IBM

Reference Manual

FORTRAN II

for the IBM 704 Data Processing System

Reference Manual

FORTRAN II

for the IBM 704 Data Processing System

MINOR REVISION

This edition, C28-6000-2, is a minor revision of the previous edition, C28-6000-1, but does not obsolete it or C28-6000. The principal change is the substitution of a new discussion of the COMMON statement.

TABLE OF CONTENTS

GENERAL INTRODUCTION

The original FORTRAN language was designed as a concise, convenient means of stating the steps to be carried out by the IBM 704 Data Processing System in the solution of many types of problems, particularly in scientific and technical fields. As the language is simple and the 704, with the FORTRAN Translator program, performs most of the clerical work, FORTRAN has afforded a significant reduction in the time required to write programs.

The original FORTRAN language contained 32 types of statements. Virtually any numerical procedure may be expressed by combinations of these statements. Arithmetic formulas are expressed in a language close to that of mathematics. Iterative processes can be easily governed by control statements and arithmetic statements. Input and output data are flexibly handled in a variety of formats.

The FORTRAN II language contains six new types of statements and incorporates all the statements in the original FORTRAN language. Thus, the FORTRAN II system and language are compatible with the original FORTRAN, and any program used with the earlier system can also be used with FORTRAN II. The 38 FORTRAN II statements are listed in Appendix A, page 59.

The additional facilities of FORTRAN II effectively enable the programmer to expand the language of the system indefinitely. This expansion is obtained by writing subprograms which define new statements or elements of the FORTRAN II language. All statements so defined will be of a single type, the CALL type. All elements so defined will be the symbolic names of single-valued functions. Each new statement or element, when used in a FORTRAN II program, will constitute a call for the defining subprogram, which may carry out a procedure of any length or complexity within the capacity of the computer.

The FORTRAN II subprogram facilities are completely general; subprograms can in turn use other subprograms to whatever degree is required. These subprograms may be written in source program language. For example, subprograms may be written in FORTRAN II language such that matrices may be processed as units by a main program. Also, for example, it is possible to write SAP (SHARE Assembly Program) subprograms which perform double precision arithmetic, logical operations, etc.

Certain additional advantages flow from the above concept. Any program may be used as a subprogram (with appropriate minor changes) in FORTRAN II, thus making use, as a library, of programs previously written. A large program may be divided into sections and each section written, compiled, and tested separately. In the event it is desirable to change the method of performing a computation, proper sectioning of a program will allow this specific method to be changed without disturbing the rest of the program and with only a small amount of recompilation time.

There are two ways FORTRAN II links a main program to subprograms, and subprograms to lower level subprograms.

The first way is by statements of the new CALL type. This type may be indefinitely expanded, by means of subprograms, to include particular statements specifying any procedures whatever within the power of the computer. The defining subprogram may be any FORTRAN II subprogram, SAP subprogram, or program written in any language which is reducible to machine language. Since a subprogram may call for other subprograms to any desired depth, a particular CALL statement may be defined by a pyramid of multi-level subprograms. A particular CALL statement consists of the word CALL, followed by the symbolic name of the highest level defining subprogram and a parenthesized list of arguments.

A FORTRAN II subprogram to be linked by means of a CALL statement must have a SUBROUTINE statement as its first statement. SUBROUTINE is followed by the name of the subprogram and by a number of symbols in parentheses. The symbols in parentheses must agree in number, order, and mode with the arguments in the CALL statement used to call this subprogram. A subprogram headed by a SUBROUTINE statement has a RETURN statement at the point where control is to be returned to the calling program. A subprogram may, of course, contain more than one RETURN statement.

The second way in which FORTRAN II links programs together is by means of an arithmetic statement involving the name of a function with a parenthesized list of arguments. The function terminology in the FORTRAN II language may be indefinitely expanded to include as elements of the language any single-valued functions which can be evaluated by a process within the capacity of the computer. The power of function definition was available in the original FORTRAN but has been made much more flexible in FORTRAN II.

As in the original FORTRAN, library tape functions and built-in functions may be used in any FORTRAN II program. The library tape functions may be supplemented as desired. Two new built-in functions have been added in FORTRAN II, and provision has been made for the addition of up to ten by the individual installation. The most flexible and powerful means of function definition in FORTRAN II is, however, the subprogram headed by a FUNCTION statement. The FUNCTION statement specifies the function name, followed by a parenthesized list of arguments corresponding in number, order, and mode to the list following the function name in the calling program. This new facility enables the programmer to define functions in source language in a subprogram which can be compiled from alphanumeric cards or tape in the same way as a main program. Function subprograms may use other subprograms to any depth desired. A subprogram headed by a FUNCTION statement is logically terminated by a RETURN statement(s) in the same manner as the SUBROUTINE subprogram. Subprograms of the function type may also be written in SAP code, or in any other language reducible to machine language.

Subprograms of the function type may freely use subprograms of both the subroutine type and the function type without restriction. Similarly, the subroutine type may use subprograms of both the subroutine type and the function type without restriction.

The names of variables listed in a subprogram in a SUBROUTINE or FUNCTION statement are dummy variables. These names are independent of the calling program and, therefore, need not be the same as the corresponding variable names in the calling program, and may even be the same as non-corresponding variable names in the calling program. This enables a subprogram or group of subprograms to be used with various independently written main programs.

There are many occasions when it is desirable for a subprogram to be able to refer to variables in the calling program without requiring that they be listed every time the subprogram is to be used. Such cross-referencing of the variables in a calling program and in various levels of subprograms is accomplished by means of the COMMON statement which defines the storage areas to be shared by several programs. This feature also gives the programmer more flexible control over the allocation of data in storage.

The END statement has been added to the FORTRAN II language for multiple program compilation, another new feature of FORTRAN II. This statement acts as an end-of-file for either cards or tape so that there may be many programs in the card reader or on a reel of tape at any one time. Five digits in parentheses follow the END statement. These digits refer to the first five Sense Switches on the 704 Console, allowing the programmer, if he wishes, to indicate to the Translator which of certain options it is to take, regardless of the actual setting of the Sense Switches.

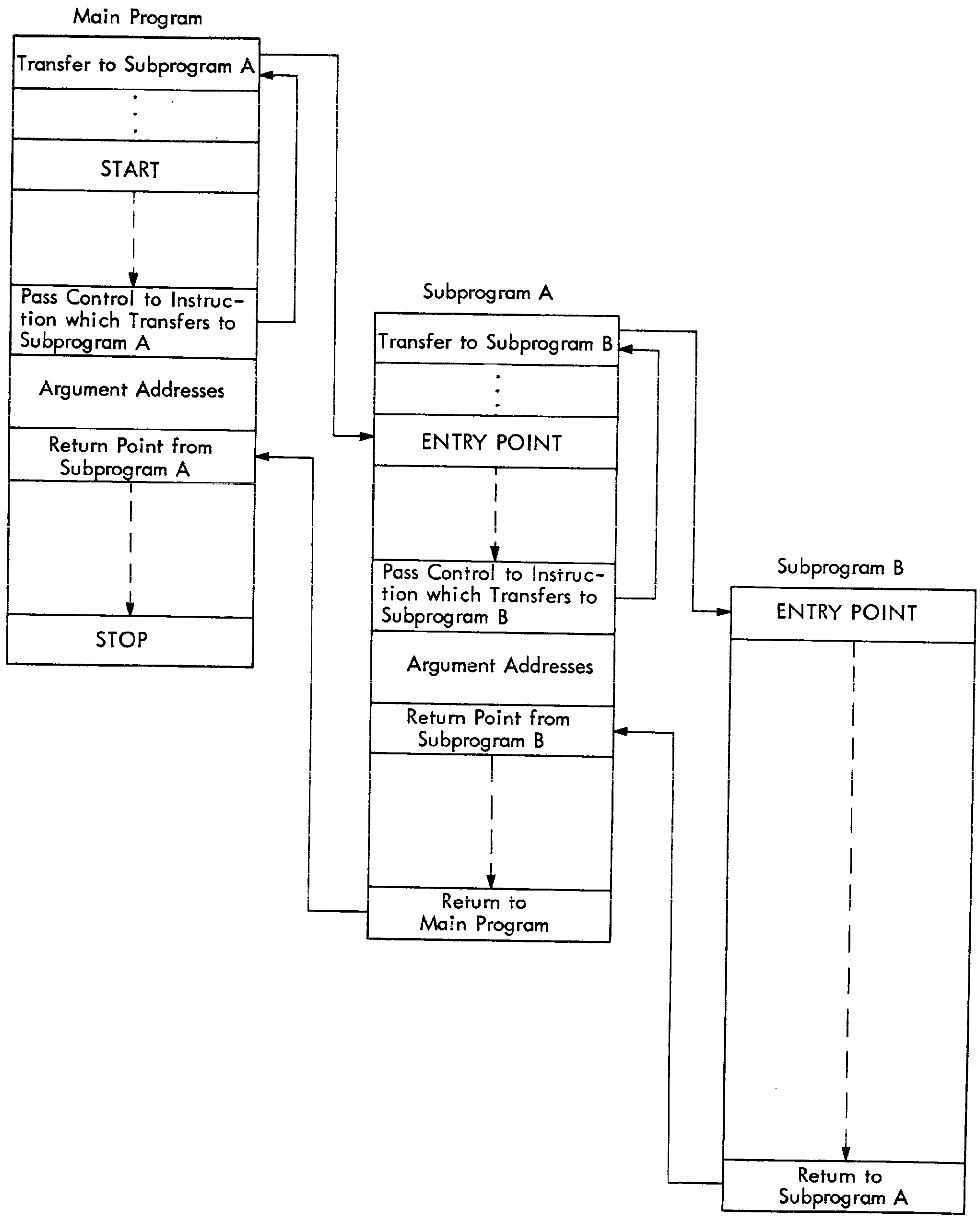
In an early phase of the FORTRAN II Translator, a diagnostic program has been incorporated which finds many types of errors much earlier during the compilation process, provides more complete information on error print-outs, and reduces the number of stops. Thus, both programming time and machine compilation time are saved.

The object programs, both main programs and subprograms, are stored in 704 memory by the Binary Symbolic Subroutine Loader. The Loader interprets symbolic references between a main program and its subprograms and between various levels of subprograms and provides for the proper flow of control between the various programs during program execution.

