File No. 1401 -25
Form C24-1455-2

IBM Systems Reference Library

Fortran Specifications and Operating Procedures IBM 1401

PROGRAM NUMBER 1401—FO—050, VERSION 3

This reference publication contains the language specifications necessary to code a 1401 Fortran source program and the procedures for assembling and running the object program. In addition to describing the 1401 Fortran language, the specifications section also contains descriptions of:

1. the control card
2. the phases of the compiler
3. the arithmetic and input/output routines generated by the compiler
4. the 1401 Fortran facility for linking programs or segments for continuous processing and
5. the input/output routine option provided in 1401 Fortran.

In addition to the procedures for assembling and running the object program, the operating procedures section also includes explanations of:

1. compiler output
2. compiler diagnostics
3. object-program storage allocation and
4. object-program halts.

The reader should be familiar with the *Fortran General Information Manual,* Form F28-8074, and the IBM 1401 configurations required for the assembly and the execution of the object program. Additional publications concerning the IBM 1401 system can be found in the IBM *1401-1460 Bibliography,* Form A24-1495.

This is a reprint of an earlier edition, and incorporates the following Technical Newsletter:

Form No.	Pages	Date
N21-0046-0	15, 20, 22, 28, 29, 33, 40, and 42	4/27/65

Copies of this and other IBM publications can be obtained through IBM Branch Offices. A form is included at the back of this manual for readers' comments. If this form has been removed, address comments to: IBM Corporation, Product Publications, Dept. 245, Rochester, Minn. 55901.

Contents

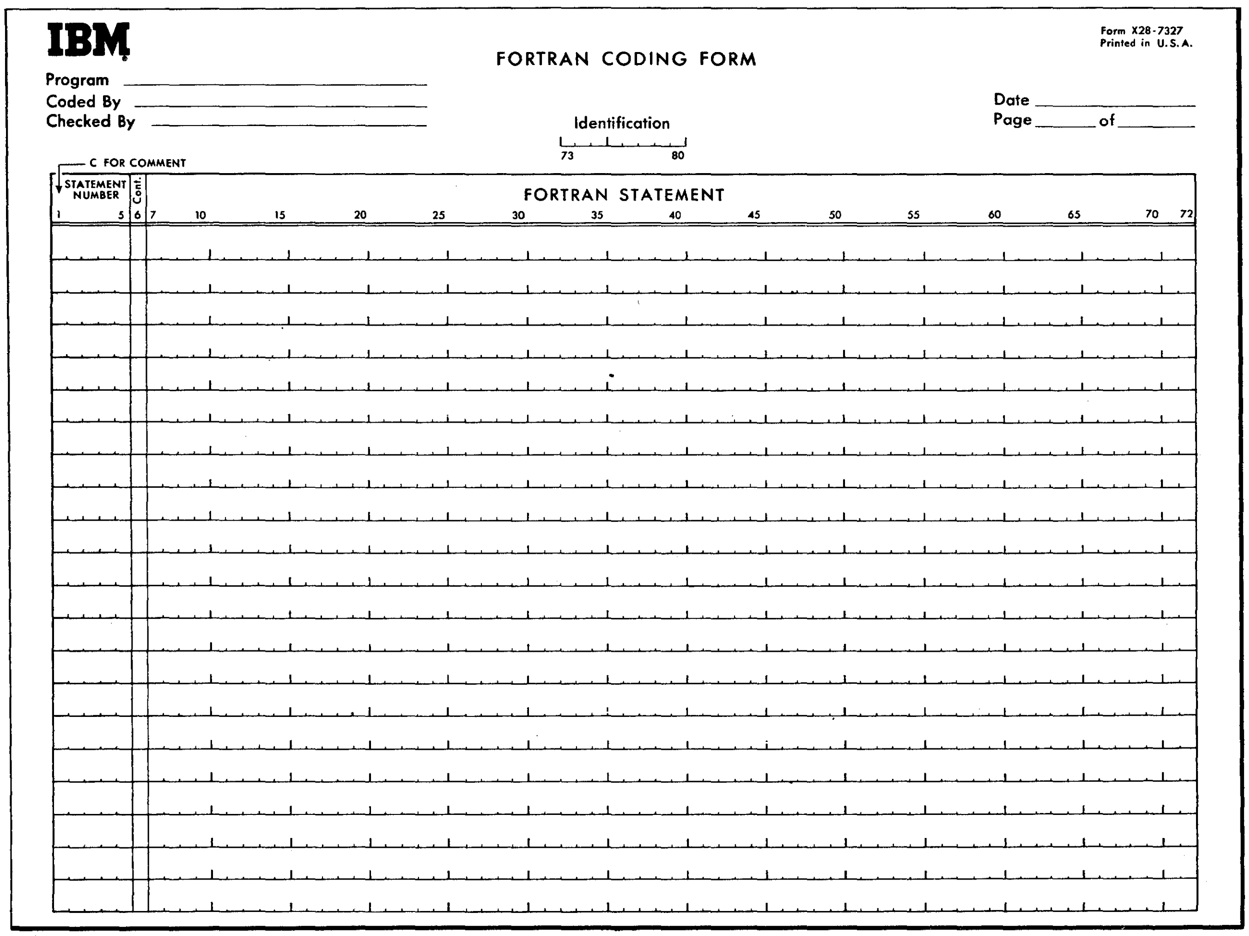
The IBM 1401 Fortran is a symbolic programming system composed of (1) a language and (2) a processor program (compiler). Symbolic or source statements are coded using the 1401 Fortran language, which closely resembles the language of mathematics. The *source program* is a particular sequence of source statements. After being coded on the Fortran Coding Form, Form X28-7327 (Figure 1), the source statements are punched into cards, which are then used as input to the 1401 Fortran compiler. The compiler translates the source program to a 1401 machine-language program *(object program)* that can be executed immediately or punched into cards for future use.

Machine Requirements

The minimum machine requirements for the compilation of a 1401 Fortran source program are as follows:

8,000 positions of core storage
Advanced Programming Feature
High-Low-Equal Compare Feature
Multiply-Divide Feature
One IBM 1402 Card Read-Punch
One IBM 1403 Printer, Model 1 or 2

One magnetic tape unit, the IBM 729 or the IBM 7330, may be used to store and load the 1401 Fortran com-



IBM

FORTRAN CODING FORM

Form X28-7327
Printed in U.S.A.

Program
Coded By
Checked By

Date
Page ____ of ____

Identification
73 80

C FOR COMMENT

STATEMENT NUMBER 1 5	Cont. 6	FORTRAN STATEMENT 7 10 15 20 25 30 35 40 45 50 55 60 65 70 72

Figure 1. Fortran Coding Form

piler. The Sense Switches feature may be used to provide a 1403 listing of the object program during various stages of compilation.

The minimum machine requirements for execution of the compiled object program are as follows:

8,000 positions of core storage
Advanced Programming Feature
High-Low-Equal Compare Feature
Multiply-Divide Feature
One IBM 1402 Card Read-Punch
One IBM 1403 Printer, Model 1 or 2

Source Program Characters

The following chart indicates the list of characters which may be used in a Fortran source program:

Character	*Card Code*	*Character*	*Card Code*
Blank		M	11-4
.	12-3-8	N	11-5
)	12-4-8	O	11-6
+	12	P	11-7
$	11-3-8	Q	11-8
*	11-4-8	R	11-9
-	11	S	0-2
/	0-1	T	0-3
,	0-3-8	U	0-4
(	0-4-8	V	0-5
=	3-8	W	0-6
A	12-1	X	0-7
B	12-2	Y	0-8
C	12-3	Z	0-9
D	12-4	0	0
E	12-5	1	1
F	12-6	2	2
G	12-7	3	3
H	12-8	4	4
I	12-9	5	5
J	11-1	6	6
K	11-2	7	7
L	11-3	8	8
		9	9

No other card codes are acceptable in 1401 Fortran source program statement cards, with the following exceptions:

4-8 will be taken to mean — (minus).

11-3-8, which normally has meaning $ only when it appears as H-conversion text in a FORMAT statement, will be taken to mean * when it appears elsewhere. In this event a message will be printed in the source program listing in the same line as the statement.

0-2-8 (prints as record mark) will be tolerated, but no characters following it in the statement will be processed even if it is merely a member of an H-conversion format specification.

Writing the Source Progam

Each Fortran statement begins a new line of the coding form. (Two statements may not appear on the same line.) Statements that are too long to fit on one line, however, may be continued on subsequent lines.

Statements and information are arranged on the coding sheet as follows (comments and continuation lines are handled separately):

1. Columns 1-5 of the first line of a statement may contain a *statement number,* which can be referenced by another Fortran statement. Statement numbers are unsigned and may range from 1 to 99999. Leading and trailing blanks and leading zeros in statement numbers are ignored by the 1401 Fortran processor. If no statement number is needed, columns 1-5 may be left blank.
2. Column 6 of the first line of a statement may be either blank or punched with a zero as the user wishes. See *Continuation Lines.*
3. Columns 7-72 contain the Fortran statements. A statement cannot consist of more than 660 characters (i. e., 10 lines — see *Continuation Lines).* The Fortran processor ignores blank characters except in the case of H-conversion (see *H-Conversion).* Blanks can be used freely to improve the readability of the source program.
4. Columns 73-80 are not processed. They can be used to punch card numbers or other identifying information.

Continuation Lines

When a Fortran statement is too long to fit on one line of the coding sheet, it may be continued on the next line or lines. A statement may take up to nine continuation lines, or a total of ten lines (660 characters).

A continuation line is coded as follows:

1. Columns 1-5 are blank.
2. Column 6 contains any character other than zero or blank.
3. Columns 7-72 contain the continuation of the Fortran statement. Column 7 can be considered as following column 72 of the preceding line.
4. Columns 73-78 are used for identification. The processor does not process these columns.

