File Number 7070-25
Form C28-6170-1

IBM Systems Reference Library

IBM 7070-Series Programming Systems

FORTRAN

This publication is a reference text for the IBM 7070-Series FORTRAN language. It provides information for writing programs in both FORTRAN I and FORTRAN II languages. In addition to describing the basic elements of the FORTRAN languages, this publication discusses the various types of FORTRAN language statements available to users of the 7070-Series FORTRAN. Also detailed is a method that will enable the user to expand the FORTRAN system by including additional routines. Separate sections describe the various limitations imposed on the size of FORTRAN source program statements, and error messages that are produced by the Basic FORTRAN and Full FORTRAN processors.

Several appendixes provide a complete description of the 7070-Series Library subroutines; admissible characters in the FORTRAN source program; method of preparing and punching a FORTRAN source program, including an illustration of a tested routine; and a summary of differences between the Full FORTRAN and Basic FORTRAN languages.

MAJOR REVISION (December, 1963)

This publication is a major revision of the previously published edition, Form C28-6170, and makes that edition obsolete. The revised version incorporates the contents of Technical Newsletter, Form N28-1092, and makes it obsolete. Several additions and corrections have been made throughout the text. This edition should be reviewed in its entirety.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D91, PO Box 390, Poughkeepsie, N. Y.

CONTENTS

PURPOSE OF THE PUBLICATION

This publication is a reference text for writing programs in the FORTRAN language for use on IBM 7070-Series Data Processing System. The FORTRAN language closely resembles the language of mathematics and, therefore, may be used to facilitate the writing of programs that will perform scientific computation and data handling.

THE FORTRAN SYSTEM

The FORTRAN System is the IBM FORmula TRANslating System and consists of two parts: the language and the processor.

Elements of the Language

The following are the basic elements of the FORTRAN language:

1. Constants (such as 27 or 3.14159).
2. Variables (such as X or Y).
3. Subscripted variables (such as x_i or y_i, which are written in FORTRAN as X(I) or Y(I)).
4. Arithmetic statements, which cause mathematical computations, (such as $a = \frac{b}{c}$, which is written in FORTRAN as A = B/C).
5. Mathematical expressions, which are meaningful to FORTRAN, (such as X + Y or 3*J).
6. Control statements, which govern the sequence of operations.
7. Specification statements, which assist the FORTRAN processor in producing the machine-language program.
8. Input/Output statements, which are used to specify and control the transmission of data between the computer and the Input/Output devices.
9. Subroutine statements, which permit the logical subdivision of programs to be incorporated within larger programs. This allows the programmer to write a program without specifying each instruction every time the sequence is to occur.

The FORTRAN source program is written using these elements of the language. The processor converts source programs written in the FORTRAN language into machine-language programs. IBM provides two processors for the IBM 7070-Series Data Processing Systems: the Basic FORTRAN processor and the Full FORTRAN processor.

Basic FORTRAN

The Basic FORTRAN processor does not require any tape units for compilation on a 7070 or 7074 Data Processing System. However, tape units must be provided when the program is to be compiled on an IBM 7072. The Basic FORTRAN processor accepts only a subset of the statements accepted by the Full FORTRAN processor and corresponds to the FORTRAN I language. The output of the Basic FORTRAN processor is a symbolic program in the Basic Autocoder language. In order to obtain a machine-language program, this output must be processed by one of the following: Basic Autocoder, Four-Tape Autocoder, or Autocoder.

Full FORTRAN

The Full FORTRAN processor is a component of the 7070/2/4 Compiler Systems Tape and consists of two major parts. The first part accepts and converts FORTRAN source statements corresponding to the FORTRAN II language into Autocoder element records for the second part. The second part follows the first part without interruption and produces a machine-language program. The Full FORTRAN processor requires at least six magnetic tape units. If multifile compilations are desired, a minimum of eight tape units or seven tape units and an IBM 7550 Card Punch are required.

The main differences between the Full FORTRAN processor and the Basic FORTRAN processor can be stated as follows:

Basic FORTRAN is the FORTRAN I Language
Full FORTRAN is the FORTRAN II Language

Full FORTRAN permits the use of subprograms whereas Basic FORTRAN does not. Full FORTRAN allows for six-character variable names, EQUIVALENCE within COMMON and triple subscripting whereas Basic FORTRAN does not.

PREREQUISITES

The use of this manual presupposes that the programmer has read IBM General Information Manual: FORTRAN, Form F28-8074. Familiarity with the material contained therein is essential to the use of this publication.

Additional Reference Material

A knowledge of the following IBM publications will be helpful in understanding this manual:

IBM 7070-7074 Data Processing Systems, Form A22-7003.
Glossary for Information Processing, Form C20-8089.
Operator's Guide, IBM 7070 Series Programming Systems: Compiler Systems, Form C28-6249.

This section describes in detail the basic elements that comprise the FORTRAN language.

CONSTANTS

Two types of constants are permissible in a FORTRAN source program -- fixed-point (restricted to integers), and floating-point (characterized by being written with a decimal point). The rules for writing each of these constants are given below.

GENERAL FORM	EXAMPLES
Fixed-Point Constants A fixed-point constant is written without a decimal point, using from 1 to 10 digits. The constant may have a preceding + or - sign. An unsigned constant is assumed to be positive. The magnitude or absolute value of the constant must be less than 10^{10}.	3 +1 -28987 0
Floating-Point Constants A floating-point constant is written with a decimal point at the beginning, or the end or between two digits. The constant may have a preceding + or - sign. An unsigned constant is assumed to be positive.	.17 5. -0.0003
An integer exponent preceded by an E may follow a floating-point constant. The exponent may have a preceding + or - sign (an unsigned exponent is assumed to be positive). The magnitude of a number must be expressed as follows: $1.0 \times 10^{-51} \leq x < 1.0 \times 10^{49}$, or the magnitude must be equal to zero.	5.0E+3 means (5.0×10^3) 5.0E3 means (5.0×10^3) 5.0E15 means (5.0×10^{15}) 5.0E-17 means (5.0×10^{-17})

VARIABLES

A FORTRAN variable is a symbolic representation of a value that may change either for different executions of the program or at different stages within the program. As with constants, a variable may be fixed-point or floating-point, depending on whether the value(s) that is represents are fixed-point or floating-point, respectively. To distinguish between fixed-point and floating-point variables, the following rules for naming each type of variable must be used.

GENERAL FORM	EXAMPLES
Naming Fixed-Point Variables The name of a fixed-point variable consists of a series of alphabetic and numerical characters (not special characters), of which the first is I, J, K, L, M, or N. The length of a fixed-point variable name is:	
1 to 6 characters in Full FORTRAN	Full FORTRAN I M1234 IOBNOM
1 to 5 characters in Basic FORTRAN	Basic FORTRAN I M1234 JOBNO
A fixed-point variable can assume any integral value provided the magnitude is less than 10^{10}. Values used for subscripts, however, are treated modulo 10^4; i.e., the right-most four digits are used.	
Naming Floating-Point Variables The name of a floating-point variable consists of a series of alphabetic and numerical characters (not special characters), of which the first is alphabetic but <u>not</u> I, J, K, L, M, or N. The length of a floating-point variable name is:	
1 to 6 characters in Full FORTRAN	Full FORTRAN A B7 DELTAT
1 to 5 characters in Basic FORTRAN	Basic FORTRAN A B7 SEVEN
A floating-point variable can assume any value expressible as a normalized floating-point number. The magnitude of such numbers must be expressed as follows: $1.0 \times 10^{-51} \leq x < 1.0 \times 10^{49}$, or the magnitude should be equal to zero.	

Restrictions on Naming Variables

The following restrictions must be observed when naming variables (either fixed-point or floating-point):

1. Subscripted variables (see below) must not be given names ending with F, because the FORTRAN processors will consider variables so named to be functions.

2. A variable must not be given a name which coincides with the name of a library function (being used in the same program) without its terminal F. Thus, if a library function is named TIMEF, no variable should be named TIME.

SUBSCRIPTS

When a variable name represents a list of numbers (or array), there must be a means of referring to a specific element of that list. The FORTRAN language provides this capability through the use of subscripts. A subscript consists of a constant, variable, or limited expression attached to a variable name and contained within parentheses.

Each subscript of a subscripted variable name must be an expression in one of the following forms:

GENERAL FORM	EXAMPLES
V C V+C V-C C*V C*V+C' C*V-C' Where: V is any fixed-point variable. C is any unsigned fixed-point constant. C' is any unsigned fixed-point constant. * denotes multiplication.	I 3 MX+2 NA-2 5*J 6*K+5 8*L-9

NOTES:

1. The variable (V) in a subscript must not itself be subscripted.
2. Where a fixed-point constant is used for the value of a subscript, it is treated modulo 10^4.

Subscripted Variables

GENERAL FORM	EXAMPLES
A subscripted variable is the name of a fixed- or floating-point variable followed by parentheses enclosing subscripts, which are separated by commas.	
Full FORTRAN permits 1, 2, or 3 subscripts.	Full FORTRAN A(I) K(3,23*L6+13) BETA(J,K,3)
Basic FORTRAN permits 1 or 2 subscripts.	Basic FORTRAN A(I) K(L,M) BETA(5*J-2,K+2)

Each variable that is subscripted must have the size of its array (i.e., the maximum values that its subscripts can attain) specified prior to the first appearance of the variable in subscripted form. This is accomplished by a DIMENSION statement (see discussion of DIMENSION statement).

The value of a subscript exclusive of its addend, if any, must be greater than zero and not greater than the corresponding array dimension.

Arrangement of Arrays in Storage

If an array, A, is 2-dimensional, it will be stored sequentially in the following order: A(1,1), A(2,1), A(3,1), . . . , A(m,1), A(1,2), A(2,2), A(3,2), . . . , A(m,2), . . . , A(m,n).

Arrays are thus placed in core storage in column order with the first of their subscripts varying most rapidly, and the last varying least rapidly. The same is true of 3-dimensional arrays. Arrays which are 1-dimensional are simply stored sequentially. All arrays are stored in order of increasing absolute storage locations.

EXPRESSIONS

An expression in FORTRAN is a sequence of constants, variables (subscripted or nonsubscripted), and functions, separated by operation symbols, commas and parentheses. An expression appears on the right-hand side of arithmetic statements and in certain types of control statements.

Rules for Constructing Expressions

The simplest expression consists of a single constant, variable or subscripted variable. If the quantity is a fixed-point quantity, the expression is said to be in the fixed-point mode. If the quantity is a floating-point quantity, the expression is in the floating-point mode. Although a FORTRAN expression may be either fixed-point or floating-point, it must not be a mixed expression. This does not mean that a floating-point quantity cannot appear in a fixed-point expression, or vice versa, but rather that a quantity of one mode can appear in an expression of the other mode only in certain ways as shown below:

1. A floating-point quantity may appear in a fixed-point expression only as an argument of a function (see "Naming of FORTRAN Function Subprograms").
2. A fixed-point quantity may appear in a floating-point expression only as an argument of a function, as a subscript, or as an exponent.

In addition to the foregoing, the following rules pertain to the formation of expressions:

1. Any constant, variable, or subscripted variable is also an expression of the same mode. Since variables with names beginning with I, J, K, L, M, or N, and integers are fixed-point, 3 and I are fixed-point expressions. However, ALPHA and A(I,J,K) are floating-point expressions.

2. If SOMEF is some function of n variables, and if E, F, . . . , H are a set of n expressions of the correct modes for SOMEF, then SOMEF (E, F, . . . , H) is an expression of the same mode as SOMEF.

3. If E is an expression, and if its first character is not + or -, then +E and -E are expressions of the same mode as E. Thus, -A is an expression, but +-A is not.

4. If E is an expression, then (E) is an expression of the same mode as E. Thus (A), ((A)), (((A))), etc., are expressions.

5. If E and F are expressions of the same mode, and if the first character of F is not + or -, then

E + F
E - F
E * F
E / F

are all expressions of the same mode, but A-+B and A/+B are not expressions. (The characters +, -, *, and / denote addition, subtraction, multiplication, and division, respectively.)

6. Exponentiation is denoted by **. Therefore, A**B is equivalent to A^B. The rules for exponentiation are as follows:

a. An exponent may not have a + or - sign as its first character. A**+B and A**-B are illegal expressions in that they contain consecutive operators. Furthermore, since +A**B and -A**B are defined by the rule of Hierarchy of Operations as +(A**B) and -(A**B), respectively, rather than (-A)**B and (+A)**B, caution must be exercised in using a + or - as the first character of an exponential expression.

b. If FLT designates a floating-point expression, and FIX denotes a fixed-point expression, then the following mode combinations are permissible:

FLT^{FLT} FIX^{FIX}. FLT^{FIX}

Thus, A**B, I**J, and A**I are all permissible, but I**A is not. That is, a fixed-point expression may not be raised to a floating-point power.

NOTE: In the case of FLT^{FLT}, the exponential expression may not be a negative quantity.

c. The notation A**B**C is not acceptable in FORTRAN. The expression (A**B)**C or A**(B**C) although acceptable will yield different results e.g., (2.0**2.0)**0.0 = 1, and 2.0**(2.0**0.0) = 2.

Hierarchy of Operations

The order of evaluation of an expression may be explicitly specified through the use of parentheses. Computation begins with the innermost parentheses and proceeds to the outermost. When the hierarchy of operations in an expression is not explicitly specified by the use of parentheses, the order of computation is understood by FORTRAN to be the following:

Exponentiation
Multiplication and Division
Addition and Subtraction

For example, the expression

A+B/C+D**E*F-G

will be taken to mean

A+(B/C)+(D^E*F)-G

SUBROUTINES

A subroutine is any sequence of instructions that performs some desired operation. A closed subroutine is one which will appear in the object program only once regardless of the number of times it is referred to in the source program. Thus, for each reference to a closed subroutine by the source program, the processor constructs only the proper linkage to the routine in the object program once for every reference to it in the source program. An open subroutine is a set of instructions that is placed in an object program, every time a reference is made to the subroutine.

All subroutines in FORTRAN are closed subroutines except the Library Functions ABSF and XABSF.

FUNCTION and SUBROUTINE Subprograms

There are four types of subprograms that may be utilized in FORTRAN. Three of the four types of subprograms are classified as functions. A function is a subprogram that may be called (i.e., used) by writing the name of the function in an expression. The fourth type of subprogram is the SUBROUTINE subprogram. Since the SUBROUTINE subprogram is not a function, it can be called only by the use of a special statement. As noted below, each type of

subprogram is defined in a different way. Table I shows the characteristics of the four types. The numbers refer to the notes below:

Table I. Function and SUBROUTINE Subprograms

Type of Subprogram	Calling Method	How Named	How Defined
Library Functions	1	2	3
Arithmetic Statement Functions (Full FORTRAN)	1	2	4
FORTRAN FUNCTION Subprogram (Full FORTRAN)	1	5	6
SUBROUTINE Subprogram (Full FORTRAN)	7	5	8

1. The function is referred to by an arithmetic expression containing its name.
2. See "Naming of Library and Arithmetic Functions."
3. The library function is a subprogram, coded in Autocoder and assembled separately. It is placed in core storage in absolute form with the object program at object time.
4. These are functions that are defined by a single FORTRAN arithmetic statement and apply only to the particular program or subprogram in which their definition appears.
5. See "Subprogram Statements."
6. These subprograms must be defined by a FUNCTION statement.
7. The SUBROUTINE subprogram may be called only by a CALL statement.
8. These subprograms must be defined by a SUBROUTINE statement.

Calling Subroutines

As indicated in Table I, there are two distinct ways to call subprograms. Following are examples of arithmetic statements which contain function names:

```
Y = A-SINF(B-C)
C = MIN0F (M, L)+ABC(B*FORTF(Z), E)
```

The names of library, arithmetic statement, and FORTRAN function subprograms are used in this way. The appearance in the arithmetic expression serves to call the function; the value of the function is then computed using the arguments that are supplied in the parentheses following the function name. Only one value is produced by these three types of functions, whereas the SUBROUTINE subprogram may produce many values. (A value is here defined to be a single numerical quantity.) The SUBROUTINE subprogram is not a function and, therefore, is not called in the same way. A SUBROUTINE subprogram is called by using a CALL statement.

Naming of Library and Arithmetic Statement Functions

Library and arithmetic statement functions are named in the same way. Note, however, that while library functions are available in both Full and Basic FORTRAN, arithmetic statement functions may be defined and used only in Full FORTRAN.

GENERAL FORM	EXAMPLES
The name of the function consists of a series of alphabetic and numerical characters (not special characters), of which the last must be F and the first must be alphabetic. Furthermore, the first must be X if and only if the value of the function is to be fixed-point. The name of the function is followed by parentheses enclosing the arguments separated by commas. The length of the name is:	
In Full FORTRAN, 4 to 7 characters for library functions and arithmetic statement functions.	ABSF(B) XMODF(M/N, K) XFIRSTF(L, Y, R10)
In Basic FORTRAN, 4 to 6 characters for library functions.	COSF(A) XMODF(M/N, K) FLOATF(I)

Naming of FORTRAN FUNCTION Subprograms

Although FORTRAN FUNCTION subprograms are referred to in arithmetic expressions in the same manner as the two types previously mentioned, the rules for naming them are different. These functions are named in exactly the same way as ordinary variables of the program; that is, the name of a fixed-point FORTRAN function must have I, J, K, L, M, or N for its first character. The only restriction is that the name of a FORTRAN FUNCTION subprogram which is 4 to 6 characters long may not end with F.

Mode

Consider a function of a single argument. It may be desired to state the argument either in fixed or floating point; similarly the function itself may be in either of these modes. Thus, a function of a single argument has 4 possible mode configurations; in general, a function of n arguments will have 2^{n+1} configurations.

A separate name must be given, and a separate routine must be available, for each of the mode configurations which are used. Thus, a complete set of names for a given function might be:

SOME0F	Fixed argument, floating function
SOME1F	Floating argument, floating function
XSOME0F	Fixed argument, fixed function
XSOME1F	Floating argument, fixed function

The Xs and Fs are mandatory, but the rest of the naming is arbitrary.

Library Functions

Library functions are subprograms that are prewritten and included with the compiled program at object time. A hand-coded library function subprogram may be used in a FORTRAN program by referring to its name in an arithmetic statement and then including the subprograms in absolute relocatable form with the object program at object time. In Basic FORTRAN, symbolic as well as absolute subprograms can be used.

Table II shows the functions that are available and whether the function is an open subprogram or contained in the FORTRAN Package or the FORTRAN Library. For a complete description of the FORTRAN Library routines, see Appendix D.

Arithmetic Statement Functions

Arithmetic statement functions are available only in Full FORTRAN. These are functions which are defined by a single FORTRAN arithmetic statement and apply only to the particular program or subprogram in which their definitions appear.

GENERAL FORM	EXAMPLES
The statement which defines an arithmetic statement function is of the form "a b" where a is a function name followed by parentheses enclosing its arguments (which must be distinct nonsubscripted variables) separated by commas, and b is an expression which does not involve subscripted variables. Any functions appearing in b must be library functions or already defined by previous arithmetic statements.	FIRSTF(X) = A*X+B SECONDF(X, B) = A*X+B THIRDF(D) = FIRSTF(E)/D FOURTHF(F, G) = SECONDF (F, THIRDF(G)) FIFTHF(I, A) = 3.0*A**I SIXTHF(J) = J+K XSIXTHF(J) = J+K

An arithmetic statement function should be defined prior to any reference to it. That is, it should be in logical sequence in the program (definition followed by the use). As with the other functions, the answer will be expressed in fixed- or floating-point format, depending on whether the name does or does not begin with X.

The right-hand side of a statement that defines an arithmetic statement function may be any expression not involving subscripted variables. In particular, it may involve functions which are either library functions or previously defined arithmetic statement functions.

Variables appearing in the expression on the right-hand side can be stated, as the arguments of the function on the left-hand side. Since the arguments are really only dummy variables, their names are unimportant (except when indicating fixed- or floating-point mode) and may even be the same as names appearing elsewhere in the program.

Those variables on the right-hand side which are not stated as arguments are treated as parameters. Thus, if FIRSTF is defined in a function statement as FIRSTF(X) = A*X+B then a later reference to FIRSTF(Y) will cause ay+b to be computed, based on the current values of a, b, and y. The naming of parameters, therefore, must follow the normal rules of uniqueness.

An arithmetic statement function may be used just as any other function. In particular, its arguments may be expressions and may involve subscripted variables; thus a reference to FIRSTF(Z+Y(I)), with the above definition of FIRSTF, will cause $a(z+y_i)+b$ to be computed on the basis of the current values of a, b, y_i and z.

FORTRAN FUNCTION Subprograms

A FORTRAN FUNCTION subprogram is defined by a completely independent FORTRAN program.

Since FORTRAN FUNCTION and SUBROUTINE subprograms are defined in similar ways, a discussion of their definitions is included in Part II.

Table II. Library Function

Type of Function	Definition	Number of Arguments	Name	Mode of Argument	Mode of Function	Where Found
*Trigonometric Cosine	Cosine Arg	1	COSF	Floating	Floating	Library
*Trigonometric Sine	Sine Arg	1	SINF	Floating	Floating	Library
*Arcsine	Arcsine Arg	1	ASINF	Floating	Floating	Library
*Arctangent	Arctan Arg	1	ATANF	Floating	Floating	Library
*Hyperbolic Tangent	Tanh Arg	1	TANHF	Floating	Floating	Library
Square Root	$\sqrt{\text{Arg}}$	1	SQRTF	Floating	Floating	Library
Choosing largest value	Max (Arg_1, Arg_2, ...)	≥2	MAX0F MAX1F XMAX0F XMAX1F	Fixed Floating Fixed Floating	Floating Floating Fixed Fixed	Library Library Library Library
Choosing smallest value	Min (Arg_1, Arg_2, ...)	≥2	MIN0F MIN1F XMIN0F XMIN1F	Fixed Floating Fixed Floating	Floating Floating Fixed Fixed	Library Library Library Library
Transfer of sign	Sign of Arg_2 times Arg_1	2	SIGNF XSIGNF	Floating Fixed	Floating Fixed	Library Library
Positive difference	Arg_1-Min (Arg_1, Arg_2)	2	DIMF XDIMF	Floating Fixed	Floating Fixed	Library Library
**Remaindering	Arg_1 (mod Arg_2)	2	MODF XMODF	Floating Fixed	Floating Fixed	Library Library
Truncation	Sign of Arg. times largest integer ≤ |Arg|	1	INTF XINTF	Floating Floating	Floating Fixed	Library Package
Natural Logarithm	$\text{Log}_e x$	1	LOGF	Floating	Floating	Package
Base 10 Logarithm	$\text{Log}_{10} x$	1	LOGF	Floating	Floating	Package
Base e Exponential	e^x	1	EXPF	Floating	Floating	Package
Base 10 Exponential	10^x	1	EXPXF	Floating	Floating	Package
Float	Floating a fixed number	1	FLOATF	Fixed	Floating	Package
Fix	Same as XINTF	1	XFIXF	Floating	Fixed	Package
Absolute value	|Arg|	1	ABSF XABSF	Floating Fixed	Floating Fixed	Open Open

*Trigonometric functions

**The function MODF (Arg_1, Arg_2) is defined as $Arg_1 - [Arg_1/Arg_2] Arg_2$, where [x] = integral part of x.

PART II: THE FORTRAN LANGUAGE

The basic unit of the FORTRAN language is called a "statement." There are 36 different types of statements that are accepted by the Full FORTRAN processor. Each statement deals with one aspect of a given problem; it may cause data to be fed into the computer, calculations to be performed, decisions to be made, results to be printed, etc. Some statements do not cause specific computer action, but rather provide information to the processor.