Because of the function of the Loader, the programmer need know only the symbolic name of an available subprogram and the procedure which it carries out; he does not need to be concerned with the constitution of the machine language deck, nor with the location of the subprogram in storage. In machine language decks, symbolic references are retained in a set of names, or "Transfer List," at the beginning of each program which calls for subprograms. The symbolic name of each subprogram is also retained on a special card, the "Program Card," at the front of each subprogram deck. At the beginning of loading, a call for a subprogram is a transfer to the appropriate symbolic name in the Transfer List. Before program execution commences, the Loader replaces the Transfer List names with transfers to the actual locations occupied in storage by the corresponding subprogram entry points.

The order in which the decks are loaded determines the actual locations occupied by the main program and subprograms in storage but does not affect the logical flow of control. The order in which decks are loaded is therefore arbitrary.

The following diagram illustrates the flow of control between a main program and two subprograms, each of which may be of either the function type or the subroutine type. The main program calls for subprogram A, and subprogram A calls for subprogram B.


Main Program
Transfer to Subprogram A
START
Pass Control to Instruction which Transfers to Subprogram A
Argument Addresses
Return Point from Subprogram A
STOP
Subprogram A
Transfer to Subprogram B
ENTRY POINT
Pass Control to Instruction which Transfers to Subprogram B
Argument Addresses
Return Point from Subprogram B
Return to Main Program
Subprogram B
ENTRY POINT
Return to Subprogram A

NOTE ON ASSOCIATED PUBLICATIONS

Part I of this manual supplements the FORTRAN Reference Manual, Form No. 32-7026. Taken together, Part I and Form No. 32-7026 completely define the FORTRAN II language.

Part II of this manual supplements the FORTRAN Programmer's Primer, Form No. 32-0306. Part II assumes familiarity with the following types of FORTRAN statements:

Arithmetic: a = b
GO TO n
IF (a) n_1, n_2, n_3
STOP
DO n i = m_1, m_2
CONTINUE
FORMAT (Specification)
READ n, List
PRINT n, List
DIMENSION v, v, v, ...

Part II, Chapter 3, which deals with the use of non-FORTRAN subprograms, assumes in addition a basic knowledge of the symbolic code acceptable to the SHARE Assembly Program, presented in SHARE Distribution No. 347.

Part III of this manual describes the Binary Symbolic Subroutine Loader and supersedes all information previously distributed to FORTRAN users on the loading of object programs produced by the original FORTRAN system.

The FORTRAN Reference Manual and Programmer's Primer may be obtained from the IBM Sales Representative. SHARE Distribution No. 347 may be obtained from:

SHARE Program Librarian
International Business Machines Corporation
590 Madison Avenue
New York 22, New York

PART I: THE FORTRAN II LANGUAGE

CHAPTER 1 - GENERAL PROPERTIES OF A FORTRAN II SOURCE PROGRAM

Types of Statements

A FORTRAN II source program consists of a sequence of FORTRAN II statements. There are 38 different types of statements in FORTRAN II, including the 32 types of statements in the original FORTRAN and 6 additional types. The 6 new types of statements are described in detail in Part I, Chapter 3. Since the original FORTRAN statements are part of the FORTRAN II language, all programs written for the original FORTRAN system are proper FORTRAN II programs.

Types of Source Programs

A FORTRAN II source program can be either a main program or a subprogram. Source subprograms are of two types:

a. Subroutines, which must have an initial SUBROUTINE statement (see page 17).
b. Functions, which must have an initial FUNCTION statement (see page 18).

Preparation of Input to FORTRAN II Translator

There are no changes in the way statement cards are punched. Main programs and subprograms are prepared as distinct sets of cards. The actual input to the Translator may be a deck consisting of one or more programs, or may be a binary-coded decimal (BCD) tape written from a deck on the card-to-tape equipment with the standard SHARE 80 x 84 board. On both cards and tape, the END statement (see page 22) is treated as an end-of-file indication, thereby permitting multiple program compilation with a single loading of the Translator.

Classification of the New FORTRAN II Statements

The 6 new types of statements may be included in two of the original FORTRAN classifications as follows:

<u>Control Statements</u>

CALL

RETURN

END

<u>Specification Statements</u>

SUBROUTINE

FUNCTION

COMMON

Arithmetic Statements

As in the previous FORTRAN system, there are two kinds of arithmetic statements. The terms used in previous FORTRAN literature for the two kinds of arithmetic statements are arithmetic formula and function statement. The latter should not be confused with the new FUNCTION statement. To avoid confusion of terms, the second kind of arithmetic statement will be referred to subsequently in this manual as a function definition.

The formal description of the function definition is as follows:

General Form	Examples
"a = b" where a is a function name followed by parentheses enclosing its arguments (which must be distinct non-subscripted variables) separated by commas, and b is an expression which does not involve subscripted variables. The function name on the left side of the function definition consists of 4 to 7 alphabetic or numeric characters (not special characters), of which the last must be F and the first must be alphabetic. Also, the first must be X if and only if the value of the function is to be fixed point. Any functions appearing on the right side must be built-in, or available on the library tape, or already defined by preceding function definitions, or defined by a function subprogram.	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

In the left column on the following page are examples of arithmetic formulas as defined in the FORTRAN manual.

In the right column are the equivalent algebraic formulas.

FORTRAN	ALGEBRA
S(I) = A**2.0 - (B/C) * D * * X	$s_i = a^2 - \frac{b}{c} d^x$
Y(3) = A - COSF (C/B)	$y_3 = a - \cos \frac{c}{b}$
P = MAX1F (A, B, C, D)	$p = \max (a, b, c, d)$
R = ARGF (S, T, W) + A/B	$r = \arg (s, t, w) + \frac{a}{b}$

The second FORTRAN example involves COSF, a library tape function. The third FORTRAN example involves MAX1F, a built-in function. Assuming ARGF is not a library tape function, the last of the above FORTRAN examples is valid only if ARGF has previously appeared in the program as a function name, with three arguments, on the left side of a function definition; for example:

ARGF (X, Y, Z) = (D/E) * Z + X * * F + Y/G

This FORTRAN statement is equivalent to the algebraic function definition:

$$\arg (x, y, z) = \frac{d}{e} z + x^f + \frac{y}{g}$$

The function definition is very convenient, but it is limited to functions which can be defined in one statement. FORTRAN II provides a new means of defining functions - the subprogram with an initial FUNCTION statement. The new FORTRAN II function is not subject to the foregoing limitation.

Types of Functions

In summary, there are four types of functions in FORTRAN II:

1. Built-in Functions: These functions are pre-defined as part of the FORTRAN II system and are compiled by the FORTRAN II Translator as open subroutines. In all, there are 20 built-in functions in FORTRAN II, the 18 functions listed on page 13 of the FORTRAN Reference Manual and 2 new functions:

Type of Function	Definition	# of Args	Name	Mode of Argument	Mode of Function
Diminishing (See note below)	Arg_1 (dim Arg_2)	2	DIMF XDIMF	Floating Fixed	Floating Fixed

NOTE: The function DIMF (Arg_1, Arg_2) is defined as Arg_1 - Min (Arg_1, Arg_2).

Provision has been made for the addition of up to ten built-in functions by the individual installation. Details are given in Appendix B, page 61. Also in Appendix B, the twenty built-in functions provided with FORTRAN II are listed for convenient reference.

2. Library Tape Functions: These functions are pre-defined in 704 language and stored on the FORTRAN II library tape. In FORTRAN II, library tape functions are compiled as closed subroutines, relocatable relative to 0. The library tape functions to be distributed with FORTRAN II, which may be supplemented, altered, or replaced by the individual installation, are the following:

LOG	Logarithm
SIN	Sine
COS	Cosine
EXP	Exponential
SQRT	Square Root
ATAN	Arctangent
TANH	Hyperbolic Tangent

3. Functions Defined by a Single Arithmetic Statement: These functions are as described above.
4. Functions Defined by a Function Subprogram: These functions are defined either by a FORTRAN II subprogram headed by a FUNCTION statement specifying the name of the function, or by an equivalent non-FORTRAN subprogram. Compiled function subprograms, like compiled library tape functions, are relocatable relative to 0. The FUNCTION statement is described in detail on page 18.

Once defined, the four types of functions are used in the same way in a FORTRAN II source program; that is, the function is specified in an arithmetic formula. For example, the arithmetic formula

```
Y(3) = A - COSF (C/B)
```

will cause the cosine of C/B to be computed, using the library tape cosine function, and subtracted from A; the result will then be assigned as the current value of Y(3).

Function Names

The FORTRAN II Translator distinguishes a function name with a parenthesized list of arguments from a subscripted variable by examining the DIMENSION statements in the program. A subscripted variable must be listed in a DIMENSION statement, and a function name must not be so listed. Having determined that an alphanumeric symbol is a function name, the Translator determines the type of function by examining the characters used to specify the function.

1. A built-in function is specified by a name which is uniquely reserved for it and listed in a dictionary on the system tape.
2. A library tape function is specified by its name with a terminal F added. With the terminal F added, the name consists of 4 to 7 alphabetic or numeric characters, of which the first is alphabetic and the last is F. Also, the first must be X if and only if the value of the function is to be fixed point.
3. A function defined by an arithmetic statement (function definition) is specified by the name on the left side of the function definition. This name is not distinct in form from a library tape function name, but is recognized because of its previous occurrence in a function definition.
4. A function defined by a FUNCTION subprogram is specified by a name which is formally distinct from the names of other types of functions. This name consists of 1 to 6 alphabetic or numeric 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 last character must not be F if the total number of characters is 4 or more.

Additional Examples

1. <u>Built-in Function</u>
   Means of definition: The open subroutine MAX1F on the FORTRAN II system tape.

   Example of use in a source program:

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
		READ 2, A, B, C, D, E
		BIGX = MAX1F (A, B, C, D, E,)
		PRINT 2, BIGX
2		FORMAT (1P5E14. 5)
		STOP

2. Library Tape Function

Means of definition: The closed subroutine SQRT on the FORTRAN II library tape.

Example of use in a source program:

```
C FOR COMMENT / STATEMENT NUMBER | CONTINUATION | FORTRAN STATEMENT
      READ 2, A, B, C
    2 FORMAT (1P5E14.5)
      X=(-B - SQRTF (B ** 2.0 - 4.0 * A * C))/(2.0*A)
      PRINT 2, X
      STOP
```

Note that from the point of view of the writer of the source program the terminal F is part of the name of the library tape function.

