal point, the actual position of the decimal overrides d.

(2) Output: The output form of an E conversion consists of a space for a sign followed by a mantissa and an exponent. The mantissa is a signed decimal fraction preceded by a blank and a decimal point; the exponential part consists of the letter E followed by a sign and a two digit exponent. If the exponent is positive, the sign is omitted and a blank space is printed.

For format scaling it is necessary to add seven to the number of digits in the field width, w, to format an E conversion correctly.

One space for sign of the number

One blank space

One space for decimal point

One space for letter E

One space for sign of exponent

Two spaces for exponent

The output number is right-justified when the field width is wider than necessary.

7.4.4 G (Generalized E or F) Conversion

Form: Gw.d

(1) Input: The G-conversion may be used in place of the F-conversion since the processing of the G- and F-conversion codes are identical. The incoming number is stored internally as if an E-conversion code had been used.

(2) Output: The G-conversion output is a real constant expressed without an exponent, as in F-conversion, if the number is between 1 and 10 inclusive. Otherwise, E-conversion is used.

7.4.5 L(Logical) Conversion

Form: Lw

(1) Input: On input, the L-conversion allows logical quantities to be entered (i.e., .TRUE. or .FALSE.). The first non-blank character of the input field must be either a T (for .TRUE.) or an F (for .FALSE.). If the T or F are not right-justified in the field, all other letters are ignored.

7.4.6 A (ALPHANUMERIC) Conversion

Form: Aw

(1) Input: The A-conversion reads a list containing any allowable FORTRAN Characters. On the 810A, a word is filled with two 8-bit characters. On the 840A, a word is filled with four 6-bit characters. The input data is left-justified in the word, and the remainder of the word is filled with blanks. (4 corrections).

(2) Output: If the I/O list element specifies an entire array (i.e., an array name without subscript) the characters specified are stored continuously starting at the first word of the array and upward until the entire alphanumeric string has been stored. When separate array elements are

specified, either by the use of an implied DO-loop or directly by specifying a particular element; each element requires a new FORMAT term.

(3) I/O Example:

```
    DIMENSION A(10)

    INTEGER A

    READ (ID, 10) A

    . . .

    WRITE (OD, 11) A

    STOP

        810A

10  FORMAT (10A2)

11  FORMAT (1HC, 10A2)

        840A

10  FORMAT (10A4)

11  FORMAT (1HC, 10A4)
```

where C is carriage control character 0, 1, +, or blank.

7.4.7 D (Double-precision) Conversion

Form: Dw.d

(1) Input: The basic form of D-conversion is the same as for real conversion, except that the data is stored in two words, instead of one word, for better accuracy.

(2) Output: The form of D-conversion of output is the same as the output of E-conversion except that a character D, replaces the character E in the exponent.

7.4.8 Complex Conversion

Form: Ew.d, Ew.d
Ew.d, Fw.d
Fw.d, Ew.d
Fw.d, Fw.d

Complex data consists of a pair of separate real data, the first of which supplies the real part of the complex number, the second supplies the imaginary portion of the complex number.

7.5 EDITING SPECIFICATIONS

The following specifications are used for editing of input and output. When used with input, they allow the programmer some flexibility in preparing coding sheets. For output, they allow the labeling of the results, and also allow for the arrangement of the output quantities.

7.5.1 H (Hollerith) Conversion

Form: wH

(1) Input: This specification allows for the input of any set of characters, including blanks, in the form of comments, headings, and titles. When a Hollerith field is referenced by an input statement, the field is replaced by whatever characters appear in the field of the input record. When the same FORMAT statement is later referenced by an output statement, the new field of characters is transferred to the output record.

Example:

Source Program:

```
     READ (1, 10)

 10  FORMAT (22HXXXXXXXXXXXXXXXXXXXXXX)
```

Input Record:

```
     INPUT RECORD
     ^THIS IS A NEW HEADING
     <------22 columns------>
```

A later call for the same FORMAT statement number: WRITE (2, 10) produces this output record: THIS IS A NEW HEADING.

Note that characters read by a Hollerith specification are used only for input/output. They may not be manipulated in any way.

(2) Output: On output the field width, w, specifies the number of characters to the right of H that are transmitted. The first characters of each line is considered a carriage control character and does not print.