THE ARITHMETIC STATEMENT

The arithmetic statement defines a numerical calculation. It very closely resembles a conventional arithmetic or algebraic formula.

GENERAL FORM	EXAMPLES
a = b a is a variable that may or may not be subscripted. b is an arithmetic expression.	Q1 = K A(I) = B(I)+SINF(C(I))

In a FORTRAN arithmetic statement, the equal sign (=) specifies replacement, rather than equality. For example,

Y = N-LIMIT (J-2)

means that the value of N-LIMIT (J-2) is to replace the value of Y. The result is stored in fixed-point or floating-point format according to the variable to the left of the equal sign.

If the variable on the left is fixed-point and the expression on the right is floating-point, the result will be computed in floating-point format and then truncated and converted to a fixed-point integer. Thus, if the result is +3.872 the fixed-point number stored will be +3, not +4. If the variable on the left is floating-point and the expression on the right fixed-point, the latter will be computed in fixed-point format, and then converted to floating-point.

Examples of Arithmetic Statement

Arithmetic Statement	Explanation
A = B	Store the value of B in A.
I = B	Truncate B to an integer, convert to fixed-point and store in I.
A = I	Convert I to floating-point, and store in A.
I = I+1	Add 1 to I and store in I. This example illustrates the fact that an arithmetic statement is not an equation, but is a command to replace a value.
A = 3.0*B	Replace A by 3.0(B).
A = 3*B	Not permitted. The expression is mixed; i.e., contains both fixed- and floating-point quantities.
A = I*B	Not permitted. The expression is mixed.

CONTROL STATEMENTS

Control statements enable the user to control the flow of his program. The following section provides a description of the control statements that are available to the users of this program.

DO Statement

GENERAL FORM	EXAMPLES
"DO n i = m_1, m_2" or DO n i = m_1, m_2, m_3" where n is a statement number, i is a nonsubscripted fixed-point variable, and m_1, m_2, m_3 are each either an unsigned fixed-point constant or nonsubscripted fixed-point variable. If m_3 is not stated, it is taken to be 1.	DO 30 I = 1, 10 DO 14 JOB = 2, M, 3

The DO statement is a command to execute repeatedly the statements which follow, up to and including the statement with statement number n.

The first time the statements are executed with $i = m_1$; for each succeeding execution i is increased by m_3. After these statements have been executed with i equal to the highest value which does not exceed m_2, control passes to the statement following statement number n.

The range of a DO is that set of statements which will be executed repeatedly (i.e., the sequence of consecutive statements immediately following the DO, up to and including the statement numbered n).

The index of a DO is the fixed-point variable i, which is controlled by the DO in such a way that its value begins at m_1 and is increased each time by m_3 until it is about to exceed m_2. Throughout the range, it is available for computation, either as an ordinary fixed-point variable or as the variable in a subscript. After the last execution of the range, the index is not available for use until it is redefined. The index of a DO cannot exceed 9999.

DOs Within DOs

A DO can be contained within another DO. This is called a nest of DOs. If the range of a DO contains another DO, then all statements in the range of the enclosed DO must be within the range of the enclosing DO. The maximum number of DOs that can be contained within a nest is 50; i.e., a DO can contain a second DO, the second can contain a third, and so on.

Transfer of Control

Transfers of control that are permitted in and out of the range of a DO are illustrated in Figure 1. In this illustration 1, 2, and 3 are permitted transfers. However, 4, 5, and 6 are permitted only if the values of the indexes or parameters are unchanged; and the returns are made as follows:

4 is permitted, if the return is made from 1
5 is permitted, if the return is made from 2
6 is permitted, if the return is made from 3
7 is never permitted, since it is a transfer into the range of a DO from outside its range.

Restrictions on Statements in the DO Range

1. Any statement which redefines the value of the index or any of the indexing parameters (m's) is not permitted in the range of a DO.
2. The first statement in the range of a DO must not be one of the non-executable FORTRAN statements.
3. The range of a DO must not end with a transfer statement (see "Assigned GO TO").

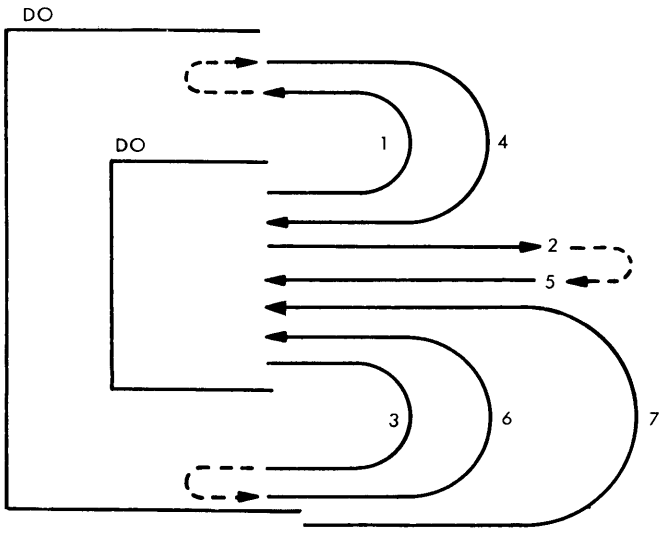


Figure 1

Ending of the DO Statement

The DO statement must end with a Continue statement, Format statement, or an Arithmetic statement.

Unconditional GO TO Statement

GENERAL FORM	EXAMPLES
"GO TO n" where n is a statement number	GO TO 3

This statement causes transfer of control to the statement with statement number n.

Computed GO TO Statement

GENERAL FORM	EXAMPLES
"GO TO (n_1, n_2, . . ., n_m), i" where n_1, n_2, . . ., n_m are statement numbers and i is a non-subscripted fixed-point variable. The limits of the value of i are $1 \leq i \leq m$.	GO TO (30, 42, 50, 9), I

This statement causes control to be transferred to the statement with statement number n_1, n_2, $n_3, \ldots, n_m$, depending on whether the value of i at time of execution is 1, 2, 3, ..., m, respectively. Thus, in the example, if i is 3 at the time of execution, a transfer to the statement whose number is third in the list (namely, statement 50) will occur. This statement is used to obtain a computed many-way fork.

Assigned GO TO Statement

GENERAL FORM	EXAMPLES
"GO TO n, (n_1, n_2, . . ., n_m)" where n is a nonsubscripted fixed-point variable appearing in a previously executed ASSIGN statement, and n_1, n_2, . . ., n_m are statement numbers.	GO TO K, (17, 12, 19)

This statement causes transfer of control to the statement with statement number equal to that value of n which was last assigned by an ASSIGN statement; n_1, n_2, . . . , n_m are a list of the values that n may have assigned.

The Assigned GO TO is used to obtain a preset many-way fork. When an Assigned GO TO exists in the range of a DO, the statement to which it may refer must all be in the exclusive range of a single DO, or all outside the DO nest.

ASSIGN Statement

GENERAL FORM	EXAMPLES
"ASSIGN i TO n ' where i is a statement number and n is a nonsubscripted fixed-point variable that appears in an assigned GO TO statement.	ASSIGN 12 TO K

This statement causes a subsequent GO TO n, (n_1, . . . , n_i, . . . , n_m) to transfer control to the statement with statement number i.

IF Statement

GENERAL FORM	EXAMPLES
"IF (a) n_1, n_2, n_3" where (a) is an expression and n_1, n_2, n_3 are statement numbers.	IF (A(J, K)-B) 10, 4, 30

This statement will cause control to be transferred to statement number n_1, n_2, or n_3, depending on the result of the computed expression (a):

- if the result is less than zero, statement number n_1 will be executed
- if the result is equal to zero, statement number n_2 will be executed
- if the result is greater than zero, statement number n_3 will be executed.

IF ACCUMULATOR OVERFLOW Statement

GENERAL FORM	EXAMPLES
"IF ACCUMULATOR OVERFLOW n_1, n_2" where n_1 and n_2 are statement numbers.	IF ACCUMULATOR OVERFLOW 30, 40

This statement will cause control to be transferred to statement number n_1 if an accumulator overflow condition has occurred, or to statement number n_2 if no overflow has occurred. For fixed-point operations, accumulator two is tested for overflow. For floating-point operations, accumulator two is tested for overflow; additional tests are made for floating-point overflow and floating-point underflow. The execution of this statement will cause the overflow and/or underflow indicators to be turned OFF. (Also see Note under "END Statement.")

IF QUOTIENT OVERFLOW Statement

GENERAL FORM	EXAMPLES
"IF QUOTIENT OVERFLOW n_1, n_2" where n_1 and n_2 are statement numbers.	IF QUOTIENT OVERFLOW 30, 40

This statement will cause control to be transferred to statement number n_1 if a quotient overflow condition has occurred or to statement number n_2 if no overflow condition has occurred. The execution of this statement will cause the quotient overflow condition indicator to be turned OFF. Also see note under "END Statement.")

IF DIVIDE CHECK Statement

GENERAL FORM	EXAMPLES
"IF DIVIDE CHECK n_1, n_2" where n_1 and n_2 are statement numbers.	IF DIVIDE CHECK 84, 40

This statement will cause control to be transferred to statement number n_1 if a divide check indicator is ON (i.e., if an attempt has been made to divide by zero); otherwise statement number n_2 will be executed. An internal indicator is used to denote the divide check condition; it is reset after the execution of this statement. The execution of this statement will turn the indicator OFF. (Also see note under "END Statement.")

SENSE LIGHT Statement

GENERAL FORM	EXAMPLES
"SENSE LIGHT i" where i is 0, 1, 2, 3, or 4.	SENSE LIGHT 3

Sense Lights are simulated using Electronic Switches in the 7070, 7072 and 7074. If i is 0, all Sense Lights will be turned OFF. If i is a digit from 1 through 4, the corresponding Sense Light will be turned ON.

IF (SENSE LIGHT) Statement

GENERAL FORM	EXAMPLES
"IF (SENSE LIGHT i) n_1, n_2" where n_1 and n_2 are statement numbers and i is 1, 2, 3, or 4.	IF (SENSE LIGHT 3) 30, 40

This statement will cause control to be transferred to statement number n_1 when the Sense Light corresponding to i is ON or to statement number n_2 when the designated Sense Light is OFF. The execution of this statement will cause the Sense Light to be turned OFF. (Also see note under "END Statement.")

IF (SENSE SWITCH) Statement

GENERAL FORM	EXAMPLES
"IF (SENSE SWITCH i) n_1, n_2" where n_1 and n_2 are statement numbers and i is 1, 2, 3, or 4.	IF (SENSE SWITCH 3) 30, 40

The Alteration Switches on the 7070, 7072 and 7074 consoles serve as Sense Switches in FORTRAN. Control is transferred to statement number n_1, when the Sense Switch corresponding to i is ON or to statement number n_2 when the designated Sense Switch is OFF.

CONTINUE Statement

GENERAL FORM	EXAMPLES
"CONTINUE"	CONTINUE

CONTINUE is a dummy statement and does not produce any executable instructions. It is used as the last statement in the range of a DO to provide a transfer address for IF and GO TO statements that are to begin another repetition of the DO range.

PAUSE Statement

GENERAL FORM	EXAMPLES
"PAUSE" or "PAUSE n" where n is an unsigned fixed-point constant less than 10^5.	PAUSE PAUSE 77777

This statement is used when the execution of the program is to be temporarily interrupted. The machine will halt and type out the number n. If n is not specified it is understood to be zero. Depressing the Start key causes the object program to resume execution at the next instruction.

STOP Statement

GENERAL FORM	EXAMPLES
"STOP" or "STOP n" where n is an unsigned fixed-point constant less than 10^5.	STOP STOP 77777

This statement is used where a terminal stop is desired. When the program halts, number n is typed out. If n is not specified, it is understood to be zero.

END Statement

GENERAL FORM	EXAMPLES
"END" or "END (I_1, I_2, I_3, I_4, I_5)" where I is 0, 1, or 2.	END END (2, 2, 2, 2, 2) END (1, 2, 0, 1, 1)

The END statement defines the end of a program or subprogram. Only the word END has any significance. The remainder of the statement, if included, is ignored by the processor.

This statement may only appear as the last statement in a program. In a multifile compilation, the END statement must appear as the last statement of all programs except the last program. In unifile compilation, the use of an END statement is optional, since the physical tape mark or end-of-file will signify the end of the source program.

This statement does not cause a halt instruction to appear in the object program.

NOTE: When using the

IF [ACCUMULATOR OVERFLOW
QUOTIENT OVERFLOW
DIVIDE CHECK
SENSE LIGHT i] n_1, n_2

the condition specified may be initialized by using the statement in question with $n_2 = n_1$, where n, is the statement number of the next desired executable statement. These statements then serve as "Unconditional GO TO" statements.

INPUT/OUTPUT STATEMENT

Input/output statements specify and/or control the handling of information between the computer and the input/output devices. This section describes each of these statements and their respective functions:

Specifying Lists of Quantities

Most I/O Statements call for the transmission of information and must, therefore, include a list of the quantities to be transmitted. The order of this list must be the same as the order in which the words of information exist (for input), or will exist (for output) in the input/output medium.

The formation and meaning of a list can be described by an example

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

Suppose that this list is used with an output statement. Then the information will be written on the input/output medium in this order:

A, B(3), C(1), D(1,K), C(2), D(2,K),...,
C(10), D(10,K),
E(1,1), E(3,1),..., E(9,1), F(1,3),
E(1,2), E(3,2),..., E(9,2), F(2,3),...,
F(K,3).

Similarly, if this list were used with an input statement, the successive words, as they were read from the external medium, would be placed in storage in the sequence just given.

Thus, the list reads from left to right with repetition of variables enclosed within parentheses. Only variables and not constants may be listed. The execution is exactly that of a DO loop, as though each opening parenthesis (except subscripting parentheses) were a DO, with indexing given immediately before the matching closing parenthesis, and with the DO range extending up to that indexing information. The order of the above list can thus be considered to be the equivalent of the "program":

```
1  A
2  B(3)
3  DO 5 I = 1, 10
4  C(I)
5  D(I,K)
6  DO 9 J = 1, K
7  DO 8 I = 1, 10, 2
8  E(I,J)
9  F(J,3)
```

Note that indexing information, as in DOs, consists of three constants or fixed-point variables, and that the last of these may be omitted, in which case it is taken to be 1.

For a list of the form K, (A(K)) or K, (A(I), I = 1, K) where an index or indexing parameter itself appears earlier in the list of an input statement, the indexing will be carried out with the newly read-in value.

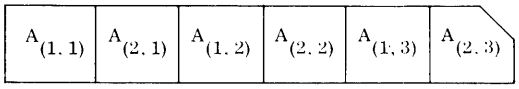
Input/Output in Matrix Form

FORTRAN treats variables according to conventional matrix practice. Thus, the input/output statement

READ 1, ((A(I,J), I = 1, 2), J = 1, 3)

causes the reading of I x J (in this case 2 x 3) items of information. The data items will be read into storage in the same order as they are found on the input medium.

For example, if punched on a data card in the form:



the items will be stored in locations N, N - 1, N - 2,..., N - 5, respectively, where N is the highest absolute location used for the array of information to be read in.

Input/Output of Entire Matrices

When input/output of an entire matrix is desired, an abbreviated notation may be used for the list of the input/output statement; only the name of the array need be given and the indexing information may be omitted.

Thus, if A has previously been listed in a DIMENSION statement, the statement,

READ 1, A

is sufficient to read in all of the elements of the array A. In 7070 FORTRAN the elements, read in by this notation, are stored in order of increasing storage locations. If A has not previously appeared in a DIMENSION statement, only the first element will be read in.

FORMAT Statement

GENERAL FORM	EXAMPLES
"FORMAT (Specification)" where specification is as shown in the examples. The specification must be enclosed in parentheses.	FORMAT (I2/(E12.4, F 10.4)) FORMAT (E 10.6, (I8))

FORMAT statements are used in conjunction with other input/output statements. These statements specify the arrangement of the data that is to be transferred. They also specify the type of conversion to be performed between the internal and external representations for each element in the list. FORMAT statements are non-executable and, therefore, may be placed anywhere in the source program.

For the sake of clarity, the details of writing a FORMAT specification are given below for use with PRINT statements. However, the description is valid for any case simply by generalizing the concept of "printed line" to that of unit record in the input/output medium. A unit record may be:

1. A printed line with a maximum of 120 characters.
2. A punched card with a maximum of 72 characters.
3. A BCD tape record with a maximum of 120 characters.

Conversion of Numeric Data

The three types of conversion that can be used with numerical variables are E, F, and I. Each type can convert data from external to internal notation and vice versa, depending on whether the data is entering or leaving the computer.

Internal Representation	Conversion Code	External Representation
Floating-Point	E	Floating-point (with exponent)
Floating-Point	F	Floating-point (without exponent)
Integer	I	Integer

The three types of conversion codes are specified in the following manner:

FORMAT (Ew.d)
FORMAT (Fw.d)
FORMAT (Iw)

E, F, and I specify the type of conversion desired.

W

is an unsigned integer constant specifying the field width of the data.

d

is an unsigned integer constant specifying the number of positions of the field that are to appear as a fractional part (d is treated module 10).

E-TYPE CONVERSION: External representation suitable for E Conversion is of the form

±x E ±e

±x

is a decimal number of not more than eight digits including the decimal point.

E

denotes the start of the exponent field.

±e

is a one- or two-digit decimal number indicating the power of ten to which ±X is to be raised. The magnitude of the number should be

$1.0 \times 10^{-51}, \leq X < 1.0 \times 10^{49}$

or it may be zero.

Examples:

+1.E-1	is interpreted as +0.1
-14376210.E+32	is interpreted as -1437621×10^{33}
.0003572E +5	is interpreted as +35.72

F-CONVERSION: External representation for F Conversion is of the form

±X

where

X is a decimal number of not more than eight digits, including a decimal point.

Examples:

+1
35267.453
-.23456
+4356789.

I-TYPE CONVERSION: I-type conversion is used only with fixed-point variables. External notation suitable for I conversion is of the form

±X

where

X is a decimal number of not more than ten digits which must not contain a decimal point.

Examples:

1
+1234567890
-5634945
39495

NOTE: For additional information on preparing data see the discussion on "Data Input to the Object Program."

Alphameric Fields

FORTRAN provides two ways to read or write alphameric information; the specifications for this purpose are nAw and wH. Both result in storing

the alphameric information internally. The basic difference is that information handled with the A specification is given a variable or array name and, hence, can be referred to by means of this name for processing and/or modification. Information handled with the H specification is not given a name and may not be referred to or manipulated in storage in any way.

The specification nAw causes n fields of w characters to be read into, or written from, a variable or array, where $0 \leq n*w \leq 120$. The name of the variable or array must be constructed in the same manner as the name of either a fixed-point or a floating-point variable.

On input, n successive fields of w characters each will be stored in double-digit form. If w is greater than 5, only the five right-most characters of each field will be transmitted. If w is less than 5, the w left-most characters of each field will be read and each word will be filled on the right with blanks.

On output, the next n successive fields each containing w characters will be transmitted to the external medium without conversion. If w exceeds 5, only five characters of output will be transmitted (for each field) preceded by w-5 blanks. If w is less than 5, the w left-most characters of each field will be transmitted.

The specification wH is followed in the FORMAT statement by w alphameric characters. For example:

24H THIS IS ALPHAMERIC DATA

Note that blanks are considered alphameric characters and must be included as part of the count w.

1. Input. The w characters are extracted from the input record and replace the w characters included with the specification.

2. Output. The w characters following the specification, or the characters which replaced them, are written as part of the output record.

Example: The statement FORMAT (3HXY = F8.3, A8) might produce the following lines:

```
XY=b-93.210bbbbbbbbbb
XY=9999.999bbbbOVFLO
XY=bb28.768bbbbbbbbbb
```

This example assumes that there are steps in the source program which read the data "OVFLO," store this data in the word to be printed in the format A8 when overflow occurs, and stores five blanks in the word when overflow does not occur. (The b is used to indicate blank characters.)

Blank Fields

The specification wX introduces blank characters as follows:

For an input record, w number of characters are skipped over, and are considered to be blank regardless of what they actually are.

For an output record, w number of blank characters are placed in the record.

The number of blanks inserted or characters skipped must be less than 120. The control character X need not be separated by a comma from the specification of the next field.

Basic Field Specifications

These basic field specifications are given in the forms:

Iw, Ew.d, and Fw.d

The specifications for successive fields are separated by commas. Thus, the statement

```
1 FORMAT(12,E12.4,F10.4)
```

might give the line:

```
27 -0.9321E 02   -0.0076
```

As in this example, the field widths may be made greater than necessary in order to provide spacing blanks between numbers. In this case, there is 1 blank following the 27, there is 1 blank after the E (automatically supplied except in cases of a negative exponent, when a minus sign will appear), and there are 3 blanks after the 02. Within each field the printed output will always appear in the right-most positions.

It may be desired to print n successive fields within one record, in the same fashion. This may be specified by giving n before E, F, or I. Thus, the statement FORMAT (I2, 3E12.4) might give:

```
27 -0.9321E 02 -0.7580E-02  0.5536E 00
```

Repetition of Groups

A parenthetical expression is permitted in order to enable repetition of data fields according to certain format specifications within a longer FORMAT statement specification. Thus, FORMAT (2(F10.6, E10.2), I4) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2, I4). Only one level of these parentheses is allowed.