3. Function Defined by an Arithmetic Statement

Example of the definition (first statement) and use of this type of function:

```
C FOR COMMENT / STATEMENT NUMBER | CONTINUATION | FORTRAN STATEMENT
      TANF (R) = SINF (R) / COSF (R)
      READ 2, X, Y
      A = TANF (X)
      B = TANF (Y)
      C = TANF (X + Y)
      PRINT 2, A, B, C
    2 FORMAT (1P5E14.5)
      STOP
```

4. Function Defined by a FUNCTION Subprogram

Example of a FUNCTION subprogram, defining the function SUM:

```
C FOR COMMENT / STATEMENT NUMBER | CONTINUATION | FORTRAN STATEMENT
      FUNCTION SUM (A, NA, B, NB)
      DIMENSION A(500), B(500)
      SUM=A(1)
      DO 5 J=2, NA
    5 SUM= SUM + A(J)
      DO 10 I=1, NB
   10 SUM= SUM + B(I)
      RETURN
```

Example of the use of the SUM function in a main program (NX, NY, NV, and NW are each ≤ 500):

```
C FOR COMMENT / STATEMENT NUMBER | CONTINUATION | FORTRAN STATEMENT
      DIMENSION X(500), Y(500), V(500), W(500)
      READ 2, NX, NY, NV, NW, X, Y, V, W
      AVERG= (SUM(X,NX,Y,NY) + SUM(V,NV,W,NW))/FLOATF (NX + NY + NV + NW)
      PRINT 10, AVERG
    2 FORMAT (4I8/ (1P5E14.5))
   10 FORMAT (35H AVERAGE OF X, Y, V, AND W LISTS IS 1PE14.5)
      STOP
```

CALL

General Form	Examples
CALL NAME (a_1, a_2, ..., a_n) where NAME stands for the symbolic name of a subroutine, and the arguments a_1, a_2, ..., a_n, if any, may each have one of seven forms, described in the text 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 causes transfer of control to the subroutine NAME and presents the subroutine with the arguments, if any, enclosed in parentheses. The order of the arguments is taken from the list, reading from left to right. There must be agreement in number, order, and mode between the argument list of the corresponding SUBROUTINE statement.

An argument in the CALL statement must 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. A FORTRAN II arithmetic expression
6. An argument of the following form:

$$nHx_1\ x_2 \ldots x_n$$

where the x's are any n Hollerith characters.

A Hollerith argument is interpreted in the same way as a Hollerith field in a FORMAT statement. It is not the name of a variable but, as with constants, is itself the data to be operated on. A Hollerith argument is stored as follows:

1. The characters nH are dropped.
2. The first Hollerith character x_1 is stored as the first character of the first word.
3. The remaining characters, including blanks, are stored as successive characters, six to a word, in successive words.
4. If the last word contains less than six characters, it is filled out with blanks.
5. A word consisting of 36 binary 1's is stored immediately after the last word.

SUBROUTINE

General Form	Examples
SUBROUTINE NAME (a_1, a_2, ..., a_n) where NAME stands for the symbolic name of a subroutine, and the arguments a_1, a_2, ..., a_n, if any, are non-subscripted variable names. The subroutine name consists of 1 to 6 alphanumeric characters, the first of which is alphabetic; the final character must not be F if the total number of characters is 4, 5, or 6. Also, the subroutine name must not occur in a DIMENSION statement in the subroutine, nor in a DIMENSION statement in any program having a CALL for the subroutine. The arguments may be any variable names occurring in executable statements in the subroutine.	SUBROUTINE MATMPY (A, N, M, B, L, C) SUBROUTINE QDRTIC (B, A, C, ROOT1, ROOT2)

This statement, when used, must be the first statement in a program and defines the program to be a subroutine. A subroutine introduced by a SUBROUTINE statement must be a FORTRAN II program and may contain any FORTRAN II statements except a FUNCTION statement or another SUBROUTINE statement. If several programs are stacked together to form a single source language deck for multiple program compilation, each SUBROUTINE or FUNCTION statement marks the beginning of a new program, and successive programs must be separated by an END statement. Thus, it is not permissible to insert a subprogram between two statements of a higher level program.

A subroutine introduced by a SUBROUTINE statement is called into the main program by a CALL statement specifying the name of the subroutine. For example, the subroutine introduced by SUBROUTINE MATMPY (A, N, M, B, L, C) could be called into the main program by the statement

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

In the above, X, Y, and Z are matrices which are given the same size dimensions in the DIMENSION statement of the calling program as A, B, and C are given in the DIMENSION statement of the subroutine. The operations specified in the subroutine for A, N, M, B, L, and C would be performed on the X matrix, 5, 10, the Y matrix, 7, and the Z matrix, respectively. Note the correspondence between the list in the main program and the list in the subroutine. There must be agreement in number, order, and mode between the argument list following the subroutine name in the CALL statement and the argument list in the SUBROUTINE statement.

If an argument is the name of an array, it must appear in a DIMENSION statement following the SUBROUTINE statement. A DIMENSION statement must be given in the main program, specifying the same dimensions for the corresponding CALL statement argument. The actual dimensions of the array must be less than or equal to the specified dimensions. If the actual dimensions are less, this information can be conveyed to the subprogram by means of arguments which, in the subroutine, are indexing parameters. In this case, of course, some of the locations reserved for the array will be unused.

FUNCTION

General Form	Examples
FUNCTION NAME (a_1, a_2, ..., a_n) where NAME stands for the symbolic name of a single-valued function, and the arguments a_1, a_2, ..., a_n are non-subscripted variable names. The function name consists of 1 to 6 alphanumeric characters, the first of which is 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 the total number of characters is 4, 5, or 6. Also, the function name must not occur in a DIMENSION statement in the FUNCTION subprogram, nor in a DIMENSION statement in any program which uses the function. The arguments may be any variable names occurring in executable statements in the subprogram. There must be at least one argument.	FUNCTION ARCSIN (RADIAN) FUNCTION ROOT (B, A, C) FUNCTION INTRST (RATE, YEARS)

This statement, when used, must be the first statement in a program and defines the program to be a function subprogram. A subprogram introduced by a FUNCTION statement must be a

FORTRAN II program which evaluates a single-valued function, that is, a function which has one and only one value for a given set of arguments. The FUNCTION subprogram may contain any FORTRAN II statements except a SUBROUTINE statement or another FUNCTION statement. If several programs are stacked together to form a single source language deck for multiple program compilation, each FUNCTION or SUBROUTINE statement marks the beginning of a new program, and successive programs must be separated by END statements. Thus, it is not permissible to insert a subprogram between two statements of a higher level program.

A subprogram introduced by a FUNCTION statement is called for in the main program by an arithmetic formula involving the function name. For example, the subprogram introduced by FUNCTION ARCSIN (RADIAN) could be called for in the main program by the arithmetic formula:

A = B - ARCSIN (X)

The current value of the argument X would be assigned to RADIAN in the subprogram. The arcsine of X would be computed and subtracted from the current value of B, and the difference assigned as the value of A. Note the correspondence between X in the main program and RADIAN in the FUNCTION statement. There must be agreement in number, order, and mode between the argument list following the function name in the main program and the argument list in the FUNCTION statement.

If an argument is the name of an array, it must appear in a DIMENSION statement following the FUNCTION statement. A DIMENSION statement must be given in the main program, specifying the same dimensions for the corresponding function argument appearing in an arithmetic formula. The actual dimensions of the array must be less than or equal to the specified dimensions. If the actual dimensions are less than the specified dimensions, some of the locations reserved for the array will be unused.

In a FUNCTION subprogram, the name of the function must be evaluated as a variable on the left side of an arithmetic formula; for example, by means of a DO loop:

```
C FOR COMMENT
STATEMENT NUMBER | CONTINUATION | FORTRAN STATEMENT
      .
      .
      .
      .
      NAME = 0.0
      DO 5 I = 1, 10
    5 NAME = NAME + L(I) + M(I)
      .
      .
      .
      .
```

It is the final value of the function name, used as a variable, that is returned as the function value.

COMMON

GENERAL FORM	EXAMPLES
"COMMON A, B,..." where A, B,.... are the names of variables and non-subscripted array names.	COMMON X, ANGLE, MATA, MATB

Variables, including array names, appearing in COMMON statements are assigned to upper storage. They are stored in locations completely separate from the block of program instructions, constants, and data. This area is assigned separately for each program compiled. The area is assigned beginning at location 77462_8 and continuing downwards. This separate (COMMON) area may be shared by a program and its subprograms. In this way, COMMON enables data storage area to be shared between programs in a way analogous to that by which EQUIVALENCE permits data storage sharing within a single program. Where the logic of the programs permit, this can result in a large saving of storage space.

Array names appearing in COMMON 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. 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.

Arguments in Common Storage

Because of the above, COMMON statements may be used to serve another important function. They may be used as a medium by which to transmit arguments from the calling program to the called FUNCTION subprogram or SUBROUTINE subprogram. In this way, they are transmitted implicitly rather than explicitly by being listed in the parentheses following the subroutine name.

To obtain implicit arguments, it is necessary only to have the corresponding variables in the two programs occupy the same location. This can be obtained 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 valid to place dummy variable names in a COMMON statement. These dummy names, which may be dimensioned, will cause reservation of the space necessary to cause correspondence.

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

   The entire COMMON area may be relocated downward for any one problem by means of a Control Card (See FORTRAN Operations Manual).

   When a variable is made equivalent to a variable which appears in a COMMON statement, the first variable will also be located in COMMON storage. When COMMON variables also appear in EQUIVALENCE statements, the ordinary sequence of COMMON variables is changed and priority is given to those variables in EQUIVALENCE statements, in the order in which they appear in EQUIVALENCE statements. For example,

```
COMMON A, B, C, D

EQUIVALENCE (C, G), (E, B)
```

   will cause storage to be assigned in the following way.

77462_8	C and G
77461_8	B and E
77460_8	A
77457_8	D

RETURN

General Form	Examples
RETURN	RETURN

This statement terminates a subprogram and returns control to the calling program. A RETURN statement must be the last statement to which control passes in a function subprogram or a subroutine; that is, it must be the last statement logically, but not necessarily physically.

END

General Form	Examples
END (I_1, I_2, I_3, I_4, I_5) where I is 0, 1, or 2 (fixed point integers).	END (2, 2, 2, 2, 2) END (1, 2, 0, 1, 1)

An END statement, when used, must be the physically last statement in a program. The I's in the END statement, all of which must be specified, control the interrogation of Sense Switches 1 through 5, respectively, on the 704 console:

I = 0 means "Ignore the Sense Switch and assume it to be in UP position."

I = 1 means "Ignore the Sense Switch and assume it to be in DOWN position."

I = 2 means "Interrogate the Sense Switch."

If I_n = 0 or 1, the console operator's setting of Sense Switch n will be overridden by the programmer's option. Thus either the programmer or the operator may, at the option of the programmer, control certain FORTRAN II operations. If the END statement is omitted in single program compilation, the absence of the END statement is equivalent to an END statement with all I's equal to 2.