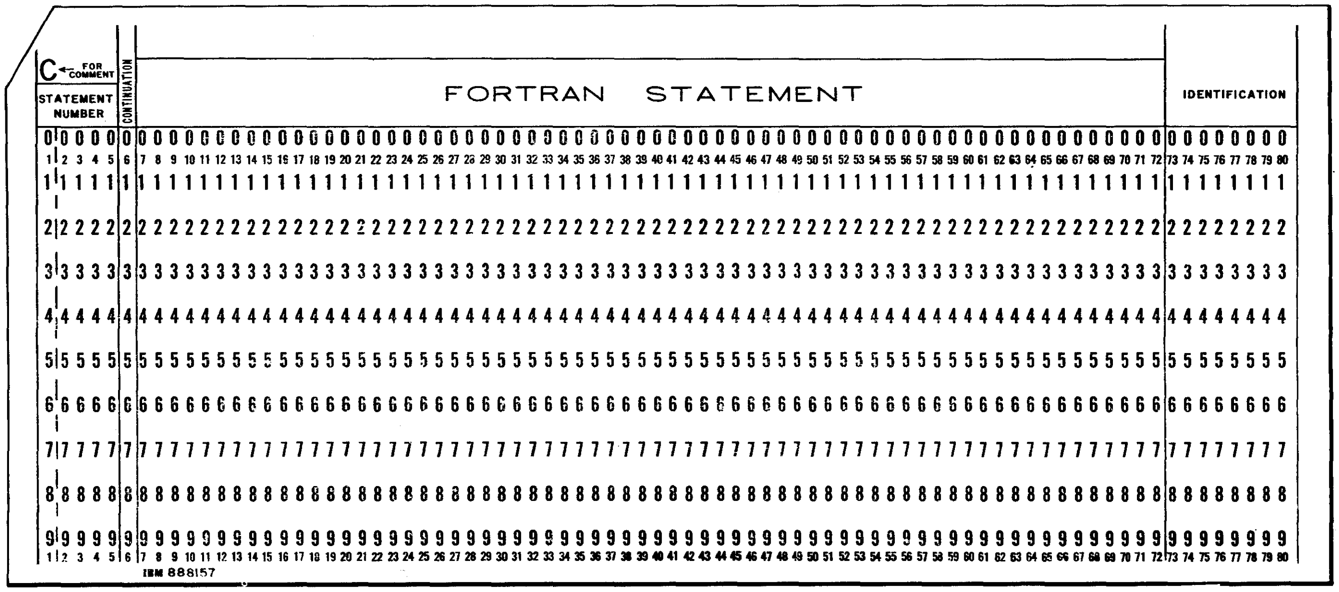


Figure 2. Fortran Statement Card

Comments Line

If the user wishes to have comments or notes appearing in the source program listing, he may use a line (or lines) of the coding sheet strictly for comments. A comments line is coded as follows:

1. Column 1 contains a C. This identifies the comments line to the processor.
2. Columns 7-72 may be used to contain the comments or notes.
3. Columns 73-80 may be used for identification. The processor does not process these columns.

Punching a Source Program

Each line of the coding sheet is punched, column for column, into a separate Fortran statement card. The Fortran statement card is shown in Figure 2. These punched cards form the Fortran source program deck.

Constants, Variables, Subscripts, Functions, and Expressions

Constants, variables, and functions separated and related by arithmetic operation signs form an arithmetic expression. (Variables can be subscripted to express one- or two-dimensional variable arrays.) The degree of precision of the value of an arithmetic expression is called *arithmetic precision.*

Arithmetic Precision

The degree of precision of an arithmetic expression (the number of digits retained) can be set by control card. The degree of precision for fixed-point and floating-point arithmetic calculations is set separately. Fixed-point precision (designated by the letter *k*) can be set to any value from 1 through 20. If any result from a fixed-point calculation exceeds *k* digits, the leftmost (high-order) extra digits are dropped.

Floating-point precision (designated by the letter *f*) can be set to any value from 2 through 20. All floating-point calculations are performed to a precision of f + 2 digits, and ultimately rounded to *f* digits.

Where no specification of precision is made:

1. fixed-point precision is 5 decimal digits.
2. floating-point precision is 8 decimal digits.

Constants

Two types of constants are permitted in a 1401 Fortran source program: *fixed-point* and *floating-point.*

Fixed-Point Constants

General Form: A fixed-point constant consists of from 1 to *k* decimal digits written without a decimal point (as integers).

Examples:

1
2
+ 524267
− 28987

Floating-Point Constants

General Form: A floating-point constant consists of any number of digits with a decimal point. *E* followed by an integer (signed or unsigned) designates multiplication by a power of 10. Floating-point constants can contain any number of digits but only a maximum of *f* significant digits are retained. Floating-point constants of *n* significant digits, where n < f, will have *n* digits of precision. The magnitude of a floating-point constant may lie between the limits 10^{-100} and $(1 - 10^{-f}) \times 10^{99}$ or be exactly zero.

Examples:

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}

Within storage, a floating-point constant of *n* significant digits consists of *s* + *2* digits, where *s* is the smaller of *n* or *f*. For example, if *f* is defined as *18*, a number in the source program having 18 or more significant digits results in a 20-digit real number, 18 for the mantissa and 2 for the characteristic. If the constant contains 13 significant digits (*f* = *18*), the internal representation will have the 15 digits: 13 for the mantissa and 2 for the characteristic.

Variables

Variable quantities are represented in 1401 Fortran statements by symbolic names. Variable names consist of from one to six alphameric characters (no special characters), of which the first character must be alphabetic. The first character of a variable name denotes which of the two types of variables a particular variable is: (1) fixed-point or (2) floating-point.

Fixed-Point Variables

General Form: All variables whose symbolic name begins with the letter I, J, K, L, M, or N are fixed-point variables.

Examples:

I
M2RB3
JOBNO

A fixed-point variable can assume any integral value (1, 2, 3, etc.) less than 10^{k} (where k is the integer precision).

When the value assumed by a fixed-point variable has fewer than k digits, high-order zeros are added. When the value exceeds k digits, only the *k* rightmost digits are retained.

Floating-Point Variables

General Form: A variable whose symbolic name begins with an alphabetic letter *other than* I, J, K, L, M, or N is a floating-point variable.

Examples:

A
B7
DELTA1

A floating-point variable can assume any value expressible as a normalized floating-point number. That is, it can be between the limits 10^{-100} and $(1 - 10^{-f}) \times 10^{99}$, or be exactly zero. A precision of *f* digits is carried in the mantissa.

Cautions in Naming Variables

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

Subscripts

A variable can be made to represent any element of a one- or two-dimensional array of quantities by appending one, or two subscripts, respectively, to the variable name. The variable is then a *subscripted variable* (see *Subscripted Variables*). The subscripts are expressions of a special form whose value determines the member of the array to which reference is made.

Form of Subscripts

General Form: A subscript may take *only* one of the following forms, where *v* represents any unsigned, nonsubscripted fixed-point variable, and *c* and *c'* represent any unsigned fixed-point constant:

```
v
c
v + c
v − c
c * v
c * v + c' or c * v − c'
```

(The * denotes multiplication.)

Examples:

```
IMAS
J9
K2
N + 3
8 * IQUAN
5 * L + 7
4 * M − 3
7 + 2 * K (invalid)
9 + J     (invalid)
```

Subscripted Variables

General Form: A subscripted variable consists of a variable name (fixed- or floating-point) followed by parentheses enclosing one or two subscripts, separated by commas.

Examples:

```
A (I)
K (3)
BETA (8 * J + 2, K − 2)
MAX (I, J)
```

1. Any subscripted variable must have the size of its array (i.e., the maximum values its subscripts can attain) specified in a DIMENSION statement preceding the first appearance of the variable in the source program. See *DIMENSION.*
2. The variable in a subscript must be greater than zero, but not greater than the corresponding array dimension.

Arrangements of Arrays in Storage

One-dimensional variable arrays are stored sequentially. Example: The array A(I), where *I* takes the integer values from 1 to 10, is stored in the sequence, A(1), A(2), A(3), . . . , A(10).

A two-dimensional variable subscript can be thought of as designating rows and columns of variables, for example, the two-dimensional array designation A(I, J) can be thought of as A (I *row,* J *column).* Two-dimensional arrays are stored sequentially by columns. Example: If A (I, J) represents a 3x2 array (I=1, 2, 3 and J=1, 2), the array is stored in the sequence A(1, 1), A(2, 1), A(3, 1), A(1, 2), A(2, 2), A(3, 2).

Functions

A function consists of a function name and a function routine. One argument is appended (in parentheses) to each function name. The argument can be any valid 1401 Fortran expression, either a fixed- or floating-point expression (as the function routine requires). The function name links the argument to the function routine.

Function routines are *closed* routines, which appear in the object program only when called, and then only once, regardless of the number of references.

The function name can be comprised of from 4 to 7 alphameric characters (not special characters). The first character must be alphabetic, and the last character must be the letter F. The first character must be X if and only if the value of the function is to be fixed point.

Examples:

```
SINF(A)
LOGF(C)
XFIXF(B)
FLOATF(I)
```

1401 Fortran Functions

1401 Fortran includes ten function subroutines:

Function	*Result*
SINF	trigonometric sine of argument
COSF	trigonometric cosine of argument
ATANF	trigonometric arctangent of argument
LOGF	natural logarithm of argument
EXPF	argument power of e
SQRTF	positive square root of argument
ABSF	absolute value of floating-point argument
XABSF	absolute value of fixed-point argument
FLOATF	convert fixed-point argument to floating point
XFIXF	convert floating-point argument to fixed point

The first seven functions listed require that both the argument and the computed value of the function be in floating-point form. For XABSF, both argument and function are fixed-point. For FLOATF and XFIXF, argument and function are of opposite form as specified.

User Functions

1401 Fortran allows the addition of up to twelve user functions. (Each function consists of a function name and a corresponding function routine.)

Function Name

The user may choose any name he wishes as long as it conforms to the specifications previously discussed under the general form of a function.

The function name is added to the 1401 Fortran table of functions. See *Adding the Function Name.*

Argument

The argument of the function may be any valid fixed- or floating-point expression. No single function may take both fixed- and floating-point arguments.

If a given operation is to give:

1. fixed- and floating-point functions, using
2. both fixed- and floating-point arguments, four separate functions must be set up, one for each function-argument combination.