Repetitions are made from the last open parenthesis including the number (if any) preceding the parenthesis if the LIST in the I/O statement is not satisfied. For additional information, see the discussion on ending a FORMAT statement.

For example, consider the following FORMAT statement:

FORMAT (3F10.2, 2(F15.8, I10,I5))

Nine fields will be read from the first record, of which the first 3 will be 10 digits, the next in order will be 15, 10, 5, 15, 10, and 5 digits, respectively. If the I/O LIST is not satisfied, a new record will be read assuming 6 fields of 15, 10, 5, 15, 10, and 5 digits. Additional records will be read following this pattern until the LIST is satisfied.

Scale Factors

To permit more general use of F-type conversion, a scale factor followed by the letter P may precede the specification. The scale factor is defined in such a way that:

$$\text{Printer number} = \text{Internal number} \times 10^{\text{scale factor}}$$

Thus, the statement FORMAT (I2, 1P3F11.3) used with the data of the preceding example, would give:

```
27   -93.096   -0.076   5.536
```

FORMAT (I2, -P3F11.3) would give:

```
27   -9.321   -0.001   0.055
```

A scale factor may be used with E-type conversion. Thus, with the same data, FORMAT I2, (1P3E12.4) would produce:

```
27 -9.3210E 01 -7.5804E-03  5.5361E-01
```

Once a value has been given, it will hold for all E- and F-type conversions following the scale factor within the same FORMAT statement. This applies to both single-record and multiple-record formats. Once a scale factor has been given, a subsequent scale factor of zero in the same FORMAT statement must be specified by 0P. Scale factors have no effect on I-type conversion.

NOTE: When a scale factor is used with F-type conversion, the absolute value of the number is changed by the scale factor. However, when used with E-type conversion, the scale factor only expresses the same number in a different form. Thus, if the number -93.2096 were printed with a specification 1PF11.3, it would become -932.096. However, if the same number were printed with the specification E12.4, it would appear as -0.9321E 02; and with the specification 1PE12.4, it would be -9.3210E 01. (The control character X need not be separated by a comma from the specification of the next field.)

Multiple-Record Formats

To deal with a block of more than one line of print, a FORMAT specification may have several different one-line formats, separated by a slash (/) to indicate the beginning of a new line. Thus, FORMAT (3F9.2, 2F 10.4/8E14.5) would specify a multi-line block of print in which lines 1, 3, 5, ... have format (3F9.2, 2F10.4), and lines 2, 4, 6, ... have format 8E14.5.

If a multiple-line format is desired in which the first two lines are to be printed according to a special format and all remaining lines according to another format, the last line-specification should be enclosed in a second pair of parentheses; e.g., FORMAT (I2, 3E12.4/2F10.3, 3F9.4/ (10F12.4)). If data items remain to be transmitted after the format specification had been completely "used," the format repeats from the last open parenthesis, including the number (if any) preceding the parenthesis.

As these examples show, both the slash and the closing parenthesis of the FORMAT statement indicate a termination of the record.

Blank lines may be introduced into a multiline FORMAT statement by listing consecutive slashes. N + 1 consecutive slashes produce N blank lines.

FORMAT and Input/Output Statement Lists

The FORMAT statement indicates, among other things, the maximum size of each record to be transmitted. In this connection, it must be remembered that the FORMAT statement is used in conjunction with the list of some particular input/output statement, except when a FORMAT statement consists entirely of alphameric fields. In all other cases, control in the object program switches back and forth between the list (which specifies whether data remains to be transmitted) and the FORMAT statement (which gives the specifications for transmission of that data).

Ending a FORMAT Statement

An input/output operation under the control of a FORMAT statement will end:

1. When a specification of a numerical field is given and all the items in the list have been transmitted, the execution of the particular statement is terminated.
2. When the end of the FORMAT statement is encountered.

FORMAT Statements Read in at Object Time

FORTRAN accepts the name of a variable as a FORMAT address. This provides the facility of entering a statement at object time.

Example: C for Comment

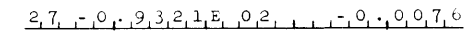
Statement Number	FORTRAN STATEMENT
1	DIMENSION FMT (12) FORMAT (12A5) READ 1, FMT READ FMT, A, B, (C (I), I = 1, 5)

Thus, A, B, and the array C would be converted and stored, according to the FORMAT specification read into the array FMT, at object time.

The FORMAT read in at object time must take the same form as a source-program FORMAT statement, except that the word FORMAT is omitted; i.e., the variable format begins with a left parenthesis.

Data Input to the Object Program

On-line input data to be read by means of a READ or READ INPUT TAPE statement, when the object program is executed, must be in essentially the same format as given in the previous examples. Thus, a card to be read according to FORMAT (I2, E12.4, F10.4) might be punched:



Within each field, all information must appear at the extreme right. Plus signs may be omitted or indicated by a blank or plus. Minus signs may be punched with an 11-punch or an 8-4 punch. Blanks in numerical fields are regarded as zeros. Numbers for E- and F-type conversion may contain any number of digits, but only the high-order eight digits will be retained (rounding will be performed). Numbers for I-type conversion will be treated modulo 10^{10}.

To permit economy in punching, certain relaxations in input data format are allowed:

1. Numbers of E-type conversion need not have 4 columns devoted to the exponent field. The start of the exponent field must be marked by an E, or if that is omitted, by a plus or minus sign. Thus, E2, E02, +2, +02, E 02, and E + 02 are all permissible exponent fields.
2. Numbers for E- or F-type conversion need not have their decimal point punched, because the FORMAT specification automatically makes such an insertion. For example, the number -09321+2 with the specification E 12.4 will be treated as though the decimal point had been punched between the 0 and the 9. If the decimal point is punched, its position will over-ride the indicated position in the FORMAT specification.

READ Statement

GENERAL FORM	EXAMPLES
"READ n, List" where n is the statement number of a FORMAT statement or the name of a variable, and List is an input list.	READ 1, ((ARRAY (I, J), I = 1, 3), J = 1, 5)

The READ statement causes the reading of cards from the card reader. Record after record (i.e., card after card) is read until the complete list has been transmitted; i.e., brought in, converted, and stored in the locations specified by the list of the READ statement. The FORMAT statement to which the READ refers describes the arrangement of information on the cards and the type of conversion to be made.

PUNCH Statement

GENERAL FORM	EXAMPLES
"PUNCH n, List" where n is the statement number of a FORMAT statement or the name of a variable, and List is an output list.	PUNCH 30, (A(J), J = 1, 10)

The PUNCH statement causes the object program to punch cards according to specifications in a FORMAT statement. The cards are punched until the entire list is exhausted.

PRINT Statement

GENERAL FORM	EXAMPLES
"PRINT n, List" where n is the statement number of a FORMAT statement or the name of a variable and List is an output list.	PRINT 2, (A(J), 1, 10)

The PRINT statement causes the object program to print output data on an on-line printer according to specifications in a FORMAT statement. Successive lines are printed until the entire list has been exhausted.

TYPE Statement

GENERAL FORM	EXAMPLES
"TYPE n, List" where n is the statement number of a FORMAT statement or the name of a variable and List is an output list.	TYPE 56, (A(J), J = 1, 10)

The TYPE statement causes the object program to type out data on the console typewriter in accordance with specifications in a FORMAT statement. The data is typed out until the entire list is exhausted.

Tape Input and Output

FORTRAN includes a complete set of statements which will read or write tapes. It makes a distinction between logical tape unit numbers specified in a FORTRAN source program, and the actual tape units which will be affected by the resulting object program.

Any 7070-Series Data Processing System may have one of two conventions for denoting tape units depending on whether or not the system is equipped with an optional device; i.e., Feature Code 12. Without Feature Code 12, only unit numbers 0-5 are legal on each channel. For example, a reference by a 7070 to channel 1 unit 9 would be permissible only if the machine were equipped with a Feature Code 12 device.

FORTRAN allows the programmer to substitute the name of a fixed-point variable for a logical tape number in the tape statements explained later in this chapter. The value of the variable when referenced by the tape statement must be the logical number of an available tape unit.

FORTRAN can read and write tape in two ways. One set of tape read/write statements (READ INPUT TAPE i, n, List and WRITE OUTPUT TAPE i, n, List), reads or writes information in external notation which is especially suitable for off-line printing. These statements must refer to a FORMAT statement.

The second set of statements (READ TAPE i, List and WRITE TAPE i, List) reads or writes tape in internal notation which normally is used as input to a 7070, 7072 or 7074, or can be printed off-line if internal notation is acceptable to the particular application.

All FORTRAN tapes on the 7070, 7072 or 7074 are written in binary coded decimal. Each tape contains one or more physical records that are combined to form one or more logical records. A physical record is a group of data on tape preceded and followed by an inter-record gap. A logical record is one or more physical records that are written or read by a single list. Thus, one list produces one logical record. The number of characters per physical tape record depends on the type of statement that was used to write the record.

A tape written in internal notation contains logical records, each of which is preceded and followed by a one-character record called a segment mark. The format of a tape written in internal notation might be as follows:

sm 240 240 240 (or less) smsm 240 (or less) smsm 240 240 (or less) sm

where

240 is the number of characters
sm is a segment mark

A tape written in external notation contains logical records which are not separated by segment marks. The only identifying characteristic of a logical record in external notation is the programmer's knowledge that a group of physical records was written by a single list.

READ INPUT TAPE Statement

GENERAL FORM	EXAMPLES
"READ INPUT TAPE i, n, List" where i is an unsigned fixed-point constant or the name of a fixed-point variable; n is the statement number of a FORMAT statement or the name of a variable, and List is an input list.	READ INPUT TAPE 4, 3, K, A(J) READ INPUT TAPE N, 8, K, A(J)

The READ INPUT TAPE statement causes the object program to read information in external notation from logical tape unit i. Each record is brought in, in accordance with specifications in a FORMAT statement, until the entire list has been read.

A READ INPUT TAPE statement accepts records of less than 120 characters. This enables FORTRAN to read card images put on tape by peripheral equipment. When the record to be read is less than 120 characters, the FORMAT statement referred to must describe the record exactly; i.e., it must not specify more than the number of characters actually present in the record.

WRITE OUTPUT TAPE Statement

GENERAL FORM	EXAMPLES
"WRITE OUTPUT TAPE i, n, List" where i is an unsigned fixed-point constant or the name of a fixed-point variable, n is the statement number of a FORMAT statement or the name of a variable, and List is an output list.	WRITE OUTPUT TAPE 2, 3, (A(J), J = 1, 10) WRITE OUTPUT TAPE L, 10, (A(J), J = 1, 10)

The WRITE OUTPUT TAPE statement causes the object program to write information in external notation on symbolic tape unit i. Successive physical records are written in accordance with the specifications in a FORMAT statement, until the list has been exhausted.

Programming Carriage Control

The WRITE OUTPUT TAPE statement prepares a BCD tape that can be used later to obtain off-line printed output. The off-line printer is manually set to operate in one of three modes: single space, double space, and Program Control. Under Program Control, which gives the greatest flexibility, the first character of each BCD record is used to control the spacing of the off-line printer and that character is not printed. The following control characters may be used to specify the format:

Character	Explanation
Blanks	Single-space before printing
0	Double-space before printing
+	No space before printing
1-9	Skip to printer control channel 1-9
J-L	Short skip to printer control channels 1-9

Thus, a FORMAT specification, for WRITE OUTPUT TAPE for off-line printing with Program Control, will usually begin with 1H followed by the appropriate control character. (This procedure is not required for the PRINT statement.)

READ TAPE Statement

GENERAL FORM	EXAMPLES
"READ TAPE i, List" where i is an unsigned fixed-point constant or a fixed-point variable, and List is an input list.	READ TAPE 14, (A(J), J = 1, 10) READ TAPE K, (A(J), J = 1, 10)

The READ TAPE statement causes the object program to read information expressed in internal notation from logical tape unit i into locations specified in the list.

A logical record is read completely only if the list specifies as many words as the tape record contains; no more than one logical record will be read. The tape, however, always moves to the beginning of the next logical record.

NOTE: A tape written by a WRITE OUTPUT TAPE statement must have been read by a READ INPUT TAPE statement. A tape written by a WRITE TAPE statement must have been read by a READ TAPE statement.

WRITE TAPE Statement

GENERAL FORM	EXAMPLES
"WRITE TAPE i, List" where i is an unsigned fixed-point constant or a fixed-point variable, and List is an output list.	WRITE TAPE 14, (A(J), J = 1, 10) WRITE TAPE K, (A(J), J = 1, 10)

The WRITE TAPE statement causes the object program to write information in internal notation on a symbolic tape unit i. One logical record is written consisting of all the words specified in the list.

If a list specifies a number of characters other than a multiple of 240, the final physical record will be written consisting of only the remaining characters in the list.

END FILE Statement

GENERAL FORM	EXAMPLES
"END FILE i" where i is an unsigned fixed-point constant or a fixed-point variable.	END FILE 5 END FILE K

The END FILE statement causes the object program to write an end-of-file mark on logical tape unit i.

REWIND Statement

GENERAL FORM	EXAMPLES
"REWIND i" where i is an unsigned fixed-point constant or a fixed-point variable.	REWIND 3 REWIND K

The REWIND statement causes the object program to rewind logical tape unit i.

BACKSPACE Statement

GENERAL FORM	EXAMPLES
"BACKSPACE i" where i is an unsigned fixed-point constant or a fixed-point variable.	BACKSPACE 18 BACKSPACE K

If the BACKSPACE statement is used on a tape that has been written in internal notation, it will cause the tape to backspace one logical record. If BACKSPACE is used on a tape written in external notation, it will cause the tape to backspace one physical record.

SPECIFICATION STATEMENTS

Specification Statements provide the processor with necessary information to make storage allocations. The three specification statements discussed in this section are the DIMENSION statement, the EQUIVALENCE statement, and the COMMON statement. All these statements are non-executable and are available to the users of the Full FORTRAN processor. The Basic FORTRAN processor cannot handle the COMMON statement.

DIMENSION Statement

GENERAL FORM	EXAMPLE
"DIMENSION v, v, v, . . . " where v is the name of a variable subscripted with fixed-point constants.	Full FORTRAN B34 (10, 25, 15) B(5, 15)
1, 2, or 3 subscripts are permitted in Full FORTRAN.	JOBNUM (8)
1 or 2 subscripts are permitted in Basic FORTRAN.	Basic FORTRAN TOR(23, 7) I(305)

The DIMENSION statement provides information necessary to allocate storage for arrays in the object program. Each variable that appears in subscripted form in a source program must appear in a DIMENSION statement of the source program. The DIMENSION statement normally precedes the first appearance of each subscripted variable whose array size it is to specify. Since the DIMENSION statement lists the maximum dimension of arrays, programming references to these arrays should not exceed the specified dimensions.

The above example indicates that B is a 2-dimensional array for which the subscripts never exceed 5 and 15. The DIMENSION statement therefore, causes 75 (i.e. 5 x 15) storage locations to be set aside for array B.

A single DIMENSION statement may specify the dimensions of any number of arrays, provided it is within the bounds of the statement card.

EQUIVALENCE Statement

GENERAL FORM	EXAMPLES
"EQUIVALENCE (a, b, c, . . .), (d, e, f, . . .), . . ." where a, b, c, d, e, f, . . . are variables optionally followed by a single unsigned fixed-point constant in parentheses.	EQUIVALENCE (A(1), B(1)), (B(1), C(5)), (D(17), E(3))

The EQUIVALENCE statement causes two or more variables to be assigned the same core-storage location and, thus in effect, increases the storage capacity. This statement may be placed anywhere in the program, but is generally placed in the beginning.

Each pair of statement list parentheses encloses the names of two or more quantities which are to be stored in the same locations during execution of the object program; any number of equivalences (i.e., sets of parentheses) may be given.

The subscript in an EQUIVALENCE statement has a different meaning from that of normal FORTRAN subscripts. For example, the EQUIVALENCE subscripts, C(5), means the fourth storage location following C(1) or C(1, 1) or C(1, 1, 1), depending on whether the array is one-, two-, or three-dimensional. In general, C(p) is defined for p>0 to mean the (p - 1)th location after C or after the beginning of the C-array. Generally, p is specified, but if p is not specified, it is taken to be 1. Also p cannot exceed the maximum number of elements in the array. Nonsubscripted variables in an EQUIVALENCE statement need not be followed by parentheses. Quantities or arrays that are not mentioned in an EQUIVALENCE statement will be assigned unique locations.

For example, if R and S are arrays, which are defined by the statement DIMENSION R(10, 9), S(15, 15, 3), and if it is desired to have R(2, 3) occupy the same location as S(1, 1, 1) then the following statement would have the desired effect:

EQUIVALENCE (R(22), S)

Locations can be shared only among variables, not among constants. The sharing of storage locations cannot be planned safely without a knowledge of which FORTRAN statements, when executed in the object program, will cause a new value to be stored in a location. These statements are:

1. An arithmetic formula, which stores a new value of the variable for the left-hand side of the formula.

2. An ASSIGN i TO n, which stores a new value in n.

3. A DO, which stores a new indexing value in the cell containing i.

4. A READ, READ INPUT TAPE, or READ TAPE, which stores new values for the variables mentioned in the statement list.

The dummy variables that are used as arguments of FUNCTION or SUBROUTINE statements should not appear in EQUIVALENCE or COMMON statements in a subprogram.

For an example of the use of EQUIVALENCE, see "Using COMMON and EQUIVALENCE Together."

COMMON Statement

The COMMON statement is available only in Full FORTRAN.

GENERAL FORM	EXAMPLES
"COMMON A, B, . . ." where A, B, . . . are the names of variables and nonsubscripted array names.	COMMON X, ANDMN, MATA, ATB

The COMMON statement refers to a common area of core storage and has a twofold function. It allows the programmer to specify variables and arrays to be placed in upper storage; and it serves as a medium to transmit arguments from the calling program to the called FORTRAN FUNCTION or SUBROUTINE Subprogram.

The common area is assigned normally starting with the highest location in storage and continuing downward. However, the programmer can specify any location as the highest location of storage by modifying control cards at compilation time.

The common area is assigned separately for each program compiled. Therefore, it can be shared between a program and its subprograms. Thus, the COMMON statement enables a data storage area to be shared between programs in a way analogous to that by which the EQUIVALENCE statement permits data storage sharing within a single program.

Array names appearing in the COMMON statement must also appear in a DIMENSION statement in the same program.

The programmer has complete control over the locations assigned to the variables appearing in common area. The locations are assigned in the sequence in which the variables appear in the COMMON statements, beginning with the first COMMON statement of the problem. If, however, a variable expressed in a COMMON statement also appears in an EQUIVALENCE statement, the assignment of a common area varies as described under "Using COMMON and EQUIVALENCE Together."

Arguments in Common Storage

The arguments in common storage are implicitly transmitted. To obtain these arguments, the corresponding variables in the two programs must occupy the same location. This can be done by having them occupy corresponding positions in COMMON statements of the two programs.

NOTES:

1. In order to force correspondence in storage locations between two variables which otherwise will occupy different relative positions in common storage, it is permissible to place dummy variable names in a COMMON statement. These dummy names, which can be dimensional arrays, may be used to provide the reservation of space that is necessary to cause correspondence.

2. While implicit arguments can take the place of all arguments in CALL-type SUBROUTINES, there must be at least one explicit argument in a FORTRAN FUNCTION subprogram. Here, too, a dummy variable may be used for convenience.

3. No dummy variables that are arguments of FUNCTION or SUBROUTINE statements may appear in EQUIVALENCE or COMMON statements in a subprogram.

Using COMMON and EQUIVALENCE Statements Together

When a variable is made equivalent to another variable that appears in a COMMON statement, the first variable will also be located in common storage. When variables appearing in a COMMON statement also appear in EQUIVALENCE statements, the ordinary sequence of common variables is changed and priority is given to variables that appear in the EQUIVALENCE statements. These variables follow the order in which they appear in the EQUIVALENCE statements. To illustrate the use of an EQUIVALENCE statement, consider a program in which five separate arrays (A, B, C, D, E) are to be placed in core storage. The maximum amount of storage that must be allocated for these arrays is specified in a DIMENSION statement as follows:

```
DIMENSION A(20), B(30), C(40), D(50), E(60)
```

If this DIMENSION statement appeared in the program without an EQUIVALENCE or COMMON statement, 200 storage locations would be reserved as shown in Figure 2. The addition of the statement

```
EQUIVALENCE (A(5), B(10)), (C(5), D(1))
```

will indicate to the compiler that the fifth element of

the array A is to occupy the same location as the tenth element of the array B, with the remaining elements similarly equated.

For arrays C and D, the EQUIVALENCE statement indicates that the fifth element of C and the first element of D are to occupy the same location, and so on for the succeeding elements. The use of the EQUIVALENCE statement thus reduces the storage required for the five arrays from 200 locations to 144 (Figure 3).

The addition of the following statement to the above illustration will cause storage allocations to be altered further as shown in Figure 4.

SUBPROGRAM STATEMENTS

Subprogram statements can only be used in Full FORTRAN. By using the Full FORTRAN language, the user can write his own subroutines that can be referred to by other programs. These subroutines, known as subprograms, may, in turn, refer to still other subprograms also written in the Full FORTRAN language.

This section discusses the two types of FORTRAN subprograms; FORTRAN FUNCTION subprograms (referred to also as FUNCTION subprogram); and SUBROUTINE subprograms. Also included is a description of four statements that are necessary for the definition and usage of these subprograms.

FUNCTION and SUBROUTINE Subprograms

Although FUNCTION subprograms and SUBROUTINE subprograms are treated together and may be viewed as similar, they differ in two fundamental respects.

1. The FUNCTION subprogram is always single-valued, whereas the SUBROUTINE subprogram may be multi-valued.
2. The FUNCTION subprogram is called or referred to by an arithmetic expression containing its name; the SUBROUTINE subprogram is referred to by a CALL statement.

Each of the two types of subprograms must be regarded as independent FORTRAN programs, and must conform to the rules for FORTRAN programming. These subprograms may be compiled separately or along with the main program by means of a multifile run.

