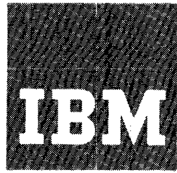
File No. 1410-25
Form J24-1468-1



Systems Reference Library

IBM 1410 FORTRAN

This publication contains the specifications for the FORTRAN programming system for the IBM 1410. The FORTRAN language enables the user to express problems in a symbolic source language similar to the language of mathematics.

Included in this publication are descriptions of the various types of arithmetic, control, input/output, subprogram, and specification statements that are translated by the 1410 FORTRAN Processor into machine language instructions.

© 1962 by International Business Machines Corporation

MINOR REVISION (May, 1963)

This publication, J24-1468-1, supersedes the bulletin, FORTRAN for the IBM 1410: Preliminary Specifications, Form J24-1468-0. It includes the information contained in Technical Newsletters N28-1051, N28-1062, and N28-1064.

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

CONTENTS

INTRODUCTION 5
Machine Requirements 5
Punching a Source Program 6
IBM 1410 FORTRAN Statements 6

ARITHMETIC OPERATIONS 7
Arithmetic Precision 7
Constants 7
Variables 7
Subscripted Variables 8
Expressions 8
Hierarchy of Operations 8

IBM 1410 FORTRAN STATEMENTS 10
Arithmetic Statements 10
Subprogram Statements 10
Functions 10
Library Functions 10
Arithmetic Statement Functions 10
Subprograms 12
FUNCTION Subprograms 12
SUBROUTINE Subprograms 13
CALL Statement 14
RETURN Statement 14
Control Statements 15
Unconditional GO TO 15
Computed GO TO 15
Assigned GO TO 15
ASSIGN Statement 15
IF Statement 15
SENSE LIGHT Statement 15
IF (SENSE LIGHT) Statement 16
IF (SENSE SWITCH) Statement 16
IF ACCUMULATOR OVERFLOW Statement 16
IF QUOTIENT OVERFLOW Statement 16
IF DIVIDE CHECK Statement 16
DO Statement 16
CONTINUE Statement 18
PAUSE Statement 18
STOP Statement 18
END Statement 18
Input/Output Statements 18
List of Quantities 18
Matrices 19
FORMAT Statement 19
FORMAT Specifications 19
READ Statement 21
READ INPUT TAPE Statement 22
PUNCH Statement 22
PRINT Statement 22
WRITE OUTPUT TAPE Statement 22
READ TAPE Statement 22
WRITE TAPE Statement 22
TYPE Statement 23
END FILE Statement 23
REWIND Statement 23
BACKSPACE Statement 23
1301 Disk Storage Statement 23
RECORD Statements 23
FETCH Statement 24
Status of (I) 24
FIND Statement 24
Specification Statements 24
DIMENSION Statement 24
EQUIVALENCE Statement 25
COMMON Statement 25
DEFINE FILE Statement 26

This publication contains the specifications for the FORTRAN programming system for the IBM 1410. FORTRAN is a programming system designed primarily for scientific applications. The system consists of the FORTRAN language, which is similiar to standard mathematic notation, and the FORTRAN processor, which converts FORTRAN language statements into machine language.

FORTRAN statements are written on the FORTRAN Coding Form (shown in Figure 1) and then punched into cards. These cards constitute the "source program," and serve as input to the FORTRAN processor. The processor compiles an "object program" from the FORTRAN statements in the source program.

Two versions of the FORTRAN processor are available. One version, the FORTRAN (20K) processor, requires a minimum of 20,000 positions of core storage for compilation. The other version, the FORTRAN (40K) processor, requires a minimum of 40,000 positions of core storage, but provides faster compilation and more complete diagnostic checking than the 20K version. Intermediate output from the FORTRAN (20K) processor is an Autocoder language program, which is converted into a machine-language object program by the Autocoder processor. The FORTRAN (40K) processor compiles directly into machine language, and does not require the Autocoder processor. The language specifications in this bulletin apply to both versions of the FORTRAN processor.

Machine Requirements

In addition to the core-storage sizes specified above for each version of the FORTRAN processor, execution of either version of the processor requires the following input/output units:

1 IBM 1402 Card Read Punch, Model 2
1 IBM 1403 Printer, Model 2
4 magnetic tape units

IBM
FORTRAN CODING FORM
Form X28-7327
Printed in U.S.A.
Program
Coded By
Checked By
Identification
73 80
Date
Page of
C FOR COMMENT
STATEMENT NUMBER
Cont.
FORTRAN STATEMENT
1 5 6 7 10 15 20 25 30 35 40 45 50 55 60 65 70 72

Figure 1. FORTRAN Coding Form

Punching a Source Program

Each statement of a FORTRAN source program is punched into a separate FORTRAN statement card (Figure 2) using character set H. Character set H consists of the regular 0-9 and A-Z punching, plus these special characters.

Card Code	Printed Symbol	Meaning
blank		blank
11	-	minus
12	+	plus
0-1	/	divide
11-4-8	*	multiply
12-4-8	)	right parenthesis
0-4-8	(	left parenthesis
0-3-8	,	comma
11-3-8	$	dollar sign
12-3-8	.	decimal point
3-8	=	equal sign
4-8	'	apostrophe (prime)

If a statement is too long for one card, it can be continued on as many as nine continuation cards.

The character C is punched in column 1 to identify a comment card. Such cards appear when the source program deck is listed, but do not become a part of the object program.

When coding a source program, it is sometimes necessary to refer one statement to another. This cross referencing is facilitated by assigning unique statement numbers. Any unsigned number from 00000 to 99999 can be used as a statement number. Because the order of the statements is governed solely by the order of cards in the source deck, statement numbers need not be in numerical sequence. Columns 1-5 are punched with the assigned statement number. If no statement number is needed, these columns can be left blank.

Column 6 of the initial card of a statement must be left blank or be punched with a zero. Continuation cards (other than comment cards) are punched 1 through 9 as required.

The statements are punched in columns 7-72. A statement may consist of not more than 660 characters (that is, 10 cards). The FORTRAN processor ignores blank characters (see "FORMAT" for exception); thus, they can be used freely to improve the readability of the source program listing.

Columns 73-80 are not processed by the compiler and can be punched with any desired identifying information.

IBM 1410 FORTRAN Statements

There are five classes of statements that can be used in a 1410 FORTRAN source program.

1. Arithmetic statements specify a numerical computation.
2. Subprogram statements define and specify subprograms.
3. Control statements govern the flow of the program.
4. Input/output statements.
5. Specification statements required for object program storage allocation.

C ← FOR COMMENT
STATEMENT NUMBER
CONTINUATION
FORTRAN STATEMENT
IDENTIFICATION
IBM 888157

Figure 2. FORTRAN Statement Card

Arithmetic statements define numerical calculations that the object program is to perform. An arithmetic statement in FORTRAN resembles a conventional arithmetic equation. The statement consists of a variable, followed by an equal sign, followed by an expression.

An arithmetic expression is a sequence of constants, variables, and functions, arranged according to a set of rules.

Arithmetic Precision

In most types of scientific computation, the precision of the quantities used is important. In the 1410 FORTRAN system, any desired degree of precision (f) can be assigned to floating-point values by means of a control card. Where no specification is made, floating-point precision is eight decimal digits. The maximum number of significant digits (k) for fixed-point values can also be specified by a control card. If no specification is made, k is five decimal digits.

NOTE. The specification for (f) cannot exceed 45; (k) cannot exceed 99.

Constants

Numbers are written in one of two forms, fixed point or floating point, indicating the mode of arithmetic to be used. Numbers expressed as integers are fixed point. Numbers with decimal points are floating point.

The general form and examples of fixed-point constants are shown in Figure 3.

GENERAL FORM	EXAMPLES
1 to k digits. A preceding + or − sign is optional. The magnitude or absolute value of the constant must be less than 10^k or be zero. When a fixed point constant is used for the value of a subscript, only five digits will be used. A fixed point constant that appears in an expression is limited to 45 digits.	3 +1 −28987

Figure 3. Fixed-Point Constants

The general form and examples of floating-point constants are shown in Figure 4.