In addition to controlling the interrogation of Sense Switches, the END statement is treated as an end-of-file on either the card reader or tape, thereby permitting multiple program compilation with a single loading of the FORTRAN II Translator. In multiple program compilation, the physically last statement in each program must be an END statement.

Sense Switch 6 in the DOWN position causes FORTRAN II to assume multiple program compilation ("batch compiling"). Sense Switch 6 in the UP position causes FORTRAN II to assume a single program is to be compiled. The functions of Sense Switches 1 through 5 are as follows:

Sense Switch 1	UP	Binary cards for the object program(s) are punched on-line. If not batch compiling, tape unit 3 contains the binary program. If batch compiling, tape unit 3 contains the binary output for the last program compiled. Tape unit 7 contains no binary programs.
	DOWN	Binary cards for the output program(s) are not punched. Tape unit 3 contains the binary program for the last or only source program compiled. If batch compiling, tape unit 7 contains the binary programs for all the source programs compiled in the order they were compiled.
Sense Switch 2	UP	Produces, on tape unit 2, two files for the source program compiled, containing the source program and a map of object program storage. If batch compiling, tape unit 6 will contain two files for each program compiled and tape unit 2 will contain two files for the last program compiled.
	DOWN	Adds a third file for each program compiled (see above) containing the object program in SAP (SHARE Assembly Program) type language on tape unit 2 (and 6, if batch compiling).
Sense Switch 3	UP	No on-line listings are produced.
	DOWN	Lists on-line the first two or three files of tape unit 2, depending on the setting of Sense Switch 2.
Sense Switch 4	UP	Causes FORTRAN II to produce a program optimized with respect to index registers.

	DOWN	Causes FORTRAN II to produce a program not fully optimized but which will be translated from a source to an object program more rapidly.
Sense Switch 5	UP	Library Routines are not to be punched out or written on tape unit 3.
	DOWN	Causes Library Routines to be punched on-line or written on tape unit 3, depending on whether Sense Switch 1 is in the UP or DOWN position.

PART II: PRIMER ON THE NEW FORTRAN II FACILITIES

Purpose of FUNCTION Subprograms

In FORTRAN II, functions which are not available as built-in functions or library tape functions and which cannot, or cannot conveniently, be defined by a function definition may be defined by a subprogram headed by a FUNCTION statement. Like the other three types of functions, the function defined by a subprogram must be single-valued; that is, it must have one and only one value for a given set of arguments. Lists and two- or three-dimensional arrays can, however, be computed and returned to the main program by a FORTRAN II subroutine - a subprogram headed by a SUBROUTINE statement. On the object program level, the only difference between a FUNCTION subprogram and a SUBROUTINE subprogram is that the single result of the FUNCTION subprogram is left in the Accumulator for further computation in the main program, and the result or results produced by the SUBROUTINE subprogram are assigned to storage locations.

Example 1: Function of an Array

One of the principle uses of the FUNCTION subprogram is to define a function of one or more arrays. The following simple example illustrates this use.

It is desired to write a subprogram to compute the average value of a one-dimensional array of N floating-point numbers, where N is less than or equal to 500. The following subprogram will carry out this procedure.

```
C FOR COMMENT
STATEMENT NUMBER | CONTINUATION | FORTRAN STATEMENT
    1      FUNCTION AVRG (ALIST, N)        FUNCTION IFIND (X)
           DIMENSION ALIST (500)
           SUM = ALIST (1)
           DO 10 I=2, N
   10      SUM=SUM+ALIST (I)
           AVRG=SUM / FLOATF (N)
           RETURN
           END (2, 2, 2, 2, 2)
```

Dummy Variables

The arguments listed in parentheses after the function name in the FUNCTION statement are dummy variables. In Example 1, the dummy variables listed as arguments of the AVRG function are ALIST and N. The dummy variable names may be different from the corresponding arguments listed after the function name in the calling program. The only requirements are:

1. The dummy variable list in the subprogram must agree in

number, order, and mode with the corresponding argument list in the calling program.

2. If a dummy variable name represents an array, equivalent DIMENSION statements must be made for the dummy variable in the subprogram and the corresponding argument in the calling program.

For example, the AVRG function could be called for in a main program as follows:

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT
		DIMENSION SET (500)
		READ 2, (SET (I), I = 1, 200)
		TEXT = AVRG (SET, 200)
		PRINT 10, TEXT
10		FORMAT (18H AVERAGE OF SET IS 1PE14.5)
2		FORMAT (5E12.5)
		STOP

Note that the DIMENSION statement in the main program specifies the same length (500) for the array named SET as the DIMENSION statement in the subprogram specifies for the dummy variable ALIST. This is required even though the actual length of SET is only 200. The argument 200 is supplied to the subprogram from the main program and is used in the subprogram as an index maximum.

Restriction on Statements Involving Dummy Variables

A dummy variable in a FUNCTION subprogram should not normally appear on the left side of an arithmetic statement, except as a variable subscript. The reason for this is that it is generally undesirable to change the value of the arguments supplied to the subprogram by the calling program. Similarly, a fixed-point dummy variable should not normally appear in an ASSIGN statement.

Example 2: Series Evaluation

The following subprogram is an example of a series evaluation with IF-type branching. The function defined by this subprogram cannot be defined in a single statement. The subprogram computes the value of arctan x, correct to 4 decimal places, for any given argument x greater than or equal to zero. Actually, the arc-tangent function is available on the FORTRAN II library tape and, in practice, would not normally be rewritten as a function sub-program. It will, however, serve to illustrate a type of problem.

Where $0 \leq x \leq 1$, the following series equation is used in the subprogram:

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

Where $x > 1$, the following equation is used:

$$\arctan x = \frac{\pi}{2} - \frac{1}{x} + \frac{\left(\frac{1}{x}\right)^3}{3} - \frac{\left(\frac{1}{x}\right)^5}{5} + \frac{\left(\frac{1}{x}\right)^7}{7} - \ldots$$

STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
1		FUNCTION ARCTAN(X)
		IF(X) 2, 3, 3
2		STOP
3		ARCTAN = 0.0
		IF (X - 1.0) 10, 10, 5
5		TERM = - 1.0/X
		ARCTAN = 1.57079
		GO TO 11
10		TERM = X
11		PREVXP = 1.0
		Y = TERM ** 2.0
12		ARCTAN = ARCTAN + TERM
		PRESXP = PREVXP + 2.0
13		TERM = - PREVXP/PRESXP* Y * TERM
		PREVXP = PRESXP
14		IF (TERM - 0.00005) 15, 12, 12
15		IF (-TERM - 0.00005) 16, 12, 12
16		RETURN
20		END (2, 2, 2, 2, 2)

Statement 13 is an arithmetic formula which calculates successive terms of either series. In this equation, the variable PRESXP stands for the exponent of X in the term currently being calculated, and PREVXP for the exponent of X in the previous term. Statements 14 and 15 test for the desired accuracy. When enough terms have been taken, the iterative calculation and summation of terms ceases. Statement 16 returns control to the calling program.

The following is a main program calling for the ARCTAN function.

```
DIMENSION A(2), B(2), C(2)
READ 2, A, B, C
ANGLE1 = ARCTAN (MAX1F (A(1), B(1), C(1)))
ANGLE2 = ARCTAN (MIN1F (A(2), B(2), C(2)))
ANGLE3 = ANGLE1 - ANGLE2
PRINT 2, ANGLE1, ANGLE2, ANGLE3
STOP
```

Note that the FUNCTION subprogram will be executed twice in carrying out the above procedure. In the computation of ANGLE1, the operations specified for X in the subprogram will be performed on the value returned by the built-in MAX1F function. Similarly, in the computation of ANGLE2, the operations specified for X in the subprogram will be performed on the value returned by the built-in MIN1F function.

Main Program Function Arguments

The main program in example 2 illustrates the use of functions as FUNCTION arguments. The dummy variable names in a subprogram argument list must be non-subscripted variables. However, any legitimate FORTRAN II constant, variable (subscripted or non-subscripted), function, expression, or name of an array may be used as a FUNCTION argument in a calling program, provided the corresponding dummy variable in the subprogram has the same mode. A Hollerith argument (see page 16) may also be used. As a Hollerith argument does not have a mode, the naming of the corresponding dummy variable is arbitrary with respect to mode. Although Hollerith arguments can be supplied as data to FORTRAN II subprograms, Hollerith arguments are useful principally in conjunction with non-FORTRAN subprograms.

Purpose of FORTRAN II Subroutines

FORTRAN II subroutines, that is, subprograms headed by a SUBROUTINE statement, may be written to carry out procedures for which the FUNCTION-type subprogram is inappropriate. In general, the subroutine is used for the computation of lists and arrays, which cannot be computed in a single run of a FUNCTION subprogram, and for the performance of segments of a total problem (e. g., complex input and output operations) which are more conveniently written separately or are applicable to more than one problem. The subroutine may also be used as an alternative to the FUNCTION subprogram in cases where there is no particular advantage to using the function notation in the calling program.

Example 1: Matrix Multiplication

Subroutine:

```
    1 SUBROUTINE MATMPY (A, N, M, B, L, C)
      DIMENSION A (10, 15), B(15, 12), C(10, 12)
      DO 5 I = 1, N
      DO 5 J = 1, L
    3 C (I, J) = 0.0
      DO 5 K = 1, M
    5 C (I, J) = C(I, J) + A(I, K) * B(K, J)
      RETURN
    7 END (2, 2, 2, 2, 2)
```

Main Program

```
    1 DIMENSION X(10, 15), Y(15, 12), Z(10, 12), D(10, 15), E(15, 12), F(10, 12)
      READ 15, X, Y
    5 CALL MATMPY (X, 5, 10, Y, 7, Z)
      READ 15, D, E
   10 CALL MATMPY (D, 6, 8, E, 5, F)
      DO 13 J = 1, 7
   13 PRINT 15, (Z (I, J), I = 1, 5)
      DO 14 J = 1, 5
```

(continued on next page)

(continued from preceding page)

C FOR COMMENT / STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
14		PRINT 15, (F(I, J), I = 1, 6)
15		FORMAT (6 E 15.6)
16		STOP

After the first transfer (statement 5 in the main program) to the subroutine MATMPY, values in the X matrix will be substituted for the dummy variable A, 5 for N, 10 for M, values in the Y matrix for B, and 7 for L. The values in the X and Y matrices will not all be used by the subroutine, since the CALL statement specifies the values 5, 10, and 7 for N, M, and L, respectively, which function in the subroutine as subscript maxima. These maxima are less than the values (10, 15, and 12, respectively) which would cause the subroutine operations to be performed on the entire X and Y matrices. The actual Z matrix computed by the subroutine will be

$$Z_{IJ} = \sum_{K=1}^{10} X_{IK} Y_{KJ}$$

for I = 1, 2, ..., 5 and J = 1, 2, ..., 7. The resulting 5 by 7 matrix will be stored in the locations for $Z_{1,1}$, $Z_{2,1}$, $Z_{3,1}$, $Z_{4,1}$, $Z_{5,1}$, $Z_{1,2}$, $Z_{2,2}$, ..., $Z_{5,7}$. The arguments listed in the second CALL statement will be similarly operated on after the second transfer to the subroutine, and the resulting 6 by 5 F matrix will be stored in the locations for $F_{1,1}$ through $F_{6,5}$. After control is returned the second time to the main program, the computed 5 by 7 Z matrix will be printed out in natural order ($Z_{1,1}$ through $Z_{5,7}$, as listed above), followed by the computed 6 by 5 F matrix in natural order ($F_{1,1}$ through $F_{6,5}$).