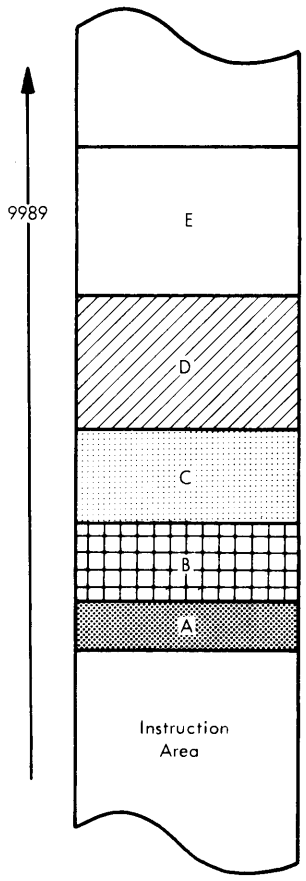


Figure 2

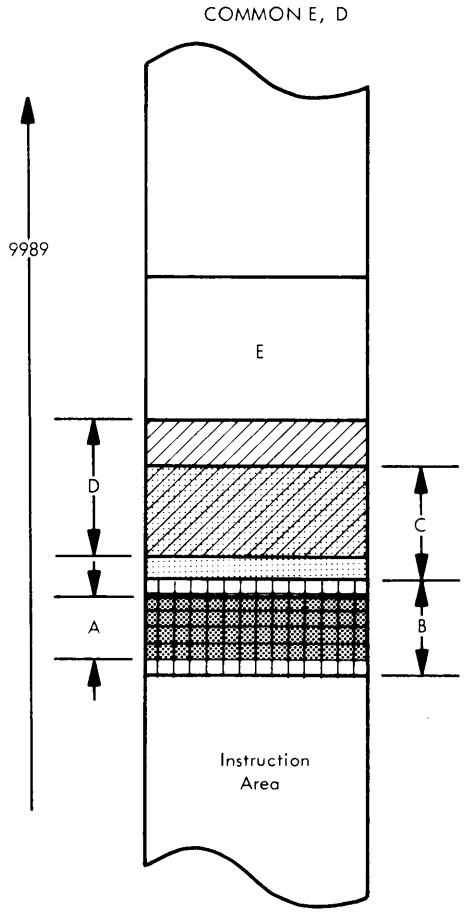


Figure 3

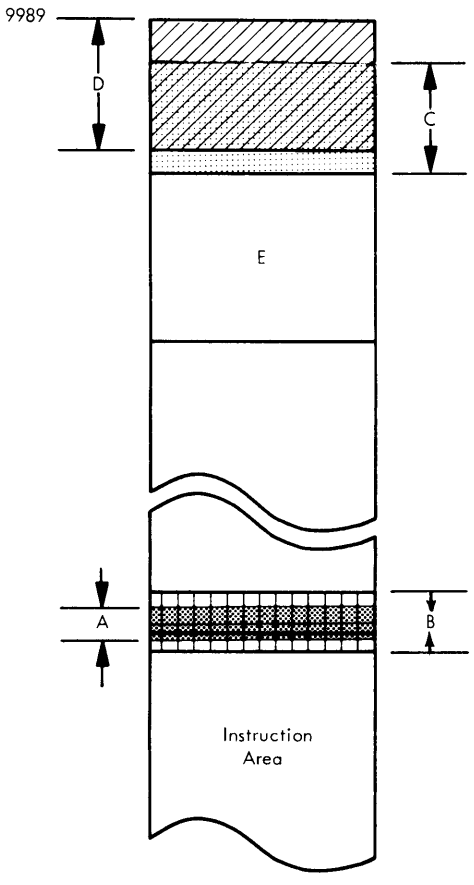


Figure 4

Schematically, the relationship between the main program and the nested subprograms is illustrated in Figure 5. The flow of control begins at "START" and continues through the numbered arrows.

FUNCTION Statement

GENERAL FORM	EXAMPLES
"FUNCTION Name (a_1, a_2, . . ., a_n)" where Name is the symbolic name of a single-valued function, and the arguments a_1, a_2, . . ., a_n, of which there must be at least one, are nonsubscripted variable names only. The function name consists of 1 to 6 alphameric characters, the first of which must be alphabetic. The first character must be I, J, K, L, M, or N if, and only if, the value of the function is to be fixed point, and the final character must not be F if there are more than three characters in the name. The function name must not occur in a DIMENSION statement in the subprogram, or in a DIMENSION statement in any program that uses the function. The arguments may be any variable names occurring in executable statements of the subprogram.	FUNCTION ARCSIN (RADIAN) FUNCTION ROOT (B. A. C) FUNCTION INTRST (RATE. YEARS)

The FUNCTION statement must be the first statement of a FUNCTION subprogram. A subprogram introduced by a FUNCTION statement must be a Full FORTRAN program and may contain any Full FORTRAN statements except SUBROUTINE or another FUNCTION statement.

In a FUNCTION subprogram, the name of the function must appear at least once as the variable on the left-hand side of an arithmetic statement or alternately, in an input statement list. By this means, the computed value of the function is returned to the calling program.

The arguments following the name in the FUNCTION statement may be considered as dummy variable names. That is, during object program execution, other actual arguments are substituted for them. Therefore, the arguments which follow the function reference in the calling program must agree with those in the FUNCTION statement in the subprogram in number, order, and mode. Furthermore, when a dummy argument is the name of an array, the corresponding actual argument must also be used as the name of an array. Each of these array names must be of the same dimension and appear in a DIMENSION statement of the respective program.

None of the FUNCTION statement arguments (i. e., the dummy variable names) may appear in EQUIVALENCE or COMMON statements in the subprogram.

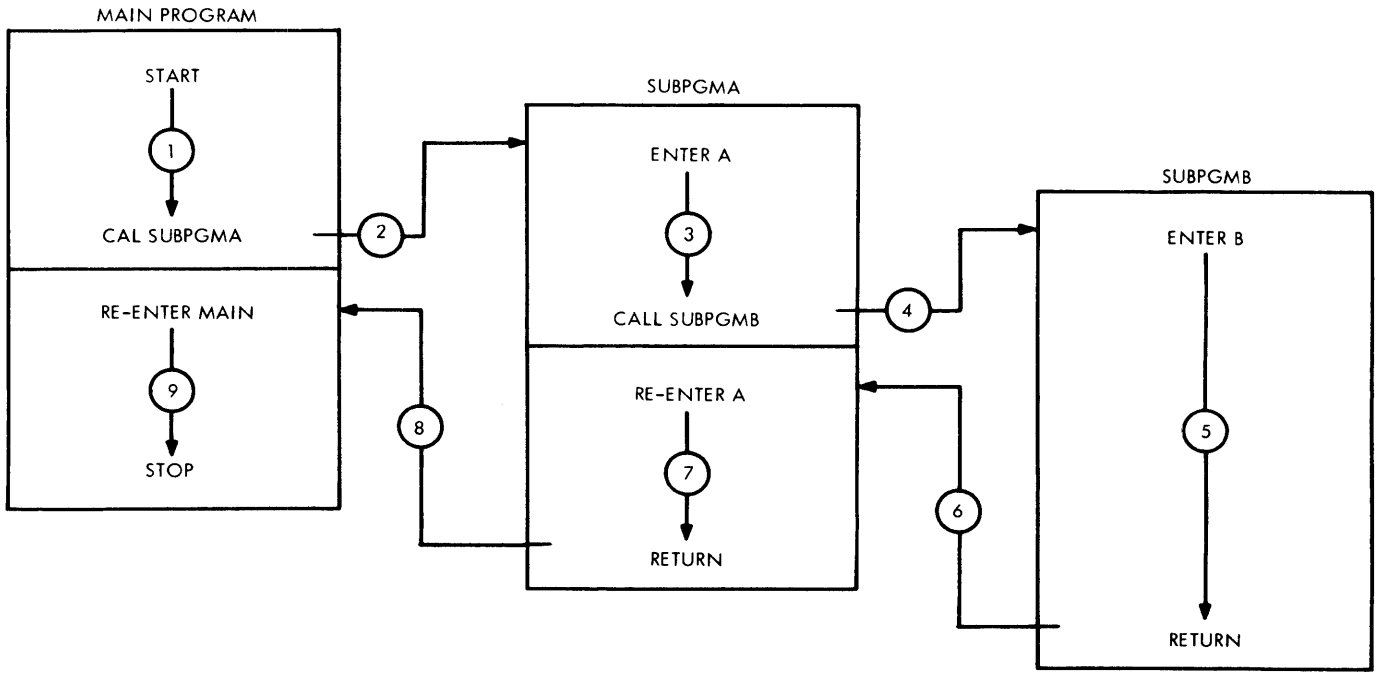


Figure 5

SUBROUTINE Statement

GENERAL FORM	EXAMPLES
"SUBROUTINE Name (a_1, a_2, . . . , a_n)" where Name is the symbolic name of a subprogram, and the arguments a_1, a_2, . . . , a_n, if any, are nonsubscripted variable names only. The name of the subprogram may consist of 1 to 6 alphameric characters, the first of which is alphabetic. Its final character must not be F if there are more than three characters in the name. Also, the name of the subprogram must not appear in a DIMENSION statement of any program which calls the subprogram, or in a DIMENSION statement of the subprogram itself. The arguments may be any variable names occurring in executable statements in the subprogram.	SUBROUTINE MATMPY (A, N, M, B, L, C) SUBROUTINE QDRTIC (B, A, C, ROOT1, ROOT2)

The SUBROUTINE statement must be the first statement of a SUBROUTINE subprogram. A subprogram introduced by a SUBROUTINE statement must be a Full FORTRAN program and may contain any Full FORTRAN statements except FUNCTION or another SUBROUTINE statement.

A SUBROUTINE subprogram must be referred to by a CALL statement in the calling program. The CALL statement specifies the name of the subprogram and its arguments.

The SUBROUTINE subprogram uses one or more of its arguments to return the computed values. These arguments, therefore, must appear on the left side of an arithmetic statement or in an input statement list within the subprogram.

The arguments of the SUBROUTINE statement are dummy variables and are replaced, at execution, by the actual arguments supplied by the CALL statement. Therefore, the arguments between the two sets must agree in number, order, and mode.

Furthermore, when a dummy argument is the name of an array, the corresponding actual argument must also be the name of an array. Each of these array names must appear in a DIMENSION statement of the respective program with the same dimensions. For example, the subprogram headed by

SUBROUTINE MATMPY (A, N, M, B, L, C)

could be called by the main program through the statement

CALL MATMPY (X, 5, 10, Y, 7, Z)

where the dummy variables, A, B, and C are the names of matrices. A, B, and C must appear in a DIMENSION statement in subprogram MATMPY and X, Y, and Z must appear in a DIMENSION statement in the calling program. The dimensions assigned must be the same in both statements.

None of the SUBROUTINE statement arguments (i. e., the dummy variable names) may appear in the EQUIVALENCE or COMMON statements in the subprogram.

The SUBROUTINE subprogram must not contain more than one entry point. However, one SUBROUTINE subprogram may be written to call another SUBROUTINE subprogram. This approach can usually be used to get the same effect as one routine having more than one entry point.

CALL Statement

GENERAL FORM	EXAMPLES
"CALL Name (a_1, a_2, . . . , a_n)" where Name is the name of a SUBROUTINE subprogram and a_1, a_2, . . . , a_n are arguments which take one of the forms described below.	CALL MATMPY (X, 5, 10, Y, 7, Z) CALL QDRTIC (P*9. 732, Q/4. 536, R - S**2. 0, X1, X2)

This statement is used only to call SUBROUTINE subprograms; the CALL transfers control to the subprogram and presents it with the parenthesized arguments. Each argument may be one of the following:

1. Fixed-point constant.
2. Floating-point constant.
3. Fixed-point variable, with or without subscripts.
4. Floating-point variable, with or without subscripts.
5. Arithmetic expression.
6. Alphameric characters. Such arguments must be preceded by nH where n is the count of characters included in the argument; e.g., 9HEND POINT. Note that blank spaces and special characters are considered characters when used in alphameric fields. Alphameric arguments can, of course, be used only as input to hand-coded programs.

The arguments presented by the CALL statement must agree in number, order, mode and array size with the corresponding arguments in the SUBROUTINE statement of the called subprogram.

RETURN Statement

GENERAL FORM	EXAMPLES
"RETURN"	RETURN

This statement terminates the execution of a subprogram and returns control to the calling program. A RETURN statement must be the last subprogram statement to be executed although not necessarily the final statement in the series of instructions of the subprogram. Any number of RETURN statements may be used in a subprogram.

PART III: EXPANDING THE FORTRAN SYSTEM

The FORTRAN system may be expanded by using subprograms and library-function subroutines in conjunction with the IBM 7070/72/74 FORTRAN Loader/Package program.

The FORTRAN Package is a set of subroutines used as part of the object program. These subroutines are loaded by the FORTRAN Loader prior to the object program, and appear in storage at a predetermined location.

The IBM 7070/72/74 FORTRAN Loader/Package is distributed as one program. However, the loader portion and the package portion function separately and are discussed as such in this section.

THE FORTRAN LOADER

When additional routines are written and compiled for inclusion in the main program, FORTRAN uses the principle of relocatability to ensure the proper assignment of storage locations. In FORTRAN, relocatability may be considered as a feature that allows a number of distinct routines to be compiled independently of each other, and yet operate simultaneously and communicate with each other as needed. The FORTRAN Loader is relocatable and performs the following functions:

1. Loads the FORTRAN package.
2. Loads the main program and subprograms sequentially into core storage.
3. Records the number of routines loaded and the number of locations each routine requires.
4. Notes the address of the first location (the base address) of each routine as it was originally assembled.
5. Computes new addresses for each routine in order to place the routine in any part of storage as needed.
6. Constructs linkages where needed so that the routines can communicate with one another.

Requirements of Relocatable Routines

Each routine handled by the FORTRAN Loader must meet the following requirements:

1. Every routine loaded must be preceded by a Title Card.
2. If a routine calls or refers to any other routine (subroutine or subprogram), the calling program cards must include a Transfer Entry Card for each called routine.
3. Each routine loaded must have an Execute Card as its last card.
4. Each routine loaded by the FORTRAN Loader must be punched in the standard Autocoder Relocatable Condensed Card format.
5. Each routine that calls another routine must have space reserved in it for a Branch List which is constructed by the FORTRAN Loader using the Transfer Entry Card.

All necessary Title Cards, Transfer Entry Cards, Branch Lists, etc., are produced by the FORTRAN processor at the time of compilation.

The Library subroutines include the necessary Branch Lists, Title Cards, and Transfer Entry Cards that are required by the FORTRAN Loader.

The Title Card

The Title Card is compiled as the first card of every routine processed by FORTRAN. It must be supplied by the user for every routine loaded by the Loader which has not been compiled by the FORTRAN processor. This card may be supplied either in symbolic form at compilation time or in actual form at object time. Each form of Title Card is described separately below.

Symbolic Title Card

Label	Operation	Operand	Comments
EXECUTE	CNTRL	7	
	DC	@ NAME @	1. Name of the routine, left-justified, in double-digit form.
		+000000LLLL	2. Number of locations to be reserved for the data area immediately following the last instruction of the routine (right-justified).
		+000000TTTT	3. Number of locations to be reserved at the top of storage for the common data area (right-justified).
		+000000BBBB	4. Base address (right-justified).
		+0000000000	5. Contents of this word may be any number.
		+910000000d	6. Value of d is 0, 1 or 2 as explained under "Relocation Indicators."

To include identification in the Title Card, change the operand of the EXECUTE CNTRL entry to 8 and add a subsequent entry under the DC.

Although these entries do not result in a program card, the assignment counters of the assembly program are increased when EXECUTE CNTRL entries are processed; therefore, they should be followed by an ORIGIN CNTRL entry to resume processing at the proper location.

Actual Title Card

Card Columns	Contents
1-20	Name of the routine, left-justified, in double-digit form (see example below).
21-30	Number of locations to be reserved for the data area immediately following the last instruction of the routine (right-justified).
31-40	Number of locations to be reserved at the top of storage for the common data area (right-justified).
41-50	Base address (right-justified).
51-60	The contents of this word may be any number.
61-62	Must contain 91 which identifies this card as Title Card to the FORTRAN Loader.
63-69	Zeros.
70	Digit 0, 1 or 2 as explained under "Relocation Indicators."
71-75	Card identification number.
76-80	Program identification.
10, 20, 30, 40, 50, 60, 70, 79, 80	12-punch in each column.

Example: A subroutine called SINF would be entered into columns 1-20 in this form:

82 69 75 66 00 00 00 00 00 00
S I N F b b b b b b

For a complete list of double-digit codes, see Appendix B.

The Transfer Entry Cards

Transfer Entry Cards are used by the Loader to complete linkages between routines after they have been loaded into storage. If a routine uses or calls no other routines, no Transfer Entry Cards are included. This card may be supplied either in symbolic form at compilation time or in actual form at object time. Each form of Transfer Entry Card is described separately below.

Symbolic Transfer Entry Card

Label	Operation	Operand	Comments
EXECUTE	CNTRL	7	
	DC	@NAME @	1. Name of the routine to be called, left-justified, in double-digit form.
		+0000000000	2. Contents of this word may be any number.
		+0000000000	
		+0000000000	
		+0000000000	
		+9200000000	

To include identification in the Transfer Entry Card, change the operand of the EXECUTE CNTRL entry to 8 and add another subsequent entry under the DC.

Although these entries do not result in a program card, the assignment counters of the assembly program are increased when EXECUTE CNTRL entries are processed; therefore, they should be followed by an ORIGIN CNTRL entry to resume processing at the proper location.

Actual Transfer Entry Card

Card Columns	Contents
1-20	Name of the routine to be called, left-justified, in double-digit form (see example above).
21-30	Contents of this word may be any number.
31-60	Zeros.
61-62	Must contain 92 (together with the zero in column 65) to identify to the FORTRAN Loader that this is a Transfer Entry Card.
63-70	Zeros.
71-75	Card identification number.
76-80	Program identification.
10, 20, 30, 40, 50, 60, 70, 79, 80	12-punch in each column.

The Execute Card

Every routine loaded by the Loader must have an Execute Card as its last card. This card is processed automatically for programs compiled by the

FORTRAN processor. The format of the Execute Card is as follows:

Card Columns	Contents
1-10	A branch to the first instruction of the routine.
11-55	Zeros.
56	The relocation indicator for the Branch instruction in columns 1-10.
57-60	Zeros.
61-62	Must contain zeros together with the zero in column 65 to indicate to the FORTRAN Loader that this is an Execute Card.
63-70	Zeros.
71-75	Card identification number.
76-80	Program identification.
10, 20, 30, 40, 50, 60, 70, 79, 80	12-punch in each column.

The End-of-File Card

The End-of-File Card allows object program data to be placed into a card reader at the same time as a program deck. This card is placed at the end of a program followed immediately by object program data that is to be read from the same input unit. The FORTRAN Loader regards the End-of-File card as a substitute for a physical end-of-file condition. Similarly, on card-to-tape operation, the use of the End-of-File Card eliminates the need for writing a tape mark between the program and the data for the object program.

The format of the End-of-File Card is as follows:

Card Columns	Contents
1-60	Zeros.
61-62	Must contain 93, which, together with the zero in column 65, identifies this card to the Loader as an End-of-File Card.
63-80	Zeros.
10, 20, 30, 40, 50, 60, 70, 79, 80	12-punch in each column.

There is no symbolic form of the End-of-File Card because the assembly program will not put out anything following the Execute Card which ends a program. The End-of-File Card is never produced during a FORTRAN compilation.

NOTE: When a relocated program deck or tape (produced by the FORTRAN Loader) is being loaded, an End-of-File Card must not be used. Instead, the relocated program should be loaded by the standard Condensed Card Load Program. If the data for the object program is to be read from the same unit, it should be placed immediately after the relocated program. The execute card produced at the end of the relocated program will cause the object program to be executed without interruption after it is loaded.

The Data-Area Card

The Data-Area Card is produced only by the Basic FORTRAN processor after the entire program has been processed. This card indicates to the loader the number of locations to be reserved for data, and by its position in the compiled program, also indicates where the data area begins.

NOTE: Since Basic FORTRAN is a one-pass Compiler, it must put out the Title Card at the beginning of the compilation (even though it does not contain all the required information). The Title Cards produced by Basic FORTRAN always contain zeros in the data field (columns 21-30).

The symbolic and actual forms of the Data-Area Card are described separately below:

Symbolic Data-Area Card

Label	Operation	Operand	Comments
EXECUTE	CNTRL		
	DC	+000000LLLL	(The number of locations to be reserved for the data area.)
		+0000000000	
		+0000000000	
		+0000000000	
		+0000000000	
		+0000000000	
		+9400000000	(Identifies this as a Data-Area Card.)

To include identification in the Transfer Entry Card, change the operand of the EXECUTE CNTRL entry to 8 and add a subsequent entry under the DC.

Although these entries do not result in a program card, the assignment counters of the assembly programs are increased when EXECUTE CNTRL entries are processed. The EXECUTE CNTRL entries, therefore, should be followed by an ORIGIN CNTRL entry to resume assignment at the proper location.

NOTE: Basic FORTRAN puts out the DC subsequent entries all on one line, with high-order zeros omitted, as shown below:

+LLLL+0+0+0+0+0+9400000000

Actual Data-Area Card

Card Columns	Contents
1-10	The number of locations to be reserved for the data area immediately following the highest location used up to this point (right-justified).
11-60	Zeros.
61-62	Must contain 94, which, together with the zero in column 65, identifies this to the FORTRAN Loader as a Data-Area Card.
63-70	Zeros.
71-75	Card identification number.
76-80	Program identification.
10, 20, 30, 40, 50, 60, 70, 79, 80	12-punch in each column.

Whenever the Data-Area Card is used, the information it provides regarding the size of the data area (columns 1-10) override the information in the corresponding field of the Title Card (columns 21-30). The Data-Area Card also introduces a new assumption about the location of the data area. In the absence of this Card, the FORTRAN Loader assumes that the data area, if any, is at the end of the program, as it is in all programs compiled by Full FORTRAN. However, when the FORTRAN Loader encounters a Data-Area Card, it assumes that the data area begins immediately after the highest location used up to that point, which may or may not be the end of the program. In programs compiled by Basic FORTRAN, the literals follow the data area; this means that only programs that produce no literals have the data area at the end.