The comma following the H specification is optional.

7.5.2 X (Blank) Conversion

Form: wX

(1) Input: The X specification causes a column of the input record to be skipped.

(2) Output: The X specification in an output field causes w spaces to be inserted in the output record.

1X may be written as X. The comma following the X specification field is optional.

7.5.3 New Record

The slash, (/), signals the end of a record. Successive slashes may be used to skip lines, cards, or tape records.

7.6 nP SCALE FACTOR

To permit more general use of the D-, E-, F-, and G-conversion, a scale factor may be used. The scale factor for input is defined as follows: $10^{-\text{scale factor}}$ x external quantity = internal quantity. The scale factor for output is: external quantity = internal quantity x $10^{\text{scale factor}}$.

For input, the scale factor has an effect only on F-conversion. When using D-, E-, or G-conversion with a scale factor on input, the value of the exponent is modified to compensate for the shift of the decimal point.

When using G-conversion for output, the scale factor is taken into consideration in the formula for determing whether F- or E-conversion is used. If F-conversion is used, the actual value of the number is changed when the decimal point is shifted, as if F-conversion had been originally specified. If E-conversion is used, because the value does not fit into the output field under F-conversion, the scale factor has the same effect as if E-conversion had been specified.

The general form of a scale factor is:

n P E w . d

n P F w . d

n P G w . d

n P D w . d

where n is a signed integer constant.

Examples of F w . d Scaling:

The input quantity 314. 1593 read under the specification 2PF8. 4 would produce the internal value of 314. 1593 x 10^{-2} = 3. 141593.

Output Examples:

Specification	Output Representation
F8.5	3.14159
1PF8.5	31.4159
3PF8.5	3141.59
-1PF8.5	.314159

Examples of Ew.d Scaling

Specification	Output Representation
E15.2	3.14E+00
1PE15.2	31.42E-01
3PE15.2	3141.59E-03
-1PE15.2	.314E+01

When no scale factor is present in a FORMAT statement, the scale factor is assumed to be zero. However, once a scale factor is given in a FORMAT statement, it applies to all following field specifications involving D-, E-, F-, and G-conversion within the same FORMAT statement until a new scale factor is given. To reset the scale factor within a FORMAT statement, 0P must be used.

Examples:

3PF8.4, A2, I5, 3PF5.2

is equivalent to:

3PF8.4, A2, I5, F5.2

If the field specification contains a repeat count, the scale factor precedes the repetition constant.

3P5F8.4, A2, 0PF5.2, 2P3F6.0

7.7 REPEATED FORMAT SPECIFICATIONS

Any FORMAT specification except X, H, and nP may be repeated by using an unsigned non-zero integer constant preceeding the field specification.

FORMAT (I7, I7, I7, F4.1, E6.3, E6.3, E6.3, E6.3, E6.3) is equivalent to:

FORMAT (3I7, F4.1, 5E6.3)

When a group of FORMAT specifications are repeated, as in:

FORMAT (I5, F6.2, I5, F6.2, I3, F5.0, F5.0)
using a repetition constant produces:

FORMAT (2(I5, F6.2), I3, 2F5.0)

7.7.1 Multiple-Record Formats

On input, a slash (/) in a FORMAT statement causes a new record (i.e., card) to be read. If n slashes are used, n-1 records are skipped.

Example:

```
READ (1, 10) A, B, C, D, E, F, G

10 FORMAT (3F8.2/F5.3, 3F6.2)
```

This FORMAT statement reads one card containing three fields of data that are stored in A, B, and C. The remainder of the card is not read; the next card contains four fields of data to be stored in D, E, F, and G.

If the FORMAT statement reads:

```
10 FORMAT (3F8.2//F5.3, 3F6.2)
```

one entire record is skipped and D, E, F and G are read on the following record.

On output, n consecutive slashes in a FORMAT statement cause n-1 blank lines to be written, except when the slashes are unscanned. In that case, an additional blank line is printed.

In a multiple-record FORMAT statement, it is possible to specify that the first record has one format and that all following records have another format by enclosing the last record specification in parentheses.

Example:

```
FORMAT (5I5/ (3F10.2, 3E10.6))
```

When this FORMAT statement is executed, the first record is printed under control of the 5I5 field specification and all subsequent records are printed under control of the other two field specifications, until the output list is satisfied.