Within storage, a floating-point constant consists of f + 2 digits. For example, if f is defined as 18, a number in the source program having 18 or more

GENERAL FORM	EXAMPLES
Any number of digits with a decimal point. A preceding + or − sign is optional. E followed by an integer (signed or unsigned) designates multiplication by a power of 10. Floating-point constants may contain any number of digits but only f significant digits are retained. The magnitude of a floating-point constant may lie between the limits 10^{-100} and $(1 - 10^{-f}) \times 10^{99}$, or may be zero.	17. 5.0 −.0003 5.0E3 i.e., 5.0×10^3 5.0E + 3 i.e., $5.0 \times 10^{+3}$ 5.0E − 3 i.e., 5.0×10^{-3}

Figure 4. Floating-Point Constants

significant digits results in a 20-digit floating-point number, 18 for the mantissa and 2 for the characteristic.

Variables

Variable quantities are represented in FORTRAN statements by symbolic names. The quantities that the names of the variables assume are either fixed point or floating point. All variables whose first letter is I, J, K, L, M, or N are fixed-point mode.

The general form and examples of fixed-point variables are shown in Figure 5. A fixed-point variable can assume any integral value provided the magnitude is less than 10^k. When the value assumed by a fixed-point variable has less than k digits, high-order zeros will be added. When the value exceeds k digits, it is treated modulo 10^k.

GENERAL FORM	EXAMPLES
1 to 6 alphabetic or numerical characters (not special characters) of which the first is I, J, K, L, M or N.	I M2 JOBNO

Figure 5. Fixed-Point Variables

The general form and examples of floating-point variables are shown in Figure 6. A floating-point variable can assume any value expressible as a normalized floating-point number. That is, the magnitude can lie between the limits 10^{-100} and $(1 - 10^{f}) \times 10^{99}$. A precision of f digits will be carried in the mantissa.

To avoid the possibility that a variable name may be considered by the compiler to be a function name,

GENERAL FORM	EXAMPLES
1 to 6 alphabetic or numerical characters (not special characters), of which the first is alphabetic but not I, J, K, L, M, or N.	A B7 DELTA

Figure 6. Floating-Point Variables

two rules should be observed with respect to naming fixed- or floating-point variables:

1. A variable should not be given a name that is identical to the name of a function without its terminal F. Thus, if a function is named TIMEF, no variable should be named TIME (see "FUNCTIONS").
2. Subscripted variables should not be given names ending with F, unless their names are less than four characters in length.

Subscripted Variables

A variable can represent any element of a one-, two-, or three-dimensional array by appending to it one, two, or three subscripts. Subscripts must be fixed-point expressions in one of the general forms shown in Figure 7. A variable in a subscript cannot itself have a subscript.

GENERAL FORM	EXAMPLES
Let v represent any fixed-point variable and C or (C') an unsigned fixed-point constant with five or fewer digits. Then a subscript is an expression in one of the forms: v c v + c or v − c c*v c*v + c' or c*v − c' (The symbol * denotes multiplication.)	I 3 MU + 2 MU − 2 5*J 5*J + 2 5*J − 2

Figure 7. Subscripts

The general form and examples of subscripted variables are shown in Figure 8.

Any subscripted variable must have the size of its array specified in a DIMENSION statement preceding the first appearance of the variable in the source program (see "DIMENSION"). A subscripted variable may not appear without a subscript. That is, A is not equivalent to A (1).

The value of a subscript must be greater than zero but not greater than the corresponding array dimension.

GENERAL FORM	EXAMPLES
A fixed- or floating-point variable, followed by parentheses enclosing 1, 2 or 3 subscripts which are separated by commas.	A(1) K(3) ALPHA (I, J + 2) BETA(5*J-2,K-2,L)

Figure 8. Subscripted Variables

One-dimensional arrays are stored sequentially. Two-dimensional arrays are stored sequentially by columns. Three-dimensional arrays are stored sequentially first by columns and then by planes. All arrays are stored in reverse order; that is, in order of decreasing storage locations.

Expressions

An expression is a meaningful sequence of constants, variables (subscripted or non-subscripted), and functions separated by algebraic operation symbols. In addition to normal algebraic rules, the following must be observed when writing FORTRAN expressions:

1. The mode of arithmetic in an expression can be either floating-point or fixed-point, but with certain exceptions the modes must not be mixed in the same expression.
   a. A floating-point quantity can appear in a fixed-point expression as an argument of a function; for example XFIXF (C).
   b. A fixed-point quantity can appear in a floating-point expression as a function argument, such as FLOATF (I); as a subscript, such as A(J, K); or as an exponent, such as A**N.
2. The five basic algebraic operations are specified by the symbols +, -, *, /, and **, which denote addition, subtraction, multiplication, division, and exponentiation, respectively. Two operation symbols cannot appear in sequence unless they are separated by parentheses. Thus, A* -B, and + -A are not valid expressions; A* (-B) and + (-A) are valid expressions.

Hierarchy of Operations

The use of parentheses in an algebraic expression clearly establishes the intended sequence of operations. The hierarchy of operations in an expression not specified by the use of parentheses is in the usual order:

Function computation and Substitution
Exponentiation
Multiplication and Division
Addition and Subtraction

For example, the expression

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

is taken to mean

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

Parentheses that have been omitted from a sequence of consecutive multiplications and divisions (or consecutive additions and subtractions) are understood to be grouped from the left. Thus, if ϕ represents either * or / (or either + or -), then

A ϕ B ϕ C ϕ D ϕ E

will be taken by FORTRAN to mean

((((A ϕ B) ϕ C) ϕ D) ϕ E)

The expression A^{B^C} which is sometimes considered meaningful, cannot be written as A**B**C. It should be written as (A**B) **C or A** (B**C), whichever is intended.

ARITHMETIC STATEMENTS

The arithmetic statement (Figure 9) defines a numerical calculation. A FORTRAN arithmetic statement closely resembles a conventional arithmetic equation. However, in a FORTRAN arithmetic statement the = sign means is to be replaced by, not is equivalent to. Thus, the arithmetic statement

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 in floating-point mode according to the mode of the variable to the left of the = sign (in this case, floating point).

GENERAL FORM	EXAMPLES
"a = b" where a is a variable (subscripted or not subscripted) and b is an expression.	Q1 = K A(I) = B(I) + SINF (C (I))

Figure 9. Arithmetic Statements

If the variable on the left is fixed-point and the expression on the right is floating-point, the result will first be computed in floating-point and then truncated to an 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 and then converted to floating-point.

Arithmetic statements can produce a number of useful effects. Here are some examples:

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 3B.

However, be careful to avoid invalid statements such as:

A = 3* B	Not accepted. The expression is mixed (contains both fixed-point and floating-point quantities).
A = I* B	Not accepted. The expression is mixed.
J = I** B	Not accepted. The expression is mixed.

SUBPROGRAM STATEMENTS

Functions

Two types of functions can be used in 1410 FORTRAN.

1. Predefined library functions (subroutines) provided by IBM, or coded in machine language by the user and added to the library.

2. Functions that are defined by the user with a FORTRAN arithmetic statement function.

In a FORTRAN source program, both types of functions are called by the appearance in an arithmetic expression of a function name and its arguments (enclosed in parentheses). For example, the arithmetic statement:

Y = A - SINF (B) + FIRSTF (C)

calls a library subroutine to evaluate the trigonometric sine of B, and an arithmetic statement function, which is defined previously in the program, to evaluate FIRSTF (C).

Library Functions

Figure 10 shows the library functions for which subroutines are provided by IBM. In some cases, two or more variations of the same function appear to provide for combinations of fixed- and floating-point mode for both the function and its arguments.

