UNIVAC 1050 SYSTEMS

MAGNETIC TAPE SYSTEM

DESIGN SPECIFICATIONS

FORTRAN

LANGUAGE SPECIFICATIONS

UP 3940.21

This document is provisional in nature and is intended as a vehicle for meeting immediate needs with regard to system familiarization and orientation. UNIVAC ® Division of Sperry Rand Corporation reserves the right to change and/or modify such information contained herein as may be required by subsequent system developments.

PRINTED IN U.S.A.

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 GENERAL

FORTRAN (FORmula TRANslator) is a problem oriented programming language which provides a precise statement form for mathematical, scientific and data processing applications. The FORTRAN compiler interprets the FORTRAN source language and produces executable machine language object programs.

1.2 THE FORTRAN PROGRAM

A FORTRAN program consists of statements to handle communication with external units, description and definition of data, and manipulation and computation of data. Also included in the FORTRAN system are procedures for including subprograms such as functions and subroutines. The FORTRAN statements available will be discussed in the following sequence:

a. Specification Statements
b. Assignment Statements
c. Control Statements
d. Input/Output Statements
e. Functions, Subprograms and Subroutines
f. DATA Statement and BLOCK DATA Subprogram

1.3 WRITING THE PROGRAM

A FORTRAN program may be written using the following characters:

a. Alphabetics	A through Z
b. Numerics	0 through 9
c. Special Characters	+ Plus
	- Minus
	* Asterisk
	/ Slash
	= Equal
	(Left Parenthesis
	) Right Parenthesis
	, Comma
	. Decimal Point
d. Blank Space	

The alphabetics must be written as capitals. Blank spaces may be used for clarity but they are ignored by the FORTRAN processor, except when used in a Hollerith constant (See Section 2.1.5).

The FORTRAN Coding form is shown on page 1-3.

Each FORTRAN statement is written on a line in columns 7-72. If the statement exceeds one line, any non-blank character in column 6 designates continuation.

Statement numbers may be written in columns 1-5 to establish reference points within the program. It is not necessary to number every statement. In general, statement numbers are used to establish transfer of control. Explanatory comments may be incorporated within a FORTRAN program by placing a "C" in column 1 and the comment in 2-80.

Columns 73-80 may be used for card number, program identification, etc; they are not processed but will appear on the printed listing.

1.4 STATEMENT ORDER

The user must maintain the following statement sequence when writing a FORTRAN program to insure successful compilation.

SUBROUTINE or FUNCTION Statements
EXTERNAL Statement
TYPE Statements
DIMENSION Statements
COMMON Statements
EQUIVALENCE Statements
BODY Statements
END Statement

UNIVAC
DIVISION OF SPERRY RAND CORPORATION

FORTRAN

PROGRAMMING FORM

PROGRAM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____ PAGES

"C" FOR COMMENT

STATEMENT NUMBER 1–5	Cont. 6	FORTRAN STATEMENT 7–72	80–90

UP-2508 Rev. 1

(80–90 COLUMN FORM)

2.0 FORTRAN CONSTANTS, VARIABLES AND ARRAYS

2.1 CONSTANTS

A FORTRAN constant is a quantity whose written representation is fixed and invariable. A constant defines both value and data type. There are five types of constants permissible within the UNIVAC 1050 FORTRAN language: integer, real, double precision, logical, and Hollerith. An integer, real, and double precision constant is considered to be signed when immediately preceded by a plus or minus.

2.1.1 Integer Constants

Integer constants are formed by strings of from one to five significant digits with optional leading zeros. They may be defined as having positive, negative, or zero integral values. For example:

```
    0
   12
99999
00055555
```

Computation or input operations may not result in a value larger than 99999.

2.1.2 Real Constants

Real constants are defined as positive or negative decimal values ranging in magnitude between 10^{-50} and 10^{+49}. A real constant consists of an integer part, a decimal point, a fractional part, and optionally an exponent. Either the integer part or the fractional part may be blank, but not both. A UNIVAC 1050 FORTRAN real constant may not exceed 8 digits in length (excluding the decimal point and exponentiation). Any real constant exceeding eight digits will be truncated from the right. Examples of real constants are:

```
   0.1
   1.2
   0.95
123.45678
```

A decimal exponent may be appended to a real or integer constant by using the letter "E" followed by an optionally signed one or two digit integer constant. This exponent is interpreted as a scale factor of ten raised to the power specified by the integer after E and applied to the constant. For example:

7.6E-5(i.e.7.6 x 10^{-5})
12.5E25(i.e.12.5 x 10^{+25})

2.1.3 Double-Precision Constants

Double-precision constants are defined as positive or negative decimal values ranging in magnitude between 10^{-50} and 10^{+49} with a maximum of 14 digits of precision for the integer and fractional parts. The letter "D" followed by an optionally signed one or two digit integer constant must be appended to all double-precision constants. The interpretation of "D" is the same as for the "E" exponent of a real constant. Examples of double-precision constants are:

```
0.0D0             (i.e. 0.0 x 10^0)
0.095D10          (i.e. .095 x 10^10)
3.14159265359D0   (i.e. 3.14159265359 x 10^0)
```

2.1.4 Logical Constants

There are only two logical constants, either .TRUE. or .FALSE. representing the truth values true and false respectively.

2.1.5 Hollerith Constants

Hollerith constants are strings of alphanumeric data written in form

n H c

where

- n is the number of characters in the constant.
- H is the letter indicating a Hollerith constant.
- c is the Hollerith constant consisting of any symbol combination capable of representation in the processor. A blank is a valid and significant character in a Hollerith constant.

Examples of Hollerith constants are:

```
1H+
10HNET PROFIT
5HSTOP.
```

2.2 VARIABLES

A FORTRAN variable is a quantity identified by a programmer-established name and for which the value and type (real, integer, double-precision, etc.) are defined by the programmer. Variables may be redefined by the programmer or program, but the values assumed are always of the same type once type is defined. All FORTRAN variables (integer, real, double precision or logical) are symbolic names of from one to six alphanumeric characters, where the first character must be alphabetic.

2.2.1 Integer Variables

In FORTRAN, integer variables may be specified in two ways:

a) By a variable name beginning with the alphabetic characters I, J, K, L, M, or N.

b) By an INTEGER type statement which overrides any alphabetic in the first character of the variable name. See Section 4.1.2.

Examples of integer variables are:

```
INTEGER  A, B, REAL (which defines A, B, and REAL as
         integer variables)
J2
MONEY
```

2.2.2 Real Variables

Real variables may be represented in FORTRAN as follows:

a) By a variable name beginning with any alphabetic character except I, J, K, L, M, and N.

b) By a REAL Type statement which overrides any alphabetic in the first character of the variable name. See Section 4.1.2.

Examples of real variables are:

```
REAL  INPUT, I1, I2 (which defines INPUT, I1, and I2 as real
                     variables)
AVGE
PROFIT
```

2.2.3 Double-Precision Variables

Double-precision variables must be specified by the DOUBLE PRECISION Type statement. For example,

```
DOUBLE PRECISION I1, A, AVGE
```

See Section 4.1.2.