Note: If more than one function routine for a given operation is to appear in the same program (or program segment), the name of the routines may be similar, but not the same. For example, the following four function names might be used when all four function-argument combinations are required for a cube-root operation:

1. CUBRTF (fixed-point argument)
2. CUBRTOF (floating-point argument)
3. XCBRTF (fixed-point argument)
4. XCBRTOF (floating-point argument)

Function Routine

The function routine is to be coded in 1401 Autocoder. Each function routine is assembled separately, and the assembled routine is then placed in the 1401 Fortran compiler. See *Incorporating the User's Function into 1401 Fortran.*

The user must consider the following restrictions when coding his function routines:

1. The routine's origin must be at position 2000.
2. The routine's length must be less than 2000 positions.
3. Any actual address is *not* relocated.
4. Any symbolic address that is assembled below position 2000 is not relocated.
5. No DA, XFR, or EX statements may be used.
6. No address constant whose operand is relocatable may be used.

In coding a function routine, the user must study the construction of arithmetic strings and the 1401 Fortran arithmetic routine. The Fortran function subroutines can be used as examples of how the functions must be coded to fit into the compiled program.

In the simplest case for example, where the source statement is:

Y = FUNCF(X),

xxx is the address of the variable X, yyy is the address of the variable Y, and R is the identifier of the function FUNCF; the arithmetic string compiled from the arithmetic statement is:

$$\underline{\text{B}}700\underline{\text{y}}\text{yy}=\underline{\text{x}}\text{xxR}\,\underline{\ddagger}$$

At the time an arithmetic statement is executed, the three-character machine address of the compiled statement within the arithmetic string (the first position after $\underline{\text{B}}$ 700) is stored in positions 084-086. Therefore in his function routine the user can refer to the address of the compiled arithmetic statement containing the reference to the function by using the contents of positions 084-086.

For convenience, the address stored in locations 084-086 will be referred to as *ARADR*. Therefore the address of yyy in our example is *ARADR*+2, the address of xxx is *ARADR*+6, and the address of the next statement to be executed in the program is *ARADR*+9. Any zone bits present in the tens position of xxx and yyy do *not* refer to address modification by an index register.

At the time of the branch to the function, the 1401 Fortran arithmetic routine has processed the argument X. The value of X is stored in a field whose address is 279+X3 (index register 3). Position 280 of this field contains a word mark. If X is a fixed-point variable, index register 3 contains the fixed-point precision value, *k*. If X is a floating-point variable, index register 3 contains the floating-point precision value plus two (f+2), position 280 contains the most significant (leftmost) digit of the mantissa, and the characteristic (exponent) is stored in positions 1677-1679. (If the mantissa is zero, the equal-compare latch is set.)

The following space is available to the user:

1. positions 1-80.
2. the index register positions.
3. positions 100-332.
4. any unused storage.
5. any area reserved by the control card.

Arithmetic Expressions

An expression is a meaningful sequence of constants, variables (subscripted or non-subscripted), and functions, separated by arithmetic operation symbols.

Examples:

```
1
A
I
A(I)
A(I)+(B/C)*2.0
A**I-(2.*B)/C
```

Arithmetic Operation Symbols

The five basic arithmetic operations are expressed by the following symbols:

+	(plus sign; addition)
—	(minus sign; subtraction)
*	(asterisk; multiplication)
/	(slash; division)
**	(two asterisks; exponentiation).

Rules for Writing Expressions

The following rules must be observed when writing 1401 Fortran expressions:

1. The mode of arithmetic in an expression can be either fixed-point or floating-point, and must not be mixed except in the following cases:
   a. A floating-point quantity can appear in a fixed-point expression as an argument of a function such as, 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. Two arithmetic-operation symbols cannot appear together, unless they are separated by parentheses. Therefore A*—B and + —A are *not* valid expressions; however A* (—B) and +(—A) *are* valid expressions.
3. In exponentiation:
   a. A floating-point exponent should not be used with a base that is a negative number, because a non-integer power of a negative number can lead to imaginary values. Also, if a floating-point exponent of a negative number is integral, the result will be a positive number regardless of whether the exponent is odd or even.
   b. A fixed- or floating-point negative number raised to a fixed-point power gives the answer with the correct sign.
   c. A fixed-point zero raised to a fixed-point power other than zero results in a fixed-point zero answer. A floating-point zero raised to either a fixed- or floating-point power other than zero results in a floating-point zero answer.
   d. Zero to the zero power will give the results indicated in each of the following cases:
      0**0 = 1
      0.**0 = 1.
      0.**0. = 1.

Note: Zero to the zero power also causes the error message ZTZ to be printed.

Hierarchy of Operations

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

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 o represents either * or / (or either + or —), then

A o B o C o D o E

will be taken by Fortran to mean

((((A o B) o C) o D) o 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.

1401 Fortran Statements

There are 25 different statements in the 1401 Fortran language. They are divided into four groups:

1. The *arithmetic statement* specifies a numerical computation.
2. *Control statements* govern the flow of the program. There are eleven different control statements.
3. *Input/Output statements* provide data input and output in a specified format. There are eleven different input/output statements.
4. *Specification statements* provide information about the storage allocation of the variables used in the program. There are two specification statements.

Arithmetic Statement

The 1401 Fortran arithmetic statement defines a numerical calculation. It closely resembles a conventional arithmetic formula; however, the equal sign of the statement specifies replacement rather than equivalence.

General Form: a = b, where:

1. *a* is fixed- or floating-point subscripted or nonscripted variable.
2. *b* is an expression.

Examples:

```
Q1 = K
A (I) = B (I) + SINF (C (I) )
```

The result of the arithmetic calculation specified by the expression (b) is stored in the field designated by the variable (a) on the left in fixed- or floating-point, according to whether the variable is fixed point or floating point.

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 an instruction 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, i.e., contains both fixed-point and floating-point quantities.
A = I*B	*Not accepted.* The expression is mixed.

Note: If characters that were read under the A-conversion format-specification (see *A-Conversion*) are referenced in an arithmetic statement, only the numeric portion of these characters (except for the sign) are considered. For example, MIN would be equivalent to − 495.

Control Statements

The second category of 1401 Fortran statements is a set of eleven statements enabling the user to control the sequence in which the program statements are to be executed.

Unconditional GO TO

General Form: GO TO *n*.
n is a statement number.

Example:

```
GO TO 3
```

The unconditional GO TO statement transfers control of the program to the specified statement.

Computed GO TO

General Form: GO TO ($n_1, n_2, \ldots, n_m$), *i*
$n_1, n_2, \ldots, n_m$ are statement numbers and *i* is a non-subscripted fixed-point variable. The range of *i* must be such that the value of *i* is $1 \leq i \leq 10$.

Example:

```
GO TO (30, 42, 50, 9), I
```

The computed GO TO statement transfers control to statement number $n_1, n_2, n_3, \ldots, n_m$, 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.

IF

General Form: IF (a) n_1, n_2, n_3
a is an expression and n_1, n_2, n_3 are statement numbers.

Example:

```
IF (A (J, K) –B) 10, 4, 30
```

The IF statement 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* is 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.

Sense Light

General Form: SENSE LIGHT *i*

i is 0, 1, 2, 3, or 4.

Example:

```
SENSE LIGHT 3
```

The term *sense light* refers to symbolic binary switches in the 1401 system. If *i* is 0, all sense lights are turned off; otherwise SENSE LIGHT *i* is turned on.

IF (Sense Light)

General Form: IF (SENSE LIGHT *i*) n_1, n_2
n_1 and n_2 are statement numbers and *i* is 1, 2, 3, or 4.

Example:

```
IF (SENSE LIGHT 3) 30, 40
```

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.

IF (Sense Switch)

General Form: IF (SENSE SWITCH *i*) n_1, n_2
n_1 and n_2 are statement numbers and *i* is 1, 2, 3, 4, 5, or 6.

Example:

IF (SENSE SWITCH 3) 30, 108

Control is transferred to statement number n_1 if sense switch *i* is on, or statement number n_2 if sense switch *i* is off. Sense switches B through G correspond to the values of *i*, 1 through 6, respectively.

Last Card Test. A test for the last card can be made using the statement IF (SENSE SWITCH 0) n_1, n_2. With sense switch A on, the IF (SENSE SWITCH 0) n_1, n_2 statement will transfer program control to statement n_1 when the last card indicator is on; otherwise control will transfer to n_2. (This particular form of the statement is unique to 1401 Fortran.)

DO

General Form: DO *n i* = m_1, m_2 or DO *n i* = m_1, m_2, m_3
n is a statement number, *i* is a nonsubscripted fixed-point variable, and m_1, m_2, m_3 are each either an unsigned fixed-point constant or nonsubscripted fixed-point variable. If m_3 is not stated, it is taken to be 1.

Examples:

```
DO 30 I = 1, 10
DO 30 I = 1, M, 3
```

The DO statement 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 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. If, in the initial setup, $m_1 > m_2$, there is no execution of the loop.

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's satisfying this rule is called a *nest of DO's* (Figure 3).

No transfer is permitted into the range of any DO from outside its range. For example, in Figure 3, 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

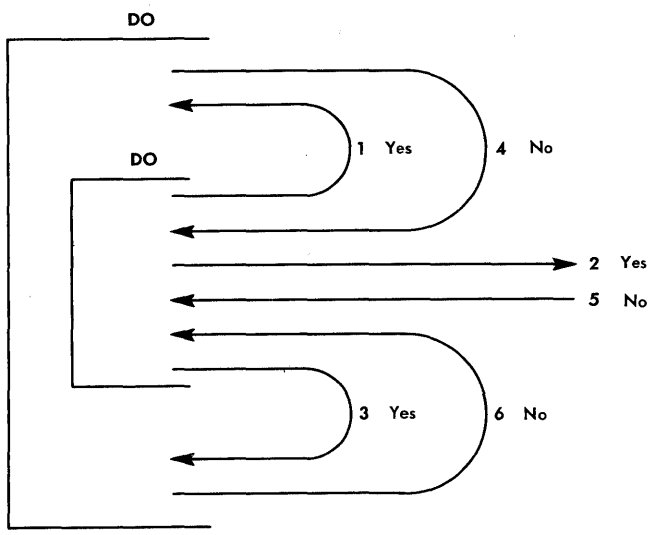


Figure 3. Nest of DO's

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's, the current values of all the indexes controlled by those DO's 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 is the range of a DO cannot be a branch instruction (see *Continue*).