Evaluation of a library function produces, from one or more arguments, a single value that is made available to the expression in which it appears. Provision will be made for the user to incorporate other machine-language subroutines required for evaluating functions.

Figure 11 shows the general form and examples of naming both library functions and arithmetic statement functions.

Arithmetic Statement Functions

An arithmetic statement function is defined by the user with a single FORTRAN arithmetic statement written according to rules similar to those applying to any arithmetic statement (see "EXPRESSIONS" and "ARITHMETIC STATEMENTS"). When called in the source program, it produces a single value from one or more arguments. Figure 12 shows the general form and examples of this type of function.

All arithmetic statement function definitions must precede the first executable statement of the source program. Library functions can be used in these definitions, as can other arithmetic statement functions, provided they are defined in a previous statement.

NAME	DESCRIPTION	NUMBER OF ARGUMENTS	MODE OF	
			ARGUMENT(S)	FUNCTION
SINF	Trigonometric sine of argument	1	Floating	Floating
COSF	Trigonometric cosine of argument	1	Floating	Floating
LOGF	Natural logarithm of argument	1	Floating	Floating
EXPF	Argument power of ϵ	1	Floating	Floating
SQRTF	Square root of argument	1	Floating	Floating
ATANF	Arc tangent of argument	1	Floating	Floating
ABSF XABSF	Absolute value of argument	1 1	Floating Fixed	Floating Fixed
INTF XINTF	Truncation (sign of argument times largest integer $\leqq$ argument)	1 1	Floating Floating	Floating Fixed
MODF XMODF	Argument 1 modulus argument 2	2 2	Floating Fixed	Floating Fixed
MAX0F MAX1F XMAX0F XMAX1F	Maximum value of 2 or more arguments	$\geqq 2$ $\geqq 2$ $\geqq 2$ $\geqq 2$	Fixed Floating Fixed Floating	Floating Floating Fixed Fixed
MIN0F MIN1F XMIN0F XMIN1F	Minimum value of 2 or more arguments	$\geqq 2$ $\geqq 2$ $\geqq 2$ $\geqq 2$	Fixed Floating Fixed Floating	Floating Floating Fixed Fixed
FLOATF	Convert fixed-point argument to floating point	1	Fixed	Floating
XFIXF	Convert floating-point argument to fixed point	1	Floating	Fixed
SIGNF XSIGNF	Absolute value of argument 1 times sign of argument 2	2 2	Floating Fixed	Floating Fixed
DIMF XDIMF	Argument 1 minus the lesser of argument 1 and argument 2.	2 2	Floating Fixed	Floating Fixed

Figure 10. Library Functions

GENERAL FORM	EXAMPLES
The name of the function consists of 4 to 7 alphabetic or numerical characters (not special characters). The last character must be F and the first must be alphabetic. The first character 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.	SINF(A) XFIXF(B) MODF(A, B) FIRSTF(C, D) XTHIRDF(J, K)

Figure 11. Library and Arithmetic Statement Function Names

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. Any functions appearing in b must be available to the program or already defined by preceding 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

Figure 12. Arithmetic Statement Functions

Variables that appear in both sides of an arithmetic statement function definition are arguments of the function. These are dummy variables for which values of actual arguments are substituted when the function is called. Dummy variables can be assigned any acceptable variable names, including those used elsewhere in the program; but the names assigned must correspond in number, order, and mode to the actual variable names in a calling statement.

Variables that appear in the right side only of a function definition are parameters for which unique names must be assigned in accordance with the rules for naming variables. Thus, for example, the function:

FIRSTF (X) = A*X + B

can be called as FIRSTF (Y), at which time the function FIRSTF is evaluated by substituting the current value of the argument Y for the dummy variable X. At the same time, current values of the parameters A and B are used.

Actual arguments used in a calling statement can be subscripted or non-subscripted variables, and can also be FORTRAN expressions. Thus, in the previous example, FIRSTF (Z + Y (I)) is evaluated by computing the current value of Z + Y (I) and substituting this value for X in the function definition.

Subprograms

Two types of subprograms can be used in 1410 FORTRAN.

1. FUNCTION subprograms that are defined by the user with a series of FORTRAN statements and are called by the appearance in an arithmetic expression of a function name and its arguments.
2. SUBROUTINE subprograms that are also defined by the user with a series of FORTRAN statements, but are called by a CALL statement.

There is a further major distinction between the two types of subprograms. A FUNCTION subprogram returns a single value to the calling program by means of a FORTRAN arithmetic statement in which the left-hand side is the function name. A SUBROUTINE subprogram can return multiple values to the calling program by utilizing one argument of the subprogram for each value desired, each argument appearing in the left-hand side of an arithmetic statement.

FUNCTION Subprograms

A FUNCTION subprogram is defined by the identifying statement

FUNCTION NAME (Arguments)

followed by a series of FORTRAN statements that compute the value of NAME. Figure 13 shows the general form and examples of the naming of FUNCTION subprograms.

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 non-subscripted variable names, the name of a SUBROUTINE subprogram, the name of a FUNCTION subprogram, or the name of a FORTRAN function. The NAME consists of 1 to 6 alphamerical characters (not special 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. The final character must not be F if there are more than three characters in the NAME.	FUNCTION ARCSIN (RADIAN) FUNCTION ROOT (B, A, C) FUNCTION INTRST (RATE, YEARS)

Figure 13. FUNCTION Subprogram Names

An example of the general form of a FUNCTION subprogram is:

```
FUNCTION ACME (A, B, C)
.
.
.
ACME = arithmetic expression
.
.
RETURN
.
.
END
```

Other statements (represented by dots) are sequences of FORTRAN statements required to compute the value of ACME. The RETURN statement terminates the subprogram and returns control to the calling program (see "RETURN"). The END statement signifies the end of the subprogram during compiling (see END).

A FUNCTION subprogram is called by the appearance of the FUNCTION NAME and its arguments in the calling program. For example, the arithmetic statement

Q = ACME (A, B, C) + C - D

calls the subprogram ACME in the process of computing Q.

Additional rules for the use of FUNCTION subprograms are:

1. Each variable name used as an argument of a FUNCTION subprogram must appear in at least one executable statement of the subprogram.

2. The NAME of the function must appear at least once in the FUNCTION subprogram as the variable on the left-hand side of an arithmetic statement, or in an input-statement list.

3. A FUNCTION subprogram must not be inserted between two statements of another program.

4. The arguments in a FUNCTION subprogram are dummy variables for which actual arguments are substituted when the subprogram is called.

5. Dummy arguments must correspond in number, order, and mode to the actual arguments in the calling statement.

6. If a dummy argument is an array name, the corresponding actual argument must be an array name. Each array name must appear in a DIMENSION statement of the program or subprogram in which it appears and the dimensions must be the same (see "DIMENSION").

7. Dummy arguments must not appear in EQUIVALENCE or COMMON statements of a FUNCTION subprogram. When a COMMON statement is used to implicitly transmit arguments from a calling program to a FUNCTION subprogram, at least one argument must be transmitted explicitly. That is, at least one of the arguments must appear in parentheses following the subprogram NAME (see "COMMON").

SUBROUTINE Subprograms

A SUBROUTINE subprogram is defined by the identifying statement

SUBROUTINE NAME (Arguments)

followed by a series of FORTRAN statements that compute one or more values, each expressed as an argument of the subprogram. Figure 14 shows the general form and examples of the naming of SUBROUTINE subprograms.