7.8 VARIABLE FORMAT

FORMAT lists may be specified at the time of execution. The specification list including left and right parentheses, but not the statement number nor the word FORMAT, is read into a dimensioned array under control of an A (Alphanumeric) field specification. The name of the array containing the specification may then be used in place of a FORMAT statement number.

Example:

Assume the following FORMAT specification:

(F10.2, I3, F6.0, F9.2)

This information is read:

```
    DIMENSION INP(5)
    READ (1,10) (INP(I), I = 1, 5)
10  FORMAT (5A4)
```

This input record places in storage the following elements:

```
INP     : (F10
INP + 1 : .2, I
INP + 2 : 3, F6
INP + 3 : .0, F
INP + 4 : 9.2)
```

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

```
WRITE (2, INP) A, I, B, C
```

This would write A, I, B, C, in the following format:

F10.2, I3, F6.0, F9.2

7.9 CARRIAGE CONTROL

The first character on every line of output is considered a carriage control indicator. Below is a list of carriage control indicators and their significance.

Carriage Control Indicator	Significance
blank	Single space prior to printing current line.
0	Double space prior to printing current line.
1	Space to top of form prior to printing current line.
+	Do not advance form prior to printing current line.

7.10 MANIPULATIVE STATEMENTS

7.10.1 END FILE Statement

> General Form
>
> END FILE i
>
> where:
>
> i is an unsigned integer or integer variable specifying the device code of a peripheral unit.

When addressing a magnetic tape unit, the END FILE statement causes an end-of-file record to be written on the designated tape unit.

If i is an integer variable, it must be assigned a value corresponding to a peripheral unit prior to the execution of this statement.

7.10.2 REWIND Statement

> General Form
>
> REWIND i
>
> where:
>
> i is an unsigned integer variable specifying the device code of a peripheral magnetic tape unit.

This statement is used to rewind to the beginning of the tape mounted on tape unit i. If i is an integer variable, it must be assigned a value corresponding to a peripheral tape unit prior to the execution of this statement.

7.10.3 BACKSPACE Statement

> General Form
>
> BACKSPACE i
>
> where:
>
> i is an unsigned integer or integer variable specifying the device code of a peripheral magnetic tape unit.

This statement causes the tape mounted on the magnetic tape unit i to move backward one logical record. If i is an integer variable, it must be assigned a value corresponding to a peripheral tape unit prior to the execution of this statement.

APPENDIX A

SEL 810A AND 840A CHARACTER CODES AND WORD STRUCTURE

CHARACTER CODES

CHARACTER	CARD	ASR-33	ASR-35	ASCII	IBM/ BCD	CHARACTER	CARD	ASR-33	ASR-35	ASCII	IBM/ BCD
0	0	260	060	60	12	A	12-1	301	101	01	61
1	1	261	261	61	01	B	12-2	302	102	02	62
2	2	262	262	62	02	C	12-3	303	303	03	63
3	3	263	063	63	03	D	12-4	304	104	04	64
4	4	264	264	64	04	E	12-5	305	305	05	65
5	5	265	065	65	05	F	12-6	306	306	06	66
6	6	266	066	66	06	G	12-7	307	107	07	67
7	7	267	267	67	07	H	12-8	310	110	10	70
8	8	270	270	70	10	I	12-9	311	311	11	71
9	9	271	271	71	11	J	11-1	312	312	12	41
BLANK	BLANK	240	240	40	20	K	11-2	313	113	13	42
=	1-3	275	275	75	13	L	11-3	314	314	14	43
'	8-4	247	047	47	14	M	11-4	315	115	15	44
+	12	253	053	53	60	N	11-5	316	116	16	45
.	12-8-3	256	056	56	73	O	11-6	317	317	17	46
)	12-8-4	251	251	51	74	P	11-7	320	120	20	47
–	11	255	055	55	40	Q	11-8	321	321	21	50
$	11-8-3	244	044	44	53	R	11-9	322	322	22	51
*	11-8-4	252	252	52	54	S	0-2	323	123	23	22
,	0-8-3	254	254	54	33	T	0-3	324	324	24	23
(	0-8-4	250	050	50	34	U	0-4	325	125	25	24
/	0-1	257	257	57	21	V	0-5	326	126	26	25
						W	0-6	327	327	27	26
						X	0-7	330	330	30	27
						Y	0-8	331	131	31	30
						Z	0-9	332	132	32	31