Dummy Variables and Main Program Arguments

The same general restrictions on the naming and use of dummy variables apply to subroutines as apply to FUNCTION subprograms. Note, however, that the SUBROUTINE list will usually contain one or more dummy variables standing for the result or results to be returned to the calling program; in example 1, C is the dummy variable for the matrix to be computed. The dummy variable for a result may be freely used on the left side of arithmetic statements, as C is used in statements 3 and 5 in the MATMPY subroutine. In addition to the use of constants,variables (subscripted or non-subscripted), functions, expressions, and the names of arrays as arguments in CALL statements, a Hollerith argument may be used, as explained on page 16. As a Hollerith argument does not have a mode, the naming of the corresponding dummy variable is arbitrary with respect to mode. In practice, as stated in the preceding chapter, Hollerith arguments are useful principally in conjunction with non-FORTRAN subprograms.

Example 2: Computation of Arrays

The condition for critical damping of a circuit consisting of resistance, capacitance, and inductance in series is given by the relation

$$C = 4L/R^2$$

where R is the resistance in ohms
L is the inductance in henrys
C is the capacitance in farads.

The instantaneous current of an LCR circuit meeting the critical damping condition is given by the relation

$$i = \frac{\bar{E} - \frac{q_0}{C}}{L} e^{-Rt/2L} t$$

where i is the instantaneous current in amperes
$\bar{E}$ is the average voltage in volts
q_0 is the initial charge on the capacitor in coulombs
t is the time elapsed in seconds since the circuit was closed
e is 2.71828
R, L, and C are as above.

A subroutine is to be written which will do the following:

1. Compute the values of capacitance which satisfy the critical damping condition for all pairs that can be formed from up to 50 values of inductance and up to 50 values of resistance.
2. Compute the corresponding values of the current, given a set of conditions consisting of the average voltage, the initial charge, and 10 values of the time elapsed since the circuit was closed.
3. Call in another subroutine which will transmit the computed values and the various data as output in some desired form.
4. Return control to the calling program.

A subroutine to carry out this procedure could be written as follows:

C ← FOR COMMENT / STATEMENT NUMBER	CONTINUATION	FORTRAN STATEMENT
1		SUBROUTINE LCRDMP (VOLTAV, BEGINQ, TIME, HENRYS, FARADS,
	X	AMPS, N, M)
		DIMENSION HENRYS (50), OHMS (50), FARADS (50, 50),
	X	AMPS (50, 50, 10), TIME (10)
		DO 10 K = 1, 10
		DO 10 J = 1, N
		DO 10 I = 1, M
		FARADS (I, J) = 4.* HENRYS (I) / OHMS (J) ** 2
10		AMPS (I, J, K) = (VOLTAV - BEGINQ / FARADS (I, J)) / HENRYS (I) *
	X	TIME (K) * 2.71828 ** (-OHMS (J)* TIME (K) / 2.* HENRYS (I))
		CALL OUTPUT (VOLTAV, BEGINQ, TIME,HENRYS, FARADS, OHMS, AMPS)
		RETURN
		END (2, 2, 2, 2, 2)

Partitioning of Problems

FORTRAN II is particularly adaptable to the partitioning of a problem into convenient segments. This is illustrated by example 2, in which the subroutine LCRDMP calls for the subroutine OUTPUT, to be written separately. Moreover, the subroutine LCRDMP, rather than being of general utility, might well be a segment of a larger problem. The chief advantage of partitioning is that various parts of the total problem may be coded, compiled, tested, and debugged at different times. In the event that changes are required in one portion of a problem, proper initial partitioning can result in a considerable saving of programming time and machine time.

Joining Pre-Written Programs

The new subprogram facilities of FORTRAN II make it possible to convert pre-written programs to subprograms and use them jointly. A main program can be written for the particular problem at hand; this main program may be essentially a call program, directing the flow of control among various subprograms.

A FORTRAN program can readily be converted to a FORTRAN II subprogram, of either the FUNCTION or SUBROUTINE type, whichever is appropriate. For example, the program described in the Programmer's Primer for FORTRAN, Form No. F28-6019, pp. 53-58, might well be useful as a subprogram in the solution of a larger problem.

This program reads in a set of values x_i, y_i, where $i = 1, \ldots,$ n, and $n \leq 100$. It then calculates and prints out the $m + 1$ coefficients a_0, a_1, $\ldots$, a_m obtained by fitting the n points (x_i, y_i) by the least-squares method to the m-degree polynomial

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m$$

where $m \leq 10$.

Assume that this program is to be converted to a FORTRAN II subroutine with the symbolic name POLFIT, and that input and output is to be handled by the main program having a CALL for POLFIT. The program can then be converted to a subroutine by deleting the READ, PRINT, and FORMAT statements and placing the following statement at the head of the program:

SUBROUTINE POLFIT (X, Y, M, N, A)

and the following statement at the end of the program, replacing the STOP statement:

RETURN

An END statement may, if desired, be written after the RETURN statement. Where m = 10 and n = 100, the main program might have the following CALL statement:

CALL POLFIT (XSET, YSET, 10, 100, COEFNT)

After return of control to the main program from the subroutine POLFIT, called by the above statement, the array of computed coefficients would be referred to subsequently in the main program by the name COEFNT. For example, the main program could print the array by means of the following statements:

C ← FOR COMMENT / STATEMENT NUMBER 1–5	CONTINUATION 6	FORTRAN STATEMENT 7–72
		PRINT 50, (COEFNT (I), I = 1, 11)
50		FORMAT (5E15.6)

Note that POLFIT cannot be converted to a subprogram of the function type, since the result of the computation is not a single value.

CHAPTER 3 - SUBPROGRAMS CODED IN 704 SYMBOLIC LANGUAGE

Use of Non-FORTRAN Subprograms

Programs assembled by a system other than FORTRAN or FORTRAN II can also be linked to FORTRAN II programs by means of the new subprogram facilities. This chapter is concerned with object subprograms whose source language was the symbolic code acceptable to the SHARE Assembly Program (SAP).

SAP subprograms can be used in the same way as FORTRAN II subprograms, of either the FUNCTION or SUBROUTINE type, provided the SAP coding answers properly to the calling sequence produced by the FORTRAN II Translator, returns results in the manner of the appropriate type of FORTRAN II subprogram, and preserves the index register settings. Thus, SAP subprograms coded in conformity to the SUBROUTINE type can be linked to a FORTRAN II program by a CALL statement, and SAP subprograms coded in conformity to the FUNCTION type can be linked to a FORTRAN II program by an arithmetic formula.

Calling Sequence

A calling program produced by FORTRAN II always has a calling sequence equivalent to the following SAP instructions:

H	Location	Op	Address, Tag, Decrement	Comments
		TSX	NAME, 4	
		TSX	LOCX1	
		TSX	LOCX2	
		.		
		.		
		.		
		TSX	LOCXn	

There are n + 1 words in the calling sequence. The first causes transfer of control to the subprogram. The remaining n words include one word for each argument. In the case of an array, there is one word for the entire array, containing in the address field the location of the first member of the array, i.e., the member whose subscripts are all 1's.

SAP Transfer List and Prologue

Immediately preceding the main part of the sequence of machine language instructions, a translated FORTRAN II subprogram, of either the function or the subroutine type, has a Transfer List if it refers to one or more lower level subprograms, and always has a prologue. The coding below gives the general form of a Transfer

List and prologue produced by FORTRAN II. If the subprogram refers to N lower level subprogram names, there are N names in the Transfer List. If the subprogram does not refer to lower level subprograms, there is no Transfer List. Similarly, there is a pair of CLA and STA instructions for each argument. If there are no arguments (permissible only for the subroutine type of sub-program), there are no CLA and STA instructions in the prologue.

LOCATION	OP	ADDRESS, TAG, DECREMENT	COMMENTS
SUBP1	BCD	1SUBP1	
SUBP2	BCD	1SUBP2	
	.		
	.		Transfer List
	.		
SUBPN	BCD	1SUBPN	
	HTR		Storage for contents of index register 4
	HTR		Storage for contents of index register 2
	HTR		Storage for contents of index register 1
NAME	SXD	NAME - 3, 4	Save IR4 contents in location (NAME-3)
	SXD	NAME - 2, 2	Save IR2 contents in location (NAME-2)
	SXD	NAME - 1, 1	Save IR1 contents in location (NAME-1)
	CLA	1, 4	
	STA	X1	Location of 1st argument $\rightarrow X1_{21-35}$
	CLA	2, 4	
	STA	X2	Location of 2nd argument $\rightarrow X2_{21-35}$
	.		
	.		
	.		
	CLA	n, 4	
	STA	Xn	Location of n th argument $\rightarrow Xn_{21-35}$

A SAP subprogram to be used with FORTRAN II programs must obtain its arguments and preserve the index register contents in a similar but not necessarily identical way. A SAP subprogram must conform to the above coding in the following respects.

1. If the subprogram modifies the contents of any of the three index registers, it must contain instructions which will save and restore the original contents, so that the index register settings after execution of the subprogram are the same as they were upon entry to the subprogram. In a FORTRAN II subprogram, the restoration of the index register contents is effected by the equivalent of the following SAP instructions,

immediately preceding the transfer of control back to the main program:

H	Location	Op	Address, Tag, Decrement	Comments
		LXD	NAME-3, 4	Restore contents of IR4 from (NAME-3)
		LXD	NAME-2, 2	Restore contents of IR2 from (NAME-2)
		LXD	NAME-1, 1	Restore contents of IR1 from (NAME-1)

2. The subprogram must obtain the locations of its n arguments from the address fields of locations (1, 4), (2, 4), . . . , (n, 4), where index register 4 is as set by the main program upon transfer to the subprogram. In the case of an array, the subprogram must obtain the location of the first member only in this way, i.e., the member whose subscripts are all 1's. An array is considered one argument.
3. The subprogram must begin with a Transfer List if it calls for other subprograms, assembled separately from it, by instructions of the form:

H	Location	Op	Address, Tag, Decrement	Comments
		TSX	SUBPN, 4	

Results

A subprogram of the function type must place its single result in positions S, 1, 2, . . . , 35 of the Accumulator prior to return of control to the main program.