2.3 ARRAYS

An array is an ordered set of data (integer, real, double precision or logical) of one, two or three dimensions. It is identified by a programmer-established variable name. Each element of the array is referenced by appending a subscript to the variable. A subscript is a list enclosed in parentheses immediately following the array name. The number of expressions in a subscript must correspond to the declared dimensionality of the array. Subscript expressions may be written in any one of the following forms:

```
c*v+k  (i.e. c x v+k)
c*v-k  (i.e. c x v-k)
c*v    (i.e. c x v)
v+k
v-k
v
k
```

where c and k are positive integer constants and v is a previously defined integer variable. For example,

```
INPUT (I, J, K)
A (10)
REST (2, I+1)
NET  (5*NET1 +3)
THETA (7*BETA4 -6)
```

3.0 FORTRAN EXPRESSIONS

3.1 GENERAL

This section presents the formulation rules for FORTRAN arithmetic, relational and logical expressions. The simpler type of expression is defined as any single term (i.e. constant, variable or function). Compound expressions are formed from combinations of terms and operators. The general form of a FORTRAN compound expression is:

Term Operator Term
or
Term Operator Term Operator . . . . . Operator Term

3.2 ARITHMETIC EXPRESSIONS

3.2.1 Definition

An arithmetic expression is any single arithmetic term or combination of arithmetic terms and arithmetic operators. The permissible arithmetic terms are:

a. Integer, real, and double-precision constants

b. Integer, real, and double-precision variables and array elements

c. Integer, real, and double-precision function references

The FORTRAN arithmetic operators are:

Operator	Represents		Example
+	Addition	A+B =	The value of A plus the value of B
-	Subtraction	A-B =	The value of A minus the value of B
*	Multiplication	A*B =	The value of A multiplied by the value of B
/	Division	A/B =	The value of A divided by the value of B
**	Exponentiation	A**B =	The value of A raised to the power B

Using the arithmetic operators, a term may be combined with another term of the same type, yielding a result of the same type. Also, a real term may be combined with a double-precision term producing a double-precision result. Examples of integer and real arithmetic expressions follow:

$$Z - SIN(X) / COS(Y) **2.0 \quad 1 + I/J$$

Note: When a fractional part occurs from the division of two integers, it is truncated leaving an integer quotient.

For example:

$$10/4 = 2.5$$

this is truncated to 2.

3.2.2 Formulation Rules

When writing compound arithmetic expressions, the following formulation rules must be observed.

a. No two operators may appear contiguously. If a plus or minus sign is used to specify a positive or negative term and not addition or subtraction, it must be denoted as such by using parentheses and not written contiguous to the preceding operator. For example, the following are invalid expressions:

```
     A ** - B
X1 + Y2 / - C**Z
```

which must be written as follows to be correct:

```
     A ** (-B)
X1 + Y2 / (-C)**Z
```

b. All elements of an arithmetic expression must be of the same type, either integer or real, with the following two exceptions:

(1) In a real expression, real elements may be exponentiated to an integer power. For example,

```
(A + B)**J
```

is valid.

(2) The arguments of an arithmetic function may differ in type from the other elements in the expression, as long as the function result is of the same type as the rest of the expression. For example:

```
SUM/FLOAT (N)
```

is a valid real expression.

3.2.3 Evaluation Rules

a. An arithmetic expression is evaluated according to an established priority of operators and in cases of equal operator priorities from left to right. The established priorities of the arithmetic operators are:

Operator	Priority
**	1 (highest)
*/	2
+-	3 (lowest)

For example, the FORTRAN evaluation of the expression

```
4 + 6 * 2**3
```

according to priority is as follows:

1. 2**3 = 8
2. 6*8 = 48
3. 4+48 = 52

b. Sets or levels of parentheses must be used in order to override the evaluation of operator priorities defined above. Parentheses are always evaluated first from the innermost out. For example the FORTRAN expression for the formula

$$\left(\frac{a - b}{a + b}\right)^{c}$$

is

```
( (A - B) / (A + B) ) **C
```

and would be evaluated as follows:

```
(1)  A - B
(2)  A + B
(3)  /
(4)  **C
```

3.3 RELATIONAL EXPRESSIONS

3.3.1 Definition

A relational expression consists of two arithmetic expressions separated by a relational operator. Its general form is:

Arithmetic expression .Relational operator. Arithmetic expression

The two arithmetic expressions must conform to the following:

a. both must be integer
b. both must be real
c. both must be double precision
d. one may be real and the other double precision

The relational operators are:

Operator	Represents
.LT.	less than
.LE.	less than or equal to
.EQ.	equal to
.NE.	not equal to
.GT.	greater than
.GE.	greater than or equal to

3.3.2 Evaluation Rules

a. The value of a relational expression is either .TRUE. or .FALSE. depending on whether the designated relation is true or false. For example:

```
A.LE.1.0    If A less than or equal to 1.0, the value is .TRUE.
            If A greater than 1.0, the value is .FALSE.
```

b. When evaluating relational expressions, all arithmetic operators are of higher priority than the relational operators. Therefore, the values of the two arithmetic expressions are determined before the relational comparison takes place. All of the relational operators are of the same priority.

c. If both a real and double precision expression appears in a relational expression, evaluation occurs as if a double precision zero were to the right of the operator and the difference between the two terms were on the left. For example:

```
A.EQ.D
```

where A is real and D is double precision would be evaluated as if it were written

```
(A)-(D).EQ. 0.0D0
```

Note that the second term specified (D) is subtracted from the first (A).

3.4 LOGICAL EXPRESSIONS

3.4.1 Definition

A logical expression consists of logical elements combined with logical operators. The general form is

Logical expression .Logical operator. Logical expression

The permissible logical elements are:

a. logical constants
b. logical variables or logical array elements
c. logical functions
d. relational expressions

The logical operators are:

Operator	Represents	Example	
.AND.	logical conjunction	A.AND.B	If A and B are both .TRUE., the value of the expression is .TRUE. Otherwise, it has the value .FALSE.
.OR.	logical disjunction	A.OR.B	If either A or B or both have the value .TRUE., the value of the expression is .TRUE. Otherwise, it has the value of .FALSE.
.NOT.	logical negation	.NOT.A	If A has the value .FALSE., the expression has the value .TRUE. If A has the value .TRUE., the expression has the value .FALSE.

3.4.2 Evaluation Rules

In any evaluation of expressions, arithmetic operators are of highest priority, relational operators are of a lower priority, and the logical operators are of the lowest priority. The priorities of the logical operators are:

.NOT.	1
.AND.	2
.OR.	3

For example, the logical expression

```
.NOT.A+B.EQ.1.0.AND..TRUE.
```

would be evaluated as follows:

a. arithmetic expression A+B
b. relational expression .EQ.1.0
c. logical expressions; first .NOT. (highest priority), then .AND..TRUE.

Assuming A+B is equal to 1.0, the value of the above expression is .FALSE.

4.0 FORTRAN STATEMENTS

4.1 SPECIFICATION STATEMENTS

The Specification Statements are used to define the type, quantity and arrangement of the data to be processed. There are five types of Specification Statements.

a. EXTERNAL Statements
b. Type Statements
c. DIMENSION Statements
d. COMMON Statements
e. EQUIVALENCE Statements

These non-executable statements must precede all executable program statements and, when used, must be specified in the order given above.

4.1.1 EXTERNAL Statements

The EXTERNAL Statement is used to declare subroutine and function names as external procedures. It must be used to declare as external procedures all subroutine and/or function names which are used as arguments to other external procedures in CALL Statements.

The general form of the EXTERNAL Statement is:

```
EXTERNAL     name list
```

where **"name list"** contains the subroutine and/or function names. The type of a function, i.e., if it is integer, real, double precision or logical, is established by the Type Statement in which it appears or by alphabetic convention. For example, the real function TCATE is declared as an external procedure by

```
EXTERNAL TCATE
```

4.1.2 Type Statements

The Type Statements are used to declare the types of variables, arrays, non basic external functions, and statement functions as INTEGER, REAL (single precision floating point), DOUBLE PRECISION (double precision floating point), or LOGICAL. (The types of the basic external functions and the intrinsic functions are as specified in section 4.5.2.)