WORD STRUCTURE

840A FLOATING POINT QUANTITIES

SINGLE-PRECISION FLOATING POINT DATA

S	2^{-1} FRACTION 2^{-23}
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

WORD 1

0	2^{-24} FRACTION 2^{-37}	S	2^{7} EXPONENT 2^{0}

WORD 2

DOUBLE-PRECISION FLOATING POINT DATA

S	2^{-1} FRACTION 2^{-23}
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

WORD 1

0	2^{-24} FRACTION 2^{-37}	S	2^{7} EXPONENT 2^{0}
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14	15	16 17 18 19 20 21 22 23

WORD 2

0	2^{-38} 2^{-60}
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

WORD 3

INTEGER QUANTITIES

INTEGER DATA

S	2^{22} 2^{0}
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

DOUBLE-PRECISION FIXED POINT DATA

S	2^{45} 2^{23}
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

WORD 1

0	2^{22} 2^{0}
0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

WORD 2

WORD STRUCTURE

810A FLOATING POINT QUANTITIES

SINGLE-PRECISION FLOATING POINT DATA

S	2^{-1} FRACTION 2^{-15}			WORD 1
0	2^{-16} FRACTION 2^{-21}	S	2^{7} EXPONENT 2^{0}	WORD 2
0	1 6	7	8 15	

COMPLEX FLOATING POINT DATA

S	2^{-1} FRACTION 2^{-15}			WORD 1
0	2^{-16} FRACTION 2^{-21}	S	2^{7} EXPONENT 2^{0}	WORD 2
0	1 6	7	8 15	

(REAL PART)

S	2^{-1} FRACTION 2^{-15}			WORD 3
0	2^{-16} FRACTION 2^{-21}	S	2^{7} EXPONENT 2^{0}	WORD 4
0	1		15	

(IMAGINARY PART)

DOUBLE-PRECISION FLOATING POINT DATA

S	2^{-1} FRACTION 2^{-15}			WORD 1
0	2^{-16} 2^{-21}	S	2^{7} EXPONENT 2^{0}	WORD 2
0	2^{-22} FRACTION 2^{-36}			WORD 3
0	1 6	7	8 15	

INTEGER QUANTITIES

INTEGER DATA

S	2^{14}	2^{0}
0	1	15

LOGICAL

1	UNUSED	
0		15

APPENDIX B

STATEMENT INDEX

SPECIFICATION STATEMENTS		PAGE
COMPLEX List	N *	3-1
DOUBLE PRECISION List	N	3-1
INTEGER List	N	3-1
LOGICAL List	N	3-1
REAL List	N	3-1
STORAGE ALLOCATIONS		
DIMENSION $v_1, v_2, v_3, \ldots$	N	3-2
COMMON / Ii / List	N	3-3
EQUIVALENCE (a, b, c, ..), (d, e, f, ..)	N	3-4
DATA LIST / a, b, c, ... /, $List_2$ / d, e, f, ..) a	N	3-6
SUBPROGRAM STATEMENTS		
Entry Points		
SUBROUTINE Name	N	6-6
SUBROUTINE Name ($p_1, p_2, \ldots$)	N	6-6
FUNCTION Name ($p_1, p_2, \ldots$)	N	6-3
REAL FUNCTION Name ($p_1, p_2, \ldots$)	N	6-3
INTEGER FUNCTION Name ($p_1, p_2, \ldots$)	N	6-3
COMPLEX FUNCTION Name ($p_1, p_2, \ldots$)	N	6-3
LOGICAL FUNCTION Name ($p_1, p_2, \ldots$)	N	6-3
DOUBLE PRECISION FUNCTION Name ($p_1, p_2, \ldots$)	N	6-3
Inter-Subroutine		
EXTERNAL $Name_1$, $Name_2$	N	3-1