Relocatable Program Cards

The FORTRAN Loader loads object programs and subroutines in relocatable form. Relocatable program cards are punched with up to five contiguous instructions of the routine. The relocatable codes of each instruction, and the relocatable address of the instruction in word 1 are also punched. The card format is as follows:

Card Columns	Contents
1-10	Instruction one.
11-20	Instruction two.
21-30	Instruction three.
31-40	Instruction four.
41-50	Instruction five.
51-60	The relocation codes for words 1, 2, 3, 4 and 5, respectively (see below).
61-64	Zeros.
65	The number of instructions (1-5) punched in the card. If this column is punched with zeros, it is identified by the FORTRAN Loader as an Execute Card.
66	Zero.
67-70	The address of the instruction in word 1 as it was originally assembled.
71-75	Card identification number.
76-80	Program identification.
10, 20, 30, 40, 50	11- or 12-punch in each column.
60, 70, 79, 80	12-punch in each column.

Relocation Indicators

Relocation indicators are necessary because whenever the location of an instruction, or constant, or any data is changed, all references to that location which occur in other instructions must also be changed. The relation indicators for a given instruction tell the FORTRAN Loader which portions of that instruction are to be changed.

Instructions punched in the standard Autocoder condensed card format are limited to five instructions per card with word six (columns 51-60) reserved for relocation indicators.

The FORTRAN Loader recognizes either of two sets of relocation indicators; the set produced by Four-Tape Autocoder and Basic Autocoder (also produced by Autocoder prior to October 1961), and the set produced by Autocoder starting in October 1961.

In the following discussion of Relocation indicator:

I_1 represents the relocation increment for the upward relocation of programs and ordinary data (it may be negative).

I_2 represents the relocation decrement for the downward relocation of common area and the FORTRAN Loader itself (it is never negative).

The values of I_1, and I_2 are computed by the FORTRAN Loader. I_1, and I_2 are used to adjust the addresses in a routine being loaded.

The value of I_1 is computed separately for each program. This value is equal to the relocated starting location of the program minus the original starting location as found in the compilation listing or in the Title Card.

The value of I_2 is the same for all programs in a given run. It is equal to 9989 (the compiled top of common area for all FORTRAN programs) minus the top of storage for the run, as indicated to the Loader by the user.

If it is desired to compare a relocated program with the original program, the original location of an instruction may be found by subtracting I_1 from the relocated address of the instruction. This address appears on the storage map produced by the typewriter with Alternation Switch No. 3 ON, or in a listing of the relocated deck or tape optionally produced by the FORTRAN Loader.

The original values of the relocated instructions depend on the relocation indicators, and can be calculated as follows:

1. For upward relocation indicators, subtract the value of I_1 from the part(s) of the instruction that the relocation indicator shows has been adjusted.
2. For downward relocation indicators, add the value of I_2 to the part(s) of the instruction that the relocation indicator shows has been adjusted.

Relocation Indicators Produced by Four-Tape Autocoder and Basic Autocoder

This set of relocation indicators will be produced for the following types of programs:

1. Basic Autocoder programs or the symbolic output of Basic FORTRAN programs processed by Basic Autocoder starting in October 1961. These programs can be identified by a +2 in column 70. Programs processed by Basic Autocoder before October 1961 contain a +0 in column 70 and do not contain any relocation indicators.
2. Four-Tape Autocoder programs or the symbolic output of Basic FORTRAN programs processed by Four-Tape Autocoder. These programs can be identified by +0, +1, or +2 in column 70. Programs having +0 in column 70 begin with a Branch instruction that bypasses the Branch List located at the beginning of the program. Programs having +1 or +2 in column 70 begin following the Branch List and thereby eliminate the need for a Branch instruction to bypass the Branch List. The Branch List provides linkages with subprograms and subroutines.
3. FORTRAN programs processed before October 1961. These programs can be identified by a +0 in column 70.
4. Autocoder programs or subroutines processed by Autocoder before October 1961. These programs can be identified by a +0 in column 70.

The relocation indicators produced for the programs listed in 1 through 4 above and their location in the card are as follows:

Card Columns	Relocation Indicator for:
56	Word 1, card columns 1-10
57	Word 2, card columns 11-20
58	Word 3, card columns 21-30
59	Word 4, card columns 31-40
60	Word 5, card columns 41-50

Let I_1 be the relocation increment, and I_2 be the relocation decrement. I_1 and I_2 are numbers formed by the FORTRAN Loader and are used to adjust the addresses in a routine being loaded. The meanings of the codes in card columns 56 through 60 are as follows:

Value	Meaning
0	Do not alter instruction.
1	Add I_1 to digit positions 6-9.
2	Add I_1 to digit positions 2-5.
3	Add I_1 to digit positions 2-5 and 6-9.
5	Subtract I_2 from digit positions 6-9.
6	Subtract I_2 from digit positions 2-5.
7	Subtract I_2 from digit positions 2-5 and 6-9.

I_1 is always added to the contents of card columns 67-70, which gives the relocatable location of the instruction in word 1.

NOTE: Relocation indicators 5, 6, and 7 are produced only by Four-Tape Autocoder.

Relocation Indicators Produced by Autocoder

This set of relocation indicators will be produced for the following types of programs:

1. FORTRAN programs processed from October 1961. These programs can be identified by a +1 in column 70.
2. The symbolic output of Basic FORTRAN programs processed by Autocoder from October 1961. These programs can be identified by a +1 in column 70.
3. Autocoder programs or subroutines processed by Autocoder from October 1961. These programs can be identified by a +1 in column 70.

The relocation indicators produced for the programs listed in 1 through 3 above and their location in the card are as follows:

Card Columns	Relocation Indicator for:
51, 56	Word 1, card columns 1-10.
52, 57	Word 2, card columns 11-20.
53, 58	Word 3, card columns 21-30.
54, 59	Word 4, card columns 31-40.
55, 60	Word 5, card columns 41-50.

Left-Hand Digit

The following table lists the meanings of the digits punched in card columns 51 through 55:

Value	Meaning
0	Do not alter instruction.
1	Add I_1 to digit positions 1-5.
3	Add I_1 to digit positions 2-5.
4	Add I_1 to digit positions 0-4.
5	Subtract I_2 from digit positions 1-5.
7	Subtract I_2 from digit positions 2-5.
8	Subtract I_2 from digit positions 0-4.

Right-Hand Digit

The following table lists the meanings of the digits punched in card columns 56 through 60:

Value	Meaning
0	Do not alter instruction.
1	Add I_1 to digit positions 6-9.
2	Used in additional-storage mode only. Add I_1 to digit positions 3 and 6-9.
4	Add I_1 to digit positions 5-9.
5	Subtract I_2 from digit positions 6-9.
6	Used in additional-storage mode only. Subtract I_2 from digit positions 3 and 6-9.
8	Subtract I_2 from digit positions 5-9.

THE FORTRAN PACKAGE

The FORTRAN Package is not relocatable and contains subroutines that are loaded into storage beginning at location 0325. The following subroutines are included in this package:

1. Input/Output Subroutines -- These subroutines are used by the object program to read in data required for processing and to write or punch out results of calculations. The input/output subroutines are of two types: those for unit record input/output equipment and those for tape input/output.

2. Exponentiation Subroutines -- These subroutines perform the three types of exponentiation permissible in an arithmetic expression. Quantities may be raised to powers as follows:

FLT^{FLT} FLT^{FIX} FIX^{FIX}

3. Function Subroutines -- The library function subroutines in the package are:

LOGF	$(\log_e x)$
LOGXF	$(\log_{10} x)$
EXPF	(e^x)
EXPXF	(10^x)
FLOATF	(fixed-point to floating-point conversion)
XFIXF	(floating-point to fixed-point conversion)
XINTF	(same as XFIXF)

4. Floating-Decimal Arithmetic Subroutines -- These subroutines perform the floating-point add, subtract, multiply, and divide required by the object program on machines not equipped with the floating device. On machines with the floating device, these subroutines are omitted from the package (except for a few instructions which check for a zero divisor before a floating divide).

In general, the order of input to the object machine consists of the following decks in the indicated order.

1. The FORTRAN Loader.
2. The FORTRAN Package.
3. The compiled main program; any subroutines; and any required function or subprograms placed in any order. (See discussion below for the exact requirements of hand-coded subroutines to be used as FORTRAN library functions.)

AUTOCODER ROUTINES FOR USE WITH FORTRAN PROGRAM

In a FORTRAN system, the user can code his own library function subroutines and subprograms in the 7070-Series Autocoder languages. These routines may then be used with a Full FORTRAN program. However, when such routines* are coded in the Autocoder language, the following conventions must be observed:

Library Functions

When library functions are written, the called program must assume:

1. The conditions of all indicators, electronic switches, and index registers used in the subroutine must be saved, and then restored after the subroutine is executed. Also, if accumulators are being used in the subroutine, their contents must be saved, as well as their overflow-underflow conditions. The machine is assumed to be in the "sense" mode for the Field Overflow and Sign Change latches. However, the subroutine may put the machine into stop mode at any time, providing the sense mode is restored before exit is made.

2. Priority masks, and latches such as the High, Low, Equal, or Zero latch may be used freely and changed at will. Control must not be returned to a compiled FORTRAN program in the priority mode.

Arguments

The rules governing the location of arguments in a user's library function subroutine are given below.

*In addition, these routines must conform, at object time, to specifications discussed earlier in this section.

Single Argument Library Functions

Upon entry into the library function subroutine, the argument will be in:

1. Accumulator 1 if the mode of the argument is floating-point.
2. Accumulator 2 if the mode of the argument is fixed-point.

Upon exit from the library function subroutine, the result must be in:

1. Accumulator 1 if the mode of the function is floating-point.
2. Accumulator 2 if the mode of the function is fixed-point.

Entry into the library function subroutine is effected by the following instruction in the object program:

```
BLX   94, NAME
```

where NAME is the designation of the called routine. Return to the main program from the subroutine should be effected by the following instruction in the subroutine:

```
B   0 + X94
```

Multiple Argument Library Functions

As in the case of a single argument function, entry into a subroutine evaluating a function of multiple arguments is by means of a BLX 94, NAME from the object program. The locations in storage immediately following this instruction contain the addresses of the arguments in the sequence in which they appear in the source statement. Thus, if the statement

```
ANYF   (A, B, C, D)
```

appeared in the source statement, 0 + X94 would contain the address of the first argument, (A), 1 + X94 would contain the address of the second argument, (B), etc.

The return to the main program is accomplished by a branch to n + X94, where n is the number of arguments in the library function.

As in the case of the library function of a single argument, the result of the multiple argument subroutine must be in:

1. Accumulator 1 if the mode of the function is floating-point.
2. Accumulator 2 if the mode of the function is fixed-point.

Branch List

All programs which call other routines must contain a Branch List. Thus, a main program which refers to any library function subroutines or any subprograms must have a Branch List. If a hand-coded program refers to any other routines, the programmer must provide space for a Branch List. This Branch List is filled in with branch instructions by the FORTRAN Loader after it places various routines in storage. The Branch List has one storage location for each different routine called, regardless of the number of times it is called.

When a routine transfers control to another routine, it does so by a BLX 94 to the location in the Branch List which has been previously filled in by the FORTRAN Loader. The branch instruction in this location, in turn, transfers control to the called subroutine. The order in which branch instructions appear in the Branch List is specified in the Transfer Entry Cards.

The Branch List can be placed anywhere in a routine. The Full FORTRAN processor compiles a Branch List at the beginning of a program, while the Basic FORTRAN processor places it immediately after the last executable instruction of a program.

The Branch List in a hand-coded routine need not come at the beginning of a program, nor need it be in contiguous locations. However, the Transfer Entry Card for a particular subroutine must be at a point in the calling program that is not overlaid by data or an instruction. The FORTRAN Loader will place a branch to the subroutine in question at this point. Also, the first executable instruction may occupy any location used by the program (the Execute Card provides the FORTRAN Loader with this location, whereas the Title Card provides the lowest location used by the program, which may or may not be the same).

The following example illustrates how the programmer can provide space for a Branch List as he codes a subroutine using an Autocoder system:

```
Label      Operation    Operand
EXECUTE    CNTRL        7
           DC
                        @ SUBAF   @
                        +0000000050
                        +0000000010
                        +0000000325
                        +0000000000
                        +9100000001
EXECUTE    CNTRL        7
           DC
                        @ SUBBF   @
                        +0000000000
                        +0000000000
                        +0000000000
                        +0000000000
                        +9200000000
```

```
Label       Operation    Operand
EXECUTE     CNTRL        7
            DC
                         @ SUBCF    @
                         +0000000000
                         +0000000000
                         +0000000000
                         +0000000000
                         +9200000000
ORIGIN      CNTRL        325
            DA           1
SUBBF                    00,09
SUBCF                    10,19
SUBAF       XXX          XXXXXXXXXX
  .           .            .
  .           .            .
  .           .            .
  .         BLX          94,SUBBF
  .         HB           ARG1
  .         HB           ARG2
  .         HP           ARG3
  .           .            .
  .           .            .
  .           .            .
  .         BLX          94,SUBCF
  .           .            .
  .           .            .
  .           .            .
  .         HB             *
```

SUBBF may be in the following form:

```
Label       Operation    Operand
EXECUTE     CNTRL        7
            DC
                         @ ASUBBF   @
                         +0000000000
                         +0000000000
                         +0000000325
                         +0000000000
                         +9100000001
ORIGIN      CNTRL        325
SUBBF       XXX          XXXXXXXXXX
  .           .            .
  .           .            .
  .           .            .
  .         B            3+X94
```

SUBCF may be in the following form:

```
Label       Operation    Operand
EXECUTE     CNTRL        7
            DC
                         @ SUBCF    @
                         +0000000000
                         +0000000010
                         +0000000325
                         +0000000000
                         +9100000001
EXECUTE     CNTRL        7
            DC
                         @ SUBBF    @
                         +0000000000
                         +0000000000
                         +0000000000
                         +0000000000
                         +9200000000
```

```
Label       Operation    Operand
ORIGIN      CNTRL        325
            DA           1
SUBBF                    00,09
SUBCF       XU           94,STORE94
  .           .            .
  .           .            .
  .           .            .
  .         BLX          94,SUBBF
  .         HB           ARG1
  .         HB           ARG2
  .         HP           ARG3
  .           .            .
  .           .            .
  .           .            .
  .         XL           94,STORE94
  .         B            3+X94
```

The Transfer Entry Cards required for the foregoing example would be as follows:

SUBAF Object Deck

The routine SUBAF must contain a Transfer Entry Card for SUBBF as well as one for SUBCF.

SUBBF Object Deck

The routine SUBBF requires no Transfer Entry Cards.

SUBCF Object Deck

The routine SUBCF must contain a Transfer Entry Card for SUBBF.

The above discussion of the Branch List has been in terms of library function subroutines. However, any routine which is handled by the FORTRAN Loader and which calls another routine must have a Branch List. Each routine that a FORTRAN compiler processes is provided automatically with all necessary Branch Lists, Title Cards, and Transfer Entry Cards.

Coding Subprograms for Use with Full FORTRAN Program

When a hand-coded routine is to be used as a subprogram, the programmer must follow the conventions used by FORTRAN (i.e., transferring of arguments, saving of index words, etc.), and duplicate what would be compiled for the FORTRAN FUNCTION, SUBROUTINE, and RETURN statements.

All indicators, relocation switches, and index words used in a subprogram must be saved and then restored after the subprogram is executed. The machine is assumed to be in the "sense" mode for the Field Overflow and Sign Change latches. This

may be changed by the subprogram, but must be restored before exit is made.

An example of subprogram calling is given below.

MAINPROG calls SUBPGM and uses library functions ANYF and XANYF. The routine SUBPGM is hand-coded and uses library function ANYF and has five arguments. See Figure 6 for the schematic view of the flow of control between MAINPROG and SUBPGM. (For clarity, the library functions have been eliminated from the schematic.)

The following is the coding of MAINPROG:

```
Label       Operation   Operand                Comment
EXECUTE     CNTRL       7
            DC
                        @ MAINPROG @
                        +0000000150
                        +0000000025
                        +0000000325
                        +0000000000
                        +9100000001
EXECUTE     CNTRL       7
            DC
                        @ SUBPGM  @
                        +0000000000
                        +0000000000
                        +0000000000
                        +0000000000
                        +9200000000
EXECUTE     CNTRL       7
            DC
                        @ ANYF    @
                        +0000000000
                        +0000000000
                        +0000000000
                        +0000000000
                        +9200000000
EXECUTE     CNTRL       7
            DC
                        @ XANYF   @
                        +0000000000
                        +0000000000
                        +0000000000
                        +0000000000
                        +9200000000
ORIGIN      CNTRL       325
            DA          1
SUBPGM                  00,09                  1
ANYF                    10,19
XANYF                   20,29
MAINPROG    XXX         XXXXXXXXXX             2
  .           .           .
  .           .           .
  .           .           .
  .         BLX         94, ANYF               3
  .           .           .
  .           .           .
  .           .           .
  .         BLX         94, SUBPGM             3
  .         XZA         1, A                   4
  .         XZA         2, B
  .         XZA         3, C
  .         XZA         4, D
  .         XZA         5, E
  .         BLX         94,0+X94
  .         XXX         XXXXXXXXXX             5
```

```
Label       Operation   Operand                Comment
  .           .           .
  .           .           .
  .           .           .
            HB          *
```

Coding for SUBPGM might appear as follows:

```
Label       Operation   Operand                Comment
EXECUTE     CNTRL       7
            DC
                        @ SUBPGM  @
                        +0000000010
                        +0000000005
                        +0000000325
                        +0000000000
                        +9100000001
EXECUTE     CNTRL       7
            DC
                        @ ANYF    @
                        +0000000000
                        +0000000000
                        +0000000000
                        +0000000000
                        +9200000000
ORIGIN      CNTRL       325
            DA          1
ANYF                    00,09                  1
SUBPGM      XZA         98,1                   6
            RS          98,IWSTORAGE
            BLX         94,0+X94
            XU          94,SAVEEXIT            7
  .           .           .
  .           .           .
  .           .           .
            BLX         94, ANYF               3
  .           .           .
  .           .           .
  .           .           .
            B           RESETIW                8
IWSTORAGE   DRDW        -*+1,*+6
            DA          1
                        00,59                  9
SAVEEXIT                60,69
RESETIW     XZA         98,1                   10
            RG          98,IWSTORAGE
            XL          94,SAVEEXIT
            B           0+X94
```

Comments

1. Branch List which is filled by the FORTRAN Loader.
2. First executable instruction of MAINPROG.
3. A branch to the Branch List which, in turn, transfers control to the appropriate routine.
4. Temporary re-entry into MAINPROG. A, B, C, D, E are the addresses of the arguments appearing in the CALL statement. MAINPROG inserts them into index words beginning with word 1, which makes them available to SUBPGM, and then returns to SUBPGM.
5. Re-entry point of MAINPROG.

6. Entry point of SUBPGM, where the subprogram saves the contents of the index words it requires, and then returns temporarily to MAINPROG.

7. Re-entry into SUBPGM, which begins its computation after saving the contents of index word 94, for use in returning to MAINPROG. The contents of index word 94 must be saved because it is used by SUBPGM to go to ANYF.

8. End of SUBPGM.

9. Space is reserved to store the contents of index words used by SUBPGM, and the re-entry location of MAINPROG.

10. A routine executed after the SUBPGM has been completed. It resets the index words used by the subprogram, restores the re-entry location of MAINPROG to index word 94, and returns to MAINPROG.

NOTE: A FORTRAN FUNCTION subprogram can have only a single result. Thus, upon exit from a FORTRAN FUNCTION subprogram, the result must be left in:

Accumulator 1, if it is floating-point
Accumulator 2, if it is fixed-point

A SUBROUTINE subprogram can have more than one result. The names of these results are included in the argument lists of the CALL statement in the calling program and the SUBROUTINE statement in the called program.

The following schematic diagram shows MAINPROG and SUBPGM as they appear in storage. Numbers in parentheses correspond to comment numbers in the coding example. Arrows indicate the flow of control (beginning with Start).

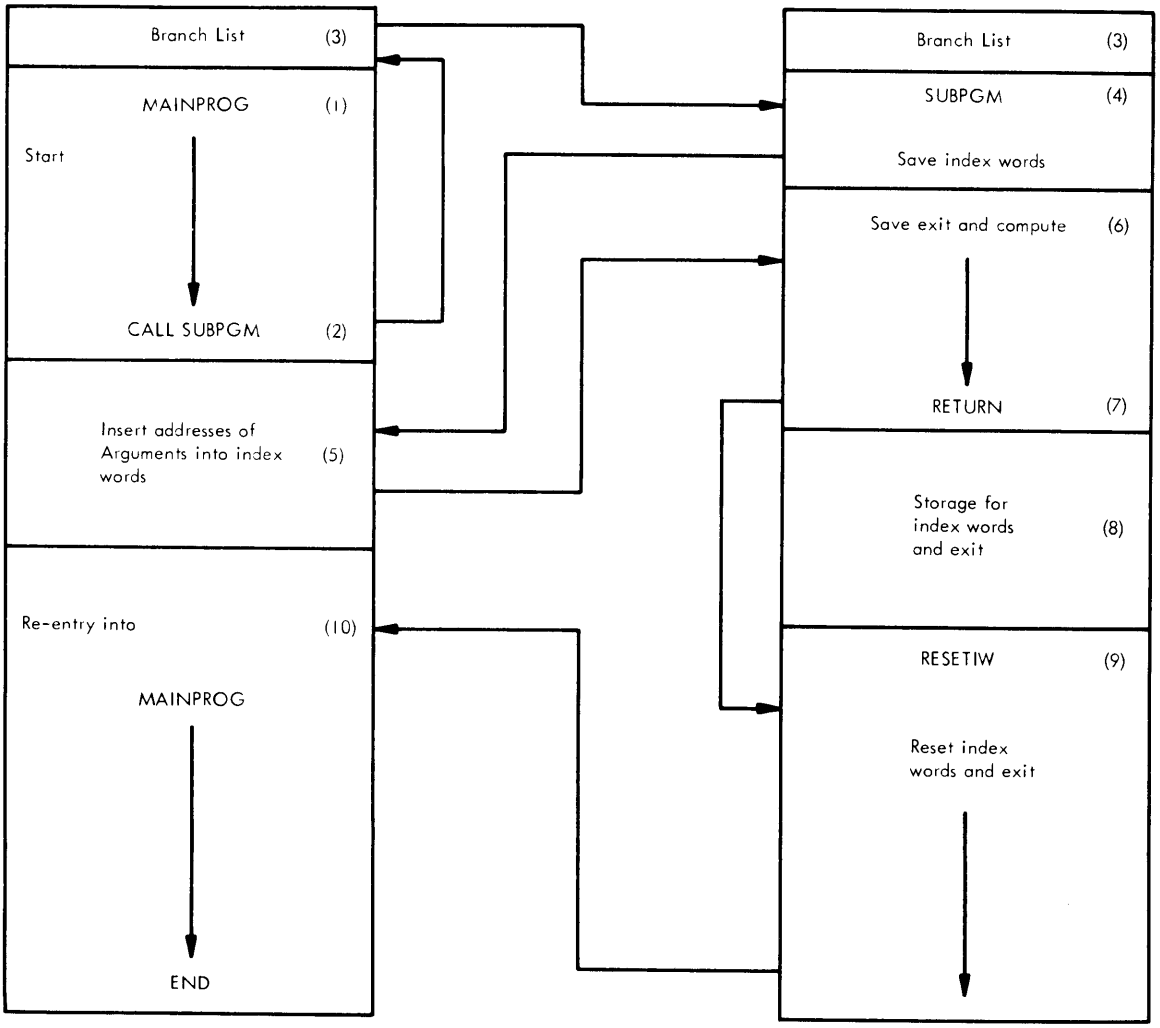


Figure 6

FIXED-POINT ARITHMETIC

The use of fixed-point arithmetic is governed by the following considerations:

1. Fixed-point constants specified in the source program must have magnitudes that are less than 10^{10}, or 10^4 if used as a subscript.
2. Fixed-point data read in by the object program itself is treated modulo 10^{10}.
3. The output from fixed-point arithmetic in the object program is modulo 10^{10}. However, if, during computation of a fixed-point arithmetic expression, an intermediate value occurs which is greater than 10^{10}, it is possible that the final result will be inaccurate even though modulo 10^{10}.
4. Indexing in the object program is modulo 10^4.
5. Any fixed-point number raised to the zero power produces an answer of one.
6. Any fixed-point number raised to a negative fixed-point power other than zero produces an answer to zero.