In the absence of a Type Statement for a variable, array, or function, the name will be considered of type INTEGER if its first character is an I, J, K, L, M or N and otherwise it will be considered to be of type REAL.

The general form of the Type Statements is

```
INTEGER              name list
REAL                 name list
DOUBLE PRECISION     name list
LOGICAL              name list
```

where **"name list"** may contain variable names, array names (with or without its dimensions declared), and/or function names each separated by a comma. It should be noted that if a variable, array or function name appears in a Type statement, it must precede all other references to the name, (with the exception of references in SUBROUTINE and/or FUNCTION and EXTERNAL statements) and also that a variable, array or function name may only appear in one Type statement.

Examples of Type statements are:

```
REAL INPUT(20),MEAN,KILL
DOUBLE PRECISION I,A,B1
```

4.1.3 DIMENSION Statements

The DIMENSION Statement is used to define the maximum size of each array in a FORTRAN program. The general form of the DIMENSION statement is:

```
DIMENSION variable (subscript list), variable (subscript list),...
```

where "variable" may be of integer, real, double precision or logical type. "Subscript list" may be of 1, 2, or 3 integer constants separated by commas specifying a one, two or three dimensional array. When specifying a one-dimensional array, the integer subscript may not exceed 5 digits. When specifying a two or three dimensional array, the product of the integer subscripts may not exceed 5 digits. Zero (0) or negative values may never appear as array subscripts.

Example:

```
DIMENSION I(5),A(10,10),ARRAY(3,3,3)
```

If a variable has already been dimensioned in a Type statement, it is not necessary to restate it in a DIMENSION statement.

4.1.4 COMMON Statements

The COMMON Statement is used to economize on data storage among <u>separately</u> compiled portions of a FORTRAN program allowing different variables to share storage locations during the execution of a program.

The general form of a COMMON statement is:

```
COMMON/Block name/variable list/Block name/variable list/...
```

where "Block name", the symbolic name given to the block, is from 1 to 6 alphanumeric characters the first of which is alphabetic. "Variable-list" is a list of variable names and/or array names (with or without dimension declarators) whose values are to be placed in the common block named. Each entry in the variable list must be separated by commas. Arrays may be DIMENSIONed in a COMMON statement; however, if this is done, they must not have been DIMENSIONed previously.

For compatability with earlier FORTRAN compilers, "Blank Common" will be provided though its use is discouraged. "Blank Common" may be specified in either of the following forms:

1) COMMON variable list (i.e., no block-name)

2) COMMON // variable list (i.e., two consecutive slashes)

Labeled common blocks may be added at the end of either of the above "Blank Common" statements or be specified ahead of the two consecutive slashes in the second form.

The rules for assigning storage to labeled and blank common blocks within a FORTRAN program and among separately compiled FORTRAN programs are:

a. Within any single FORTRAN program, **symbolic common block names** or blank common may occur more than once. The FORTRAN processor will extend the named common blocks or Blank common to include the variables and/or arrays in the order of their appearance.

b. The size of a common block in a single FORTRAN program is the sum of the storage units required for each element introduced through COMMON and EQUIVALENCE. (EQUIVALENCE statements may be used to lengthen common blocks, labeled or blank, only beyond the last assignment for that block made directly by common statements.)

The storage units are:

No. of Units	Type of Element
1	Logical
5	Integer
10	Real
16	Double Precision

c. In separately compiled FORTRAN programs, common blocks of the same name must be of the same size. The sizes of blank common do not have to be the same in separately compiled FORTRAN programs.

Examples:

```
COMMON/A/FOUR(4),TEACH,PLAY(5),/B/INCH,JAR
COMMON BETA,GAMMA,OMEGA
COMMON ALIPH,BET,/GIMEL/DALID,HEY
```

4.1.5 EQUIVALENCE Statements

The EQUIVALENCE statement is used to permit sharing of data storage within separately compiled FORTRAN programs or subprograms. The general form of an EQUIVALENCE statement is:

```
EQUIVALENCE (variable list), (variable list)...
```

where "variable list" contains the variables, arrays and/or array elements whose values are to share the same storage (or part of the same storage). The names in the variable list must be separated by commas and each pair of parentheses enclosing the variable lists must be separated by commas.

It is recommended that only variables of the same mode be equivalenced. However, if it becomes necessary to EQUIVALENCE variables or arrays of different modes, the variables or array elements requiring less storage space will share space starting with the first character of the variable requiring greater storage space.

Arrays or a particular element in an array may be used in the variable list. If an array name is used without referring to a specific numbered element of that array, the element treated will be the first member of that array. If one element of an array is equivalenced to one element of another array, the entire arrays are equivalenced.

Examples:

```
EQUIVALENCE (A,B,C (2,5)),(INK,JACK,MINE)
EQUIVALENCE (INDIAN(10),SQUAW(5),CHIEF(2))
```

4.2 ASSIGNMENT STATEMENTS

There are three types of assignment statements:

Arithmetic assignment statement
Logical assignment statement
Control assignment statement

4.2.1 Arithmetic Assignment Statement

The general form of the arithmetic assignment statement is:

Arithmetic variable=Arithmetic expression

The arithmetic assignment statement causes the evaluation of the "arithmetic expression" on the right and the replacement of the "arithmetic variable" on the left with the expression result. Note that in FORTRAN the meaning of the equal sign (=) is replacement and that in an arithmetic assignment statement the left hand side must always be a single variable or array element. The rules governing the expression evaluation and resultant conversion to the mode of the arithmetic variable are as follows:

Rules For Arithmetic Statement Evaluation

Type of "Arithmetic-variable"	Type of "Arithmetic-expression"	Assignment Rule
Integer	Integer	Assign expression result without change
Integer	Real	Truncate expression result to integer and assign
Integer	Double Precision	Truncate expression result to integer and assign
Real	Integer	Convert expression result to real
Real	Real	Assign expression result without change
Real	Double Precision	Evaluate expression in DP and assign most significant part
Double Precision	Integer	Convert expression result to double precision and assign
Double Precision	Real	Evaluate expression extending to double precision and assign
Double Precision	Double Precision	Assign expression result without change

Examples:

```
I = A * B
A = K(J)-I
D = SQRT(B**2-(4.0*A/C))
```

4.2.2 Logical Assignment Statement

The logical assignment statement whose general form is

Logical variable=Logical expression

causes the evaluation of the "Logical expression" on the right and the replacement of the "Logical variable" on the left with the expression result. The left hand side of a logical statement must be a single logical variable name or logical array element. The value of the "logical expression" which is assigned to the "logical variable" must be either .TRUE. or .FALSE.

Examples:

```
FLAG= RED.AND.WHITE.AND.BLUE
BEST= (GOOD.LT.BETTER.LT.BEST).OR.NONE
```

4.2.3 Control Assignment Statement

The general form of the control assignment statement is:

ASSIGN Statement Number TO Integer variable

The ASSIGN statement is used only for the assignment of a "statement number" to a non-subscripted "integer variable" which is later used by an assigned GO TO statement. (See Section 4.3.1.b). Execution of an ASSIGN statement sets the value of the "integer variable" to the designated

"statement number". The subsequent execution of any assigned GO TO statement using that "integer variable" causes control to be transferred to the executable statement labeled by the assigned "statement number". For example,

```
          ASSIGN 10 TO J
          .
          .
    8     GO TO J
          .
          .
          .
   10     Z=(X+Y)**Z
```

causes the statement number "10" to be assigned to variable J and the assigned GO TO is executed as GO TO 10 transferring control to the arithmetic statement labeled 10.