*N = Non-executable
E = Executable

			PAGE
Transfer Statements			
CALL Name		E	6-6
CALL Name (p_1, p_2, ..)		E	6-6
RETURN		E	5-6
ARITHMETIC STATEMENT FUNCTION			
Function (p_1, p_2, ... p_n) = EXPRESSION		E	6-1
SYMBOL MANIPULATION, CONTROL			
Replacement			
V = E	Arithmetic	E	4-1
	Logical/Relational	E	4-3
	Multiple	E	4-2
Inter-program	GO TO n	E	5-1
Transfers	GO TO n, (m_1, m_2, ... m_n)	E	5-2
	GO TO (m_1, m_2, ... m_n) n	E	5-1
	IF (A) n_1, n_2, n_3	E	5-3
	IF (L) s	E	5-3
MISCELLANEOUS PROGRAM CONTROL			
ASSIGN I TO N		E	5-2
CALL SLITE (I)		E	5-6
CALL SLITET (I, K)		E	5-7
CALL SSWTCH (I, K)		E	5-7
CALL OVERFL (J)		E	5-7
CONTINUE		E	5-5
PAUSE ; PAUSE n		E	5-5
STOP ; STOP n		E	5-6
DO n i = m_1, m_2, m_3		E	5-4

I/O STATEMENTS		PAGE
FORMAT ($spec_1$, $spec_2$, ...)	N	7-2
READ (n, F) LIST	E	7-2
READ (n, F)	E	7-2
READ (n) LIST	E	7-2
READ (n)	E	7-2
WRITE (N, F) LIST	E	7-2
WRITE (N, F)	E	7-2
WRITE (n)	E	7-2
I/O Tape Handling		
END FILE I	E	7-14
REWIND I	E	7-14
BACKSPACE I	E	7-14
TERMINATION		
END	N/E	5-5

APPENDIX C

LIBRARY FUNCTIONS

FORM	MODE OF		DEFINITION
	ARGUMENT	RESULT	
ABS (X) CABS (C) DABS (D) IABS (I)	REAL INTEGER DOUBLE COMPLEX	REAL INTEGER DOUBLE COMPLEX	Absolute Value |ARG|
AIMAG (C)	COMPLEX	REAL	Obtain imaginary part of complex number
AINT (X) INT (X) IDINT (D)	REAL REAL DOUBLE	REAL INT INT	Truncation Sign of ARG Times Largest integer $\leq$ |ARG|
ALOG (X) DLOG (D) CLOG (C)	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX	Natural Log LOG_e (ARG)
ALOGIO (X) DLOGIO (D)	REAL DOUBLE	REAL DOUBLE	Common Log LOG_e (ARG)
AMAXO ($I_1, I_2, \ldots$) AMAX1 ($X_1, X_2, \ldots$) MAXO ($I_1, I_2, \ldots$) MAX1 ($X_1, X_2, \ldots$) DMAX1 ($D_1, D_2, \ldots$)	INTEGER REAL INTEGER REAL DOUBLE	REAL REAL INTEGER INTEGER DOUBLE	Determine Maximum Argument
AMINO ($I_1, I_2, \ldots$) AMIN1 ($X_1, X_2, \ldots$) MINO ($I_1, I_2, \ldots$) MIN1 ($X_1, X_2, \ldots$) PNIN1 ($D_1, D_2, \ldots$)	INTEGER REAL INTEGER REAL DOUBLE	REAL REAL INTEGER INTEGER DOUBLE	Determine Minimum Argument
AMOD (X_1, X_2) MOD (I_1, I_2)	REAL INTEGER	REAL INTEGER	Remaindering ARG_1 - (ARG_1/ARG_2) ARG_2, where (X) is the integral part of X.
ATAN (X) DATAN (D) ATAN2 (X_1, X_2) DATAN2 (D_1, D_2)	REAL DOUBLE REAL DOUBLE	REAL DOUBLE REAL DOUBLE	Arctangent (ARG) Arctangent (ARG1/ARG2)
CMPLX (X_1, X_2)	REAL	COMPLEX	Form complex number $X_1 + X_2 \sqrt{-1}$