GENERAL FORM	EXAMPLES
"SUBROUTINE NAME (a_1, a_2, . . . , a_n)" where NAME is the symbolic name of a subprogram, and each argument, if any, is a nonsubscripted variable name, the name of another SUBROUTINE subprogram, the name of a FUNCTION subprogram, or the name of a FORTRAN function. The NAME of the subprogram may consist of 1 to 6 alphamerical characters (not special characters). The first character must be alphabetic. The final character must not be F if there are more than three characters in the NAME.	SUBROUTINE MATMPY (A, N, M, B, L, C) SUBROUTINE QDRTIC (B, A, C, ROOT1, ROOT 2)

Figure 14. SUBROUTINE Subprogram Names

An example of the general form of a SUBROUTINE subprogram is:

```
SUBROUTINE STRESS (V,W,X,Y,Z)
.
.
.
V = arithmetic expression
.
.
W = arithmetic expression
.
.
RETURN
.
.
END
```

Other statements (represented by dots) are sequences of FORTRAN statements required to compute the values of arguments V and W based on current

values of the remaining arguments X, Y, and Z. The RETURN and END statements serve the same purpose as in FUNCTION subprograms.

A SUBROUTINE subprogram is called by a CALL statement in the calling program. For example, the statement

CALL STRESS (V, W, X, Y, Z)

calls the subprogram described in the previous example (see "CALL").

Additional rules for the use of SUBROUTINE subprograms are:

1. Each variable name used as an argument of a SUBROUTINE subprogram must appear in at least one executable statement of the subprogram.
2. A SUBROUTINE subprogram must not be inserted between two statements of another program.
3. Each of the arguments for which a value is computed must appear at least once in the SUBROUTINE subprogram as the variable on the left-hand side of an arithmetic statement, or in an input-statement list.
4. The arguments in a SUBROUTINE subprogram are dummy variables that are replaced by actual arguments in the CALL statement of the calling program.
5. Dummy arguments must correspond in order, number, and mode to the actual arguments in the CALL statement.
6. If a dummy argument is an array name, the corresponding actual argument must be an array name. Each array name must appear in a DIMENSION statement of the program or subprogram in which it appears, and the dimensions must be the same (see "DIMENSION").
7. Dummy arguments must not appear in EQUIVALENCE or COMMON statements of SUBROUTINE subprograms. However, a COMMON statement can be used to implicitly transmit arguments from a calling program to a SUBROUTINE subprogram, instead of these arguments appearing in parentheses following the subprogram name (see "COMMON").

A dummy argument in a SUBROUTINE subprogram can be replaced by an actual argument that is the name of another subprogram. For example, assuming that SECANT is a subprogram with a single floating-point argument and is supplied by the user, the SUBROUTINE subprogram

```
SUBROUTINE TRIG (DUMMY, Y)
.
.
.
B = DUMMY (Y)
.
.
```

can be called by the CALL statement

CALL TRIG (SECANT, A)

This causes DUMMY (Y) to be replaced by SECANT (A).

When FUNCTION and SUBROUTINE subprogram names appear as arguments of a CALL statement, the subprogram names must appear in F cards. The F cards must be placed before the first CALL statement using the argument. An F must be punched in column 1, and the subprogram names, separated by commas, are punched in columns 2 - 72. For example, assuming TANG and SECANT are user-supplied subprograms, an F card punched F TANG, SECANT permits either the tangent or secant to be computed within SUBROUTINE TRIG, depending on which is specified in the CALL statement.

CALL Statement

The CALL statement (Figure 15) takes the form

CALL NAME (Arguments)

and is required to call a SUBROUTINE subprogram.

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 actual arguments of the subprogram.	CALL MATMPY (X, 5, 10, Y, 7, Z) CALL QDRTIC (P*9.732, Q/4.536, R - S**2.0, X1, X2

Figure 15. CALL Statement

The arguments in a CALL statement must correspond in number, order, and mode to the arguments in the called SUBROUTINE subprogram. These arguments can take any of the forms:

1. Fixed-point constant
2. Floating-point constant
3. Fixed-point variable (subscripted or non-subscripted)
4. Floating-point variable (subscripted or non-subscripted)
5. Arithmetic expression
6. The name of a FUNCTION subprogram or a another SUBROUTINE subprogram, but not the name of the called subprogram

RETURN Statement

The RETURN statement (Figure 16) terminates both FUNCTION subprograms and SUBROUTINE subprograms, and returns control to the calling program.

A RETURN statement must be the last executed statement of a subprogram but need not be the last statement of the program. Any number of RETURN statements can be used.

GENERAL FORM	EXAMPLES
"RETURN"	RETURN

Figure 16. RETURN Statement

CONTROL STATEMENTS

The third class of 1410 FORTRAN statements enables the programmer to control the sequence of the program.

Unconditional GO TO

The unconditional GO TO statement (Figure 17) transfers control of the program to the specified statement

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

Figure 17. Unconditional GO TO Statement

Computed GO TO

The computed GO TO statement (Figure 18) transfers control to statement number n_1, n_2, n_3, . . . n_m, depending on whether the value of i at the 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, statement 50, will occur. This statement is used to obtain a computed many-way branch.

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 $1 \leqq i \leqq m$.	GO TO (30, 42, 50, 9) ,I

Figure 18. Computed GO TO Statement

Assigned GO TO

The assigned GO TO statement (Figure 19) transfers control to statement number n, where n can have any of the values n_1, n_2, . . . n_m, and is assigned its current value by a prior ASSIGNED statement.

GENERAL FORM	EXAMPLES
"GO TO n, (n_1, n_2, . . . , n_m)" where n is a non-subscripted 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)

Figure 19. Assigned GO TO Statement

ASSIGN Statement

The ASSIGN statement (Figure 20) assigns the value i to n in a subsequent GO TO statement, where i can have any of the values n_1, n_2, . . . , n_m specified in the GO TO statement.

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

Figure 20. ASSIGN Statement

IF Statement

The IF statement (Figure 21) conditionally transfers control to another statement of the program. Control is transferred to the statement number n_1, n_2, or n_3, depending on whether the value of a less than, equal to, or greater than zero. Thus, in the example, if (A(J, K) - B) is zero at the time of execution, transfer to statement number 4 occurs.

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

Figure 21. IF Statement

SENSE LIGHT Statement

The term sense light refers to symbolic binary switches in the 1410 system. Figure 22 shows the general form and an example of the SENSE LIGHT statement. If i is 0, all sense lights are turned OFF; otherwise Sense Light i is turned ON.

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

Figure 22. SENSE LIGHT Statement

IF (SENSE LIGHT) Statement

Figure 23 shows the general form and an example of the IF (SENSE LIGHT) statement. Control is transferred to statement number n_1 if Sense Light i is ON, or statement number n_2 if Sense Light i is OFF. If Sense Light i is ON, it is turned OFF.

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

Figure 23. IF (SENSE LIGHT) Statement

IF (SENSE SWITCH) Statement

Figure 24 shows the general form and an example of the IF (SENSE SWITCH) statement. Control is transferred to statement n, if Sense Switch i is ON, or statement number n_2 if Sense Switch i is OFF. In the 1410 system, sense switches are symbolic, and their settings are specified by a control card.

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, 4, 5, or 6.	IF (SENSE SWITCH 3) 30, 108

Figure 24. IF (SENSE SWITCH) Statement

IF ACCUMULATOR OVERFLOW Statement

The IF ACCUMULATOR OVERFLOW statement (Figure 25) transfers control to statement number n_1 if an overflow condition is present, otherwise to statement number n_2. The pseudo indicator can be turned ON only by floating-point operations, and is turned OFF by the test statement.

GENERAL FORM	EXAMPLES
"IF ACCUMULATOR OVERFLOW n_1, n_2," where n_1 and n_2 are statement numbers.	IF ACCUMULATOR OVERFLOW 30, 49

Figure 25. IF ACCUMULATOR OVERFLOW Statement

IF QUOTIENT OVERFLOW Statement

The IF QUOTIENT OVERFLOW statement (Figure 26) transfers control to statement number n_1 if an overflow condition is present; otherwise to statement number n_2. The pseudo indicator can be turned ON only by floating-point operations, and is turned OFF by the test statement.