Note: In an ASSIGN statement the "statement-number" must refer to an executable statement in the same FORTRAN program and the "integer-variable" may not be referenced in any statement other than an assigned GO TO until it has been redefined.

4.3 CONTROL STATEMENTS

The Control Statements are used to alter the sequential order in which FORTRAN statements are normally executed. There are eight types of Control Statements:

a. GO TO statements
b. Arithmetic IF statement
c. Logical IF statement
d. DO statement
e. CONTINUE statement
f. CALL statement
g. RETURN statement
h. Program control statements

The statement labels used within control statements must refer to executable statements within the same FORTRAN program.

4.3.1 GO TO Statements

All of the GO TO statements are used to transfer control to a statement other than the next in sequence. There are three types of GO TO statements:

Unconditional GO TO statement
Assigned GO TO statement
Computed GO TO statement

a. Unconditional GO TO Statement

This statement is used to unconditionally transfer control to a designated statement. The form of the unconditional GO TO statement is:

GO TO Statement number

Execution of an unconditional GO TO statement causes the statement indicated by the "statement number" to be executed next.

Example:

```
6 A= (B+C)/(B-C)
  GO TO 8
7 A= (B-C)/(B+C)
8 X= SQRT(A)
```

The "GO TO 8" in the above example transfers control to the arithmetic statement labeled "8" bypassing the arithmetic statement labeled "7".

b. Assigned GO TO Statement

The assigned GO TO statement is used to transfer control conditionally based upon the ASSIGNED value of a variable to a designated statement. An assigned GO TO statement is of the form:

```
GO TO integer variable, (statement number list)
```

The "integer variable" may not be subscripted. A "statement number list" may be present and must be enclosed in parentheses. Each list element is separated by a comma.

At the time of execution of an assigned GO TO statement, the value of the "integer variable" must have been assigned by the execution of an ASSIGN statement. The "statement number list" provides a visual reference for the programmer. (See Section 4.2.3.) Execution of the assigned GO TO causes control to be transferred to the statement identified by the assigned statement number.

Example:

```
3  ASSIGN 8 TO J
   .
   .
   .
4  GO TO J, (6,8,10)
```

In the example statement 3 assigns the statement number 8 to variable J and statement 4 transfers control to statement 8 the same as if it had been written:

```
4 GO TO 8
```

c. Computed GO TO Statement

The computed GO TO statement is used to transfer control to one of several possible statements depending upon the value of a control variable. The general form of a computed GO TO statement is:

```
GO TO (Statement number list), integer variable
```

The "integer variable" must be non-subscripted and may assume only positive values. The "statement number list" specifies the statement numbers to which control may be transferred depending upon the value of the "integer variable" and must be enclosed in parentheses and followed by a comma. The statement numbers in the list must be separated by commas.

The value of the "integer variable" determines the statement number in the "statement number list" to which control will be transferred by a computed GO TO statement. If the "integer variable" has the value 1, the computed GO TO will transfer control to the statement whose number is first in the statement number list"; if the "integer variable" has the value 2, control will be transferred to the statement whose number appears second in the "statement number list", etc.

For example,

```
GO TO (17,19,21),J.
```

transfers control to statement 17 when J=1, statement 19 when J=2 and statement 21 when J=3.

Note: The maximum value that the "integer variable" may assume must be the same as the number of statement numbers in the "statement number list".

4.3.2 Arithmetic IF Statement

The arithmetic IF statement transfers control to a designated statement depending upon the value of an arithmetic expression. An arithmetic IF statement is of the form:

IF (arithmetic expression) n_1,n_2,n_3

where "n_1,n_2,n_3" are statement numbers which must be separated by commas. The "arithmetic expression" which may be of type integer, real, or double precision must be enclosed in parentheses.

Execution of an arithmetic IF statement causes the "arithmetic expression" to be evaluated. If the expression value is less than zero, control is transferred to statement "n_1", if it is equal to zero, control is transferred to statement "n_2", and if it is greater than zero, control is transferred to "n_3".

For example,

```
IF (A-B) 10,10,20
```

If the value of A-B is less than or equal to zero, statement 10 is executed next. If the value of A-B is greater than zero, statement 20 is executed next.

4.3.3 Logical IF Statement

The general form of the logical IF statement is:

IF (logical expression) FORTRAN statement

The "logical expression" must be enclosed in parentheses. Any executable "FORTRAN statement" except a DO statement or another logical IF statement may follow the parentheses.

Execution of a logical IF statement causes the "logical expression" to be evaluated. If the expression value is .FALSE., the remainder of the IF statement is skipped and control proceeds to the next statement in sequence. If the expression value is .TRUE., the "FORTRAN statement" is executed. For example,

```
IF (A.AND.B) Z=1.0
```

If both A and B have the value .TRUE., the expression is .TRUE. and Z is set to 1.0. If both A and B do not have the value .TRUE. the expression is .FALSE. and control proceeds to the next statement in sequence.

4.3.4 DO Statement

The DO statement is used to execute a defined number of times the set of statements which are in the defined range of a DO statement. In other words, a DO statement is used to define an iterative loop within a FORTRAN program. The two general forms of a DO statement are:

DO statement number Integer-variable = m_1,m_2,m_3

or

DO statement number Integer-variable = m_1,m_2

where

a. The "statement-number" is the label of the executable statement which terminates the DO statement. Execution begins with the next sequential statement following the DO and proceeds up to and including the statement whose "statement-number" appears in the DO statement. This is called the range of the DO. The terminal statement may not be a GO TO of any form, arithmetic IF, DO, RETURN, STOP, PAUSE, END, or a logical IF containing any of these statements. A dummy CONTINUE statement (see Section 4.3.5) should be used instead.

b. The unsubscripted "integer variable" is used as the control variable for the DO-loop. It assumes a new value each time the DO-loop is executed.

c. The parameters "m_1" (the initial parameter), "m_2" (the terminal parameter) and "m_3" (the incrementation parameter) are each either an integer constant or an integer variable. In the second form of the DO statement parameter "m_3" is not stated; this implies a value of 1 for the incrementation parameter. At the time of execution, "m_1", "m_2" and "m_3" must be non-zero and positive.

Example:

The statement

```
DO 12 J = 2,10,2
```

causes the execution of all statements following it up to and including statement 12 in a loop. J will have the value 2 during the first execution of the range, 4 during the second execution of the range, and the values 6, 8 and 10 during the succeeding executions of the range.

The control "integer variable" and the parameters "m_1", "m_2" and "m_3", if they are integer variables, are available for use as variables or in subscripts in statements throughout the range of the DO as long as their values are not altered. The value of the control variable is also available outside of the range of a DO statement if control is transferred outside that range before completion by a GO TO or an arithmetic IF statement.