FORM	MODE OF		DEFINITION
	ARGUMENT	RESULT	
COS (X) DCOS (D) CCOS (C)	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX	Cosine (ARG) (Radians)
CONJG (C)	COMPLEX	COMPLEX	Complex Conjugate for ARG = X + iY C = X - iY
DBLE (X)	REAL	DOUBLE	Convert Single Precision Argument to Double
DIM (X_1, X_2) IDIM (I_1, I_2)	REAL INTEGER	REAL INTEGER	Positive Difference ARG_1 - MIN (ARG_1, ARG_2)
EXP (X) DEXP (D) CEXP (C)	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX	Exponential e^{ARG}
FLOAT	INTEGER	REAL	Convert Integer to REAL
IFIX (X)	REAL	INTEGER	Convert REAL to Integer
REAL (C)	COMPLEX	REAL	Obtain REAL part of complex number
SIGN (X_1, X_2) ISIGN (I_1, I_2) DSIGN (D_1, D_2)	REAL INTEGER DOUBLE	REAL INTEGER DOUBLE	Transfer of sign Sign of ARG_2 times $\lvert ARG_1 \rvert$
SIN (X) DSIN (D) CSIN (C)	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX	Sine (ARG) Radians
SNGL (D)	DOUBLE	REAL	Double to REAL Conversion
SQRT (X) DSQRT (D) CSQRT (C)	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX	Square Root $ARG^{1/2}$
TANH (X)	REAL	REAL	Hyperbolic Tangent

APPENDIX D

TRACE

There are two types of TRACE statements available. The first is used for tracing only selected variables, and the second is used for tracing all variables within a specified area.

NOTE: If sense switch 4 is on, no TRACE is executed.

1. Item Tracing

The TRACE statement used for item tracing specifies a list of variables and/or array names. The format is:

TRACE X_1, X_2, X_3, ... X_n

where X is any variable or array name. Whenever any of these variables or array elements becomes redefined by an arithmetic expression, coding is inserted into the object program by the TRACE routine which causes a line of TRACE information to be typed. A description of the output format appears in paragraph (4). A TRACE statement of this type may be placed anywhere in the source program. As many TRACE statements as desired may be included in the program.

2. Area Tracing

The TRACE statement used for area tracing specifies a single statement number. The format is:

TRACE n

where n is any statement number not yet defined. This type of TRACE statement inserts coding into the object program which causes the results of all arithmetic expressions (including IF statements) that follow the TRACE statement inclusive to statement n, to output a line of TRACE information as described in paragraph 4. This group of statements is called the TRACE Range. In addition to tracing all arithmetic and IF statements within the TRACE Range, all statement numbers within this range are also output as a line of TRACE information.

An area TRACE statement should not be placed within the TRACE Range of another area statement unless all such TRACE statements refer to the same statement number.

3. Unconditional TRACE

If a TRACE of the entire source program is required, the format:

TRACE 99999

traces all arithmetic statements, IF statements, and statement numbers at run time.

4. TRACE Listing Format

At execution time of the object program, any TRACE coding inserted by the compiler causes a line to be typed consisting of a variable name, an array name, or a statement number, followed by an equal sign, followed by the current decimal value just assigned to that name. The decimal value is typed in either integer, floating point or complex format. Array names are followed by a subscript indicating the element within the array first modified, as if it were a single dimensioned variable. (For converting double and triple dimensions to single. See Section 3.4.)

5. Sample TRACE Program

```
      DIMENSION A (3, 3)
      TRACE Y, A
      X = 3.24
      Y = X + 1.5
      Z = Y**2
      DO 48 I = 1, 3
      A(I, 2) = Y/2.0
48    Y = Y + 1.0
      X = 0.0
      K = 2
      TRACE 62
50    X = X + 1.0
      IF (X - 3.0) 51, 53, 53
51    K = K*K
      GO TO 50
53    IF (X.LE.Y) X = X + 100.0
63    X = X - 1.0
      Z = 2.0*X
      Y = 0.0
      .
      .
      .
      .
```

The output generated by this program would appear as:

```
Y       = 60.4740000000E601
A       (4) = 60.2370000000E601
X       = 60.5740000000E601
A       (5) = 60.2870000000E601
Y       = 60.6740000000E601
A       (6) = 60.3370000000E601
Y       = 60.7740000000E601
(50)
X       = 60.1000000000E601
(IF)    = -0.2000000000E601
(51)
K       = ←6b→4
(50)
X       = 60.2000000000E601
(IF)    = -0.1000000000E601
(51)
K
(50)
X       = 60.3000000000E601
(IF)    = 60.0000000000E600
(53)
(IF)    = ←6b →1
X       = 60.1030000000E603
(62)    =
X       = 60.1020000000E603
Y       = 60.0000000000E600
```

APPENDIX E

CHAINING

The CHAINING feature of SEL 810A/840A FORTRAN IV allows a FORTRAN object program that is too large to fit into the available memory space to be divided into segments. Each segment is run separately and inter-segment communication of data is done through COMMON storage.