GENERAL FORM	EXAMPLES
"IF QUOTIENT OVERFLOW n_1, n_2," where n_1 and n_2 are statement numbers.	IF QUOTIENT OVERFLOW 30, 49

Figure 26. IF QUOTIENT OVERFLOW Statement

IF DIVIDE CHECK Statement

The IF DIVIDE CHECK statement (Figure 27) transfers control to statement number n_1 if an overflow condition is present, otherwise to statement number n_2. The pseudo indicator can be turned ON only by floating-point operations, and is turned OFF by the test 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

Figure 27. IF DIVIDE CHECK Statement

DO Statement

The DO statement (Figure 28) is a command to execute repeatedly the statements that follow, up to and including statement number n. The first time the statements are executed with $i = m_1$. For each

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 non-subscripted fixed-point variable, and m_1, m_2, m_3 are each either an unsigned fixed-point constant or non-subscripted fixed-point variable. If m_3 is not stated, it is taken to be 1.	DO 30 I = 1, 10 DO 30 I = 1, M, 3

Figure 28. DO Statement

succeeding execution, i is increased by m_3. After they have been executed with i equal to the highest value that does not exceed m_2, control passes to the statement following the last statement in the range of the DO.

The range of a DO is that set of statements that will be executed repeatedly; that is, it is 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 of a DO, i is available as data for any computations, either as an ordinary fixed-point variable or as the variable of a subscript. After the last execution of the range, the DO is said to be satisfied.

As an example of the use of a DO statement, suppose that control has reached statement 10 of the program:

```
.
.
.
10 DO 11 I = 1, 10
11 A(I) = I* N(I)
12
.
.
.
```

The range of the DO is statement 11, and the index is I. The DO sets I to 1, and control passes into the range. The value of N_1 is converted to floating point, and stored in location A_1. Because statement 11 is the last statement in the range of the DO and the DO is unsatisfied, I is increased to 2, and control returns to the beginning of the range, statement 11. The value of $2N_2$ is then computed and stored in location A_2. The process continues until statement 11 has been executed with I = 10. Because the DO is now satisfied, control passes to statement 12.

Among the statements in the range of a DO can be other DO statements. If the range of a DO includes another DO, then all of the statements of the included DO must also be in the range of the inclusive DO. A set of DO statements satisfying this rule is called a nest of DO statements (Figure 29).

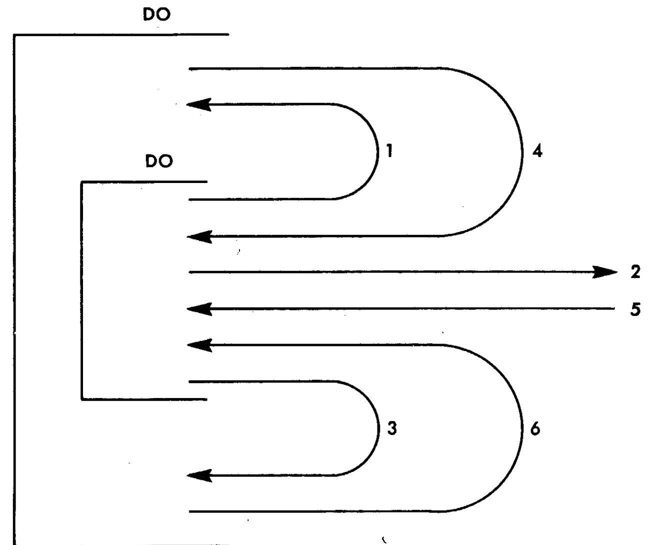


Figure 29. Nest of DO Statements

No transfer is permitted into the range of any DO from outside its range. For example, in Figure 29, 1, 2, and 3 are permitted transfers, but 4, 5, and 6 are not.

When control leaves the range of a DO in the ordinary way (that is, when the DO becomes satisfied and control passes on to the next statement after the range) the exit is said to be a normal exit. After a normal exit from a DO occurs, the value of the index controlled by that DO is not defined, and the index cannot be used again until it is redefined.

However, if exit occurs by a transfer out of the range, the current value of the index remains available for any subsequent use. If exit occurs by a transfer out of the ranges of several DO statements, the current values of all the indexes controlled by those DO statements are preserved for any subsequent use.

Restrictions on statements in the range of a DO are:

1. Any statement that redefines the value of the index (i) or of any of the indexing parameters (m's) is not permitted.
2. The first statement in the range of a DO must be an executable FORTRAN statement.
3. The last statement in the range of a DO cannot be a branch instruction (see "CONTINUE").

CONTINUE Statement

CONTINUE (Figure 30) is a dummy statement that causes no additional instructions in the object program. It is most frequently used as the last statement in the range of a DO to provide a branch address for IF and GO TO statements that are intended to begin another repetition of the DO range.

An example of a program that requires a CONTINUE is:

```
.
.
.
10 DO 12I = 1, 100
   IF (ARG - VALUE (I)) 12,20,12
12 CONTINUE
.
.
.
```

This program will scan the 100-entry VALUE table until it finds an entry that equals the value of the variable ARG, whereupon it exits to statement 20 with the value of I available for subsequent use. If no entry in the table equals the value of ARG, a normal exit to the statement following the CONTINUE occurs.

GENERAL FORM	EXAMPLES
"CONTINUE"	CONTINUE

Figure 30. CONTINUE Statement

PAUSE Statement

During the execution of the object program, the PAUSE statement (Figure 31) causes the machine to halt and print on the 1415 I/O printer the number n. If n is not specified, it is understood to be zero. Pressing the start key causes the object program to resume execution at the next instruction.

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

Figure 31. PAUSE Statement

STOP Statement

The STOP statement (Figure 32) causes a halt in such a way that pressing the start key has no effect. Therefore, in contrast to PAUSE, this statement is used where a terminal, rather than a temporary, stop is desired. When the program halts, the number n is printed on the 1415 I/O printer. If n is not specified, it is understood to be zero.

GENERAL FORM	EXAMPLES
"STOP" or "STOPn" where n is an unsigned fixed-point constant less than 10^5.	STOP STOP 33333

Figure 32. STOP Statement

END Statement

The END statement (Figure 33) is the last statement of a program or subprogram. Although the general form of this statement, as specified for other FORTRAN systems, is permissible when used in a 1410 source program, only the word END has any significance.

GENERAL FORM	EXAMPLES
"END"	END

Figure 33. END Statement

INPUT/OUTPUT STATEMENTS

The fourth class of 1410 FORTRAN statements specifies the transmission of information, during execution of the object program, between storage and input/output units:

1. READ, READ INPUT TAPE, PUNCH, PRINT, WRITE OUTPUT TAPE, TYPE RECORD and FETCH cause transmission of a specified list of data between storage and an external input/output medium such as cards, printed sheet, or magnetic tape.
2. FORMAT is non-executable. It specifies the arrangement of the information in the external input/output medium with respect to the input/output statements of group 1, and converts the information being transmitted to or from an internal notation, if necessary.
3. READ TAPE and WRITE TAPE cause the transmission of information that is already in internal machine notation, and thus need not be converted under control of a FORMAT statement.
4. END FILE, REWIND, And BACKSPACE control magnetic tape units.
5. FIND controls 1301 Disk Storage Units.

Lists of Quantities

The input/output statements that call for the transmission of information must include a list of the quantities to be transmitted. The order 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.

For example, if the list:

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

is used with an output statement, the information will be written on the output medium in the 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),
. . . .
. . . .
E(1, K), E(3, K), . . . , E(9, K), F(K, 3)

If the list is used with an input statement, the information is read into storage from the input medium. The order of the list can be considered equivalent to the "program":

```
1 A
2 B(3)
3 DO5I = 1,10
4 C(I)
5 D(I, K)
6 DO9J = 1,K
7 DO8I = 1,10,2
8 E(I, J)
9 F(J, 3)
```

Note that the parentheses in the original list define the ranges of the implied DO loops.

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. Where a single subscripted variable appears in a list preceded by the subscript itself, the entire subscripted variable must be enclosed within parentheses, if the indexing is to be carried out with the newly read-in subscript value.