Example:

The DO-loop

```
      DO 100 I = 1,25
100 A(I) = B(2* I-1)
```

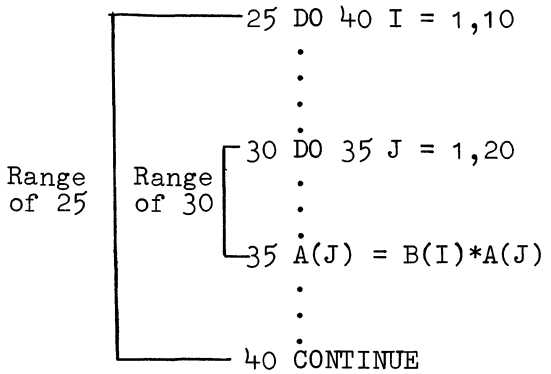
is executed equivalent to the execution of the statements:

```
A(1) = B(1)
A(2) = B(3)
A(3) = B(5)
.
.
.
A(25) = B(49)
```

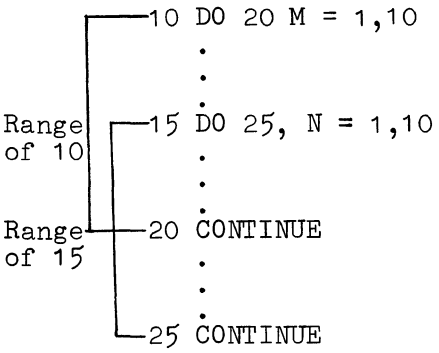
One or more DO-loops may be contained or nested within the range of another DO-loop. The following rules must be observed when nesting DO-loops.

Rule 1: If the range of a DO includes another DO, the range of the inner DO must lie completely within the range of the outer DO.

Example: Permitted by Rule 1



Example: Violation of Rule 1



Rule 2: Transfer of control by IF or GO TO statements into the range of a DO statement from outside its range is not permitted except in the case of an extended DO defined by Rule 3. Such transfers would not allow the control variable to be set up and evaluated properly.

Example:

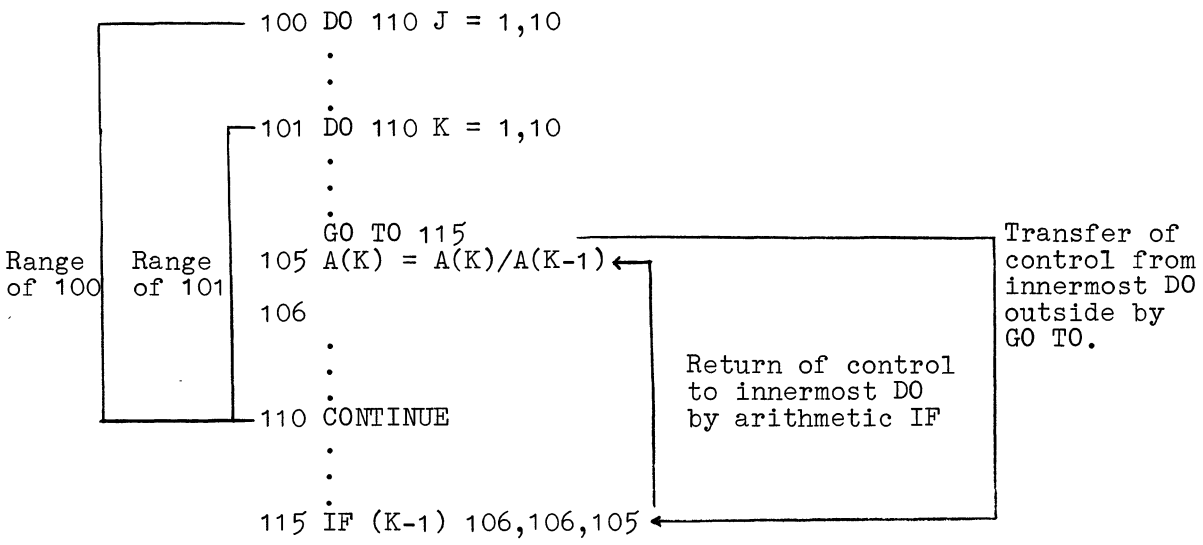
```
                GO TO 10                    Illegal transfer
                                            of control into
        5       DO 20 K = 2,20,21           the range of
                                            a DO
Range   10      Z(K) = Z(K)/X(K-2)
of 5
        20      CONTINUE
```

Rule 3: A DO statement has an extended range only when both of the following conditions exist:

(a) An arithmetic IF or a GO TO statement within the range of the innermost DO of a completely nested nest transfers control outside of the nest.

(b) An arithmetic IF or a GO TO statement outside of the nest described in (a) causes control to be returned into the range of the innermost DO of the completely nested nest.

Example of an extended DO:



It should be noted that the extended range of a DO may not contain another DO that has an extended range.

4.3.5 CONTINUE Statement

The CONTINUE statement is a dummy statement which is used as the terminating statement of a DO, when the DO-loop would end on a GO TO, arithmetic IF, DO, RETURN, STOP, PAUSE, END, or logical IF containing any of the preceding. The form of the CONTINUE statement is:

```
CONTINUE
```

Execution of a CONTINUE simply continues the normal execution sequence.

Example:

```
      DO 113 I = 1,100
      .
      .
      .
  111 IF (A(I).EQ.0.0)GO TO 13
  112 PROD = PROD * A(I)
  113 CONTINUE
```

4.3.6 CALL Statement

The CALL statement is used to reference any designated SUBROUTINE subprogram. The general form of a CALL statement is:

```
CALL Subroutine name (Argument list)
```

The "argument list" containing the actual arguments separated by commas must be enclosed in parentheses. See Section 4.5.2 for the description of and rules for the actual arguments to a SUBROUTINE subprogram. If there are no arguments, the "argument list" and its enclosing parentheses are omitted.

Execution of a CALL statement transfers control to the named subroutine which performs its operation upon the input arguments. Upon completion the subroutine returns control to the sequential statement following the CALL statement.

Examples of CALL statements are:

```
CALL INVERT (A,B,C,D)
CALL TYPE (10HSINE ERROR)
CALL PSIGN (Y,Z,M,N).
```

4.3.7 RETURN Statement

The RETURN statement is used to logically end a procedure subprogram and, therefore, it may be used only in a procedure subprogram. A RETURN statement is of the form:

```
RETURN
```

Execution of a RETURN statement in a SUBROUTINE subprogram causes control to be returned to the statement directly following the subroutine CALL statement.

In a FUNCTION subprogram, the execution of a RETURN statement causes control to be returned to the statement which contained the function reference and the current value of the function is made available to that statement.

Example

```
SUBROUTINE SWAP (A,B)
TS=A
A=B
B=TS
RETURN
```

4.3.8 Program Control Statements

The three program control statements are:

```
PAUSE
STOP
END
```

a. PAUSE Statement

The two general forms of a PAUSE statement are:

```
PAUSE Integer constant
```

or

```
PAUSE
```

Execution of the PAUSE statement causes execution of the object program to be halted and "PAUSE iiiii" to be logged on the printer. If the "integer constant" is not specified, only the word "PAUSE" will be printed. Upon manual restart by the computer operator, the program resumes at the sequential statement following the PAUSE statement.

b. STOP Statement

The STOP statement is used to terminate the execution of the object program. The STOP statement should be used at the logical end rather than at the physical end of the program. The two general forms of the STOP statement are:

```
STOP Integer constant
```

or

```
STOP
```

Execution of the STOP statement causes immediate termination of the object program, "STOP iiiii" (or just "STOP") to be displayed on the PRINTER, and control to be transferred to the executive routine.

c. END Statement

The END statement is used to specify the physical end of the program and indicates the completion of a compilation to the FORTRAN processor. The form of the END statement is:

```
END
```

The END statement must be used as the last physical statement of every FORTRAN compilation (i.e. main-program or subprogram). Both the STOP and END statements will terminate the program in the same manner. Therefore, if the physical end of a program is also the logical end, only an END statement need be used and the STOP statement may be omitted.