Continue

General Form: CONTINUE

Example:

```
CONTINUE
```

CONTINUE 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 12 I = 1, 100
11 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.

Pause

General Form: PAUSE or PAUSE *n*

n is an unsigned fixed-point constant less than 10^3.

Examples:

```
PAUSE
PAUSE 777
```

During the execution of the object program, the PAUSE statement causes the machine to halt and display at the console the number *n* (see *Object Time Halts or Error Conditions*). 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.

Stop

General Form: STOP or STOP *n*

n is an unsigned fixed-point constant less than 10^3.

Examples:

```
STOP
STOP 333
```

The STOP statement 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 displayed on the console. (See *Object Time Halts or Error Conditions.*) If *n* is not specified, it is understood to be zero.

End

General Form: END

Example:

```
END
```

The END statement is the last statement of the source program. Although the general form of this statement, as specified for other Fortran systems, is permissible when used in a 1401 source program, only the word END has any significance.

Input/Output Statements

There are eleven 1401 Fortran statements available for specifying the transmission of information, during execution of the object program, between storage and input/output units:

1. Five statements (READ, READ INPUT TAPE, PUNCH, PRINT, and WRITE OUTPUT TAPE) that 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. One statement (FORMAT) that is non-executable. It specifies the arrangement of the information in the external input/output medium with respect to the five input/output statements of group 1, and converts the information being transmitted, if necessary, to or from an internal notation.

3. Two statements (READ TAPE, and WRITE TAPE) that 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. Three statements (END FILE, REWIND, and BACKSPACE) that control magnetic tape units.

Lists of Quantities

Of the eleven input/output statements, seven call for the transmission of information and 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  DO 5 I = 1, 10
4  C(I)
5  D(I, K)
6  DO 9 J = 1, K
7  DO 8 I = 1, 10, 2
8  E (I, J)
9  F (J, 3)
```

Note that 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.

Matrices

1401 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 rows × 3 columns) 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 1401 Fortran, the elements, read in by this notation, are stored in their natural order, that is, in order of increasing storage. Note that the dimensions of an array must be specified (see *Dimension*).

Format

The five input/output statements of group one (see *Input-Output Statements*) require, in addition to a list of quantities to be transmitted, reference to a FORMAT statement that describes the type of conversion to be performed between the internal machine language and the external notation for each quantity in the list.

General Form:

FORMAT (S_1, S_2, . . . , S_n/S'_1, S'_2, . . . , S'_n/ . . .)

Each field, S_i, is a format specification.

Example:

FORMAT (I2/(E12.4, F10.2))

1. FORMAT statements are not executed. They can be placed anywhere in the source program, except as the first statement in the range of a DO statement. Each FORMAT statement must be given a statement number.
2. The FORMAT statement indicates, among other things, the maximum size of each record to be transmitted. In this connection, remember that the FORMAT statement is used in conjunction with the list of some particular input/output statement, except when a FORMAT statement consists entirely of H conversion fields. In all other cases, control in the object program switches back and forth between

the list (which specifies whether data remains to be transmitted) and the FORMAT statement (which gives the specifications for transmission of that data).

3. Records must consist of one of the following:
   a. A tape record with a maximum length corresponding to the printed line of the printer.
   b. A punched card with a maximum of 80 characters.
   c. A line to be printed on-line, with a maximum of 100, or 132 characters, depending on the printer used.
4. The initial left parenthesis begins a record. In a read operation this means that a record is read. However, in a write operation, an output record is begun, but not written.
5. A slash terminates the current record. If list elements remain to be transmitted, a slash also begins a new record. In a read operation a slash means that no more information is obtained from the last record read; and in a write operation, that the output record which has been developed is written (even though blank, as when two slashes are adjacent).
6. The final right parenthesis of the FORMAT statement terminates the current record. If list elements remain to be transmitted, it also begins a new record and repeats. A repeat starts with the last repetitive group if there is one. (See *Repetition of Groups.*) Otherwise it starts with the specification immediately following the first left parenthesis of the FORMAT statement.
7. During input/output of data, the object program scans the FORMAT statement to which the relevant input/output statement refers. When a specification for a data field is found and list items remain to be transmitted, editing takes place according to the specification, and scanning of the FORMAT statement resumes. If no list items remain, the current record and execution of that particular input/output statement are terminated. Thus, an edited input/output operation is brought to an end when no items remain in the list, except when the next element to the right is an H conversion. In this case, the H conversion is transmitted.

Format Specification

FORMAT statement specifications designate:

For input:

1. The arrangement of data read in.
2. The type of conversion required for numeric data.
3. The space set aside for alphameric text to be read in.
4. The input fields to be skipped or ignored.
5. The extent of each input record.

For output:

1. The arrangement of data to be written, punched, or printed out.
2. The type of conversion and scale factor required for each numeric field.
3. The alphameric text to be written, punched, or printed out.
4. The output fields to be skipped or ignored.
5. The extent of each output record.
6. (In printing) the printer carriage-control character.

Numeric Field Specifications

Three types of conversion are available for numeric data:

Internal	*Conversion Code*	*External*
Floating point	E	Floating point with E exponent
Floating point	F	Floating point without exponent
Fixed point	I	Fixed point

These types of conversion are specified in the forms Ew.d, Fw.d, Iw, where:

1. *E, F,* and *I* represent the type of conversion
2. *w* is an unsigned fixed-point constant that represents the field width for converted data. This field width can be greater than required in order to provide spacing between numbers.
3. *d* is an unsigned fixed-point constant or zero that represents the number of positions of the field that appear to the right of the decimal point.

For example, the statement FORMAT (1Hb, I2, E12.4, F10.4) causes the following line to print (when given in conjunction with a PRINT statement):

Stored data	$\underline{0}002\overset{+}{7}$	$\underline{9}320963\overset{-}{1}0\overset{+}{2}$	$\underline{7}634352\overset{-}{6}0\overset{-}{2}$
Field specifications	I2,	E12.4,	F10.4
Printed line	27b—0.9321Eb02bbb—0.0076		

where *b* represents blanks. (See *Carriage Control* for an explanation of the specification 1Hb.)

Notes on E-, F-, and I-Conversion

1. Specifications for successive fields are separated by commas.
2. No format specification that provides for more characters than permitted for a relevant input/output record should be given. Thus, a format for a record

to be printed should not provide for more characters (including blanks) than the capabilities of the printer.

3. Information to be transmitted with E- and F-conversion must have floating-point names. Information to be transmitted with I-conversion must have fixed-point names.
4. The field width *w*, for F-conversion on output, must include a space for the sign, a space for the decimal point, and a space for a possible zero which precedes the decimal if the absolute magnitude is less than 1. Thus $w \geq d + 3$.

   Note: The maximum value of d that can be used is 20.

   The field width *w*, for E-conversion on output, must include one space for the sign, one space for possible rounding, one space for a decimal point, and four spaces for: the E, exponent sign, and exponent. Thus $w \geq$ the scale factor $+ d + 7$.
5. The exponent, which can be used with E-conversion, is the power of 10 to which the number must be raised to obtain its true value. The exponent is written with an E followed by a minus sign if the exponent is negative, or a plus sign or a blank if the exponent is positive, and then followed by one or two numbers which are the exponent. For example, the number .002 is equivalent to the number .2E-02.
6. If a number converted by I-conversion on output requires more spaces than are allowed by the field width *w*, the excess on the high-order side is lost. If the number requires fewer than *w* spaces, the leftmost spaces are filled with blanks. If the number is negative, the space preceding the leftmost digit will contain a minus sign if sufficient spaces have been reserved, otherwise the minus sign will be lost.

Scale Factors (With Output Only). A scale factor can be applied to data that is to be written, punched, or printed as a result of F-type conversion. The scale factor is the power-of-10 by which data is multiplied before 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) before conversion. Thus in the earlier example, the internal data $\underline{7}634352\bar{6}0\bar{2}$ 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 for output. 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 $9320963\bar{1}0\overset{+}{2}$ 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- or 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, have the same effect as the specifications 2PF10.4, 2PE12.4, 4PF10.4, 4PE12.4.

Alphameric Field Specifications

Fortran provides two ways by which alphameric information can be transmitted. The internal representation of the data is the same as the external for both specifications.

1. The specification A*w* causes *w* characters to be read into, or written from, a variable or array name.
2. The specification *nH* introduces alphameric information into a FORMAT statement.

   The basic difference between A- and H-conversion is that information handled by A-conversion is given a variable name or array name. Hence, it can be referred to by means of this name by more than one input or output statement list. Whereas, information handled by H-conversion is not given a name and may not be referred to or manipulated in storage in any way.

A-Conversion. The variable name used in conjunction with A-conversion must be a floating-point variable.

1. On input, A*w* will be interpreted to mean that a field of *w* characters is to be stored without conversion. If *w* is greater than *f*, the extra $(w - f)$ rightmost characters will be dropped. If *w* is less than *f*, the characters will be left-adjusted, and the words filled out with blanks.
2. On output, A*w* will be interpreted to mean that a field of *w* characters is to be the result of transmission from storage without conversion. If *w* exceeds *f*, only *f* characters of output will be transmitted followed by $w - f$ blanks. If *w* is less than *f* the leftmost *w* characters of the word will be transmitted.

Note: With $f = 8$, the format specification A10 will print an eight-character mantissa *and* a two-character exponent.

H-Conversion. The specification *nH* is followed in the FORMAT statement by *n* alphameric characters, and