TRUNCATION DURING COMPUTATION

The partial or intermediate results during the computation of an arithmetic expression are truncated, if necessary. The resultant error depends on the error in which the computations and truncations take place. To insure accuracy of fixed-point multiplication and division, it is suggested that parentheses be inserted into the expression involved.

Examples:

Fixed-point Expression	Truncated Result
((5*3)/2)	7
((5/2)*3)	6
(-(5*3)/2)	-7
(-(5/2)*3)	-6

RELATIVE CONSTANTS

A relative constant is defined as a variable in a subscript, which is not under control of a DO, or a DO-implying parenthesis in a list. For example, in the sequence:

```
A        =   B(K)
DO 10 I  =   1, 10
X        =   B(I) + C(I, 3*J+2)
```

K and J are relative constants, but I is not.

The appearance of a relative constant in any of the following ways will be called a relative constant definition.

1. On the left side of an arithmetic statement.
2. In the list of an input statement.
3. As an argument for a FORTRAN FUNCTION or SUBROUTINE subprogram.
4. In a COMMON statement.

Relative Constants in an Argument List*

A variable defined in one program may have its value transmitted to another program where it is relative constant and where, consequently, the value is used. This may be done by placing it in an argument list. The appearance of a relative constant in an argument list is sufficient to provide the necessary computation for the relative constant.

Relative Constants in COMMON Statements*

A relative constant value may be transmitted from one program to another by placing it in the common area, but only if it is being transmitted from the calling to the called subprogram.

Example:

Main Program

```
      .
      .
      .
COMMON K

K = 5
CALL ABC
      .
      .
      .
      .
```

```
      SUBROUTINE ABC
      COMMON I
      DIMENSION B(10)
      A - B(I)
            .
            .
            .
            .
```

LIMITATIONS ON SOURCE PROGRAM SIZE

During compilation, the FORTRAN processors internally store certain kinds of information about the source program. Areas of core storage (called tables), set aside to store and process this information, are of finite size. This places limits on the extent of certain kinds of information that the source program may contain. These limitations are different for each processor and so are listed separately below.

There are two general types of limitations on the size of a Full FORTRAN source program. One type of limitation restricts the complexity of a single

* These topics apply to subprograms only and, therefore, need not concern the user of Basic FORTRAN.

statement; the other type of limitation governs the complexity of a group of statements or of a program as a whole.

Limitations on the Size of a Single Statement

1. No FORTRAN statement may consist of more than 660 characters, including blanks, and must be contained on a maximum of 10 cards.

2. The total number of parameters in a single arithmetic statement must not exceed 50. A parameter is defined as any name or constant. Thus, the arithmetic expression Y = X*3.2 - ROOT*3.0* (SINF(X) + P) contains eight parameters. This maximum of 50 parameters also applies to an arithmetic expression appearing in an IF statement.

3. The CALL statement is subject to the following rules:

a. The maximum number of parameters permissible in one CALL statement is 50. Each name (or constant) in a CALL statement, excluding the word CALL but including the name of the SUBROUTINE subprogram, is counted as a parameter.

b. An argument is composed of one or more parameters. For example, the statement CALL MANEF (A, R + SIGMA *5.0, I) contains three arguments and six parameters, including the name of the SUBROUTINE subprogram. The rule

$$a + p \leq 69$$

defines the maximum number of arguments and parameters allowed in a CALL statement, where

a is the number of arguments
p is the number of parameters

From the rule, (3a) above, it must be remembered that p must never be greater than 50. It follows that the greatest number of arguments ever permitted in a CALL statement is 34 (assuming each argument to be a single parameter). The rule governing the maximum number of arguments which may be present in any subprogram also affects the CALL statement.

4. The maximum number of arguments in a FORTRAN FUNCTION subprogram is limited by the number of parameters in the arithmetic statement calling the subprogram.

a. The maximum number of parameters in the entire arithmetic statement must not exceed 50.

b. The number of arguments permissible in each of the FORTRAN FUNCTION subprograms called in a single arithmetic statement is determined as follows:

$$a + p \leq 69$$

where,

a is the number of arguments in a FORTRAN FUNCTION subprogram
p is the number of parameters in the arithmetic statement in which the FORTRAN FUNCTION is called

5. The number of consecutive nonsubscripted variables (including nonsubscripted array names) that may appear in an input or output list is limited to 46. The appearance of either a DO loop or a subscripted variable will end a sequence.

6. A computed GO TO statement may contain a maximum of 93 statement numbers.

7. The maximum number of arguments in a single arithmetic statement function may be computed as follows:

Let x be the number of arguments with names consisting of five characters or less;
Let y be the number of arguments with names consisting of more than five characters
Finally, the number of arguments in an arithmetic statement function must agree with the formula:

$$x + 2y \leq 40$$

Limitations on the Size of a Full FORTRAN Program

The following limitations are imposed by the maximum sizes of various tables that are constructed by the Full FORTRAN processor. If any of these tables are filled during the compilation of a program, an appropriate message will be typed.

1. Dimension Table -- The total number of dimensioned variables in one source program must not exceed 290. Of these, not more than 50 may be three-dimensional arrays.

2. The total number of nonsubscripted variables in a single source program must not exceed 290.

3. The innermost DO loop within a nest of DO loops must not exceed the 50th level. Implied DO's in the lists of input/output statements are also included in the level count.

4. The limit of the number of arithmetic statement functions in a single source program may be computed as follows:

Let x be the number of arithmetic statement functions with names of five characters or less;
Let y be the number of arithmetic statement functions with names of more than five characters.
The maximum number may be computed as follows:

$$x+2y \leq 75$$

5. Vector Table -- The number of different subprograms and Library functions called by a single source program must not exceed 50.

6. Transfer Table -- The total number of different statement numbers appearing in all the control statements of a program must not exceed 400.

7. Main Table -- The number of "Forvals" and DO statements together must not exceed 500.

NOTE: A Forval is a nonsubscripted fixed-point variable on the left-hand side of an arithmetic statement, in an input list, in a COMMON statement, or in any argument list of a subprogram.

8. The maximum number of arguments in a subprogram depends on the length of the name of the argument.

Let x be the number of arguments with names of five characters or less

Let y be the number of arguments with names of more than five characters.

The maximum number of arguments in a subprogram may be computed as follows:

$$x + 2y \leq 50$$

Note that the maximum number of arguments in a subprogram may also depend on the number of arguments that are stated as arithmetic expressions in the calling program.

Limitations on Source Program Size - Basic FORTRAN

1. No FORTRAN statement may consist of more than 660 characters, including blanks, and must be contained on a maximum of 10 cards.

2. No statement may contain more than 50 pairs of parentheses (those used with subscripts are not counted).

3. The maximum number of user's function subroutines that may be included in any one source program is 20. These 20 are in addition to the open function subroutines and to those available in the 7070 FORTRAN Programming Package.

4. The number of different fixed- or floating-point constants that may be used in any one statement must not exceed 40. (Constants differing only in sign are not considered different, neither are numbers such as 4., 4.0, 40. E-1, etc., considered different.)

5. The maximum number of variables allowed in a source program is determined as follows:

$$x + 2y \leq 200$$

where

- x is the number of variables whose names do not appear in a DIMENSION statement or in an EQUIVALENCE statement
- y is the number of variables whose names appear in a DIMENSION statement or EQUIVALENCE statement, or both.

6. The innermost DO loop within a nest of loops must not exceed the 27th level. Implied DOs in the lists of input/output statements are included in the level count.

Each 7070-Series FORTRAN processor has its own method of detecting errors during the compilation of a source program. In general, however, when a rule of the language is violated or the capacity of a table is exceeded, a message is typed on the console typewriter.

ERROR MESSAGES IN FULL FORTRAN

The following is a description of the error messages that are produced by the 7070-Series Full FORTRAN processor:

Types of Errors

Language Errors: These errors occur when statements in the source program do not meet FORTRAN language specifications for programs to be run on a 7070-Series machine.
Internal Errors: These errors occur when the processor detects invalid records produced as a result of machine errors, incorrect input, or source program logic errors.

When internal errors occur, review the source program, make corrections if necessary, and then retry the operation.

Contents of Message List

The messages are listed below alphabetically with an explanation that indicates:

1. Whether the error is a language error or an internal error.
2. The section of the 7070-Series FORTRAN Processor that detected the error.
3. The cause of the error.

NOTE: If a halt occurs after a message is typed and no action is indicated, processing cannot be continued.

Types of Messages

When an error occurs, the following types of messages may be typed:

Descriptive Message: indicates the nature of the error.
Statement Type: indicates the type of source statement that led to the error; e.g., DO, CALL, ARITH.
FORTRAN Number: indicates the FORTRAN number that was assigned to the source statement in error.
Card Count: indicates the number of the card that contained the source statement in error.

Message and Explanation

ARITH STMNT FNCT ARGS EXCEED TABLE
Explanation: Language error, section III. The number of arguments of a function defined by an Arithmetic Statement exceeds the table limit of 40 words.

ARITH STMNT FNCTS EXCEED TABLE
Explanation: Language error, section III. The number of Arithmetic Statement functions exceeds the table limit of 75 words.

xxxxxx ARRAY EXCEEDS STORAGE
Explanation: Language error, section IV. A variable in a Dimension Statement, indicated above by xxxxxx, has an array that will exceed 10K core storage at object time.

BLX ADDR +000000yyyy
Explanation: Internal error, section III. This error appears only if the operations digit has been turned on in the Communication region of Autocoder. The yyyy is the location in the processor that follows the instruction BLX 94, xxxx
This instruction branches to the error routine that begins at location xxxx.

BLX ADDR +00yyyy0000
Explanation: Internal error, section I. This error appears only if the operations digit has been turned on in the Communication region of Autocoder. The yyyy is the location in the processor that follows the instruction

```
BLX 94, xxxx
```

This instruction branches to the error routine that begins at location xxxx.

BLX ADDR +000000yyyy, STMNT TYPE ttttt, FORTRAN NO. nnnnn
Explanation: Internal error, section IV. This error appears only if the operations digit has been turned on in the Communication region of Autocoder. The yyyy is the location in the processor that follows the instruction

```
BLX 94, xxxx
```

where

BLX 94	is the instruction that branches to the error routine
xxxx	is the starting location of the error routine
ttttt	is the type of FORTRAN statement in error
nnnnn	is the number of the FORTRAN statement in error.

*CARD COUNT** +00xxxx0000
Explanation: Language error, section I or section III. The actual card number of the erroneous FORTRAN statement is indicated by xxxx.

*CHARACTER COUNT ZERO OR EXCEEDED xxxxxxxxxx
Explanation: Language error, section I. Either the name of a variable or the number of a statement exceeds six characters. The name of the variable or the type of statement is indicated by xxxxxxxxxx. When the character count is zero, anything typed to replace xxxxxxxxxx is meaningless. This message is also produced when a DO index exceeds four characters.

*CLASSIFY ERROR xx*TLU CHARACTERS
Explanation: Language error, section I. In the message, xx represents the first two characters of a FORTRAN statement whose format cannot be classified in the non-arithmetic table by the 7070-Series FORTRAN processor.

DIMENSION TABLE EXCEEDED xxxxxxxxxx

Explanation: Language error, section II. The total number of dimensional variables in a single source program exceeds 290. The variable that caused the error indication is represented by xxxxxxxxxx.

*DO LOOP OPEN

Explanation: Language error, section I. The FORTRAN statement number corresponding to n in the statement

DO n i = m_1, m_2, m_3

is missing.

*DO NEST EXCEEDS 50

Explanation: Language error, section I. The number of DO loops that make up a nest exceeds 50. All DO loops associated with input/output statements that appear in the DO nest have been counted.

DUPLICATE COMMON VARIABLE

Explanation: Language error, section IV. A variable has appeared more than once in a COMMON statement.

DUPLICATE VARIABLE xxxxxx

Explanation: Language error, section II. A subscripted variable (xxxxxx) has appeared more than once in one or more DIMENSION statements.

ERR 1

Explanation: Internal error, section VI. An instruction with negative address adjustment has been created.

ERR 2

Explanation: Internal error, section VI. No more index words are available.

ERROR IN ARITH STATEMENT FUNCTION

Explanation: Language error, section III. An Arithmetic Statement function has been written incorrectly.

*ERROR IN FORMAT OF STMNT

Explanation: Language error, section I. A statement does not meet FORTRAN language specifications for programs to be run on a 7070-Series machine.

ERROR xx IN FORMAT STMNT

Explanation: Language error, section I. This message is typed when an error is detected in a FORMAT statement. One of the following numbers will be typed in the message (xx above), depending upon the type of error.

Number	Error
01	A left parenthesis is not the first character (excluding blanks) to follow FORMAT.
02	There are more than two levels of parentheses.
03	The record exceeds 120 characters.
04	Format specifications follow the final right parenthesis.
05	The final right parenthesis is missing.
06	H or X field size is blank.
07	A or I field size is blank.
08	E or F field size is blank.
09	There is no decimal point after E or F specification..
10	The decimal portion of E or F exceeds field size.
11	A constant precedes a plus or a minus sign.
12	No plus or minus sign is associated with the P factor.
13	E, F, I, or A processing is not followed by a right parenthesis, a slash, or a comma.
14	An illegal character is used.

ERROR IN I/O LIST

Explanation: Language error, section III. There is an error in the list of an input/output statement.

ERROR IN USE OF CONSTANT

Explanation: Language error, section III. The message indicates misuse of a fixed- or floating-point constant.

ERRORS IN INPUT, TO CONTINUE HIT START TO COMPILE NEXT PROGRAM HIT PROGRAM RESET AND START

Explanation: Language error, section VII. Major errors in input.

Action: To continue this compilation, press START. To compile the next program, press PROGRAM RESET and START.

EXTRANEOUS TXT RCD

Explanation: Internal error, section III. There is an extra TXT record following a SAT (communication between sections) record; this TXT record is not required.

FLT CONST ERROR

Explanation: Language error, section III. An error in a floating-point constant has been detected. The exponent portion of a floating-point constant is in error, or a decimal point is included in a fixed-point number.

*FORTRAN NO** xxxxx

Explanation: Language error, section I. The number of the erroneous FORTRAN statement is typed as indicated above by xxxxx. If the FORTRAN statement is not numbered, the message is typed without a number.

FORTRAN NO** xxxxx

Explanation: Language error, section III. The number of the erroneous FORTRAN statement is typed as indicated above by xxxxx. If the FORTRAN statement is not numbered, the message is typed without a number.

*FORTRAN NO.** xxxxx

Explanation: Language error, section III. The number of the erroneous FORTRAN statement is typed as indicated above by xxxxx. If the FORTRAN statement is not numbered, the message is typed without a number.

*FUNCTION STMNT NOT FIRST

Explanation: Language error, section I. The FORTRAN FUNCTION statement must be the first statement (not including comments cards) of a FORTRAN FUNCTION subprogram.

ILLEGAL COMBINATION OF CHARS

Explanation: Language error, section III. An illegal combination of characters has been used; e.g., two operations grouped together in the format of an Arithmetic Statement.

ILLEGAL SUBSCRIPT

Explanation: Language error, section III. An invalid subscript has been used.

ILLEGAL SUBPGRM ARGUMENT xxxxxxxxxx

Explanation: Language error, section I. A COMMON EQUIVALENCE variable has been used as an argument in a subprogram. The erroneous subprogram argument is indicated by xxxxxxxxxx above.

ILLEGAL SUBPGRM ARGUMENT

Explanation: Language error, section IV. There is an error in a subprogram argument.

INCONSISTENT EQUIV

Explanation: Language error, section IV. An EQUIVALENCE statement is inconsistent.

INCORRECT RECORD NAME

Explanation: Internal error, section IV. The names of the DA and EQU records are incorrect.

INVALID TAPE RECORD xxxxx

Explanation: Internal error, section II. A FORTRAN section I output record, indicated by xxxxx, on tape C is not a DIM (dimension) or SAT (communications between sections) record.

xxxxxx IS AN UNDEFINED OPERAND

Explanation: Language error, section III. This message may be written on the output listing at the completion of the Autocoder portion of the FORTRAN compilation. In the message above, xxxxxx is replaced by the name of the operand that has not been defined by one of the following means of defining a nonsubscripted variable:

1. EQUIVALENCE statement
2. COMMON statement
3. The left-hand side of an arithmetic statement.
4. The n of ASSIGN i TO n
5. A variable in the list of a READ, READ INPUT TAPE, or READ TAPE statement.
6. An argument of a CALL statement
7. The i of DO n i = m_1, m_2, m_3, or i of an implied DO in the list of an input/output statement.

MAIN TABLE OVERFLOW

Explanation: Internal error, section V. The combined total of "Forvals" (FORTRAN variable and data format) and DO statements exceeds 50.

MISSING TXT RCD

Explanation: Internal error, section III. A TXT record, required after a SAT (communications between sections) record, is missing.

MORE THAN 290 NONSUBSCRIPTED VARIABLES

Explanation: Language error, section IV. The total number of nonsubscripted variables in any one source program exceeds 290.

MORE THAN 50 VECTORS

Explanation: Language error, section IV. The number of different subprograms and library functions called by any one source program exceeds 50.

NO DIMEN STMNT FOR SUBSCRIPTED VARIABLE IN EQUIV

Explanation: Language error, section IV. There is no DIMENSION statement defining a subscripted variable shown in an EQUIVALENCE statement.

*PAREN COUNT NZ OR MINUS

Explanation: Language error, section I. There is an excess of either left or right parentheses; the parenthesis counter was non-zero at the end of the scan or less than zero during the scan.

*READ DRUM ILLEGAL

Explanation: Language error, section I. The 7070-Series FORTRAN processor does not process the READ DRUM statement.

RECORD FORMAT ERROR xxxxxxxxxx

Explanation: Internal error, section V. The format of an internal record is incorrect. In the message above, xxxxxxxxxx represents the serial number of the erroneous record.

*RETRN STMNT ILLEGAL

Explanation: Language error, section I. A RETURN statement has been used in a main program.

*RTRN STMNT MISSING

Explanation: Language error, section I. A RETURN statement is missing in a FORTRAN FUNCTION subprogram or a SUBROUTINE subprogram.

SAME NAME FOR SBSCRPTED AND NONSBSCRPTED VARIABLE

Explanation: Language error, section IV. The same name has been used for both a subscripted variable and a nonsubscripted variable; i.e., the variable name is not in the DIMENSION table.

*STMNT NUMBER 0 OR BLANK

Explanation: Language error, section I. The n in

DO n i = m_1, m_2, m_3

is zero or blank.

*STMNT TOO LONG

Explanation: Language error, section I. A FORTRAN statement exceeds 660 characters.

*STMNT TYPE** ttttt

Explanation: Language error, section I or section III. The type of FORTRAN statement in error is indicated as shown by ttttt above.

STMNT TYPE ttttt, FORTRAN NO. nnnnn

Explanation: Language error, section IV. The type of FORTRAN statement in error is indicated by ttttt, and the FORTRAN statement number is indicated by nnnnn.

*SUBRTINE STMNT NOT FIRST

Explanation: Language error, section I. The SUBROUTINE statement must be the first statement (not including comments cards) of a SUBROUTINE subprogram.

TRANSFER TABLE EXCEEDED xxxxx

Explanation: Language error, section II. The total number of different statement numbers in all the control statements exceeds 400. The number of the statement that caused the message to be typed is represented above by xxxxx.

TRNSFR ENDS DO

Explanation: Language error, section I. A DO loop ends in a transfer statement.

TXT AND SAT MACRO NAMES MISMATCHED

Explanation: Internal error, section III. The macro-instruction names written in FORTRAN section I do not match.

+xxxxxx0000 TYPE nnn ERROR

Explanation: Language error or internal error, section VII. This message is generally produced whenever invalid records are received as a result of errors in the source program. In the message, xxxxxx represents the six high-order digits of the ten-digit internal serial number of the statement being processed at the time the invalid record is received; nnn represents a three-digit number indicating either the type of statement being processed (500-580) or the type of error (600-901). The meanings of the various numbers that replace nnn are given below along with an indication of whether the error was a language error (L) or an internal error(I).

Where a number is followed by an asterisk (*), the asterisk is not typed; it indicates that the limitations allowed by the processor have been exceeded.