4.4 INPUT/OUTPUT STATEMENTS

4.4.1 General

Input/Output Statements control and direct the transmission of data between input and output peripheral units and the Central Processor. UNIVAC 1050 input data may be supplied either on punched cards via the Card Reader or on magnetic tape; output data may be produced either on punched cards via the Card Punch Unit, in printed form via the Printer or on magnetic tape.

There are three types of input/output statements:

Statements that cause transfer of records of sequential files to and from internal storage.

Statements that provide for positioning and demarcation of external files.

Data Specification statement.

The INPUT/OUTPUT Statements are:

```
FORMAT
READ (Unit) List
READ (Unit,Format) List
WRITE (Unit) List
WRITE (Unit,Format) List
ENDFILE unit
REWIND unit
BACKSPACE unit
```

In order to conform with previous versions of the FORTRAN language, the following additional Input/Output statements are available:

```
PRINT List
PRINT Format, List
PUNCH List
PUNCH Format, List
WRITE OUTPUT TAPE Unit, List
WRITE OUTPUT TAPE Unit, Format, List
WRITE TAPE Unit, List
READ List
READ Format, List
READ INPUT TAPE Unit, List
READ INPUT TAPE Unit, Format, List
READ TAPE Unit, List
```

4.4.2 FORMAT Statement

FORMAT Statements are used in conjunction with the input/output statements to provide editing information between the internal representation and the external character strings.

The FORMAT Statement is of the form

Statement number FORMAT $(q_1 t_1 z_1 q_2 t_2 z_2 \ldots q_n t_n z_n)$

where (1) each q is a series of slashes or empty specification.

(2) each t is a field descriptor or group of field descriptors.

(3) each z is a field separator.

The field descriptors are of the forms

srFw.d
srEw.d
srGw.d
srDw.d
rIw
rLw
rAw
$nHh_1h_2...h_n$
nX
Ow

where:

1) The letters F, E, G, D, I, L, A, H, O and X indicate the manner of conversion and editing between the internal and external representations.

2) w and n are nonzero integer constants representing the width of the field in the external character string.

3) d is an integer constant representing the number of digits in the fractional part of the external character string (except for G conversion code).

4) r, the repeat count, is an optional nonzero integer constant indicating the number of times to repeat the succeeding basic field descriptor.

5) s is optional and represents a scale factor designator.

6) each h is one of the characters that can be represented on the UNIVAC 1050.

For all descriptors, the field width must be specified and the number of digits in the fractional part must be specified where applicable. The width must be greater than or equal to the fractional part.

a. Numeric Fields

The six numeric field descriptors I, E, F, G, D and O are used to specify input/output of integer, real, double precision and octal data.

Symbol	Editing Action Internal		External	Editing Code
I	Integer variable	to/from	decimal integer	Iw
E	Real variable	to/from	floating point decimal number	Ew.d
F	Real variable	to/from	fixed point decimal number	Fw.d
G	Real variable	to/from	generalized E or F	Gw.d
D	Double precision variable	to/from	double precision floating point decimal number	Dw.d
O	Binary number	to/from	octal integer	Ow

With all numeric input conversions, leading blanks are not significant and other blanks are zero. Plus signs may be omitted. With the F, E, G, D input conversions, a decimal point appearing in the input field overrides the decimal point specification supplied by the field descriptor. With all output conversions, the output field is right justified. The number of characters produced by an output conversion must not exceed the field width.

Editing codes for successive fields must be separated by commas. If a group of editing codes is to be repeated, the group can be placed inside parentheses and the number of times the group is to be repeated preceding it.

Example:

The FORMAT statement

```
5 FORMAT (2I3,2(I2,F4.2),E8.3,06)
```

might cause the output

```
123 Δ45 Δ1 9.87 27 3.45 .237EΔ04 054321
 I3 I3  I2 F4.2 I2 F4.2   E8.3     06
```

b. Alphanumeric Fields

UNIVAC 1050 FORTRAN provides two means for reading or writing alphanumeric data. The control symbols are A and H. The codes using these symbols are Aw and nH, where w and n are unsigned integer constants.

Both the A and H type editing codes transmit to the input or output medium the literal representation of Hollerith characters. The Hollerith characters in the A type, however, are specified via a transmission list in an input or output statement, while the Hollerith characters in the H type immediately follows the editing code in the FORMAT specification itself.

A Hollerith type editing code need not be followed by a comma and may be contiguous with another editing code.

Example:

The FORMAT statement

```
20 FORMAT (8HOVERFLOW)
```

will cause the output

```
OVERFLOW
```

c. Skipped Fields

The editing control symbol X provides for ignoring characters of input or insertion of space characters in output. The editing code using this symbol is nX, where n denotes the number of characters to be ignored on input or the number of spaces to be produced in output.

X need not be followed by a comma and may be contiguous with another editing code.

Example:

The FORMAT

```
555 FORMAT (4HRATE3X4HTIME3X8HDISTANCE)
```

might produce

```
RATEΔΔΔTIMEΔΔΔDISTANCE
```

d. Record Termination

If a group of editing codes begin or end with a slash (/), the current record is terminated and the next record is set to begin construction. Entire records may be ignored on input or blank (skipped) records may be produced on output by writing consecutive slashes.

Example:

```
556 FORMAT(4HRATE/4HTIME/8HDISTANCE)
```

might produce

RATE
TIME
DISTANCE

e. Scale Factor Usage

A scale factor designator is defined for use with the F, E, G, or D conversion and is of the form nP where n, the scale factor, is an integer constant or minus followed by an integer constant.

The scale factor affects the conversions in the following manner:

1) For F, E, G and D input conversions (provided no exponent exists in the external field) and F output conversions. External value = Internal value X $10^{\pm n}$

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

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

Example:

Using the following data

-.8743E+2 -.00777E0 .006736E+2

the statement FORMAT (1P3F12.3)

could cause to be printed

ΔΔΔΔ-874.300ΔΔΔΔΔΔ-0.078ΔΔΔΔΔΔΔ6.736

f. Logical Conversion

The logical field descriptor Lw indicates that the external field occupies w positions as a string of information. The external input field must consist of optional blanks followed by a T if true and any other non-blank character if false. Any characters following are ignored. The external output field consists of w-1 blanks followed by a T or F.

4.4.3 READ Statement

The READ Statement activates the input process, allowing the programmer to introduce data into the computer from an input medium. Depending upon the type of data to be processed, there may or may not be an associated FORMAT statement. For this reason, there are two general forms of the READ statement:

```
READ (Unit,Format)List
READ (Unit)List
```

where:

Unit — is the unit designation for the input medium utilized.

Format — is the designation of a FORMAT editing code specification if a Formatted control statement is employed for edited data-either a statement number of a FORMAT statement or an array name specifying an object-time introduced Format Specification.

List — is the list of elements defining the data to be transmitted and the order of transmission. (See section 4.4.6).

Example:

```
READ(4,100)A,B,I
```

The unedited READ statement will accept input according to the variable types in the list; that is,

Type	Input
Integer	5 characters
Real	10 characters
Double precision	16 characters
Logical	1 character

For both real and double precision variables, it is assumed that the value is already in excess 50, where the first two digits are the characteristic and the remaining are the mantissa.

Example:

The following READ statement

```
READ(4)A,I
```

would expect as input

511000000000001

for the real value and the integer value of 1, respectively.