Matrices

IBM 1410 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 six (2 × 3) items of information. The items will be read into storage in the same order as they are found on the input medium: $A_{1.1}$ $A_{2.1}$ $A_{1.2}$ $A_{2.2}$ $A_{1.3}$ $A_{2.3}$.

Note that the numeral 1, following READ, in this case specifies FORMAT statement number 1 (see "FORMAT").

When input/output of an entire matrix is desired, an abbreviated notation can be used for the list of the input/output statement. Only the format statement number and the name of the array are required. Thus, the statement,

READ 1, A

is sufficient to read in all of the elements of the array A, according to format statement number 1. In 1410 FORTRAN, the elements, read in by this notation, are stored in reverse order; that is, in order of decreasing storage. Note that the dimensions of an array must be specified (see "DIMENSION").

FORMAT Statement

The READ, READ INPUT TAPE, PUNCH, PRINT, WRITE OUTPUT TAPE, and TYPE require, in addition to the list of quantities to be transmitted, the number of a FORMAT statement (Figure 34). The FORMAT statement describes the information format to be used and also specifies the type of conversion to be performed between internal machine notation and external notation. FORMAT statements are not executed. They supply information to the object program. Therefore, they can be placed anywhere in the source program, except as the first statement in the range of a DO.

GENERAL FORM	EXAMPLES
"FORMAT (Specification)" where Specification is as described under *FORMAT Specification*.	FORMAT (I 2/(E 12. 4, F10.4)

Figure 34. FORMAT Statement

FORMAT Specifications

It is convenient to consider a FORMAT specification as applying to a printed line. However, the specification 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 can be:

1. A printed line with a maximum of 100 or 132 characters, depending upon the model 1403 used
2. A punched card with a maximum of 80 characters
3. A BCD tape record with a maximum of 100 or 132 characters, depending upon the model 1403 used

The terms internal and external notation are used often. Internal numerical data appears in two forms:

Integer (e.g., 314)
Floating point (e.g., 31415901)

External numerical data, to be read at object time, or constants in the source program can appear in three forms:

E(e.g., 31.4159E-1)
F(e.g., 3.14159)
I(e.g., 3)

The type of conversion required (external to internal notation or internal to external notation) depends upon the type of external notation used. For example, the floating-point constant 31.4159E-1 is expressed in external notation. E-type conversion is required to convert to the internal notation 3141590̄1.

The FORMAT specification describes the line to be printed by giving the specification for each field in the line (from left to right, beginning with the first print position). Specifications for numerical data requiring conversion include:

1. The type of conversion (E, F, or I) to be used
2. The width (w) of the field
3. For E- and F-type conversion, the number of places (d) to be printed after the decimal point.

Specifications for alphameric data or blank positions not requiring conversion include:

1. An alphabetic character designating the way the data is to be handled
2. The width(w) of the field
3. The alphameric text, when required

Field specifications are given in the forms:

Iw, Ew.d, Fw.d, wH, Aw, and wX

with the specifications for successive fields separated by commas.

Numerical Fields. An example of the printing of internally stored numerical data requiring conversion under the control of a FORMAT statement is:

Stored data	00027	9320963102	7634352602
Field specifications	12,	E12.4,	F10.4
Printed line	27b	-0.9321Eb02	bbb-0.0076

In the example, the field widths are made greater than necessary to provide spacing blanks (represented by b) between adjacent fields. The blank space following the E is automatically supplied, except in the case of a negative exponent, when a minus sign will appear. Within each field, the printed output always appears in the right-most positions. Excess low-order decimal positions, as determined by d in the FORMAT specifications, are truncated for E- and F-type conversion of output data.

If n successive fields within one record are to be printed in the same fashion, the value of n should precede E, F, or I, as required. Thus, the statement FORMAT (I2, 3E12.4) is equivalent to FORMAT (I2, E12.4, E12.4, E12.4).

A limited parenthetical expression is acceptable in order to enable repetition of data fields according to certain format specifications within a longer FORMAT specification. Thus, FORMAT (2(F10.6, E10.2), 14) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2,14).

Scale Factors. A scale factor can be applied to data that is to be printed as a result of F-type conversion. The scale factor is the power-of-10 by which data is multiplied prior to conversion. The designation nP, preceding an F-type field specification, indicates a scale factor n. For example, the specification 2PF10.4 results in multiplication of the data by 100 (10^2) prior to conversion. Thus in the earlier example, the internal data 7634352602 prints as: bbb-0.7634. Scale factor (for F-type conversion only) can be either a positive or negative number.

Scale factor can also be used with E-type conversion. However, only positive scale factors are allowed, and the magnitude of the converted data remains constant because the shifting of the decimal point to the right is offset by reduction of the E-exponent. Thus in the earlier example, the field specification 2PE12.4 causes the internal data 9320963102 to print as: -93.2096Eb00.

Scale factors have no effect on I-type conversion.

A scale factor of zero is assumed if no other factor is given. A scale factor assigned to an E- or F-type conversion applies to all subsequent E- and F-type conversions in the same FORMAT statement, until nullified by a different scale factor. Thus for example, the specifications 2PF10.4, E12.4, 4PF10.4, E12.4, has the same effect as the specification 2PF10.4, 2PE12.4, 4PF10.4, 4PE12.4.

Alphameric Fields. Alphameric text can be included in a FORMAT statement by the field specification wH. The width (w) of the field is followed by the letter H and the desired text. Thus, the statement:

```
FORMAT(11H X SQUARED = , F5.2,
    13H AND X CUBED = , F7.2)
```

might result in the printed line:

X SQUARED = 12.96 AND X CUBED = -46.66

Any valid alphameric character, including blank, can be printed. This is the only instance in which FORTRAN does not ignore blanks. It is possible to print alphameric information only (headings, page numbers, and other identifying information) by giving no list with the output statement.

If a FORMAT statement is used with an input statement, the alphameric text listed in the FORMAT statement will be replaced by whatever text is read in from the corresponding field in the input medium. When that same FORMAT statement is used for output, whatever information is then in the FORMAT statement will appear in the output data. Thus, text can be originated in the source program, or as input to the object program.

Alphameric text, designated by an input or output statement, can be read into storage or printed from storage by use of the field specification Aw. The letter A is followed by the width (w) of the field. The desired text is specified by the appropriate variable

name in the list of the input or output statement. Thus for example, the statements:

```
1 FORMAT (F8.3,A10,F10.3)
  READ 1, X, Y, Z
```

cause specific values of variables X and Z to be read into storage with F-type conversion. A ten-character alphameric field (punched in the input card between the two variable fields) is read into the storage location designated by the variable name Y.

Characters in an input field can be skipped, or blank characters can be provided in an output field by use of the field specification wX. The width (w) of the field is followed by the letter X. Thus, the statement:

```
FORMAT (F8.3, 10X, F10.3)
```

used with an input statement, causes ten characters to be skipped between two fields for which F-type conversion is specified. When used with an output statement, ten blanks are inserted between the other two fields. The maximum value of w that can be used with any input/output statement is 132.

Multiple-Line Formats. To deal with a block of more than one line of print, a FORMAT specification can have several different one-line formats, separated by a slash (/) to indicate the beginning of a new line. Thus, FORMAT (3F9.2, 2F10.4/8E14.5) specifies 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 such that the first two lines will 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, for example:

```
FORMAT (12,3E12.4/2F10.3,3F9.4/(10F12.4))
```

If data items remain to be transmitted after the format specification has been completely used, the format repeats from the preceding left parenthesis.

As these examples show, both the slash and the closing parenthesis of the FORMAT statement indicate a termination of a record. Blank lines can be introduced into a multi-line FORMAT statement by use of consecutive slashes.

Control of I/O Operations. The FORMAT statement indicates the maximum size of each record to be transmitted. Except when a FORMAT statement consists entirely of alphameric fields, the FORMAT statement is used with the list for some particular input/output statement. Control in the object program transfers repetitively between the list, which specifies whether data remains to be transmitted, and the FORMAT statement, which gives the specifications for transmission of that data.