```
XY=b-93.210bbbbbbbb
XY=9999.999bbSNSSW1
XY=bb28.768bbbbbbbb
```

Figure 4. Examples of A- and H-Conversions

should be separated from the next field by a comma. For example:

31H THIS IS ALPHAMERIC INFORMATION

Note that blanks are considered alphameric characters and must be included as part of the count *n*. The effect of *nH* depends on whether it is used with input or output.

1. On input, *n* characters are extracted from the input record and replace the *n* characters included with the source program FORMAT specification.
2. On output, the *n* characters following the specification, or the characters that replaced them, are written as part of the output record.

Figure 4 shows an example of A- and H-conversion in a FORMAT statement.

The statement FORMAT (4HbXY =, F8.3,A8) might produce the lines shown in Figure 4 where *b* indicates a blank character.

Figure 4 assumes steps in the source program read the data SNSSW1, print the data when sense switch 1 is on, and print a word containing six blanks when sense switch 1 is off.

Note: FORMAT (1Hb,3HXY=,F8.3,A8) is equivalent to FORMAT (4HbXY=,F8.3,A8) where *b* is a blank. See *Carriage Control.*

Blank Fields—X-Conversion

The specification *nX* introduces *n* blank characters into an input/output record where *n* must be less than or equal to the maximum record length.

1. On input, *nX* causes *n* characters in the input record to be skipped, regardless of what they actually are.
2. On output, *nX* causes *n* blanks to be introduced into the output record.

Repetition of Field Format

It may be desired to perform an input or output operation in the same format on *n* successive fields within one record. This can be specified by giving *n*, an unsigned integer, before E, F, I, or A. Thus, the field specification 3E12.4 is the same as writing E12.4, E12.4, E12.4.

Repetition of Groups

A repetitive group is a nonzero fixed-point constant followed by a left parenthesis, a specification list, and a right parenthesis. A repetitive group cannot itself contain a repetitive group. Thus, FORMAT (2(F10.6, E10.2), I4) is equivalent to FORMAT (F10.6, E10.2, F10.6, E10.2, I4).

Multiple-Record Formats

See *Format: General Form,* items 3, 4, and 5.

The statement FORMAT (3F9.2, 2F10.3 // I2 //) would specify a multirecord output block in which records 1, 6, 11 . . . . have the format (3F9.2, 2F10.3), records 2, 7, 12. . . . are blank, records 3, 8, 13. . . . have the format (I2), and records 4 and 5, 9 and 10, 14 and 15,. . . . are blank. On input, the same format descriptions apply and the blank records are skipped.

If a multiple-record format is desired in which the first two records are to be read or written according to a special format and all remaining records according to another format, the last record specification should be defined as a repetitive group by enclosing it in parentheses; for example,

FORMAT (I2, 3E12.4/2F10.3, 3F9.4/(10F12.4))

If data items remain to be transmitted after the format specification has been completely interpreted, the format repeats from the last previous left parenthesis. Group repetition applies again if it is present. For example, consider the FORMAT statement:

FORMAT (3E10.3, 2 (I2, 2F12.4), E28.17)

If more items in the list are to be transmitted after this format statement has been completely used, the FORMAT repeats from the left parenthesis preceding I2, and the 2 for group repetition preceding this left parenthesis applies again.

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

Carriage Control

Control of the printer carriage requires a numerical character (or blank) in the first position of the output record for each printed line:

blank	Single-space before printing
0	Double-space before printing
1-9	Skip to channel 1-9 before printing, as indicated.

The control character does not appear in the printed record. This control character is also required in output tape records that are to be used for off-line tape-to-printer operations.

The control character is usually provided by a 1H or 1X (see *Alphameric Field Specifications)* as the first field specification of a FORMAT specification. For example, the field specification *1H6* causes a 6 to be inserted in the high-order position of the output record. This in turn causes the printer carriage to skip to channel 6 before printing. The specification *1X* causes a blank to be inserted in the output record, resulting in single-spacing the printer carriage.

When alphamerical text is specified for the high-order field of an output record, the control character can be included in the alphamerical field specification. Thus the earlier example under *H-Conversion* 4HbXY = is changed to 4H6XY = to cause the printer carriage to skip to channel 6. The specification can also be written 1H6, 3HXY =.

Data Input to the Object Program

Data input to the object program is punched into cards according to the following specifications:

1. The data must correspond in order, type, and field with the field specifications in the FORMAT statement. Punching begins in card column 1.
2. Plus signs can be omitted or indicated by a +. Minus signs are indicated by an 11-punch, or an 8-4 punch.
3. Blanks in numeric fields:
   a. are regarded as zeros when no digits appear in the field (blank field).
   b. under E- and F- conversion are ignored when they are to the left or to the right of numeric characters; for example, the field 123bb under the conversion F5.2 is interpreted as 1.23.
   c. under I-conversion are regarded as zeros when they are to the left or to the right of numeric characters.
   d. are not permitted between characters.
4. Numbers for E- and F-conversion can contain any number of digits, but only the high-order *f* digits of precision will be retained. (No rounding is performed.)
5. In I-conversion only the low-order *k* digits of precision will be retained (*k* is the fixed-point precision value).

To permit economy in punching, certain relaxations in input data format are permitted.

1. Numbers for E-conversion need not have four columns devoted to the exponent field. However, if the exponent field is not four columns, the decimal point must be punched (see item 2 below). The start of the exponent field must be marked by an E or, if that is omitted, by a + or – (not a blank). Thus, E2, E + 2, + 2 and + 02 are all permissible exponent fields.
2. Numbers for E- and F-conversion need not have their decimal point punched. The format specification will supply it. For example, the number –09321 E + 02 with the specification *E12.4* will be treated as though the decimal point had been punched between the 0 and the 9. If the decimal point is punched in the card, its position overrides the position indicated in the FORMAT specification.

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 alphamerical 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.

Read

General Form: READ *n*, List

n is the statement number of a FORMAT statement, and List is as previously described under *Lists of Quantities.*

Examples:

```
READ 1, DATA
READ 1, ((ARRAY (I, J), I = 1, 3), J = 1, 5)
```

The READ statement causes data to be read from one or more cards as specified by its list and the FORMAT statement to which it refers. 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 required for each numerical data field.
3. Provides space for alphamerical text to be read from cards.
4. Specifies card columns that are to be ignored.
5. Should specify a maximum of eighty card columns for each input record (card).

See *Format Specification.*

Read Input Tape

General Form: READ INPUT TAPE *i, n,* List

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 under *Lists of Quantities.*

Examples:

```
READ INPUT TAPE 5, 30, DATA
READ INPUT TAPE N, 30, K, A (J)
```

The READ INPUT TAPE statement causes one or more tape records to be read as specified by its list and the FORMAT statement to which it refers. Data is read in *external* notation by symbolic tape unit *i,* where *i* (constant or variable) can range from 1 to 6. The list specifies storage locations for numerical input data. The FORMAT statement:

1. Specifies the arrangement of data within tape records.
2. Specifies the type of conversion required for each numerical data field.
3. Provides space for alphamerical text to be read from tape.
4. Specifies data fields that are to be ignored.
5. Should specify a maximum of 133 characters for each input tape record.

Records should be greater than 13 characters. Records of 13 characters or less are considered noise records and are bypassed. See *Format Specification.*

Punch

General Form: PUNCH *n,* List

n is the statement number of a FORMAT statement, and List is as previously described under *Lists of Quantities.*

Examples:

```
PUNCH 1, CALC
PUNCH 30, (A(J), J = 1, 10)
```

The PUNCH statement causes data to be punched into one or more cards as specified by its list and the FORMAT statement to which it refers. The list specifies storage locations of numerical output 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 alphamerical text to be punched into cards.
4. Specifies card columns that are to be skipped.
5. Should specify a maximum of eighty card columns for each output record (card).

See *Format Specification.*

Print

General Form: PRINT *n,* List

n is the statement number of a FORMAT statement and List is as previously described under *Lists of Quantities.*

Examples:

```
PRINT 1, CHART
PRINT 2, (A (J), J = 1, 10)
```

The PRINT statement causes one or more lines of data to be printed as specified by its list and the FORMAT statement to which it refers. The list specifies storage locations of numerical output data. The FORMAT statement:

1. Contains a carriage control character that is not printed (see *Printer Carriage Control*).
2. Specifies the arrangement of data to be printed.
3. Specifies the type of conversion and scale factor required for each numerical field.
4. Provides alphamerical text to be printed.
5. Specifies print positions that are to be skipped.
6. Should specify a maximum of 100 or 132 characters (exclusive of the carriage control character) depending on the model 1403 used.

See *Format Specification.*

Write Output Tape

General Form: WRITE OUTPUT TAPE *i, n,* List

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 described under *Lists of Quantities.*

Examples:

```
WRITE OUTPUT TAPE 4, 30, TOTALS
WRITE OUTPUT TAPE L, 30, (A(J), J = 1, 10)
```

The WRITE OUTPUT TAPE statement causes one or more tape records to be written as specified by its list and the FORMAT statement to which it refers. Data is written in *external* notation by symbolic tape unit *i,* where *i* (constant or variable) can range from 1 to 6. The list specifies storage locations of numerical output data. The FORMAT statement:

1. Specifies the arrangement of data within tape records.
2. Specifies the type of conversion and scale factor required for each numerical data field.
3. Provides alphamerical text to be written on tape.
4. Specifies data fields that are to be skipped.
5. Should specify a maximum of 133 characters for each input tape record. All output tape records are 133 characters long. Any record of less than 133 characters is padded with blanks to produce a 133-character tape record.

See *Format Specification.*

Read Tape

General Form: READ TAPE *i*, List

i is an unsigned fixed-point constant or a fixed-point variable, and List is as previously described under *Lists of Quantities.*

Examples:

```
READ TAPE 2, ARRAY
READ TAPE K, (A(J), J = 1, 10)
```

The READ TAPE statement causes a *single* tape record to be read as specified by its list (a FORMAT statement cannot be used). Data is read in *internal* notation by symbolic tape unit *i*, where *i* (constant or variable) can range from 1 to 6. Data read by a READ TAPE statement must have been written previously by a WRITE TAPE statement. When the list is a single non-subscripted array name, the storage space allocated to the array must be exactly equal to the tape record length. When the list contains multiple names or subscripted array names, the storage space allocated must not exceed that specified by the list of the WRITE TAPE statement that produced the tape record.

See *Input/Output Option* for further information on READ TAPE statements.

Write Tape

General Form: WRITE TAPE *i*, List

i is an unsigned fixed-point constant or a fixed-point variable, and List is as previously described under *Lists of Quantities.*

Examples:

```
WRITE TAPE 4, ARRAY
WRITE TAPE K, (A (J), J = 1, 10)
```

The WRITE TAPE statement causes a *single* tape record to be written as specified by its list (a FORMAT statement cannot be used). This statement is frequently used for temporary bulk storage of data, particularly arrays. Data is written in *internal* notation by symbolic tape unit *i*, where *i* (constant or variable) can range from 1 to 6. The length of the tape record is determined by the list.

When the list is a single non-subscripted array name, the maximum length of the record is restricted only by available storage space. The record length must not be less than 13 characters, because this is considered to be a noise record.

When the list contains multiple names or subscripted array names, the record length must not exceed 233 characters. There is no minimum record length, because an intermediate storage area is blank-filled to produce a 233-character tape record. *Note that array names must be subscripted when they appear in a multiple-name list.*

See *Input/Output Option* for further information on WRITE TAPE statements.

End File

General Form: END FILE *i*

i is an unsigned fixed-point constant or a fixed-point variable.

Examples:

```
END FILE 6
END FILE K
```

The END FILE statement causes a tape mark to be written by symbolic-tape-unit *i*.

Rewind

General Form: REWIND *i*

i is an unsigned fixed-point constant or a fixed-point variable.

Examples:

```
REWIND 3
REWIND K
```

The REWIND statement causes symbolic-tape-unit *i* to be rewound.

Backspace

General Form: BACKSPACE *i*

i is an unsigned fixed-point constant or a fixed-point variable.

Examples:

```
BACKSPACE 5
BACKSPACE K
```

The BACKSPACE statement causes symbolic-tape-unit *i* to backspace one physical record. Note that more than one physical record can be produced by a WRITE OUTPUT TAPE statement, thereby requiring more than one BACKSPACE operation.

Specification Statements

The final class of 1401 Fortran statement consists of the two specification statements: DIMENSION and EQUIVALENCE. These are non-executable statements that control and minimize storage allocation.

Dimension

General Form: DIMENSION *v*, *v*, *v*, . . .

Each *v* is the name of an array, subscripted with one or two unsigned fixed-point constants. Any number of *v*'s may be given.

Example:

```
DIMENSION A (10), B (5, 15), CVAL (3, 4)
```

The DIMENSION statement provides the information necessary to allocate array storage in the object program.

Each variable that appears in subscripted form in a program must appear in a DIMENSION statement of that program. 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.

In the example given, B is a 2-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.

Symbolic tape unit numbers must not appear in a DIMENSION statement.

Equivalence

General Form:

EQUIVALENCE *(a, b, c, . . .), (d, e, f, . . .), . . . a, b, c, d, e, f, . . .* can each be a non-subscripted variable, or a variable with a single integer subscript.

Example:

EQUIVALENCE *(A, B (1), C (5)), (D (17), E (3))*

The EQUIVALENCE statement affects core-storage assignment to the object program by indicating that two or more variables are to be assigned to the same core-storage location. Each pair of parentheses in the statement list encloses the names of the variables that are to be stored in the same location during execution of the object program.

Any number of *equivalences* (pairs of parentheses) can be used in a statement, and any number of variable names can be used within an equivalence. However, the names within the equivalence must be either all fixed-point or all floating-point, unless the floating-point size, plus two, equals the fixed-point size ($f + 2 = k$).

Arrays: An equivalence involving elements of two or more arrays completely defines the relative locations of these arrays. In the preceding example, the equivalence (D (17), E (3)) implies that D (15) and E (1) share the same location. If a nonsubscripted array name is given, the subscript is assumed to be 1. In the example, assuming A is an array name, A (1) shares core storage with B (1) and C (5).

To include an element of a two-dimensional array in an equivalence, specify its position in the stored sequence of elements of that array. Suppose that D is an array defined in the following statement:

DIMENSION D (4, 5).

If D (3,2) is to share a core-storage location with the variable E, D (7) must appear with E in an equivalence, because D (3, 2) is the seventh element of the array D. See *Arrangements of Arrays in Storage.*

Simple Variables: If a nonsubscripted variable does not refer to an array and appears in an EQUIVALENCE statement, it is treated as a one-dimensional array, and assigned a location towards the end of core storage. Like an array, it is subject to the following restrictions:

1. It must not be used to represent symbolic tape-unit numbers.
2. It must be subscripted when it appears in a multiple-name list of a READ TAPE or WRITE TAPE statement.

Input-Output Option

The user can choose the input-output format routine or designate that no format routine be included in the object program, depending on the type of input and output statements required by the program. If an I/O format routine is required, the user may choose either (1) the full I/O format routine or (2) the limited I/O format routine or (3) the full format routine plus the A-conversion format routine. (See *Control Card.*)

The full format routine occupies about 2600 positions of core storage and is capable of executing all types of input and output statements (as described under *Input-Output Statements*). The full format routine plus the A-conversion format occupy about 2900 positions of core storage.

The limited format routine occupies 300 positions of core storage, and is capable of executing only the READ TAPE and WRITE TAPE instructions of the following form:

READ TAPE i, array_1, array_2, . . . , array_j

WRITE TAPE i, array_1, array_2, . . . , array_j

Note: Only dimensioned variables can be specified in the list.

Each array is written on tape unit *i* as a single physical record, therefore these lists are not subject to the same length requirements as the lists of the ordinary READ TAPE and WRITE TAPE statements (see *Read Tape* and *Write Tape* under *Input-Output Statements).*

Records written by WRITE TAPE statements used with the limited format routine may be read by READ TAPE statements used with the full format routine, *if* the lists satisfy the restrictions of the full format routine as described under *Read Tape* and *Write Tape.*

Note: Only dimensioned variables can be specified in the list.

If the limited format routine can be used, instead of the full format routine, a considerable amount of extra storage can be saved for use in computation (because the full format routine requires 2500 positions and the limited format routine only 300 positions).

Note: The input-output option also applies to the individual segments of a segmented program, as though each segment were a separate program. See *Program Linkage.*

Program Linkage

The user may want to link two or more programs together for continuous processing; or if a program is too large to fit into core storage, and therefore broken into segments, he may want to link the segments for continuous processing. 1401 Fortran provides such a facility for linking programs or program segments. (For the following explanation, the word *segment* is used in a general sense to refer to both programs and program segments that are to be linked with other programs or program segments.)

1401 Fortran includes a linkage statement (see *Linkage Statement)* that causes:

1. the clearing of only a specified area of core storage for the next segment to be read, therefore allowing certain processed data from a segment to be preserved, *in core storage,* when the next segment is read in for execution
2. the reading of the next segment into core storage for execution.

Segments are compiled separately. The compiled segments can be read, for execution, from cards, tape, or both cards and tape (see *Preparing the Condensed Card Decks for Execution* under *Running Programs Containing Linkage Routine*).

When any compiled segments are to be read from tape, the user loads those segments with *title cards* to identify each segment (see *Title Cards*) on a tape, referred to as a library (LIB) tape. Segments are loaded on the LIB tape using Utility Deck Three (phase 95 of the 1401 Fortran compiler). In the loading process, Utility Deck Three first supplies and loads a monitor program on the LIB tape. The monitor program makes it possible to find and correct errors in segments without rewriting the LIB tape and to change the order in which segments are read from the LIB tape for execution. See *Monitor Program.*

For *each* segment, the user also has the input-output format-routine option described under *Input-Output Option.* Therefore, depending on the type of input and output statments required in a particular segment, the user can specify either the full or limited I/O format routine, or no format routine if no input and output statements are required. If the limited format routine can be used, instead of the full format routine, a considerable amount of core storage can be saved for computation in that particular program segment.

Note: The linkage statement allows the user to keep the processed data from one segment in core storage, while reading in the next segment. This enables the user to eliminate the input and output statements that otherwise would have been required to write or punch out the processed data while the next segment is read in, and to read in that data for use in the new segment.

Linkage Statement

General Form: a = XLINKF(m). a represents a fixed- or floating-point variable-name that is either nonsubscripted or subscripted with a single variable. The name a designates the location in array storage from which core storage is cleared before reading in the next segment. m represents a constant or nonsubscripted fixed-point variable whose magnitude must be $\leqq$ 999999. The contents of the field designated by m specifies the location of the next segment.

Examples:

ADUMMY(2000) = XLINKF(2)
MATRIX(I) = XLINKF(M)

The linkage statement is unique to 1401 Fortran. Although it is in the form of an arithmetic statement, it does not perform an arithmetic operation. It is a control statement that supplies information to a linkage routine that determines the location of the next segment to be executed, clears a specified area of core storage, and reads the next segment into core storage for execution.

Note: Every segment loaded on the LIB tape must contain a linkage statement.

Preservation of Array Storage

The variable-name a designates the position in array storage from which the linkage routine clears storage before reading in the next segment (see *Location of Next Segment).*

1. If a is nonsubscripted or subscripted with a constant, the linkage routine clears storage from the position preceding the first position of array storage down through position 700. Therefore, all array storage is saved for the next segment. For example: I(3) = XLINKF(M) and A = XLINKF(M) both will result in all of array storage being saved.
2. If a is subscripted with a variable, the linkage routine clears core storage from the position preceding the array represented by a down through position 700. Therefore, the portion of array storage from the array represented by a to the end of array storage is saved for the next segment. For example: A(I) = XLINKF(M) results in the portion of array storage from the beginning of the array A(1), A(2), . . . , A(i) to the end of array storage being saved, regardless of the present value of the variable I.

   a may be a two-dimensional array, but must be given a single subscript if core storage preceding the array a is to be cleared. For example, if B is a two-dimensional array, both B(I) = XLINKF(M) and B(J) = XLINKF(M) result in core storage preceding B(1, 1) being cleared.

Array Storage

The DIMENSION statement provides the compiler with the information necessary to allocate storage for arrays of variables. Each different variable name that is subscripted must appear (with its largest possible subscript) in a DIMENSION statement. Each variable in a

DIMENSION statement represents an array to the compiler, and the number of elements in the array is determined by the subscript. For example, if the variable A(2) were specified in a DIMENSION statement, A(1) and A(2) make up the corresponding array that would be allocated storage. See *Arrangements of Arrays in Storage* for more examples of arrays.

The order in which the compiler takes arrays for storage depends on two factors:

1. the order in which the DIMENSION statements appear in the source program and
2. the order in which the subscripted variables appear in the DIMENSION statement.

Note: The individual elements of each array are stored as described under *Arrangements of Arrays in Storage.*

In the following example, *Order* refers to the order in which the DIMENSION statements are read into core storage:

Order	*Statements*
1	DIMENSION MATRIX (3, 4), VECTOR(3)
2	DIMENSION A(2), B(2), C(1)
3	DIMENSION ARG(5), ANS(7)

The resulting array storage is as follows, with ARG(1) being assigned the low address, VECTOR(3) the high address, and the remaining variables being assigned addresses between them in the order specified:

ARG(1), ARG(2), . . . , ARG(5), ANS(1), ANS(2), . . . , ANS(7), A(1), A(2), B(1), B(2), C, MATRIX(1,1), MATRIX(2,1), MATRIX(3,1), MATRIX(1,2), . . . , MATRIX(1,3), . . . , MATRIX(1,4), . . . , MATRIX(3,4), VECTOR(1), VECTOR(2), VECTOR(3).

Notes on Array Storage

1. If arrays are to be saved from one segment to the next:
   a. They must be the last arrays specified in the DIMENSION statement if only one DIMENSION statement is used.
   b. If more than one DIMENSION statement is used, the DIMENSION statement(s) defining the additional arrays should occur after the DIMENSION statement defining the arrays to be saved.
2. The saved arrays from the previous segment may be given different variable-names in the current segment, as long as the size and mode (fixed-point or floating-point) of each array remains the same. The same area for both fixed-point and floating-point arrays may be reserved only if the fixed-point precision equals the floating-point precision plus two, that is, $k = f + 2$.
3. EQUIVALENCE statements containing elements of arrays affect the allocation of array storage for those elements. See *EQUIVALENCE* for an explanation and example.
4. Simple variables are not saved from one segment to another, however, they may be saved by including them in array storage by:
   a. defining them as a single-element array in a DIMENSION statement, or
   b. including them in an EQUIVALENCE statement.

Location of Next Segment

The subscript *m* specifies whether the next segment is to be taken from cards or tape and which segment on tape if tape is designated, or whether control is to pass to the monitor program.

1. If $m = 0$, the next segment will be taken from cards. Any unread data cards that precede the next segment are ignored. The following examples cause the next segment to be read from cards:

   A = XLINKF (0)

   A = XLINKF (M), where the contents of the field designated by M is zero.
2. If $m > 0$, the next segment will be taken from tape. In this case the value of *m* must be a segment number (see *Title Cards*) to identify the segment. The following examples cause segment three to be taken from tape:

   A = XLINKF (3)

   A = XLINKF(M), where the field designated by M contains 3.
3. If $m < 0$, control will pass to the monitor program. (See *Monitor Program.*) The following examples cause control to pass to the monitor program:

   A = XLINKF (−1)

   A = XLINKF (M), where the field designated by M contains −1.

Title Cards

Each program segment to be written on the LIB tape must have a *title card* to give the segment a number. The format of the title card is as follows:

Columns	*Contents*
8-10	LIB
12-17	Program segment number

The program segment number may be any number the user wishes, however, it must be six digits long. Therefore if the program number is 17, columns 12-17 must contain 000017, respectively.

The Monitor Program

If segments are to be executed from tape or both cards and tape, Utility Deck Three (phase 95) is used to load the appropriate segments on tape. Before Utility Deck Three loads segments on tape, it first supplies and loads a monitor program. The tape that contains the monitor program and segments is referred to as a LIB (library) tape. (If the segments are to be executed only from cards, Utility Deck Three is not used and, therefore the monitor program is not supplied.)

The monitor program has three main functions:

1. It initially gets the program into operation by determining the location of the first segment and reading it into core storage for execution.
2. Between segments it can provide the user with a core-storage dump of the segment just executed.
3. It provides the user with the facility to change the order in which segments are normally executed. (That order was determined by the linkage statements in each of the segments.)

In each case the monitor program requires information from a special control card, called a *call card.* Call cards are only used with the monitor program. The format of a call card is as follows:

Card Columns	*Contents*	*Explanation*
1-4	Either the letter C followed by a three-digit machine address or blanks.	The letter C followed by the three-digit machine address specifies that core storage is to be cleared from the three-digit address down through position 700 before the next segment, specified by columns 12-17, is read in for execution. Blanks indicate that no core storage is to be cleared before the next segment is read in.
8-10	LIB	LIB identifies the card.
12-17	Either a six-digit segment number or blanks.	A segment number specifies which segment *on tape* is to be read in next for execution. Blanks indicate the next segment is to be read from cards.
19-24	Either a six-digit segment number or blanks.	The contents of this field is stored and used for comparing against subsequent segments to be read from tape. When a match is made with the number of a subsequent segment, the monitor program is called instead of the segment.

Note: The word *first* can be substituted for the word *next* in each of the explanations if the monitor program is determining the location of and reading the first segment.

Initialize Operation

The procedure for running a program that has *any* segments on tape is such that the monitor program is always read into core storage first. In this case the first card in the reader should be a call card, because the monitor program reads cards until it finds a call card. Therefore, any cards preceding the call card are ignored and the information from them is lost. The monitor program checks columns 12-17 of the call card to determine the location of the first segment. The contents of columns 19-24 is stored. If a C followed by a three-digit machine address is in columns 1-4, the monitor program will clear the storage specified before reading in the first segment.

Using the Monitor Program Between Segments

The monitor program can also be called between program segments to provide a core-storage dump of the last segment executed, change the order in which segments are executed, or both. In each case, as in the case of determining the first segment, the user must provide a call card for the monitor program to read.

The linkage statement of a segment calls for the monitor program when the field designated by the variable *m* contains either:

1. a negative number, or
2. a number that equals the segment number stored from columns 19-24 of the last call card that was read.

Note: The last call card that was read would be the initial call card if the monitor program was not previously called.

After the monitor program is called, the linkage routine reads the monitor program into core storage. The monitor program will give a core-storage dump of the segment just executed if (1) the field designated by *m* in the linkage statement contains a negative number and sense switch G is on, or (2) the contents of the field designated by *m* matches the segment number stored from columns 19-24 of the last call card that was read. The user can suppress the core-storage dump only if the monitor program is called because the field designated by *m* contains a negative number *and* sense switch G is *off.*