To conform with previous versions of the FORTRAN language, the following statements are acceptable:

```
READ List
READ Format, List
READ INPUT TAPE unit, List
READ INPUT TAPE unit, Format, List
READ TAPE unit, List
```

where unit is an unsigned integer constant specifying the logical tape transport number from which reading is to take place. Any of these statements are acceptable alternates to the above READ statements.

4.4.4 WRITE Statement

The WRITE Statement activates the output process, controlling and directing the transmission of data from computer to peripheral unit. The WRITE statement has two general forms:

```
WRITE(Unit,Format)List
WRITE(Unit)List
```

where Unit is the unit designation for the output medium utilized, and Format and List is defined as in the READ statement.

Unedited WRITE statements output according to the same rules defined for unedited READ statements (See Section 4.4.3).

To conform with previous versions of the FORTRAN language, the following statements are acceptable:

```
PRINT List
PRINT Format,List
PUNCH List
PUNCH Format,List
WRITE OUTPUT TAPE Unit,List
WRITE OUTPUT TAPE Unit,Format,List
WRITE TAPE Unit,List
```

4.4.5 Magnetic Tape-Positioning Statements

There are three magnetic tape-positioning statements available in the UNIVAC 1050 FORTRAN language. They are:

REWIND Unit
BACKSPACE Unit
ENDFILE Unit

The execution of a REWIND statement causes the tape mounted on a UNISERVO specified to be rewound. The BACKSPACE statement causes the magnetic tape mounted on the specified UNISERVO to be positioned at the beginning of the logical record last written or read. The ENDFILE statement causes an end-of-file indication to be written on the tape mounted on the specified UNISERVO.

4.4.6 Input/Output Lists

The input list specifies the names of the variables and array elements to which values are assigned on input. The output list specifies the references to variables and array elements whose values are transmitted. Input and output lists are of the same form.

A list is a simple list, a simple list enclosed in parentheses, a DO-implied list, or two or more lists separated by commas. A simple list contains one or more variable names, array element names or array names, separated by commas. A variable name or array element name specifies itself. An array name specifies all of the array element names defined by the array declarator.

A DO-implied list is a list followed by a comma and a DO-implied specification, all enclosed in parentheses. A DO-implied specification is of one of the forms:

$i = m_1,m_2,m_3$ or
$i = m_1,m_2$

where the elements i, m_1,m_2,m_3 are as defined for the DO statement.

The elements of a list are specified for each cycle of the implied DO.

Example:

```
WRITE(6,100)I,A,((TWO(I,J),I=1,10),J=1,5),NEXT
```

4.5 FUNCTIONS, SUBPROGRAMS, AND SUBROUTINES

4.5.1 General

A subprogram is a set of FORTRAN procedures which may be used a number of times, either within the same program or in different programs. By designating a set of these procedures as either a FUNCTION or SUBROUTINE subprogram, a complete set of procedures may be activated by a single referencing statement and does not have to be written out each time.

Many mathematical routines are maintained in the form of a library of subroutines. These subroutines are available to the programmer whenever he may desire to utilize them in his FORTRAN program.

4.5.2 Functions

If the value of one quantity is dependent upon the value of another quantity or values of other quantities, then it is said to be a function of the other(s). The first quantity is called the function and the other quantity(ies) the argument(s).

The general form of a function reference in FORTRAN is

Function-name (Argument list)

where the actual arguments in the "argument list" must be separated by commas and the entire list must be enclosed in parentheses. A function reference evaluates the named function for the actual arguments specified and returns a single value to the function name at the point of reference.

There are two types of functions available in the FORTRAN language

External functions

Built-in functions

a. External Functions

There are two kinds of external functions: basic external functions which are available on the library and non-basic external functions which are written by the programmer as FUNCTION Subprograms (see Section 4.5.3.b).

An external function is referenced by using its reference in an arithmetic or logical expression in exactly the same manner as one would use a variable.

The actual arguments which constitute its "argument list" must agree in order, number and type with the corresponding dummy arguments in the defining FUNCTION subprogram or with the arguments given in the table of basic external functions below. An actual argument in an external function reference may be one of the following

1) A variable name
2) An array name
3) An array element
4) Any other expression
5) External procedure name

The basic external functions available in the FORTRAN library are:

FORTRAN Name	Mathematical Function	Arguments No.	Arguments Type
SIN	Trigonometric Sine	1	Real
COS	Trigonometric Cosine	1	Real
TAN	Trigonometric Tangent	1	Real
ASIN	Trigonometric Arcsine	1	Real
ACOS	Trigonometric Arccosine	1	Real
ATAN	Trigonometric Arctangent	1	Real
SINH	Hyperbolic Sine	1	Real
COSH	Hyperbolic Cosine	1	Real
TANH	Hyperbolic Tangent	1	Real
EXP	Exponential (e^X)	1	Real
EXP10	Exponential (10^X)	1	Real
ALOG	Natural Logarithm (LOGX)	1	Real
ALOG10	Common Logarithm (LOG10X)	1	Real
SQRT	Square Root	1	Real
CBRT	Cube Root	1	Real
ATANZ	Arctangent of ratio (a_1/a_2)	2	Real
DSIN	Double Precision Sine	1	D.P
DCOS	Double Precision Cosine	1	D.P
DTAN	Double Precision Tangent	1	D.P
DASIN	Double Precision Arcsine	1	D.P
DACOS	Double Precision Arccosine	1	D.P
DATAN	Double Precision Arctangent	1	D.P
DEXP	Double Precision Exponential (e^X)	1	D.P
DEXP10	Double Precision Exponential (10^X)	1	D.P
DLOG	Double Precision Natural Logarithm	1	D.P
DLOG10	Double Precision Common Logarithm	1	D.P
DSQRT	Double Precision Square Root	1	D.P
DCBRT	Double Precision Cube Root	1	D.P
DATAN2	Double precision arctangent of ratio (a_1/a_2)	2	D.P

b. Intrinsic Functions

The symbolic names of the intrinsic functions are predefined and have a special meaning and type. The actual arguments of these functions, which constitute the argument list, must agree in type, number, and order with the specification in the Table following and may be any expression of the specified type.

Execution of an intrinsic function reference results in the evaluation of the named function for the actual arguments specified, returning the single value result to the point of reference.

The form of an intrinsic function reference is

$$f(a_1,.....a_n)$$

where f is the name of the intrinsic function and a_i are the actual arguments. a_i may be any arithmetic expression.

INTRINSIC FUNCTIONS

FORTRAN NAME	NO. OF ARGS.	FUNCTION	MODE OF ARGUMENT	MODE OF FUNCTION
ABS IABS DABS	1	Determine the absolute value of argument	Real Integer Double	Real Integer Double
AINT INT IDINT	1	Truncation. Eliminate fractional portion of argument	Real Real Double	Real Integer Integer
AMOD MOD DMOD	2	a_1(mod a_2). Subtract from the 1st argument the appropriate integer multiple of the 2nd argument so that the remainder is less than the 2nd argument but non-negative	Real Integer Double	Real Integer Double
AMAX0 AMAX1 MAX0 MAX1 DMAX1	≥ 2	Select the largest value	Integer Real Integer Real Double	Real Real Integer Integer Double
AMIN0 AMIN1 MIN0 MIN1 DMIN1	≥ 2	Select the smallest value	Integer Real Integer Real Double	Real Real Integer Integer Double
FLOAT	1	Convert argument from integer to real	Integer	Real
FIX	1	Convert argument from real to integer	Real	Integer
SIGN ISIGN DSIGN	2	Replace the algebraic sign of the 1st argument by the sign of the 2nd argument	Real Integer Double	Real Integer Double
DIM IDIM	2	Positive difference. Subtract the smaller of the 2 arguments from the 1st argument	Real Integer	Real Integer
SNGL	1	Obtain the most significant part of a double precision argument	Double	Real
DBLE	1	Extend a single precision floating point argument to double precision	Real	Double

4.5.3 Subprograms

a. General

Within the UNIVAC 1050 FORTRAN System, it is possible to define subprograms which may be called upon by other FORTRAN programs. Generally, these subprograms are not commonly enough used to warrant their inclusion in a library. Subprograms must be independently compiled and incorporated into other FORTRAN programs for execution. They are not independently executable.