A subprogram of the SUBROUTINE type must place each of its results, if any, in positions S, 1, 2, . . . , 35 of a storage location. A result represented by the nth argument in a CALL statement must be stored in the location obtained from the address field of (n, 4), where index register 4 is as set by the main program upon transfer to the subprogram.

A FUNCTION subprogram always produces a single result. A subroutine may produce as many results as are specified as arguments in the CALL statement. Subroutines usually have arguments for results, but it may be desirable to write a subroutine which operates on data without returning results - for example, an output subroutine.

Return

A subprogram which has n arguments must return control to location (n + 1, 4), where index register 4 is as set by the main program upon transfer to the subprogram. The final machine instruction in a translated FORTRAN II subprogram is always equivalent to the following symbolic instruction:

H	Location		Op		Address, Tag, Decrement	Comments
1	2 6	7	8 10	11	12	72
			TRA		n + 1, 4	Return

This instruction is immediately preceded by the three LXD instructions which restore the contents of the index registers.

Entry Points

Unlike a FORTRAN II subprogram, a SAP subprogram may have more than one entry point. A SAP subprogram used with a FORTRAN II program may be entered at any desired point, provided a subprogram name acceptable to FORTRAN II is assigned to the selected entry point, and provided all the foregoing conditions are fulfilled when the subprogram is so entered. The entry point name or names by which a FORTRAN II main program refers to a SAP subprogram need not have been used in the original symbolic coding. Control information for the interpretation of the entry point name or names must always be furnished on a Program Card, as described in the following section.

Program Card

The FORTRAN II Translator automatically produces a Program Card for a FORTRAN II main program or subprogram as part of the machine language deck. For SAP subprograms, however, the Program Card must be furnished by the coder and placed at the head of the machine language deck produced by SAP. A complete description of the format of a Program Card is given in Part III. The following description specifies the information which must be punched on a Program Card for a SAP subprogram not using data in common storage:

Rows	Columns	
9	1	Must be punched.
	2, 3	Not significant.
	4-18	Number of words on this card, not counting row 9.

Rows	Columns	
9	19-21	Not significant.
	22-36	Must be blank.
	37-72	Add-and-carry-logical checksum of all words on this card, not counting row 9, cols. 37-72.
8	1-3	Not significant.
	4-18	Number of words in Transfer List.
	19-21	Not significant.
	22-36	Total number of words in subprogram, including data used by the subprogram other than data in common storage. This is the same number as the program break location, the location following the highest non-common location, relative to 0, in the program about to be loaded.
	37-57	Not significant.
	58-72	Zero for subprogram not using common storage.
7	1-36	BCD representation of the name assigned to the first entry point for purpose of reference in FORTRAN II programs. If the name has fewer than 6 characters, each unused 6-digit group at the right must be filled in with the BCD character 110000.
	37-57	Not significant.
	58-72	Location (relative to 0 within subprogram deck) of the first entry point.

Rows	Columns	
6	1-36	BCD representation of the name assigned to the second entry point, if any, (same requirements as for row 7, cols. 1-36).
	37-57	Not significant.
	58-72	Location (relative to 0 within subprogram deck) of the second entry point.
5	1-36	BCD representation of the name assigned to the third entry point, if any, (same requirements as for row 7, cols. 1-36).
.	.	.
.	.	.
.	.	.
	etc.	

Row 7, columns 1-36 must never be blank for a subprogram. If there is only one entry point, however, rows 6, 5, . . . 0, 11, and 12 must be entirely blank.

Program Break

The key location for relocation purposes is the program break. This location is the same number as the total number of words in a program, including data used by the program other than data in common storage. The program break, specified in row 8, columns 22-36, of the Program Card for a routine, is the lower limit of upper memory, relative to 0, for that particular routine.

Location references in a routine are relocated by the Binary Symbolic Subroutine Loader either as lower memory or upper memory (common storage) locations. By relocation as a lower memory location is meant incrementation by the current Loader increment, which is initially 24 and is augmented after each routine has been loaded. By relocation as an upper memory location is meant decrementation by the current Loader decrement; unless specifically set by means of a special card, the decrement of the Loader is zero. Details about the Loader increment and decrement and special cards are given in Part III.

The program break is one of the two factors which determine how a particular location reference is to be handled by the Loader. However, location references which are numerically less than the program break are not necessarily relocated as lower memory locations; nor are location references which are equal to or greater than the program break necessarily relocated as upper memory locations.

Relocation of Location References

The Loader determines whether a location reference in an address or decrement field is to be relocated as an upper or lower memory location by comparing the numerical value of the field with the program break and examining the associated relocation digits in row 8 of the relocatable binary instruction card. If the relocation digits are 10 and the corresponding field is an upper memory location relative to the program break, the field will be relocated as an upper memory location. If the relocation digits are 11 and the corresponding field is an upper memory location relative to the program break, the field will be relocated as a lower memory location. Likewise, relocation digits 10 cause a field which is a lower memory location relative to the program break to be relocated as a lower memory location and relocation digits 11 cause a field which is a lower memory location relative to the program break to be relocated as an upper memory location.

Since SAP Assembly produces the relocation digits 10 for all noncomplemented relocatable fields, some of the relocation digits may have to be changed from 10 to 11 before a SAP deck can be loaded correctly by the BSS Loader.

The relocation digits 11 cause the relocation of the address on the opposite side of the program break from where it actually is. For example, assume B is 1 less than the program break, and it is intended that B be located below the program break. Then when the instruction:

H (1)	Location (2–6)	Op (8–10)	Address, Tag Decrement (12–)	Comments (–72)
		CLA	B +10, 2	

is used in the symbolic coding of a subprogram for FORTRAN II, it must be given the relocation digits 11.

SAP assembly produces relocation digits 11 for relocatable complemented fields, that is, relocatable fields which have a negative value as written in the symbolic coding. Since relocation digits 11 are used for a special purpose by the BSS Loader, as described above, relocatable complement symbolics cannot ordinarily be used in SAP subprograms to be used with FORTRAN II.

Example: FORTRAN II Main Program and SAP Subprogram

Assume that a non-FORTRAN program has produced two sets of positive, double-precision, floating-point numbers, each set constituting one binary record on tape 6 and consisting of 500 two-word numbers. The first set is the A(I) list, each A having a high-order part AH, which is one word, and a low-order part AL, which is one word. The second set is the B(I) list, each B having a high-order part BH, which is one word, and a low-order part BL, which is one word. The FORTRAN II main program has three branches, depending on whether A(I) < B(I), A(I) = B(I), or A(I) > B(I). Since the low-order parts of the A and B values are not in normalized floating-point form (i.e., the fractional parts are not necessarily $\geq \left|\frac{1}{2}\right|$), the comparison of A(I) and B(I) cannot be expressed by FORTRAN statements. However, the new features of FORTRAN II make it possible to call in a SAP subprogram to perform the comparison. The main program might appear as follows:

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
		DIMENSION AH(500), AL(500), BH(500), BL(500)
		REWIND 6
		READ TAPE 6, (AH(I), AL(I), I=1, 500)
		READ TAPE 6, (BH(I), BL(I), I=1, 500)
		DO 20 I=1, 500
4		CALL DPCOMP (AH(I), AL(I), BH(I), BL(I), K)
		IF (K) 5, 10, 15
5		CALL ASMALL
		GO TO 20
10		CALL EQUAL
		GO TO 20
15		CALL ABIG
20		CONTINUE
		REWIND 6
		STOP

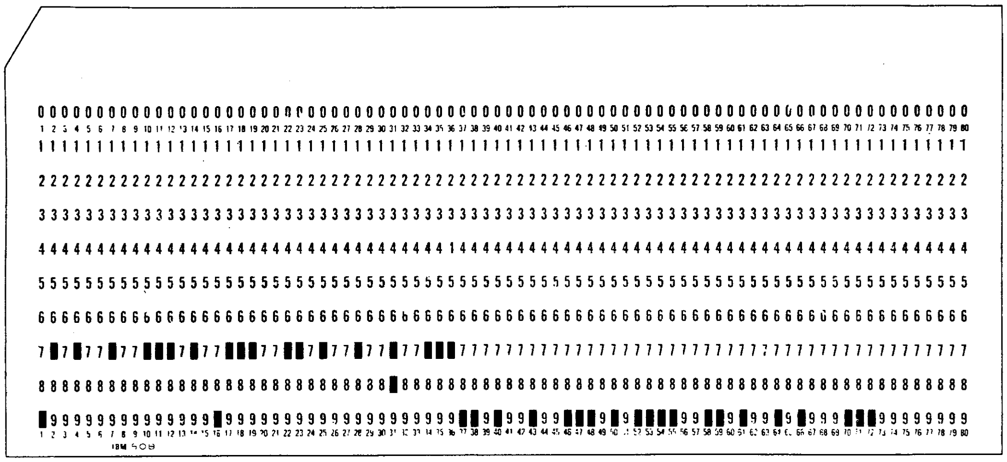
The REWIND statement rewinds the binary tape mounted on tape unit 6. The READ TAPE statements each read 1000 binary words from the tape mounted on tape unit 6, assigning them to AH(1), AL(1), AH(2), AL(2), . . . , AH(500), AL(500), BH(1), BL(1), BH(2), BL(2), . . . , BH(500), BL(500), in that order. After each of the 500 specified comparisons has been made, the main program branches to one of three subprograms with symbolic names ASMALL, EQUAL, and ABIG. These subprograms can be written and converted to machine language separately from the main program. The procedures to be followed in these subprograms are not of interest here, as the point of this example is to illustrate the use of a SAP subprogram with a FORTRAN II program. DPCOMP may have been pre-coded as a subroutine-type subprogram as follows; for simplification, it is assumed that all numbers are positive and that no values result below the lower limit of approximately 10^{-38}.

Double-precision, floating-point comparison: This subroutine places in the location obtained from the address field of (5,4): 0, if A = B; -1, if A < B; +1, if A > B.