Number	Type of Error	Type of Statement
500-502	L	GO TO
503-508	L	ASSIGNED GO TO
509-514	L	ASSIGN
515-521	L	COMPUTED GO TO
527-529	L	SENSE LIGHT
533-541	L	IF SENSE LIGHT/SWITCH
545-550	L	IF ACC/QUOT OVERFLOW, IF DIVIDE CHECK
551-553	L	PAUSE, STOP
563-565	L	READ, PUNCH, PRINT, TYPE
566-568	L	READ/WRITE TAPE, BACKSPACE, REWIND, END FILE
569-574	L	READ INPUT/WRITE OUTPUT TAPE
575-580	L	IF
		Errors in Arithmetic Statements
600	I	No TEXT record for ARITH statement
601-605	I	Error in records for ARITH
612*	L	More than 94 parameters in ARITH
620	I	ARITH-defined function has no Element Record
640*	L	More than two characters, not including the sign, after the E in a floating-point constant
650*	L	Characteristic of floating-point constant > 99
651*	L	Characteristic of floating-point constant < 0
		Errors in Control or Specification Statements
700,702	I	Same Internal Number as previous statement
701	I	Macro-instruction name on DO or subscript not legitimate
703	I	Initial record incorrect
800	I	Statement incorrectly identified as FORMAT statement
801	I	FORMAT statement has no TEXT
890	I	Subscripted variable in I/O List incorrectly processed
901*	I	More than 49 routines have been called for

VARIABLE NAME TOO LONG

Explanation: Language error, section III. The number of characters in a variable name exceeds six.

*WRITE DRUM ILLEGAL

Explanation: Language error, section I. The 7070-Series FORTRAN processor does not process the WRITE DRUM statement.

ERROR MESSAGES IN BASIC FORTRAN

The 7070-Series Basic FORTRAN processor detects two types of coding errors:

1. An error in which one of the limitations imposed on the source program has been exceeded.
2. An error in which one of the rules for writing a source statement has been violated. (This type of error concerns the format of a statement; e.g., punctuation.)

For either coding error, a message is typed and the machine continues processing with the next statement of the source program.

For the first kind of error, the following message is typed:

TYPE n ERROR

where n is the code number of the type error as given in the table below.

In addition to this message the first 40 columns of the statement will be typed for use in identifying the statement in which the error occurred.

Type	Description
1	Program begins with a continuation card.
2	Program contains too many user's function subroutines.
3	Program contains too many variables.
4	There are too many subscripted variables in use at this point.
5	A subscripted variable was not previously mentioned in a DIMENSION statement.
6	A DO loop ends with a GO TO, IF, or STOP statement.
7	A DO loop exceeds the maximum limit specified for a nest of DO loops.
8	A DO loop in the list of an input/output statement exceeds the maximum limit specified for a nest of DO loops.
9	Statement is too long.
10	Statement contains too many constants.
11	Statement contains too many parentheses.
12	Statement contains unmatched parentheses.
13	Statement contains a variable having a name that is too long.

For the second type of error, i.e., when a statement is encountered in which one of the rules of the FORTRAN language has been violated, the following message is typed:

STMNT WRITTEN INCORRECTLY

In addition to this message, the first 40 columns of the statement are typed for use in identifying the incorrect statement.

This appendix describes the rules that govern the order in which source program statements of a FORTRAN program are executed.

1. Control originates at the first executable statement.

2. If control has been with statement S, then control passes to the statement indicated by the normal sequencing of statement S (see "Table of Source Program Statement Sequencing"). However, if S is the last statement in the range of one or more DOs which are not yet completed, then the normal sequencing of S is ignored, and DO sequencing takes place.

3. The specification statements, and the FORMAT, FUNCTION, and SUBROUTINE statements are non-executable. FUNCTION and SUBROUTINE statements must precede all executable statements in a subprogram. The specification and the FORMAT statements may be placed anywhere in the program.

4. Every executable statement in the source program (except the first) must have some path of control leading to it.

TABLE OF SOURCE PROGRAM STATEMENT SEQUENCING

Statement	Normal Sequencing
a = b	Next executable statement
GO TO n	Statement n
GO TO n, $(n_1, n_2, \ldots, n_m)$	Statement last assigned to n
ASSIGN i TO n	Next executable statement
GO TO $(n_1, n_2, \ldots, n_m)$, i	Statement n_i
IF (a) n_1, n_2, n_3	Statement n_1, n_2, or n_3 if (a) < 0, (a) = 0, or if (a) > 0, respectively.
SENSE LIGHT i	Next executable statement.
IF (SENSE LIGHT i) n_1, n_2	Statement n_1, n_2 if Sense Light i is ON or OFF, respectively.
IF (SENSE SWITCH i) n_1, n_2	Statement n_1, n_2 if Alteration Switch i is ON or OFF, respectively.
IF ACCUMULATOR OVERFLOW n_1, n_2	Statement n_1, n_2 if the FORTRAN internal overflow indicator is ON or OFF, respectively.
IF QUOTIENT OVERFLOW n_1, n_2	Statement n_1, n_2 if the FORTRAN internal overflow indicator is ON or OFF, respectively.
IF DIVIDE CHECK n_1, n_2	Statement n_1, n_2 if the FORTRAN Divide Check indicator is ON or OFF, respectively.
PAUSE or PAUSE n	Next executable statement.
STOP or STOP n	Terminates program.
DO n i m_1, m_2 or DO n i m_1, m_2, m_3	DO sequencing, then next executable statement.
CONTINUE	Next executable statement.
END $(I_1, I_2, I_3, I_4, I_5)$	No sequencing; this statement terminates a problem.
† CALL Name $(a_1, a_2, \ldots, a_n)$	First statement of subroutine Name.
† SUBROUTINE Name $(a_1, a_2, \ldots, a_n)$	Next executable statement.
† FUNCTION Name $(a_1, a_2, \ldots, a_n)$	Next executable statement.
† RETURN	The statement or part of statement following the calling statement. Note that a return from a FORTRAN FUNCTION subprogram is made to the same arithmetic statement which contains the function; it is not made to the following statement.
READ n, List	Next executable statement.
READ INPUT TAPE i, n, List	Next executable statement.
PUNCH n, List	Next executable statement.
PRINT n, List	Next executable statement.
TYPE n, List	Next executable statement.
WRITE OUTPUT TAPE i, n, List	Next executable statement.
READ TAPE i, List	Next executable statement.
WRITE TAPE i, List	Next executable statement.
END FILE i	Next executable statement.
REWIND i	Next executable statement.
BACKSPACE i	Next executable statement.
FORMAT (Specification)	Non-executable statement; may be placed anywhere.
DIMENSION $v_1, v_2, v_3, \ldots$	
EQUIVALENCE (a, b, c, ...) (d, e, f, ...), ...	
† COMMON A, B, ...	

† Available only in Full FORTRAN.

APPENDIX B: ADMISSIBLE CHARACTERS IN A FORTRAN SOURCE PROGRAM

The following chart indicates the list of characters which may be used in a FORTRAN program.

Character	Card Code	Core Storage 2-Digit Alphameric Code	Magnetic Tape BCD Code	Core Storage 1-Digit Numerical Code
Blank		00	CA	
.	12-3-8	15	CBA821	
)	12-4-8	16	BA84	
+	12	20	BA	
$	11-3-8	25	B821	
*	11-4-8	26	CB84	
-	11	30	CB	
/	0-1	31	A1	
,	0-3-8	35	A821	
(	0-4-8	36	CA84	
=	3-8	45	C821	
-	4-8	46	84	
A	12-1	61	CBA1	
B	12-2	62	CBA2	
C	12-3	63	BA21	
D	12-4	64	CBA4	
E	12-5	65	BA41	
F	12-6	66	BA42	
G	12-7	67	CBA421	
H	12-8	68	CBA8	
I	12-9	69	BA81	
J	11-1	71	B1	
K	11-2	72	B2	
L	11-3	73	CB21	
M	11-4	74	B4	
N	11-5	75	CB41	
O	11-6	76	CB42	
P	11-7	77	B421	
Q	11-8	78	B8	
R	11-9	79	CB81	
S	0-2	82	A2	
T	0-3	83	CA21	
U	0-4	84	A4	
V	0-5	85	CA41	
W	0-6	86	CA42	
X	0-7	87	A421	
Y	0-8	88	A8	
Z	0-9	89	CA81	
0	0	90	82	0
1	1	91	C1	1
2	2	92	C2	2
3	3	93	21	3
4	4	94	C4	4
5	5	95	41	5
6	6	96	42	6
7	7	97	C421	7
8	8	98	C8	8
9	9	99	81	9

FORTRAN SPECIAL CHARACTERS

Only the special characters shown in the table are meaningful to the FORTRAN processor. These characters are always identified by their card codes. However, off-line peripheral equipment is available with a choice of special character sets. Depending on the set of characters supplied with a printer or a keypunch, a column punched 12-4-8 may be equated to any one of the following:

¤
<
)

The various sets of characters are known as "Type-Wheel Configurations A, B, C, D, etc." Type-Wheel Configuration F is the special character set used by FORTRAN.

When punching a source program, the character must be punched using the appropriate card code. For example, when punching

)

the card code 12-4-8 must be used, regardless of how a given printer interprets this code.

The special characters used in FORTRAN are given below with their equivalents in other type-wheel configurations.

Special Characters used in FORTRAN	Card Code	Type-Wheel Configuration								
		A	B	C	D	E	F	G	H	K
+	12	&	/	&	-	-	+	+	+	+
.	12-3-8	.	.	.	.	.	.	.	.	.
)	12-4-8	¤	¤	¤	¤	<	)	¤	)	)
-	11	-	-	-	-	/	-	-	-	-
$	11-3-8	$	$	$	$	o	$	$	$	$
*	11-4-8	*	*	*	*	*	*	*	*	*
/	0-1	/	&	0	/	&	/	/	/	/
,	0-3-8	,	,	,	,	,	,	,	,	,
(	0-4-8	%	%	%	%	%	(	%	(	(
=	3-8	#	#	#	#	#	=	+	=	=
-	4-8	@	@	@	@	>	-	-	,	@

Figure 7

NOTE: The two minus signs indicated in the table are used as follows:

FORTRAN Source
Program Cards 11 punch minus sign
Input data to the
object program 11 punch or 4-8 punch minus sign
Output data from the
object program 11 punch minus sign

The character

$

can be used in FORTRAN only as:

Alphameric text in FORMAT and CALL statements

and

Alphameric data for the object program.

The statements of a FORTRAN source program are written on a standard FORTRAN Coding Form, Form X28-7327. A sample FORTRAN source program is shown on the aforementioned coding form and is illustrated in Figure 9.

PUNCHING A SOURCE PROGRAM

Each statement of a FORTRAN source program is punched into a separate card (the standard FORTRAN card form is shown in Figure 8); however, if a statement is too long to fit on one card, it can be contained on as many as nine continuation cards. The order of the source statements is governed solely by the order of the statement cards.

Cards which contain a "C" in column 1 are not processed by the FORTRAN processor. Such cards may, therefore, be used to carry comments which will appear when the source program deck is listed.

(Columns 1-5)

Statement numbers are punched in columns 1-5 of the initial card of a statement. These statement numbers permit cross references within a source program and, when necessary, facilitate the correlation of source and object programs and compilation listing. The statement numbers need not be punched in sequence.

(Column 6)

Column 6 of the initial card of a statement must be left blank or punched with a zero. Continuation cards must have column 6 punched with a character other than a blank or zero. Continuation cards for comments need not be punched in column 6; only the "C" in column 1 is necessary.

(Columns 7-72)

The statements themselves are punched in columns 7-72 both on initial and continuation cards. Thus, a statement may consist of not more than 660 characters (i.e., 10 cards). A table of the admissible characters for FORTRAN is given in Appendix B. In general, blank characters, except in column 6, are simply ignored by FORTRAN and may be used freely to improve the readability of the source program listing (exceptions to this rule can occur in certain fields in FORMAT and CALL statements).

(Columns 73-80)

Columns 73-80 are not processed by FORTRAN and may, therefore, be punched with any desired identifying information. The identification may consist of any admissible character, except that column 79 should not contain a 12 punch. This eliminates the letters A through I, the period, the right parenthesis, and plus sign from column 79 (see Appendix B).

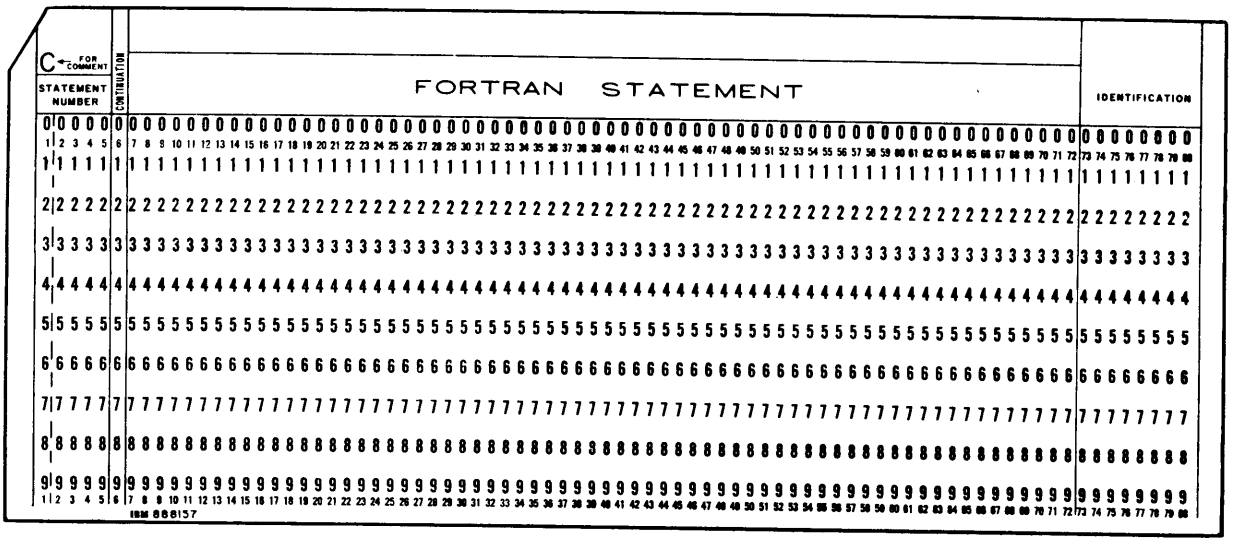


Figure 8

The input to the FORTRAN processors may be either the deck of source statement cards, or a tape prepared on peripheral card-to-tape equipment. Input specifications for Full FORTRAN are given in the Operator's Guide, 7070 Series Programming Systems, Form C28-6249. Input specifications for Basic FORTRAN are given in IBM 7070 Series Operating Instructions: Basic FORTRAN, Form J28-6171.

SAMPLE PROGRAM

The sample program (Figure 9) is designed to find all of the prime numbers between 1 and 1,000. A prime number is an integer that cannot be evenly divided by any integer except itself and 1. Thus, 1, 2, 3, 5, 7, 11, . . . are prime numbers. The number 9, for example, is not a prime number since it can evenly be divided by 3.

IBM FORTRAN CODING FORM

Program | Graphic | Card Form # * | Page 1 of 2 | Identification (73–80)
Programmer | Date | Punch

C FOR COMMENT

STATEMENT NUMBER (1–5)	CONT (6)	FORTRAN STATEMENT (7–72)
C		PRIME NUMBER PROBLEM
100		WRITE OUTPUT TAPE 6,8
8		FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
	1	19X,1H1/19X,1H2/19X,1H3)
101		I=5
3		A=I
102		A=SQRTF(A)
103		J=A
104		DO 1 K=3,J,2
105		L=I/K
106		IF(L*K-I)1,2,4

IBM FORTRAN CODING FORM

Program | Graphic | Card Form # * | Page 2 of 2 | Identification (73–80)
Programmer | Date | Punch

C FOR COMMENT

STATEMENT NUMBER (1–5)	CONT (6)	FORTRAN STATEMENT (7–72)
1		CONTINUE
107		WRITE OUTPUT TAPE 6,9
5		FORMAT (I20)
2		I=I+2
108		IF(1000-I)7,4,3
4		WRITE OUTPUT TAPE 6,9
9		FORMAT (14H PROGRAM ERROR
7		WRITE OUTPUT TAPE 6,6
6		FORMAT (31H THIS IS THE END OF THE PROGRAM)
109		STOP
		END

* A standard card form, IBM electro 888157, is available for punching source statements from this form.

Figure 9

This appendix describes the library function subroutines that are available to users of 7070 Series FORTRAN. The use of floating-point mnemonic codes is optional with floating-point function subroutines. Instructions for the use of floating-point arithmetic accompany the program tape. Detailed descriptions of these subroutines follow in alphabetical order for floating-point and fixed-point, respectively. Any error messages or programmed halts are described under the appropriate subroutine.

LIBRARY FUNCTION SUBROUTINE INDEX

The following are the floating-point subroutines. Except for MODF and TANHF, which use floating-point mnemonic codes, all other subroutines use fixed-point mnemonic codes.

ASINF
ATANF
COSF
DIMF
ERRTYPE
INTF
MAX0F
MAX1F
MIN0F (see MAX0F)
MIN1F (see MAX1F)
MODF
SIGNF (see XSIGNF)
SINF (see COSF)
SQRTF
SORTF
TANHF

The following are the fixed-point subroutines and use only fixed-point mnemonic codes:

XDIMF
XMAX0F
XMAX1F
XMIN0F (see XMAX0F)
XMIN1F (see XMAX1F)
XMODF
XSIGNF

ASINF

Purpose

This subroutine computes the arcsine of an argument, Y, in floating-decimal form.

Range

$-1 \leq Y \leq +1$

Method

The arcsine is approximated by means of the expression

$$\pi/2 - (1 - |Y|)^{1/2} \sum_{i=0}^{7} C_i \, Y^i$$

where:

$\pi/2 = +1.570\ 796\ 327$
$C_0 = +1.570\ 796\ 305$
$C_1 = -0\ 214\ 598\ 802$
$C_2 = +0\ 088\ 978\ 987$
$C_3 = -0.050\ 174\ 305$
$C_4 = +0\ 030\ 891\ 881$
$C_5 = -0\ 017\ 088\ 126$
$C_6 = +0.006\ 670\ 090$
$C_7 = -0.001\ 262\ 491$

Error Approximation

With $-1 \leq Y = +1$, the maximum absolute error is $< 5 \cdot 10^{-8}$

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The subroutines SQRTF and ERRTYPE must be available for use.
3. A minimum of 68 core storage locations must be provided for storing instructions, constants, and transfer vectors.
4. Accumulators 1, 2, and 3; index words 93, 94, and 98; indicators high, equal, and low must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

A = SINF (B/C) with $C \neq 0$

and return would be to the next FORTRAN statement, with the normalized unrounded result in A, in radians.

In any Autocoder program with the symbolic deck for ASINF as an out-of-line subroutine, an example of an entry could be

```
ZA1     Result of (B/C) in floating decimal
        form
BLX     94, ASINF
```

Return is made to the instruction following the BLX with the normalized unrounded result, in radians, in Accumulator 1.

Halt and Message

If an attempt is made to compute the arcsine of a number having an absolute value > 1, the message:

ASINF ERROR BLX ADDR XXXX

will be typed and a halt will occur where,

XXXX indicates the location following the BLX instruction in the calling program.

Action

Correct the data and start from the beginning; or place an approximate answer in floating-point form in Accumulator 1 and make a branch manually to location XXXX.

ATANF

Purpose

This subroutine computes the arctangent of an argument, N, in floating-decimal form.

Range:

$$-10^{49} < N \leq +10^{49}$$

Method

The arctangent is approximated by the continued fraction

$$N\left[\frac{A_1}{A_2 + (NA_1)^2 - \frac{A_3}{A_4 + (NA_1)^2}}\right]$$

where:

$A_1 = +0.163\ 636\ 363\ 6$ $A_3 = +0.\ 027\ 099\ 842\ 5$

$A_2 = +0.216\ 649\ 136\ 0$ $A_4 = +0.\ 051\ 119\ 459\ 05$

To improve the accuracy, the range between 0 and $\pi/2$ has been divided into five intervals:

0 to $\pi/18$, $\pi/18$ to $\pi/6$, $\pi/6$ to $5\pi/18$, $5\pi/18$ to $7\pi/18$, $7\pi/18$ to $\pi/2$

In the first interval, the arctangent is computed directly. In the other intervals, it is computed with the following relation:

ATANF (N) = $k\pi/9$ ATANF (N')

for N in the interval k, where

$$N' = A_k - B_k (N + A_k)^{-1},$$

$$A_k = \text{Cotan}\ (k\pi/9),\ B_k = 1 + A_k^2 \text{ or}$$

$$N' = \text{Cotan}\ (k\pi/9) - \left[\frac{1 + (\text{Cotan}\ (k\pi/9))^2}{N + \text{Cotan}\ (k\pi/9)}\right]$$

with each contangent value entered as a constant.

Error Approximation

The maximum absolute error is $< 2 \cdot 10^{-8}$

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. A minimum of 90 core-storage locations must be provided for storing instructions and constants.
3. Accumulators 1, 2, and 3; index words 94 and 98; and indicators high, equal, and low must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

A = ATANF (B/C) with $C \neq 0$

and return would be to the next FORTRAN statement, with the normalized unrounded result in A, in radians.

In any Autocoder program with the symbolic deck for ATANF as an out-of-line subroutine, an example of an entry into the ATANF subroutine is:

```
ZA1   Result of (B/C) in floating decimal form
BLX   94, ATANF
```

Return is made to the instruction following the BLX, with the normalized unrounded result in Accumulator 1.

COSF and SINF

Purpose

This subroutine computes the trigonometric cosine or the trigonometric sine of the argument, X, in floating-decimal radians.

Range

$$-10^{10} < X < +10^{10}$$

Method (SINF)

With $|X| \leq 10^{-4}$, SINF (X) = X. With $10^{-4} < |X|$, X is divided by $\pi/2$ and the quotient is separated into an integral and decimal part (Q_i and Q_d). If Q_i (mod 4) = 3 or 4, Q_d is replaced with the magnitude of $(1 - Q_d)$. Then, SINF (X) is evaluated by means of the polynomial.

$$\sum_{i=0}^{4} C_{2i+1} (Qd)^{2i+1}$$

where:

$C_1 = +1.570\ 796\ 318$ $C_7 = -0.004\ 673\ 766$

$C_3 = -0.645\ 963\ 711$ $C_9 = +0.000\ 151\ 484$

$C_5 = +0.079\ 689\ 679$ $\pi/2 = +1.570\ 796\ 327$

Method (COSF)

With $|X| \leq 10^{-4}$, COSF (X) = ±1.0. With $10^{-4} < |X|$, the method is identical to that for SINF except that Q_i is argumented by +1 before the quadrant analysis and polynomial evaluation.