There are three types of subprograms:

FUNCTION subprograms
SUBROUTINE subprograms
BLOCK DATA subprograms (Section 4.6.2)

b. FUNCTION Subprogram

The general form of the FUNCTION Statement is:

t FUNCTION f $(a_1,a_2,\ldots,a_n)$

where:

(1) t, the type, is either INTEGER, REAL, DOUBLE PRECISION, LOGICAL, or absent in which case the mode of f is determined by the alphabetic naming conventions.

(2) f is the symbolic name of the function.

(3) The a_i are the dummy arguments which may be a variable name, an array name or an external procedure name.

FUNCTION Subprograms are constructed with the following restrictions:

(1) The symbolic name of the function must also appear as a variable name in the defining subprogram. The value of this variable at the time of execution of any RETURN statement in this function subprogram is the value of the function.

(2) The symbolic name of the function must not appear in any nonexecutable statement, except as the symbolic name of the function in the FUNCTION statement.

(3) The symbolic names of the dummy arguments may not appear in a COMMON, EQUIVALENCE or DATA statement.

(4) All formal argument names must occur in at least one executable statement in the subprogram. The actual arguments to a FUNCTION subprogram reference may be either an arithmetic expression, a logical expression, an array name, another external function name or a SUBROUTINE name. If a FUNCTION or SUBROUTINE subprogram name appears as an argument, its name must also appear in an EXTERNAL statement in the referencing program.

(5) The FUNCTION subprogram may contain any statement except BLOCK DATA, SUBROUTINE, or another FUNCTION.

(6) The FUNCTION subprogram must contain at least one RETURN statement.

4.6 SUBROUTINE SUBPROGRAM

The general form of the SUBROUTINE statement is:

SUBROUTINE s $(a_1,a_2,\ldots,a_n)$

where:

s is the symbolic name of the subroutine.
a_i are the dummy arguments which may be a variable name, an array name or an external procedure name.

A SUBROUTINE subprogram may produce many values from one reference whereas a FUNCTION subprogram may only produce one.

SUBROUTINE subprograms are constructed with the following restrictions:

a. The symbolic name of the subroutine must not appear in any statement in this subprogram except as the symbolic name of the subroutine in the SUBROUTINE statement itself.

b. The symbolic names of the dummy arguments may not appear in a COMMON, EQUIVALENCE or DATA statement.

c. The SUBROUTINE subprogram may define or redefine one or more of its arguments so as to effectively return results.

d. The SUBROUTINE subprogram may contain any statement except BLOCK DATA, FUNCTION, or another SUBROUTINE statement.

e. The SUBROUTINE subprogram must contain at least one RETURN statement.

f. A Hollerith string may appear as an actual argument for a subroutine.

4.7 DATA STATEMENTS AND BLOCK DATA SUBPROGRAMS

4.7.1 DATA Statements

The DATA Statement enables the FORTRAN programmer to cause data to be produced internally at the time of initial loading of the object program.

The general form of the DATA statement is:

```
DATA List/Literal list/,List/Literal list/...
```

where List contains names of variables and array elements, and Literal list may consist of any combination of the following forms separated by commas:

a) Integer constants
b) Real constants
c) Alphanumeric characters (i.e. Hollerith constants)
d) Double precision constant
e) Logical constant

It is possible to repeat literals without explicitly rewriting them by indicating the number of times the literal is to be repeated followed by an asterisk.

The DATA statement must be preceded by any COMMON, DIMENSION, EQUIVALENCE or TYPE statements that relate to the variable names in the DATA statement's lists.

Example:

```
DATA(A(I),B(1),I=1,5)/0.0,1.0,8*123.4/
```

The memory locations will appear as follows:

```
A(1)   0.0          B(1)   1.0
A(2) 123.4          B(2) 123.4
A(3) 123.4          B(3) 123.4
A(4) 123.4          B(4) 123.4
A(5) 123.4          B(5) 123.4
```

4.7.2 BLOCK DATA Subprograms

The compilation of data into named COMMON blocks must be done in a separately compiled subprogram known as a BLOCK DATA subprogram. The first statement of this subprogram is a BLOCK DATA statement whose general form is:

BLOCK DATA

The only types of statements that may occur in a BLOCK DATA subprogram are the following:

```
BLOCK DATA
Type statements
DIMENSION
COMMON
DATA
END
```

All of these statements are non-executable statements. All elements appearing in a COMMON block whose name is used in a COMMON statement in a BLOCK DATA subprogram must appear in the COMMON statement in the BLOCK DATA subprogram even though this is the only appearance of such variable names in the BLOCK DATA subprogram.

Example:

```
BLOCK DATA
DIMENSION THEY(4)
COMMON/ALL/YOU,THEY/
DATA(THEY(I),I=1,4)/2.0,1.0,2*3.4/,YOU/5.0/
END
```

APPENDIX A - SPECIAL SUBROUTINE SUBPROGRAMS FOR HANDLING "HARDWARE IF" CONTROL STATEMENTS

In prior versions of FORTRAN, the language provided certain machine oriented control statements enabling control to be altered depending on certain hardware switches and indicators. These "Hardware IF" control statements have been eliminated from the FORTRAN language. UNIVAC 1050 FORTRAN incorporates a set of special SUBROUTINE subprograms whose specifications are given in the table below for simulating the "Hardware IF" control statements.

Subroutine Subprogram	Calling Sequence	Operation
OVERFL	CALL OVERFL(j)	The integer variable "j" will be set to 1 if an overflow condition exists, otherwise "j" will be set to 2. The overflow indicator is set to OFF after execution.
DVCHK	CALL DVCHK(j)	The integer variable "j" will be set to 1 if a divide-check condition exists; otherwise "j" will be set to 2. The divide-check indicator is set to OFF after execution.
SLITE	CALL SLITE(i)	If the value of the integer expression "i" is zero, all sense lights will be set to OFF. If the value of "i" is 1,2,3,etc., the corresponding sense light will be set to ON.
SLITET	CALL SLITET(i,j)	The integer expression "i" is evaluated and the corresponding sense light is tested and set to OFF. If the sense light was ON, the integer variable "j" will be set to 1; otherwise "j" will be set to 2.
SSWTCH	CALL SSWTCH (i,j)	The integer expression "i" is evaluated and the corresponding sense switch is tested. If the sense switch was ON, integer variable "j" will be set to 1; otherwise "j" will be set to 2.
PARITY	CALL PARITY(j)	The integer variable "j" will be set to 1 if an incorrectable error exists in the peripheral I/O operation; otherwise "j" will be set to 2.
EOF	CALL EOF(j)	The integer variable "j" will be set to 1 if an end-of-file record is sensed; otherwise "j" will be set to 2.
EOT	CALLΔOT(j)	The integer variable "j" will be set to 1 if an end of tape condition exists; otherwise "j" will be set to 2.

The last three subroutines listed above pertain to I/O operations and when used they must immediately follow the I/O statement they reference.