During input/output of data, the object program scans the FORMAT statement to which the input/output statement refers. When a specification for a numerical field is found and list items remain to be transmitted, input/output takes place according to the specification of the FORMAT statement. If no items remain, transmission ceases.

Input Data. Input data that is to be read by means of a READ or READ INPUT TAPE when the object program is executed must be in essentially the same format as given in the previous examples. Thus, a card that is punched:

```
27b-0.9321Eb02bbb-0.0076
```

can be read according to

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

Within each field of the input medium, all information must appear at the extreme right. Plus signs can be omitted or indicated by a blank or +. Minus signs are punched with an 11-punch. Blanks in numerical fields are regarded as zeros. Numbers for E- and F-type conversion can contain any number of digits, but only the f high-order digits will be retained (no rounding will be performed). Numbers for I-type conversions will be treated modulo 10^k.

For economy in punching:

1. Numbers of E-type conversion need not have four columns devoted to the exponent field. The start of the exponent field can be marked by an E, or if that is omitted, by a + or - (not a blank). Thus, E2, E02, + 2, + 02, E02, and E + 02 are all acceptable exponent fields.
2. Numbers for E- or F-type conversion need not have their decimal point punched. If it is not punched, the FORMAT specification will supply it. For example, the number -09321 + 2 with the specification E12.4 will be treated as -0.9321 + 2. If the decimal point is punched in the card, its position overrides the position indicated in the FORMAT specification.

READ Statement

The READ statement (Figure 35) causes data to be read from one or more cards as specified by its list and the FORMAT statement to which it refers.

GENERAL FORM	EXAMPLES
"READ n, List" where n is the statement number of a FORMAT statement, and List is as previously described.	READ 1, ((ARRAY (I, J), I = 1, 3), J = 1, 5)

Figure 35. READ Statement

The list specifies storage locations for numerical input data. The FORMAT statement:

1. Specifies the arrangement of data on the cards.
2. Specifies the type of conversion and scale factor required for each numerical data field.
3. Provides space for alphameric text to be read from cards.
4. Specifies card columns that are to be ignored.
5. Should specify a maximum of 80 card columns for each input record (card).

READ INPUT TAPE Statement

The READ INPUT TAPE statement (Figure 36) causes the object program to read information in external notation from symbolic tape unit i. Consecutive records are read in, as specified by the FORMAT statement, until the complete list has been satisfied. For all tape statements in 1410 FORTRAN, the value of i (constant or variable) must represent an actual magnetic tape unit.

GENERAL FORM	EXAMPLES
"READ INPUT TAPE i, n, List" where i is an unsigned fixed-point constant or a fixed-point variable, n is the statement number of a FORMAT statement, and List is as previously described.	READ INPUT TAPE 5, 30, K, A (J) READ INPUT TAPE N, 30, K, A (J)

Figure 36. READ INPUT TAPE Statement

PUNCH Statement

The PUNCH statement (Figure 37) causes the object program to punch cards in accordance with the FORMAT statement, until the complete list has been satisfied.

GENERAL FORM	EXAMPLES
"PUNCH n, List" where n is the statement number of a FORMAT statement, and List is as previously described.	PUNCH 30, (A (J), J = 1, 10)

Figure 37. PUNCH Statement

PRINT Statement

The PRINT statement (Figure 38) causes the object program to print one or more lines in accordance with the FORMAT statement, until the complete list has been satisfied.

GENERAL FORM	EXAMPLES
"PRINT n, List" where n is the statement number of a FORMAT statement and List is as previously described.	PRINT 2, (A (J), J = 1, 10)

Figure 38. PRINT Statement

WRITE OUTPUT TAPE Statement

The WRITE OUTPUT TAPE statement (Figure 39) causes the object program to write information in external notation on symbolic tape unit i. Successive tape records are written in accordance with the FORMAT statement, until the complete list has been satisfied. Note that an end-of-file is not written after the last record.

GENERAL FORM	EXAMPLES
"WRITE OUTPUT TAPE i, n, List" where i is an unsigned fixed-point constant or a fixed-point variable, n is the statement number of a FORMAT statement, and List is as previously described.	WRITE OUTPUT TAPE 4, 30, (A (J), J = 1, 10) WRITE OUTPUT TAPE L, 30, (A (J), J = 1, 10)

Figure 39. WRITE OUTPUT TAPE Statement

READ TAPE Statement

The READ TAPE statement (Figure 40) causes the object program to read information expressed in internal notation from symbolic tape unit i. Tapes containing information expressed in internal notation and read by a 1410-compiled FORTRAN program must have been written by a WRITE 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 as previously described.	READ TAPE 2, (A (J), J = 1, 10) READ TAPE K, (A (J), J = 1, 10)

Figure 40. READ TAPE Statement

WRITE TAPE Statement

The WRITE TAPE statement (Figure 41) causes the object program to write information in internal notation on symbolic tape unit i.

GENERAL FORM	EXAMPLES
"WRITE TAPE i, List" where i is an unsigned fixed-point constant or a fixed-point variable, and List is as previously described.	WRITE TAPE 4, (A (J), J = 1, 10) WRITE TAPE K, (A (J), J = 1, 10)

Figure 41. WRITE TAPE Statement

TYPE Statement

The TYPE statement (Figure 42) causes the object program to print data on the 1415 I/O printer in accordance with the FORMAT statement, until the complete list has been satisfied.

GENERAL FORM	EXAMPLES
"TYPE n, List" where n is the statement number of a FORMAT statement and List is as previously described.	TYPE 56, (A(J), J = 1, 10)

Figure 42. TYPE Statement

END FILE Statement

The END FILE statement (Figure 43) causes the object program to write an end-of-file mark on symbolic tape unit i.

GENERAL FORM	EXAMPLES
"END FILE i" where i is an unsigned fixed-point constant, or a fixed-point variable.	END FILE 6 END FILE K

Figure 43. END FILE Statement

REWIND Statement

The REWIND statement (Figure 44) causes the object program to rewind symbolic tape unit i.

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

Figure 44. REWIND Statement

BACKSPACE Statement

The BACKSPACE statement (Figure 45) causes the object program to backspace symbolic tape unit i.

Formated tapes are backspaced one physical record. Non-formated tapes are backspaced one logical record.

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

Figure 45. BACKSPACE Statement

1301 Disk Storage Statements

1301 Disk Storage input/output will be performed in the interpretive mode; that is, by branching to subroutines. Control cards are used to supply these interpretive routines with descriptive information about 1301 Disk Storage. The interpretive routines will operate under the following assumptions:

(1) One or more full cylinders are available.
(2) The cylinders are consecutive.
(3) The cylinders for any one program are available on any four or less modules.

There are three 1410 FORTRAN statements that call for IBM 1301 operations. They are:

(1) RECORD (I) List
(2) FETCH (I) List
(3) FIND (e)

A description of each of these follows:

RECORD Statement

The RECORD statement (Figure 46) specifies the writing of information into 1301 Disk Storage. This statement is written as follows:

GENERAL FORM	EXAMPLE
"RECORD (I) List" where (I) is a non-subscripted fixed-point variable defining the record area and List is an input/output list.	RECORD (J) A, B, C, D

Figure 46. RECORD Statement

The RECORD statement causes the program to start writing at the record area indicated by the current value of (I) and to proceed until the list is completed. If the list is too long for a single record area but within the defined file area (see DEFINE FILE statement), writing will continue into consecutive higher-numbered record areas until the list is completed.

At the completion of the write operation, the value of (I) will be the number of the next record area. For example, if the last character was placed into

record area 23, the value of (I) will be 24 after the operation is completed, whether record area 23 was filled or not.