Control is transferred from link-to-link by means of the statement "CALL CHAIN", which is the last executable statement of each link.

All blank or labeled COMMON areas used for communication between segments of the chain must be declared with a COMMON statement at the beginning of each segment. The declaration order and size of each area must agree in each chain segment.

Chain Program Example

```
C       LINK NO. 1

        COMMON A, B

        WRITE (4, 1)

        1 FORMAT (15H THIS IS LINK 1)

        A = 2.*B

        CALL CHAIN

        END

$

C       LINK NO. 2

        COMMON X, Y

        WRITE (4, 1)

        1 FORMAT (15H THIS IS LINK 2)

        Y = 2.*X

        CALL CHAIN

        END
```

```
$
C      LINK NO. 3
       COMMON E, F
       WRITE (4, 1)
       1 FORMAT (15H THIS IS LINK 3)
       E = 2.*F
       STOP
       END
```

APPENDIX F

OPERATOR COMMUNICATIONS

FORTRAN IV Diagnostics for 840A

More than 50 different error diagnostics can be indicated. They will appear on the line following a FORTRAN statement in which an error has occurred, for example; ..... ERROR DETECTED AT COLUMN.

The following list contains the different diagnostic codes and their meaning.

CODE		ROUTINE	MEANING
ADDR			ILLEGAL ADDRESS CONSTRUCTION
ADJD		SC01	ILLEGAL ADJUSTABLE DIMENSIONS
AMOD			ILLEGAL MODE FOR ADDRESS (MUST BE INTEGER)
ASOV	*	AS03	ASSIGNMENT TABLE OVERFLOW
ASTO		X301	ASSIGN TO SPELLING ERROR
BLKD		W500	NO CODE GENERATED BY A BLOCK DATA PROGRAM
CERR	*	A100	CHARACTER NO A C/R
CICD			CANNOT INITIALIZE COMMON DATA
COMM		NM01	ERRONEOUS COMMON USAGE
CRET	*	TH02	C/R WITHIN HOLLERITH STRING
DDST		NR01	DOUBLY DEFINED STATEMENT
DPFL	*	B500	DATA POOL FULL
DPOF			DATA POOL OVERFLOW
DUMM		ND01	ERRONEOUS DUMMY USAGE
EQCN		C310	ERRONEOUS EQUIVALENCE CONSTRUCTION
EQIV		C309	IMPOSSIBLE EQUIVALENCE GROUP
EQMS	*	C901	DO EQUALS (=) IS MISSING
ERDO	*	C900	ILLEGAL DO-TYPE STATEMENT
ERTN	*	R903	RETURN STATEMENT IN MAIN PROGRAM
EXS=	*	EX66	NOT FIRST EQUALS, OR EQUALS WITH PARENTHESIS, OR EQUALS NOT ALLOWED
FUNV		W501	FUNCTION NAME NEVER ASSIGNED
FRST	*	R205	NOT FIRST STATEMENT OF PROGRAM
FWAR	*	R203	FUNCTION HAS NO ARGUMENTS
HOLL			ILLEGAL HOLLERITH STRING