Error Approximation

The maximum absolute error is $\leq 10^{-7}$, where $-\pi/2 \leq X \leq +\pi/2$

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The SINF subroutine must be included in any COSF computation.
3. The ERRTYPE subroutine must be available for either the SINF or COSF computation.
4. A minimum of 87 (for SINF) and 90 (for COSF) core storage locations must be provided.
5. Accumulators 1, 2, and 3; and index words 93, 94, and 98 must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

A = SINF (B*C) or D = COSF (E+F)

and return would be to the next FORTRAN statement, with the normalized unrounded result in A or D.

In any Autocoder program with the symbolic deck for SINF or COSF as an out-of-line subroutine, an example of an entry into this subroutine is:

ZA1 Result of (B*C) in floating-decimal radians	or ZA1 Result of (E+F) in floating-decimal radians
BLX 94, SINF	BLX 94, COSF

Return is made to the instruction following the BLX, with the normalized unrounded result in Accumulator 1.

Halt and Message

If an attempt is made to compute the sine or cosine of an angle of magnitude $\geq 10^{10}$ radians, one of the following messages will be typed and a halt will occur.

SINF ERROR BLX ADDR XXXX

or

COSF ERROR BLX ADDR XXXX

where:

XXXX indicates the location following the BLX instruction in the calling program.

Action

Correct the data and start from the beginning; or place an approximate answer in floating-point form in Accumulator 1 and make a branch manually to location XXXX.

DIMF

Purpose

This subroutine determines the positive difference between two arguments, X_1 and X_2, with both arguments in floating-decimal form, which may be simulated by the floating-point routines of the 7070 Series FORTRAN Loader/Package.

Range

$$-10^{49} < X_i < +10^{49}$$

Method

Obtaining the positive difference is accomplished by subtracting as follows:

(argument 1) - (argument 2)

If this result is = 0, +0 is used as the result.

Error Approximation

No error is produced in the resultant numerical value

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. A minimum of 11 core storage locations must be provided for storing instructions.
3. Accumulators 1, 2, and 3; index words 93, 94, and 98 must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

P = DIMF (X1, X2/4.)

and return would be to the next FORTRAN statement, with the resultant numerical value in floating-point form in P.

In any Autocoder program with the symbolic deck for DIMF as an out-of-line subroutine, an example of an entry into the DIMF subroutine is:

```
BLX 94, DIMF
HB  X1
HP  Word containing X2/4
```

Return is made to the instruction following the HP, with the resultant numerical value in normalized floating form in Accumulator 1.

ERRTYPE

Purpose

This subroutine types a standard error message for known errors occurring in other routines.

Range

Input to ERRTYPE is the name of the function or subroutine that has recognized an error and the corresponding location in the main program that is the normal return for the function or subroutine.

The name of the function or subroutine has the form

NNNNN (five alphameric characters).

The corresponding location has the form

xxxx (four numeric characters).

Method

The function or subroutine name and corresponding location (converted to double-digit form) are placed in a skeletal form of the standard error message. A typeout is then executed.

The standard error message produced by ERRTYPE has the following form:

NNNNN ERROR BLX ADR XXXX

with NNNNN the alphameric name and XXXX the location of the normal return.

Error Approximation

No programming error is produced, since no computation is performed.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both the sign change and the field overflow.
2. A minimum of 18 core storage locations must be provided for storing instructions and constants.
3. Accumulators 1 and 2, and index words 93 and 94 must be available for use.

Entry and Exit

Since this subroutine has a 7-character name (ERRTYPE), it cannot be called for in the FORTRAN language.

In any Autocoder program, or any corresponding function or subroutine written in the Autocoder language, with the symbolic deck for ERRTYPE as an out-of-line subroutine, an entry example could be as follows:

If the location of the normal return is not in Index Word 94, include ...

```
        XZA   94,xxxx
              (with xxxx being the location of the normal return,
              as a four-digit number,
              or an Autocoder label)

        ZA1   NNNNN
              (the name of the function
              or subroutine)

        BLX   93,ERRTYPE
```

After the standard error message has been typed, return is made to the instruction following the BLX. The instruction in the user's function or subroutine to which ERRTYPE returns may have a halt at the user's option.

INTF

Purpose

This subroutine truncates an argument, X, by attaching the sign of X to the largest integer $\leq |X|$, with X in floating-decimal form.

Range

$$-10^{49} < X < +10^{49}$$

Method

The truncation is accomplished as follows:

(largest integer $\leq$ 1X1) with sign of X attached

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. A minimum of 12 core storage locations must be provided for storing instructions and constants.
3. Accumulators 1 and 2, and index words 93 and 94 must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

Z = INTF (X1+3.5)

and return would be to the next FORTRAN statement, with the resultant numerical value in floating-point form in Z.

In any Autocoder program with the symbolic deck for INTF as an out-of-line subroutine, an example of an entry into the INTF subroutine is:

```
ZA1     Result of X + 3.5 in
          floating decimal form
BLX     94, INTF
```

Return is made to the instruction following the BLX, with the resultant numerical value in normalized floating form in Accumulator 1.

MAX0F and MIN0F

Purpose

This subroutine determines the largest or the smallest value of two or more arguments, K_1, K_2, . . ., K_n, in fixed-point form.

Range

$$-9\ 999\ 999\ 999 \leq K_i \leq +9\ 999\ 999\ 999$$

Method

The largest (or smallest) value is chosen by comparing

K_i with K_{i+1} where i = 1, 2, . . ., n

After the first comparison, each successive comparison is made with the larger (or smaller) value of the previous comparison.

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be the sense mode for both sign change and field overflow.
2. The subroutine MAX0F should only be used, when it is provided with MIN0F.
3. The subroutine FLOATF must also be provided when MAX0F is to be used.
4. A minimum of 16 and 13 storage locations must be provided for subroutines MAX0F and MIN0F respectively.
5. Accumulators 1, 2 and 3, and index words 93, 94 and 98 must be available.

Entry and Exit

Using FORTRAN, an entry example could be

```
A = MAX0F (K1, 2*K2, K3 + 5) or
B = MIN0F (L1**3, L2, L3-16)
```

and return would be to the next FORTRAN statement, with resultant numerical value in floating-point form in A or B.

In any Autocoder program with the symbolic deck for MAX0F or MIN0F as an out-of-line subroutine, an example of an entry into this subroutine is:

```
BLX 94, MAX0F          BLX 94, MIN0F
HB  K1                 HB  Word containing
                   or        L1**3
HB  Word containing    HB  L2
      2*K2
HP  Word containing    HP  Word containing
      K3 +5                L3-16
```

Return is made to the instruction following the HP, with the resultant numerical value in normalized floating form in Accumulator 1.

MAX1F AND MIN1F

Purpose

This subroutine determines the largest or the smallest value of two or more arguments, X_1, X_2, . . ., X_n in floating decimal form.

Range

$$-10^{49} < X_i < +10^{49}$$

Method

The largest (or smallest) value is chosen by comparing

X_i with X_{i+1} where i = 1, 2, . . ., n.

After the first comparison, each successive comparison is made with the larger (or smaller) value of the previous comparison.

Error Approximation:

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The MAX1F subroutine must only be used when it is provided with MIN1F subroutine.
3. A minimum of 16 for MAX1F and 13 for MIN1F storage locations must be provided for MAX1F and MIN1F respectively.
4. Accumulators 1, 2, and 3, and index words 93 and 94 must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

```
A = MAX1F(X1, X2**2, X3 + 1., X4) or
B = MIN1F (3.*W1, W2, W3, W4/4.)
```

and return would be to the next FORTRAN statement, with the resultant numerical value in floating point form in A or B.

In any Autocoder program with the symbolic deck for MAX1F or MIN1F as an out-of-line subroutine, an example of an entry into this subroutine is:

```
BLX 94, MAX1F          BLX 94, MIN1F
HB  X1                 HB  Word containing
                   or      3.*W1
HB  Word containing    HB  W2
      X2**2
HB  Word containing    HB  W3
      X3+1.
HP  X4                 HP  Word containing
                           W4/4.
```

Return is made to the instruction following the HP, with the resultant numerical value in normalized floating form in Accumulator 1.

MODF

Purpose

This subroutine determines the remainder between two arguments, X_1 and X_2 by taking X_1 (modulo X_2), with both arguments in floating-decimal form. This subroutine uses floating point mnemonic codes, which may be simulated by the floating routines of the 7070 Series FORTRAN Loader/Package.

Range

$$-10^{49} < X_i < +10^{49}$$

Method

Obtaining the remainder is accomplished as follows:

X_1 (modulo X_2)

or

$$X_1 - \left[\frac{X_1}{X_2}\right] X_2$$

considering only the integral part of

$$\frac{X_1}{X_2}$$

If $|X_1| = 0$, the remainder is X_1
If $|X_2| = 0$, the result is set to X_1
If $|X_1| > |X_2|$, the remainder is obtained as shown
If $|X_1| < |X_2|$, the remainder is X_1

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The subroutine INTF must be included in any MODF computation.
3. A minimum of 21 storage locations must be provided for storing instructions and transfer vectors.
4. Accumulators 1, 2, and 3; index words 93, 94, and 98; and the accumulator overflow indicators must be available.

Entry and Exit

Using FORTRAN, an entry example could be

```
A = MODF (X1, X2)
```

and return would be to the next FORTRAN statement, with the resultant numerical value in floating-point form in A.

In any Autocoder program with the symbolic deck for MODF as an out-of-line subroutine, an example of an entry into the MODF subroutine is:

```
BLX 94, MODF
HB  X1
HP  X2
```

Return is made to the instruction following the HP, with the resultant numerical value in normalized floating form in Accumulator 1.

SQRTF

Purpose

This subroutine computes the square root of an argument, X, with $X \geq 0$ in floating-decimal form.

Range

$$+0 \leq X < +10^{49}$$

Method

A linear approximation of the form $Y = AX + B$ is used to determine the value of $\sqrt{X/4}$ to two decimal places. This result is used as a first approximation for two iterations of a modified form of Newton's formula where

$$Y_1 \approx \sqrt{X/4},\ Y_2 \approx \sqrt{X/2},\ \text{and}\ Y_3 \approx \sqrt{X}.$$

Error Approximation

The maximum absolute error is 10^{-8}

Entry and Exit

Using FORTRAN, an entry example could be

D = SQRTF(E-F) with $F \leq E$

and return would be to the next FORTRAN statement, with the normalized unrounded result in D.

In any Autocoder program with the symbolic deck for SQRTF as an out-of-line subroutine, an example of an entry into the SQRTF subroutine is:

```
ZA1   Result of (E -F) in floating-
        decimal form
BLX   94, SQRTF
```

Return is made to the instruction following the BLX, with the normalized unrounded result in Accumulator 1.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The subroutine ERRTYPE must be available for use with SQRTF.
3. A minimum of 45 storage locations must be provided to store instructions, constants, and transfer vectors pertaining to SQRTF.
4. Accumulators 1, 2, and 3, and index words 93, 94, and 98 must be available for use.

Halt and Message

If an attempt is made to compute the square root of a number < 0, the message

SQRTF ERROR BLX ADDR XXXX

will be typed and a halt will occur where,

XXXX indicates the location following the BLX instruction in the calling program.

Action

Correct the data and start from the beginning; or place an approximate answer in floating-point form in Accumulator 1 and make a branch manually to location XXXX.

TANHF

Purpose

This subroutine computes the hyperbolic tangent of an argument, X, in floating-decimal radians. This subroutine uses floating-point mnemonic codes, which may be simulated by the floating-point routines of the 7070 Series FORTRAN Loader/Package.

Range

$$-10^{49} < X < +10^{49}$$

Method

The hyperbolic tangent is computed by means of the expression

$$\frac{e^{2x}-1}{e^{2x}+1}$$

Error Approximation

The maximum absolute error is $< 10^{-7}$

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. A minimum of 16 storage locations must be provided to store instructions and constants.
3. Accumulators 1, 2, and 3; index words 93, 94, and 98; and the accumulator overflow indicators must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

Y = TANHF (Z**2)

and return would be to the next FORTRAN statement, with the normalized unrounded result in Y.

In any Autocoder program with the symbolic deck for TANHF as an out-of-line subroutine, an example of an entry into the TANHF subroutine is:

```
ZA1   Result of (Z**2) in floating
        decimal radians
BLX   94, TANHF
```

Return is made to the instruction following the BLX, with the normalized unrounded result in Accumulator 1.

XDIMF

Purpose

This subroutine determines the positive difference between two arguments, K_1 and K_2, with both arguments in fixed-point form.

Range

$$-9\ 999\ 999\ 999 \leq K_i \leq +9\ 999\ 999\ 999$$

Method

Obtaining the positive difference is accomplished by subtracting as follows:

(argument 1) - (argument 2)

If this result is ≤ 0, + 0 is used as the result.

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. A minimum of 10 storage locations must be provided for storing instructions.
3. Accumulators 2 and 3, and index word 94 must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

M = XDIMF (K1, K2)

and return would be to the next FORTRAN statement, with the resultant numerical value in fixed-point form in M.

In any Autocoder program with the symbolic deck for XDIMF as an out-of-line subroutine, an example of an entry into the XDIMF subroutine is:

```
BLX     94, XDIMF
HB      K1
HP      K2
```

Return is made to the instruction following the HP, with resultant numerical value in fixed-point form in Accumulator 2.

XMAX0F AND XMIN0F

Purpose

This subroutine determines the largest or the smallest value of two or more arguments, K_1, K_2, . . ., K_n, in fixed-point form.

Range

$$-9\ 999\ 999\ 999 \leq K_i \leq +9\ 999\ 999\ 999$$

Method

The largest (or smallest) value is chosen by comparing

K_i with K_i where i = 1, 2, . . ., n

After the first comparison, each successive comparison is made with the larger (or smaller) value of the previous comparison.

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The subroutine XMIN0F must be provided for usage with XMAX0F.
3. A minimum of 16 for XAX0F and 13 for XMIN0F storage locations must be provided.
4. Accumulators 1, 2, and 3, and index words 93 and 94 must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

I = XMAX0F(K1, K2, K3, K4, K5) or
J = MIN0F(L1, L2, L3, L4, L5)

and return would be to the next FORTRAN statement, with the resultant numerical value in fixed-point form in I or J.

In any Autocoder program with the symbolic deck for XMAX0F or XMIN0F as an out-of-line subroutine, an example of an entry into the XMAX0F or XMIN0F subroutine is:

```
BLX   94, XMAX0F        BLX 94, XMIN0F
HB    K1                HB  L1
HB    K2                HB  L2
HB    K3          or    HB  L3
HB    K4                HB  L4
HP    K5                HP  L5
```

Return is made to the instruction following the HP, with the resultant numerical value in fixed-point form in Accumulator 2

XMAX1F AND XMIN1F

Purpose

This subroutine determines the largest or the smallest value of two or more arguments, X_1, X_2, . . ., X_n in floating-decimal form.

Range

$$-10^{49} < X_1 < +10^{49}$$

Method

The largest (or smallest) value is chosen by comparing

X_i with X_{i+1} where i = 1, 2, . . ., n.

After the first comparison, each successive comparison is made with the larger (or smaller) value of the previous comparison.

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The subroutine XMIN1F must be provided for usage with XMAX1F.
3. The subroutine XFIXFLT (included in the 7070 Series Loader/Package) must be provided.
4. A minimum of 17 for XMAX1F and 14 for XMIN1F storage locations must be provided.
5. Accumulators 1, 2, and 3; index words 93, 94 and 98; and the accumulator overflow indicators must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

I = MAX1F(X1, X2, X3, X4) or
J = MIN1F(W1, W2, W3, W4)

and return would be to the next FORTRAN statement, with the resultant numerical value in fixed-point form in I or J.

In any Autocoder program with the symbolic deck for XMAX1F or XMIN1F as an out-of-line subroutine, an example of an entry into the XMAX1F or XMIN1F subroutine is:

```
BLX 94, XMAX1F          BLX 94, XMIN1F
HB  X1                  HB  W1
HB  X2          or      HB  W2
HB  X3                  HB  W3
HP  X4                  HP  W4
```

Return is made to the instruction following the HP, with the resultant numerical value in fixed-point form in Accumulator 2.

XMODF

Purpose

This subroutine determines the remainder between two arguments, N_1 and N_2, by taking N_1 (modulo N_2), with both arguments in fixed-point form.

Range

$$-9\ 999\ 999\ 999 \leq N_i \leq +9\ 999\ 999\ 999$$

Method

Obtaining the remainder is accomplished as follows:

$$N_1 \text{ (modulo } N_2) \quad \text{or} \quad \frac{N_1}{N_2}$$

with the result being just the remainder.

If $|N_1| = 0$, the remainder is N_1.
If $|N_2| = 0$, the result is set to N_1.
If $|N_1| \geq |N_2|$, the remainder is obtained as shown
If $|N_1| < |N_2|$, the remainder is N_1

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. A minimum of 11 storage locations must be provided for storing instructions.
3. Accumulators 1, 2, and 3; index words 93 and 94; and the accumulator overflow indicators must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

L = XMODF (N1, N2)

and return would be to the next FORTRAN statement, with the resultant numerical value in fixed-point form in L.

In any Autocoder program with the symbolic deck for XMODF as an out-of-line subroutine, an example of an entry into the XMODF subroutine is:

```
BLX 94, XMODF
HB  N1
HP  N2
```

Return is made to the instruction following the HP, with the resultant numerical value in fixed-point form in Accumulator 2.

XSIGNF AND SIGNF

Purpose

The routine XSIGNF transfers the sign of an argument K_2 to another argument K_1, with both arguments in fixed-point form.

The subroutine SIGNF transfers the sign of an argument X_2 to another argument X_1, with both arguments in floating-point form.

Range

$$-9\ 999\ 999\ 999 \leq K_i \leq +9\ 999\ 999\ 999$$

and

$$-10^{49} < X_i < +10^{49}$$

Method

The transfer of sign is accomplished by attaching the

(sign of argument 2) to |(argument 1)|

i.e.,

(sign of K_2) to $|(K_1)|$

(sign of X_2) to $|(X_1)|$

Error Approximation

No error is produced in the resultant numerical value.

Requirements for Subroutine Usage

1. The machine must be in the sense mode for both sign change and field overflow.
2. The subroutine SIGNF must be provided when subroutine XSIGNF is to be used.
3. A minimum of 10 for XSIGN and 9 for SIGNF storage locations must be provided.
4. Accumulators 1, 2, and 3, and index word 94 must be available for use.

Entry and Exit

Using FORTRAN, an entry example could be

I = XSIGNF(K1, K2 + 3) or

A = SIGNF(X1-10., X2 + 1.)

and return would be to the next FORTRAN statement, with the resultant numerical value in fixed-point form in I, or in floating-point form in A.

In any Autocoder program with the symbolic deck for XSIGNF or SIGNF as an out-of-line subroutine, an example of an entry into the XSIGNF or SIGNF subroutine is:

```
BLX 94, XSIGNF          or    BLX 94, SIGNF
HB  K1                        HB  Word containing X1-10.
HP  Word containing K2 + 3    HP  Word containing X2+1.
```

Return is made to the instruction following the HP with the resultant numerical value in fixed-point form in Accumulator 2 for XSIGNF. The resultant numerical value will be in normalized floating form in Accumulator 1 for SIGNF.

This appendix provides an explanation of condition codes that may occur using the 729 tape units or 7340 Hypertape drives. These condition codes specify whether the priority signal is due to a normal or an unusual condition.

CONDITION CODES OCCURRING WITH IBM 729 MAGNETIC TAPE UNITS

Code	Symbol	Explanation
0	TWE	Tape word error occurs during 729 tape reading if the record has: (1) less than five digits or more than ten digits before the sign character is detected (numeric code); (2) a mode change character (delta) anywhere in the record except between words
1	ERROR	Either a tape data error or a machine error
2	CLR	Correct length record
3	SLR	Short length record
4	LLR	Long length record
5	EOF	End of file occurs when reading a tape mark or writing into the foil strip, indicating that the end of the 729 tape reel has been reached
6	EOS	End of segment
7	SCLR	Short character length record

CONDITION CODES OCCURRING WITH IBM 7340 HYPERTAPE DRIVES

Code	Explanation
1	Channel error (information bus validity check, address bus validity check, inhibit validity check, translation error, data error, control error)
2	Short length record
3	Long length record
4	Unusual end caused by error in 7640 Hypertape control of Hypertape Drives; e.g., trying to write on a file protected Hypertape Drive

APPENDIX F: SUMMARY OF DIFFERENCES: BASIC AND FULL FORTRAN

In the course of this manual, differences between the Full FORTRAN language and the Basic FORTRAN language were pointed out as the various parts of the languages were defined and explained. The following list is a summary of these differences:

1. Basic FORTRAN does not include the following statements:
   FUNCTION
   SUBROUTINE
   CALL
   RETURN
   COMMON
2. Basic FORTRAN language allows a maximum of two subscripts, while the Full FORTRAN language allows a maximum of three subscripts.
3. Basic FORTRAN allows two levels of implied DOs in the list of an input/output statement; Full FORTRAN allows three levels.
4. Basic FORTRAN does not include Arithmetic Statement Functions.
5. The lengths of names differ between Full FORTRAN and Basic FORTRAN as shown in the table below:

	Number of Characters	
Type of Name	Full FORTRAN	Basic FORTRAN
Fixed-point Variable	1-6	1-5
Floating-point Variable	1-6	1-5
Library Functions	4-7	4-6

6. The maximum sizes of source programs differ in Full FORTRAN and Basic FORTRAN (see discussions on "Limitation on Source Program Size").

INDEX

Reader's Comments

IBM 7070-Series Programming Systems
FORTRAN

C28-6170-1

From

Name _____

Address _____

Your comments regarding the completeness, clarity, and accuracy of this publication will help us improve future editions. Please check the appropriate items below, add your comments, and mail.

	YES	NO
Does this publication meet the needs of you and your staff?	____	____
Is this publication clearly written?	____	____
Is the material properly arranged?	____	____

If the answer to any of these questions is "NO," be sure to elaborate.

How can we improve this publication? Please answer below.

☐ Suggested Addition (Page , Timing Chart, Drawing, Procedure, etc.)

☐ Suggested Deletion (Page)

☐ Error (Page)

COMMENTS:

No Postage Necessary if Mailed in U.S.A.

C28-6170-1

FOLD FOLD

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY
IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y.

ATTN. PROGRAMMING SYSTEMS PUBLICATIONS
DEPARTMENT D91

FOLD FOLD

CUT ALONG LINE

Printed in U. S. A. C28-6170-1

IBM

International Business Machines Corporation
Data Processing Division, 112 East Post Road, White Plains, N. Y.

STAPLE