FETCH Statement

The FETCH statement (Figure 47) causes information to be read from 1301 Disk Storage and is written as follows:

GENERAL FORM	EXAMPLE
"FETCH (I) List" where (I) and List are as defined for the RECORD statement.	FETCH (J) A, B, C, D

Figure 47. FETCH Statement

The FETCH statement causes the program to start reading at the record area indicated by the current value of (I) and to proceed until the list is completed. If the information contained in a record area is not sufficient to exhaust the list, reading continues at the next consecutive higher-numbered record area until the list is completed. At the completion of the read operation the value of (I) will be the number of the next record area.

Status of (I)

The value of (I) refers to a record area, and must be set prior to the execution of the operation. The value of (I) may be altered during the execution of a FETCH operation if I appears in the list; however, the new value of (I) does not alter the sequence of records from which input values are taken:

For example:

```
I = 5
FETCH (I) I, C (I)
```

The fifth logical record will be read. At the conclusion of the input list transmission, I will be set to 6 regardless of the value given to I by its appearance in the list. If the first value in the fifth logical record were 19, the second value would be read into C (19), since C (I) would be C (19).

As a second example suppose that each record area contains 110 values.

```
DIMENSION A (109), B (100)
I = 7
FETCH (I) A, I, B (I)
```

The seventh logical record will be read. The first 109 values in the seventh record will be transmitted to the array A. If the last value in the seventh record were 17, the first value in the eighth record would be transmitted to B (17). At the conclusion of the FETCH operation, I will have the value 9. Note that alternating the value of I in the input list does not alter the normal sequencing from record 7 to record 8, and that at the conclusion of the entire list transmission the value of I is updated to indicate the next logical record.

FIND Statement

The FIND statement (Figure 48) can be used to specify positioning of the access arm for reading and writing:

GENERAL FORM	EXAMPLE
FIND (e) where "(e)" is a fixed-point arithmetic expression. The value of this expression must be a record number specified within the DEFINE FILE statement argument.	FIND (K*L)

Figure 48. FIND Statement

This statement is not required, but its inclusion will decrease the execution time of a program. Both the RECORD and FETCH statements produce the coding required to position the access arm for writing and reading. However, if a FIND precedes either of these statements, access time will be reduced because the access arm will already be in position to write or read when the RECORD or FETCH statement is encountered. The access arm will be positioned to read or write the logical record defined by "(e)".

SPECIFICATION STATEMENTS

The fifth class of 1410 FORTRAN statement consists of the three specification statements: DIMENSION, EQUIVALENCE, COMMON, and DEFINE FILE. These are non-executable statements that control and minimize storage allocation.

DIMENSION Statement

The DIMENSION statement (Figure 49) provides the information necessary to allocate array storage in the object program.

GENERAL FORM	EXAMPLES
"DIMENSION v, v, v, . . ." where each v is the name of an array, subscripted with 1, 2 or 3 unsigned fixed-point constants. Any number of v's may be given.	DIMENSION A (10), B (5, 15), CVAL (3, 4, 5)

Figure 49. DIMENSION Statement

Each variable that appears in subscripted form in a program or subprogram must appear in a DIMENSION statement of the same program or subprogram. The DIMENSION statement must precede the first appearance of that variable. The DIMENSION statement lists the maximum dimensions of arrays. In the object program, references to these arrays can never exceed the specified dimensions.

For example, in Figure 49, B is a two-dimensional array for which the subscripts never exceed 5 and 15. The DIMENSION statement, therefore, causes 75 (5×15) storage words to be set aside for the array B.

A single DIMENSION statement can specify the dimensions of any number of arrays. A program or subprogram must not contain a DIMENSION statement that includes the name of the program or subprogram.

EQUIVALENCE Statement

The EQUIVALENCE statement (Figure 50) controls the sharing of storage by two or more variables. Usually, such variables are subscripted and represent elements of arrays. An EQUIVALENCE statement can be placed anywhere in the source program, except as the first statement of the range of a DO. Each pair of parentheses of the statement list encloses the names of two or more array elements that are to be stored in the same location during execution of the object program. Any number of equivalences (pairs of parentheses) can be given, but fixed-point and floating-point variables must not be equated.

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, B (1), C(5)), (D(17), E(3))

Figure 50. EQUIVALENCE Statement

In an EQUIVALENCE statement, C (5) is the fifth sequentially stored element of the array named C. If a parenthetical number is not specified, it is assumed to be 1. Thus, the sample statement in Figure 50 indicates that the A, B, and C arrays are to be assigned storage locations so that the elements A, B (1) and C (5) will occupy the same storage location. Also elements D(17) and E(3) are to share the same location. Note that an EQUIVALENCE statement for elements of two or more arrays completely defines the relative locations of all elements of these arrays.

Quantities or arrays that are not mentioned in an EQUIVALENCE statement will be assigned unique locations.

COMMON Statement

The COMMON statement (Figure 51) enables data storage areas to be shared by more than one program, similar to the way the EQUIVALENCE statement provides storage sharing by a single program.

A variable (or array) that appears in both a main program and a subprogram, but has been assigned different names in the two programs, can be made to share the same storage location by a COMMON statement. For example, if the main program contains the statement

    COMMON A

and a subprogram contains the statement

    COMMON X

variables A and X will share a common storage location.

Within a specific program or subprogram, variables and arrays are assigned storage locations in the sequence that their names appear in a COMMON statement. Subsequent sequential storage assignments, within the same program or subprogram, are made by using additional COMMON statements or by modifying an earlier COMMON statement. Thus, if the main program contains the statement:

    COMMON A, B, C

and a subprogram contains the statement:

    COMMON X, Y, Z

variables A, B, and C are assigned sequential storage locations, as are variables X, Y, and Z. Furthermore, A and X will occupy the same storage location, as will B and Y, and C and Z.

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

Figure 51. COMMON Statement

A dummy variable can be used in a COMMON statement to force correspondence of two variables that otherwise would occupy different storage locations. Thus, in the previous example C and Y can be equated by writing the subprogram statement

    COMMON Q, R, Y

in which Q and R are dummy variables that will not be used in the program. The same result can be achieved by rewriting the main program COMMON statement, except that COMMON statements

appearing in additional subprograms might be affected undesirably.

The sequential assignment of storage locations is controlled first by the sequence of COMMON statements within the main program and then by the sequence of variable names within each COMMON statement. If an EQUIVALENCE statement is used, it assumes priority. That is, storage locations are assigned first to meet the requirements of the EQUIVALENCE statement, and then to meet the requirements of the COMMON statement. Thus for example, the statements

COMMON A, B, C, D
EQUIVALENCE (C,E), (B, F)

cause this sequential assignment of storage locations to the variables:

location 1 C and E
" 2 B and F
" 3 A
" 4 D

Arguments may be transmitted implicitly from a calling program to a called subprogram (see "Subprograms") by using a COMMON statement in each of the two programs to equate corresponding arguments. However, at least one explicit argument is required for FUNCTION subprograms.

DEFINE FILE Statement

The DEFINE FILE statement (Figure 52) is used to define the maximum-size 1301 disk record area addressed by the object program. The entries in this statement must be compatible with the format defined by the format track. The statement specifies:

n_1, where n_1, is the exact number of values contained in each fully utilized addressable record. This number is the number of elements in an input/output list that refers to one entire record area.

n_2, where n_2 is the maximum number of record areas that will be used in the program. All disk record areas must be of the same length. They are defined on the format track that is specified by the individual computer installation. The disk record areas must be large enough to accommodate n_1 variables, integer or floating point--whichever is specified as the longer field size. For 1410 FORTRAN, the maximum size of a disk record area is 2800 characters--one full track.

The DEFINE FILE statement is written as follows:

GENERAL FORM	EXAMPLE
DEFINE FILE (n_1, n_2) where "n_1" and "n_2" are fixed-point constants representing the maximum number of values per disk record area and the maximum number of such record areas, respectively.	DEFINE FILE (280, 150)

Figure 52. DEFINE FILE Statement

The DEFINE FILE statement must appear only in the main program and only once in this program.

4-1468-1

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York

Printed in U.S.A. J28-1468-1