CODE		ROUTINE	MEANING
IDOL	*	V516	IMPROPER IMPLIED DO LOOP
IF(2	*	V307	IF (ITEM HAS OVER 6 CHARACTERS
ILBD	*	R301	ILLEGAL BLOCK DATA STATEMENT
ILEG	*	A900	NOT LEGAL FORTRAN STATEMENT
ILIF			ILLEGAL LOGICAL IF CONSTRUCTION
ILSN	*	IS04	ILLEGAL STATEMENT NUMBER
INDT			DATA CONSTRUCTION ERROR
IUSE		NU00	INCORRECT USAGE
LDOP	*	EX79	IMPROPER LEADING OPERATOR
MODE	*	OMZ5	MODE MIXING ERROR
MULT		NP02	MULTIPLE DEFINED ITEM
NAME			CONSTANT ILLEGALLY USED
NARR		AT00	ITEM NOT AN ARRAY
NCBS		C315	NEGATIVE COMMON BASE
NEXT	*	C604	IMPROPER DO NEST
NINT		IT00	ITEM NOT AN INTEGER
NNAM		NCOO	ILLEGAL USE OF CONSTANT
NOIM			OPERAND MISSING
NOIT			MUST HAVE INTEGER TYPE
NPTH		V219	NO FORMAT STATEMENT NUMBER
NOC			ILLEGAL USE OF SUBROUTINE OR ARRAY NAME
OPER	*	EX25	UNACCEPTABLE OPERATOR
OPOS	*	EX60	OPERATOR NOT ALLOWED AT THIS POSITION
PATH		NP06	PATH CANNOT EXECUTE THIS STATEMENT
RLOP	*	EX70	TWO RELATIONAL OPERATORS IN A ROW
SBIG	*	DN57	DIGIT STRING TOO LARGE
SBSC		IL01	WRONG NUMBER OF SUBSCRIPTS
SPEC	*	NP00	STATEMENT CLASS OUT OF ORDER
SPEL		A903	FORTRAN STATEMENT MISSPELLED
STNO	*	C702	STATEMENT NO. CONSTRUCTION
TAG			ILLEGAL INDEX CONSTRUCTION
TYPE	*	A304	IMPROPER USE OF TYPE STATEMENT
TMDT			TOO MUCH DATA

CODE		ROUTINE	MEANING
V/SP		NS01	ILLEGAL USE OF SUBPROGRAM NAME
XARG			EXCESSIVE NUMBER OF ARGUMENTS
)ERR	*	TS01	CHARACTER NOT A)
(ERR	*	TS02	CHARACTER NO A (
/ERR	*	TS03	CHARACTER NOT A /
, ERR	*	TS04	CHARACTER NOT A,

*IRRECOVERABLE ERROR. ENTIRE RECORD IS IGNORED.

APPENDIX G

IN-LINE CODING

To provide additional flexibility in the SEL 810A and 840A FORTRAN systems, MNEMBLER code may be interspersed with FORTRAN statements. Permissable operation codes are listed at the end of this Appendix.

The location field is either blank or may contain only a FORTRAN statement number. The statement number may appear anywhere within columns 1 - 5.

The operation code appears in columns 7 - 9.

The variable field may consist of:

(1) FORTRAN names - variables, array names, function names.

(2) FORTRAN constants - (interpreted as literal constants).

(3) FORTRAN statement numbers preceded by the character ")".

At least one blank character must separate the operation code field and the variable field. If column 10 is an asterisk, indirect addressing is indicated.

The index field may use only constants of one, two, or three and is separated from the address portion of the instruction by a comma.

NOTE: If a dummy variable or array is addressed, the indirect bit in the instruction is set by using an MEA instruction, (note, if also column 10 contains an asterisk (*), the MEA command negates the indirect bit).

The following operation codes are the only allowable assembly language.

NOTE: Instructions with an asterisk cannot be done on the 810A.

ARITHMETIC:

AMA (05)	-	ADD (M) TO (A) RESULT IN A
SMA (06)	-	SUBTRACT (M) FROM (A)
AAM (31)	-	ADD (A) TO (M) RESULT IN M
MPY (07)	-	MULTIPLY
DIV (10)	-	DIVIDE

LOAD:

LAA (01)	-	(M) TO A
LBA (02)	-	(M) TO B

STORE:

STA (03)	-	(A) TO M
STB (04)	-	(B) TO M

LOGICAL:

*MAA (27)	-	(M) AND (A)
*MEA (26)	-	(M) EXCLUSIVE OR (A)
*MOA (30)	-	(M) OR (A)

BRANCH:

BRU (11)	-	BRANCH TO M
*BAZ (22)	-	BRANCH IF (A) = 0
*BAP (24)	-	BRANCH IF (A) POSITIVE
*BAN (23)	-	BRANCH IF (A) NEGATIVE
SPB (12)	-	STORE PLACE AND BRANCH

INDEX:

*LIX (32)	-	(M) TO X