Location	Op	Address, Tag, Decrement	Comments
ENTRY	CLA	1, 4	.
	STA	AH	P
	CLA	2, 4	R
	STA	AL	O
	CLA	3, 4	L
	STA	BH	O
	CLA	4, 4	G
	STA	BL	U
	CLA	5, 4	E
	STA	INDIC	.
AH	CLA		High-order part of A ⟶ AC.
BH	FSB		FSB high-order part of B.
	TZE	AL	
NOTZRO	TMI	SMLLER	If control reaches here, A ≠ B.
BIGGER	CLA	PLUS1	If control reaches here, A > B.
	TRA	INDIC	
PLUS1	OCT	+ 1	
MINUS1	OCT	- 1	
ZERO	OCT	0	
SMLLER	CLA	MINUS1	If control reaches here, A < B.
	TRA	INDIC	
AL	CLA		Low-order part of A ⟶ AC.
BL	UFS		UFS low-order part of B.
	ALS	10	
	TZE	TESTMQ	Shift out characteristic of high-order difference
	TRA	NOTZRO	
TESTMQ	LLS	45	Shift out characteristic. $C(MQ)_{9-35}$ and sign ⟶ AC.
	TZE	EQUAL	
	TRA	NOTZRO	
EQUAL	CLA	ZERO	If control reaches here, A = B.
INDIC	STO		
EXIT	TRA	6, 4	Return

Program Card for the Example

After the subroutine in the preceding example has been assembled by SAP, the only extra step required to use this subroutine with a FORTRAN II main program is the preparation of a Program Card. Since the entry point designated ENTRY in the symbolic coding is referred to by the name DPCOMP in the FORTRAN II main program, row 7, columns 1-36 of the Program Card must contain the BCD representation of DPCOMP, and row 7, columns 58-72 must contain the location, relative to 0, corresponding to ENTRY in the symbolic coding. ENTRY appears in the location field of the first instruction in the subroutine and, therefore, corresponds to location 0. It can readily be verified that the following Program Card meets the requirements previously stated in this chapter:



NOTE: The BCD code used for the symbolic entry point names is that given for 704 storage on page 35 of the 704 Electronic Data Processing Machine Manual of Operation, Form No. 24-6661.

Alternate Form of the Example

Since the SAP subroutine in the example computes a single-valued function of the arguments, AH(I), AL(I), BH(I), and BL(I), it can readily be converted, if desired, from the subroutine type to the function type. In the latter case, statement 4 in the FORTRAN II main program could be written:

K = DPCOMP(AH(I), AL(I), BH(I), BL(I))

The subprogram itself would require slight modification. Since the subprogram, as a FUNCTION type, would have only four arguments, the fifth pair of CLA and STA statements would have to be deleted. For the same reason, the instruction in location EXIT would have to be changed to

H	Location	Op	Address, Tag, Decrement	Comments
1	2 6	7 8 10	11 12	72
	EXIT	TRA	5, 4	

Since the result is to be left in the Accumulator, the instruction in location INDIC would have to be deleted. For the same reason, the transfers to INDIC would have to be replaced by transfers to EXIT. Thus altered, the subprogram would define a function of four arguments.

The only differences between the Program Cards for the function DPCOMP and for the subroutine DPCOMP would be in the word-count and, consequently, in the checksum.

PART III: BINARY SYMBOLIC SUBROUTINE LOADER

PART III - BINARY SYMBOLIC SUBROUTINE LOADER

Introduction

The Binary Symbolic Subroutine (BSS) Loader is punched out by the FORTRAN II Translator as the first nine cards of each main object program. The BSS Loader is not punched out with subprograms.

The FORTRAN II Translator produces decks in relocatable binary form. In a relocatable binary deck, instructions are assigned to consecutive storage locations starting at 0, and all location references are relative to 0. When a relocatable binary deck is loaded, location references are altered according to the actual locations occupied by the program in storage.

All routines produced by FORTRAN II, both main programs and subprograms, are loadable by the BSS Loader. The Loader enables programs in relocatable binary form to retain symbolic references to subprograms. As a result of this feature, a main program and each of its subprograms can be independently compiled. It is thus possible to compile a main program for which some or all of the subprograms have not yet been written. After a main program and its attendant subprograms have been compiled, either jointly or independently, the resulting relocatable binary decks can be loaded together and executed. At execution time, the relocatable binary decks of the main program and its subprograms - all starting at relocatable 0 - are stacked in the card reader in any order, headed by the nine Loader cards; loaded and relocated by the BSS Loader, using control information supplied with the compiled routines; and finally run.

Control information for the relocation process is provided to the Loader by Program Cards, one at the front of the main program deck and one at the front of each subprogram deck. A Program Card is the tenth card punched out by the FORTRAN II Translator for a main program and the first card for each subprogram. A Program Card specifies the number of locations to be occupied by the routine; this number is used as an increment for relocating an immediately subsequent routine. The increments specified by successive Program Cards are cumulative. Program Cards contain other information required by the Loader to interpret symbolic cross-references between the main program and its subprograms and between levels of subprograms.

In addition to relocatable binary decks produced by FORTRAN II, the Loader can also load binary cards, both absolute and relocatable, produced by a system other than FORTRAN II.

Transfer Card

The last card of the last deck to be loaded must be a Transfer Card. Row 9, column 1, must be punched on the Transfer Card, and row 9, columns 2-36, must be blank. The rest of the card is ignored. This card will signal the BSS Loader that the main program and all subprograms have been loaded. The FORTRAN II Translator produces a Transfer Card as the last card of a main program deck.

Transfer List

If a program refers to subprograms, the first instruction of the program in the compiled deck is preceded by a Transfer List. The Transfer List consists of the names of all the subprograms referred to in the program; in the case of SAP subprograms with more than one entry point, a name is listed for each entry point to which reference is made. Note that subprograms as well as main programs may have a Transfer List, since a subprogram may call for lower level subprograms. When a program has a Transfer List, the first name in the list occupies relocatable location 0 in the compiled deck, and each name in the list is counted as one word of the program.

Execution of the BSS Loader

Initially, the BSS Loader loads itself into the computer storage, the first card into the first 24 memory locations, and the other eight cards into the last 192 locations. The first Loader pass is then executed. In the first pass, absolute locations are assigned to the instructions, data, and Transfer List names of all subprograms being loaded. A symbol table is set up in which each subprogram name is associated with the absolute location of the entry point designated by the name.

In the second pass, each Transfer List name is replaced by an instruction which transfers control to the entry point designated by the name. The symbol table provides the necessary information for this step. Execution of the main program then commences.

Control Cards for Library Tape Routines

In order that FORTRAN II compilation may produce the proper Program Cards for library tape routines, the control cards formerly required must be changed as follows, thereby converting them to the FORTRAN II Program Card format:

1. Punch a "9" in column 1.
2. If the name of the routine has fewer than six characters, complete the name entirely with binary-coded decimal blanks (octal code: 60). For example, SIN must be punched SINbbb (octal code: 623145606060), instead of SINb00 as formerly.
3. Recompute the checksum and punch it in the right half of row 9 (columns 37-72).

Card Formats Acceptable to the BSS Loader

The following types of cards are loadable by the BSS Loader:

1. Program Cards
2. Transfer Cards (indicating end of first Loader pass)
3. Common Reassignment Cards
4. Control Cards
5. Absolute Transfer Cards
6. Absolute Binary Instruction and Data Cards
7. Relocatable Binary Instruction and Data Cards.

Blank cards will be ignored. The following is a description of the required formats for the seven types of cards listed above.

Program Card

Rows	Columns	
9	1	Must be punched.
	2-3	These columns will be ignored and a checksum will be done whether or not there is a punch in column 3, which means ignore checksum on other cards. The Loader cannot be made to ignore a non-zero checksum, known to be incorrect, on the Program Card; if the sum is made blank, however, the blank sum field will be ignored.
	4-18	Count of words on this card, not including the 9 row.
	19-21	Ignored.
	22-36	Must be blank.
	37-72	Checksum (add-and-carry-logical) of all words on this card except 9R.
8	1-3	Ignored.
	4-18	Contains the number of words which are the Transfer List for this program. This list must be the next thing to be loaded; it is followed by instructions in the usual relocatable format. This field will be zero when the program being loaded does not require subprograms for its execution and, therefore, has no Transfer List.
	19-21	Ignored.
	22-36	Contains a number showing the length of lower memory. This is the program break. It is the same as the address-plus-one, relative to zero, of the last word of the program, excluding data assigned to common storage. All location references in the address

Rows	Columns	
8	22-36	or decrement fields of instructions being placed in memory are relocated as either lower memory or upper memory locations, depending on the range in which they fall with respect to the program break and on the associated relocation digits (see page 42). References to be relocated as lower memory locations are relocated by the current increment of the Loader. The Loader increment is initially 30_8 (i.e., 24_{10}), since the first card of the Loader itself occupies locations 0 through 27_8. The current increment is augmented by the number in this Program Card field when a subsequent Control Card or Program Card is read. References to be relocated as upper memory locations are relocated according to the information provided to the Loader by the last preceding Control Card or Common Reassignment Card.
	37-57	Ignored.
	58-72	Contains the address of the last piece of data to be assigned downward in upper memory (common storage). This field must be blank (zero) if no common storage data is assigned. In the case of SAP subprograms assigning common data downward from 77777_8, this address is the same number as the 2's complement of the length of common storage. In the case of FORTRAN II programs, which always assign all common data downward from 77462_8, this address is 315_8 (205_{10}) less than the 2's complement of the length of common storage. Unless otherwise instructed, the Loader will cause the common data of successive routines to be overlapped. If overlapping is not desirable, the Loader's current decrement should be reset by a Common Reassignment Card in front of the Program Card for the routine whose data is to be moved down. The Common

Rows	Columns	
8	58-72	Reassignment Card will set a new decrement into the Loader to cause relocation of upper storage on cards following the Program Card. The decrement will be retained until replaced by a new decrement given on a subsequent Control Card or Common Reassignment Card.
7	1-36	If the program is a subprogram, this field contains the BCD representation of the name assigned to the first entry point (or to the subprogram if there is only one entry point). If the name contains fewer than 6 characters, each unused 6-digit group at the right must be filled in with the BCD character 110000. If the program is a main program, this field must be blank, as a main program is considered to have a blank name.
	37-57	Ignored.
	58-72	Address, relative to zero, associated with the name in columns 1-36.
6, 5, . . . , 0, 11, 12	1-36	If the program is a SAP subprogram with more than one entry point, the names assigned to the second, third, etc. entry points are listed in these fields in order, i.e., the second in row 6, the third in row 5, etc. When all names have been listed, the remaining rows are left blank. The names are represented as described for row 7.
	37-57	Ignored.
	58-72	The address, relative to zero, associated with the name in columns 1-36.

Transfer Card

Rows	Columns	
9	1	Must be punched.

Rows	Columns	
9	2-36	Must be blank.
	37-72	Ignored
8, 7, . . . , 0, 11, 12	The rest of this card is ignored.	

Common Reassignment Card

Rows	Columns	
9	1	Must be blank.
	2	Must be punched.
	3	Ignored.
	4-12	Must be blank.
	13	Must be punched.
	14-18	Must be blank.
	19-21	Ignored.
	22-36	Must be blank.
	37-57	Ignored.
	58-72	Contains the 2's complement of the number the Loader is to use in relocating common data downward in memory. This number becomes the current decrement of the Loader and is reset each time a Common Reassignment Card or Control Card is read.
8, 7, . . . , 0, 11, 12	The rest of this card is ignored.	

Control Card

Rows	Columns	
9	1	Must be blank.

Rows	Columns	
9	2	Must be punched.
	3	Ignored.
	4-12	Must be blank.
	13	Must be punched.
	14-18	Must be blank.
	19-21	Ignored.
	22-36	Contains the number of locations to be added to the current increment of the Loader, yielding a new increment. The new increment is effective for the relocation of lower memory locations in the next routine loaded.
	37-57	Ignored.
	58-72	Contains the 2's complement of the number the Loader is to use in relocating common data downward in memory. This number becomes the current decrement of the Loader. The current decrement is reset each time a Control Card or Common Reassignment Card is read and is not related to the last previous decrement of the Loader.
8, 7, . . . , 0, 11, 12	The rest of this card is ignored.	

NOTE: When a Control Card is read, the increment of the Loader is increased by the number given in row 9, columns 22-36, of the last preceding Program Card, and further increased by the number in row 9, columns 22-36, of the Control Card. The decrement of the Loader, however, is reset to a value not related to the previous decrement. If row 9, columns 58-72, of the Control Card is blank, the decrement will be reset to 0.

Absolute Transfer Card

Rows	Columns	
9	1-21	Must be blank, except that a punch in column 3 will be ignored.

Rows	Columns	
9	22-36	Absolute location to which BSS Loader now transfers control.
	37-72	Ignored.
8, 7, . . . , 0, 11, 12		The rest of this card is ignored.

Absolute Binary Instruction or Data Card

Rows	Columns	
9	1-2	Must be blank.
	3	If punched, checksum will be ignored.
	4-13	Ignored.
	14-18	Count of words on this card, not counting row 9.
	19-21	Ignored.
	22-36	Address into which first word (i.e., 8L) is to be loaded.
	37-72	Checksum (add-and-carry-logical) of all words on this card except 9R.
8, 7, . . . , 0, 11, 12		Instructions or data to be loaded.

Relocatable Binary Instruction or Data Card

Rows	Columns	
9	1	Must be blank.
	2	Must be punched.
	3	If punched, checksum will be ignored.
	4-18	Count of words on this card, not including rows 8 and 9.

Rows	Columns	
9	19-21	Ignored.
	22-36	Address, relative to zero, into which the first word (i.e., 7L) is to be loaded.
	37-72	Checksum (add-and-carry-logical) of all words on this card except 9R.
8	1-72	Both words are read together and contain information about the relocation of location reference in the address field (columns 22-36 or 58-72) or decrement field (columns 4-18 or 40-54) of the instructions in rows 7-12 of this card.
7, 6, . . . , 0, 11, 12	Instructions or data to be loaded.	

The digits in row 8 of this card are interpreted one at a time and related to the decrement field of 7L, the address field of 7L, the decrement field of 7R, the address field of 7R, the decrement field of 6L, etc. The digits have the following significances:

0 = Ignore this field.

10 = If the number in this field is equal to or greater than the program break, relocate as an upper memory location. If less than the program break, relocate as a lower memory location.

11 = If the number in this field is equal to or greater than the program break, relocate as a lower memory location. If less than the program break, relocate as an upper memory location.

Instructions for which there is no room in the 8 row for the necessary relocation digits must be put on another card.

Additional Note on the Program Card

If there are more than ten entry point names; one or more additional Program Cards are required, containing the eleventh, twelfth, etc. names. Supplementary Program Cards must have row 9 punched as specified in the description of the Program Card, and the names must start in row 8.

APPENDIX A - SUMMARY OF FORTRAN II STATEMENTS

The following is a summary of the 38 types of FORTRAN II statements, grouped in the four classifications: arithmetic statements, control statements, input/output statements, and specification statements.

1. Arithmetic statements (arithmetic formulas and function definitions):

   a = b

2-19. Control statements:

GO TO n

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

ASSIGN i TO n

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

IF (a) n_1, n_2, n_3

SENSE LIGHT i

IF (SENSE LIGHT i) n_1, n_2

IF (SENSE SWITCH i) n_1, n_2

IF ACCUMULATOR OVERFLOW n_1, n_2

IF QUOTIENT OVERFLOW n_1, n_2

IF DIVIDE CHECK n_1, n_2

PAUSE or PAUSE n

STOP or STOP n

DO n i = m_1, m_2 or DO n i = m_1, m_2, m_3

CONTINUE

CALL Name (Argument List)

RETURN

END $(i_1, i_2, i_3, i_4, i_5)$

20-32. Input/Output statements:

FORMAT (Specification)

READ n, List

READ INPUT TAPE i, n, List

PUNCH n, List

PRINT n, List

WRITE OUTPUT TAPE i, n, List

READ TAPE i, List

READ DRUM i, j, List

WRITE TAPE i, List

WRITE DRUM i, j, List

END FILE i

REWIND i

BACKSPACE i

33-38. Specification statements:

DIMENSION v, v, v, ...

EQUIVALENCE (a, b, c, ...), (d, e, f, ...), ...

FREQUENCY n(i, j, ...), m(k, l, ...), ...

SUBROUTINE Name (Argument List)

FUNCTION Name (Argument List)

COMMON a, b, c, ...

APPENDIX B

The chart summarizes the 20 built-in functions at present available as open subroutines on the FORTRAN II system tape.

Type of Function	Definition	No. of Args.	Name	Mode of Argument	Mode of Function
Absolute value	$\lvert Arg \rvert$	1	ABSF XABSF	Floating Fixed	Floating Fixed
Truncation	Sign of Arg times largest integer $\leq \lvert Arg \rvert$	1	INTF XINTF	Floating Floating	Floating Fixed
Remaindering (see note 1 below)	Arg_1 (mod Arg_2)	2	MODF XMODF	Floating Fixed	Floating Fixed
Choosing largest value	Max (Arg_1, Arg_2, ...)	≥ 2	MAX0F MAX1F XMAX0F XMAX1F	Fixed Floating Fixed Floating	Floating Floating Fixed Fixed
Choosing smallest value	Min (Arg_1, Arg_2, ...)	≥ 2	MIN0F MIN1F XMIN0F XMIN1F	Fixed Floating Fixed Floating	Floating Floating Fixed Fixed
Float	Float fixed number	1	FLOATF	Fixed	Floating
Fix	Same as XINTF	1	XFIXF	Floating	Fixed
Transfer of sign	Sign of Arg_2 times $\lvert Arg_1 \rvert$	2 2	SIGNF XSIGNF	Floating Fixed	Floating Fixed
Diminishing (see note 2 below)	Arg_1 (dim Arg_2)	2	DIMF XDIMF	Floating Fixed	Floating Fixed

NOTES: 1. The function MODF (Arg_1, Arg_2) is defined as $Arg_1 - [Arg_1/Arg_2]\, Arg_2$, where $[x]$ = integral part of x.

2. The function DIMF (Arg_1, Arg_2) is defined as $Arg_1 -$ Min (Arg_1, Arg_2).

Provision has been made for the addition of built-in functions by the individual installation. There may be up to ten of these, since the dictionary (see 2. a below) for the built-in functions provides room for only ten additional names. To do this, certain additions and changes must be made to the edit deck, FNEDT2. The two necessary changes (described in 1. b and 2. d below) require the use of binary correction cards. The general format of a binary correction card for the edit deck is as follows:

Rows	Columns	
9	1-3	Must be blank.
	4-18	Number of words to be loaded from this card.
	19-21	Ignored.
	22-36	Location of the first word to be loaded.
	37-72	Add-and-carry logical checksum of all words on this card, not counting this field.
8	1-36	First binary correction word, to be loaded in the location given in row 9, columns 22-36.
	37-72	Second binary correction word, if any, to be loaded in the location following the first binary correction word.
7		.
.		.
.		.
.		.
0		.
11		.
12		etc.

The changes and additions required to add built-in routines to the FORTRAN II system are as follows:

1. In State C of Section I in the edit deck, FNEDT2:

a. The name of each new routine must be added to the dictionary. The name must consist of 3, 4, 5, or 6 alphanumeric characters, omitting the terminal F used in the FORTRAN II source language. The first character must be alphabetic and must be X if and only if the function is to be fixed point. The name must be punched in BCD characters. If the name has fewer than 6 characters, each unused 6-digit group at the right must be filled in with the BCD character 110000 (blank).

b. The decrement which controls the search of the dictionary must be changed by a binary correction card. This decrement must be set to the 2's complement of the number of built-in routines in the FORTRAN II system. At present, the decrement is the 2's complement of 20. If one more routine is added, the decrement will have to be reset to the 2's complement of 21.

The locations at which the foregoing addition and change to Section I must be made cannot be stated at the present time. This information will be distributed in the near future.

2. In Section III in the edit deck, FNEDT2:

a. The BCD name (same requirements as 1. a above) of each new routine must be added to the M1D2 dictionary in record 55, locations 7122_8 through 7135_8. At present, the names DIM and XDIM occupy locations 7122_8 and 7123_8. The next entry must therefore be made at 7124_8.

b. Each new routine must be placed in available space in the range 7260_8 to 7777_8.

c. A transfer to the entry point of each new routine must be added to the branching routine in record 55, locations 7106_8 thorugh 7127_8. These transfer instructions, which must be added in binary machine code, have the operation field TRA (i. e., 000000010000). Since two routines (DIM and XDIM) have already been added to the original 18 routines, the next transfer instruction must be placed in location 7100_8. The transfer instructions must be placed in the same order as the corresponding names in the M1D2 dictionary.

d. The decrement of the TXH instruction in location 7077_8 must be changed by a binary correction card. This decrement must be set to the 2's complement of the number of routines added to the original 18. At present, this decrement is set to the 2's complement of 2. The entire binary word is now $(3\ 77776\ 4\ 07073)_8$.

IBM

International Business Machines Corporation
590 Madison Avenue, New York 22, N.Y.

Printed in U.S.A. C 28-6000-2