After the core-storage dump (if any), the monitor program reads cards until it finds a call card. Data from cards preceding the call card is ignored. The monitor program stores the contents of columns 19-24 of the new call card to replace the stored contents of columns 19-24 of the previous call card. The monitor program then reads columns 12-17 of the new call card to determine the location of the next segment. If columns 1-4 contains a C followed by a three-digit machine address, the monitor program clears the core storage specified before reading in the next segment for execution.

The stored contents of columns 19-24 of the new call card will later be compared against the contents of the field designated by the variable *m* in subsequent linkage statements until the monitor program is again called and another call card is read.

The facility to call the monitor program enables the user to find errors in a segment on the LIB tape, and later when rerunning the program enter the corrected segment, without rewriting the LIB tape. The core-storage dump the monitor program provides can be used to find errors in a segment by calling the monitor program to provide the dump after the segment is executed. The core-storage dump is provided *after* the linkage statement is executed, therefore a portion of core storage has been cleared leaving only the saved arrays and the monitor program. Then, after correcting the errors and obtaining the corrected condensed card deck, the user can call the monitor program just before the segment in error on the LIB tape is read and can specify in the call card (that the monitor program will read) that the next segment is to be taken from cards.

The Processor Program

The 1401 Fortran processor program (compiler) translates the source program and compiles the object program. The user, however, must supply certain information in a control card used by the processor program.

Included in this section is a description of the control card and the logical flow of the processor program.

Control Card

It is necessary for a control card (PARAM) to precede the first card of the source program or program segment to communicate the following information to the compiler:

1. Core storage size. This specification (a three-character 1401 address) must be equal to or less than the core storage size of both the compiler machine and any object machine on which the object program is to be executed. If it is less than either machine size, that part of core storage beyond the specified address is unaffected during both compilation and execution.
2. The modulus (k) or word-size for the values of fixed-point (integer) variables in the object program.
3. The mantissa length (f) for the values of floating-point variables in the object program. Because of the two-position characteristic on the right, the word-size for floating-point variables is f + 2.
4. Whether or not a self-loading, condensed object program deck is to be punched following compilation.
5. Whether or not a snapshot of the generated program in core storage (not including the arithmetic and format [I/O] routines) is to be printed following compilation.
6. Whether or not 1401 Fortran is being compiled on the 1410 in the 1401 compatability mode.
7. Whether or not an input/output format routine other than the ordinary format routine is to be included in the object program.

Columns	*Function*
1-5	PARAM — this field identifies the control card.
6-8	The machine language for the highest core storage address (END) to be used by the compiler and object program. These are normally the physical limits; e.g., I9Z for 8,000 positions of core storage available, I9R for 12,000 positions, and I9I for 16,000 positions.
9-10	The fixed-point modulus (k) bb (blank) means k = 5 01, the minimum, means k = 1 02 means k = 2 : : 20, the maximum, means k = 20
11-12	The floating point mantissa length (f) bb (blank) means f = 8 02, the minimum, means f = 2 03 means f = 3 : : 20, the maximum, means f = 20
13	P, if condensed deck is desired; blank if not
14	S, if storage snapshot is desired; blank if not
15	T, if processing on the IBM 1410 in the 1401 compatibility mode
16	X, if no format routine is desired L, if the limited format routine (READ TAPE, WRITE TAPE operations only) is desired b (blank), if the ordinary format routine is to be used A, if the A-conversion format routine is to be added to the ordinary format routine. The A must be punched for A-conversion to operate correctly.

Logical Flow of the Processor

Snapshot Phase (00)

1. Sets word marks for constants.
2. Loads snapshot routine into positions 333-680 of core storage. (This routine performs a core-storage dump of a specified amount of core storage.) It remains there throughout compilation.

System Monitor (01)

1. Brings in next phase from system tape or initiates reading of next phase from cards, depending on whether the compiler is being used as card or tape system.
2. Clears previous phase to insure that no group-mark word-mark characters exist in the compiler area of storage when operating as a tape system.

Note: The monitor exists in storage throughout compilation. When a phase has completed its function, it transfers control back to the monitor.

Loader Phase (02)

1. Stores the information of the control card (PARAM).
2. Checks that the storage size indicated on the control card does not exceed the machine storage capacity, unless T is punched in column 15.
3. Stores the source program beginning at the address indicated on the control card. The source program is stored backwards to exploit the 1401 machine instructions that cause address registers to decrement during the scanning of the source program. Appended on the right of each statement is the statement number (if any), a one-character position which will become the statement-type code, and three positions for the internal sequence number.
4. Eliminates all non-significant blanks from the input statement while storing it. Blanks are retained only in the H-conversion part of FORMAT statements.
5. Checks that there are not more than nine continuation cards.
6. Checks for input statement characters (11-3-8 punch) or (4-8 punch), except in the H-conversion part of FORMAT statements. The former, if present, is changed to *(11-4-8 punch), the latter to —(11 punch). A record mark is treated as an end-of-card character.
7. Each statement is bounded by group-mark word-marks. The *appendage* is separated from the main body of the statement by a 5-8 punch character.
8. A STOP is generated as the last statement.

Scanner Phase (03)

1. Determines the statement type and inserts the code in the appendage of each statement.
2. Supplies a sequence number to each statement.

Sort Phase One (04)

Determines if there is enough free storage remaining to expand each statement by three characters. If not, the compilation ends. A message is printed indicating that the object program is too large.

Sort Phase Two (05)

Statements of the same type are chained. Each statement expands by three characters to contain the address of the next statement of the same type.

Sort Phase Three (06)

The source program is sorted by statement type. At the end of the sort, the source program has been shifted to the leftmost part of available storage.

Insert Group-Mark Phase (07)

The 5-8-punch which separates the main body of the statement from its appendage is replaced by a group-mark word-mark.

Squeeze Phase (08)

1. The words which defined the type of statement are eliminated, shrinking the source program. For example, the word *dimension* in DIMENSION statements is squeezed out.
2. Statements that do not begin with legal statement-defining words are noted on the printer and are eliminated from the source program.

Dimension Phase One (09)

A table of arrays is generated at the end of storage. Each table element consists of the array name, its dimensions and sufficient space for control statements and data generated by the equivalence phases and by DIMENSION Phase Two.

Equivalence Phase One (10)

1. Assures all arrays present in EQUIVALENCE statements are defined.
2. Adds simple variables present in EQUIVALENCE statements to the table of arrays generated by the previous phase. These variables are treated, in effect, as one-element arrays.

Equivalence Phase Two (11)

The dimension table is altered to show the relationship between arrays. The procedure, essentially, is to make every array whose first element is equivalent to a secondary element of another array know the distance to the first element of the latter array.

Dimension Phase Two (12)

Arrays are assigned their object-time addresses.

Variables Phase One (13)

The source program is scanned for variables. Simple variables are merely tagged for later processing by Variables Phase Four. Subscripted variables whose subscripts are constants are replaced by the object-time address of the array element. Subscripted variables whose subscripts are variable are replaced by the computation required at object time to determine the array element selected. Non-subscripted array variables appearing in lists are replaced by two machine-language addresses representing the limits of the array. Non-subscripted array variables appearing elsewhere are replaced by the address of the first element of the array.

Variables Phase Two (14)

The entire source program is shifted to the top (leftmost part) of available storage, leaving room for subsequent compiler phases. The remaining storage is cleared for tables including the array table generated by Dimension Phase Two.

Variables Phase Three (15)

This phase does housekeeping for Variables Phase Four.

Variables Phase Four (16)

The compiler first scans input-output lists and the left side of equal signs for simple variables. Each unique variable is placed in a table with its object-time address. In the second scan of this phase, all variables are matched against the table. When an entry is found, the object-time address is substituted in the statement for the variable name. Variable names not present in the table are undefined.

Variables Phase Five (17)

A check is made for unreferenced variables.

Constants Phase One (18)

Constants in the source program are noted and normalized and/or truncated.

Constants Phase Two (19)

Same as Variables Phase Two. The table of simple variables is destroyed.

Constants Phase Three (20)

Constants are placed in their object-time locations at the lower end of storage. The object-time addresses replace the constants wherever they appear.

Subscripts Phase (21)

Subscripts which must be computed at object time are reduced to the required parameters.

Statement Numbers Phase One (22)

All statement numbers that appear in the source program are reduced to a unique three-character representation. Statement numbers within the statement are moved to the beginning of each source-program statement (rightmost end of statement in storage) that contains these elements.

Tamrof Phase One (23)

FORMAT statements are checked to insure that they are referenced by input-output statements.

Tamrof Phase Two (24)

The object-time format strings are developed and stored immediately preceding the constants at the lower (rightmost) end of storage.

Lists Phase One (25)

Duplicate lists are checked and eliminated to optimize storage at object time.

Lists Phase Two (26)

The object-time list strings are developed and stored immediately to the left of the format strings at the lower end of storage.

Lists Phase Three (27)

Each input-output statement is reduced to the address of the list string (when present); the format string (when present); and the tape unit number (where applicable).

Statement Numbers Phase Two (28)

Same as Variables Phase Two.

Statement Numbers Phase Three (29)

The three-character equivalents of statement numbers appearing *within* statements (generated by Statement Numbers Phase One) are placed in a table.

Statement Numbers Phase Four (30)

The three-character equivalents of statement numbers which *identify* statements is matched against the statement number table. When the equivalent is found, the sequence number generated by the compiler for that statement is substituted in the table. Unreferenced and multi-defined statement numbers are checked.

Statement Numbers Phase Five (31)

Undefined statement numbers are noted.

Input/Output Phase One (32)

The linkage to the object format routine from the input-output statements is generated in-line.

Arith Phase One (33)

This is a housekeeping phase. The unary minus (negate) and exponentiation operators are changed to unique one-character symbols. Error checking also takes place.

Arith Phase Two (34)

All arithmetic and IF statements are unnested using a forcing table technique. Error checking continues.

Arith Phase Three (35)

Initialization for Arith Phase Four takes place.

Arith Phase Four (36)

Strings generated by Arith Phase Two are optimized to reduce the number of temporary storage areas for each statement.

Arith Phase Five (37)

IF statement exits and strings for exponentiation are created.

Arith Phase Six (38)

Optimization of temporary storage areas takes place. These areas are assigned definite locations in storage.

Input/Output Phase Two (39)

In-line instructions are generated for executing END FILE, REWIND and BACKSPACE statements.

Computed Go To Phase (40)

Statements with two to ten exits generate in-line instructions.

Go To Phase (41)

An unconditional BRANCH instruction is generated in-line in place of the original statement.

Stop/Pause Phase (42)

The proper instructions to

1. HALT
2. halt, continue, and display the number indicated are generated in-line.

Sense Light Phase (43)

In-line instructions are generated.

If (Hardware) Phase (44)

In-line instructions are generated for IF (SENSE SWITCH *i*) and IF (SENSE LIGHT *i*).

Continue Phase (45)

No object-time instructions are generated for these statements. This phase passes information required by the Resort phases of the compiler.

DO Phase (46)

Strings of unconditional BRANCH instructions and parameters are generated in-line. An unconditional BRANCH is generated to follow the last statement within the range of the DO.

Resort Phase 1 (47)

An area is made available for a table to assist in resorting the statements into their original order.

Resort Phase 2 (48)

The resort table is filled with the current location of each statement.

Resort Phase 3 (49)

The source program is resorted back to its original order. The statement number table is altered to show the current address of each statement.

Resort Phase 4 (50A)

The statements are relocated to the positions they will occupy at object time. The statement number table is adjusted to show the object time locations of the statements.

Shift Constants, Formats, and Lists (50B)

Constants, formats, and list strings are moved into their object core-storage locations above array storage. Array storage-area is cleared.

Replace Phase One (51)

Object-time instructions which reference statement numbers are corrected to the object-time addresses of the statement. Subscripts strings are cleaned up.

Load Phase (52) — Sections B and C (52A)

As the object coding may originate at 1697, the coding for phase 52 must be split into two parts, the first of which replaces the snapshot coding in positions 333-680. This phase loads the two sections.

Function/Subroutine Loader Phase (52B and 52C)
Relocatable function routines and subroutines are loaded. A table of the starting addresses of these routines is created.

Relocatable Package (53)
The relocatable routines loaded in 52B and 52C constitute phase 53A of the compiler.

Reloading Snapshot (53R)
The snapshot coding which was replaced by 52B is retained. If a snapshot is requested for phases 52 and 53, it is taken at this point.

Snapshot (53S)
Same as snapshot in phase 00.

Format Package Loader Phase (54A)
This phase selects the proper I/O routine and loads it into its object core-storage location.

Object Time Limited I/O Format (54B)
This is the limited I/O routine loaded by 54A.

Object Time Format (54C)
This is the regular I/O routine.

Object Time A Format (54D)
This is the A-format routine.

Replace Phase 2 (55)
Addresses of the fixed- and floating-word work-areas are inserted into the generated object program. Instructions which branch to the relocatable routines are corrected to show the object core-storage addresses of these routines. Unused core storage is cleared.

Snapshot Phase (56)
A snapshot of the generated program is printed if requested (if there were no source program errors which would make program execution unrewarding).

Condensed Deck Phase One (57)
When requested (if there are no input errors), the compiler will punch a self-loading card deck. The deck is listed on the printer if sense switch B is on. This phase punches only the clear-storage and bootstrap cards.

Condensed Deck Phase Two (58)
This phase punches the cards that will initialize the index registers and sense lights, the snapshot or the linkage routine, the arithmetic routine, and certain fixed addresses and constants.

Copy of Snapshot Routine (59A)
This is the object-time snapshot coding loaded by 58.

Fixed XLINK Routine (59B)
This is the object-time linkage routine.

Arithmetic Operations (59C)
This is the object-time arithmetic routine.

Condensed Deck Phase Three (60)
This phase punches the generated instructions, the constants, lists and format strings, and the i format routine.

Geaux Phase One (61)
This phase prints the end of compilation message, initializes the sense lights, and prepares the branch into the object program coding.

Geaux Phase Two (62)
The arithmetic routine is loaded. Communication is established between that routine and the generated coding. The index registers are initialized.

Arithmetic Package (63)
This phase is comprised of the arithmetic routine which is loaded by Geaux Phase Two.

Arithmetic Operations

The fixed- and floating-point arithmetic operations necessary for the execution of the compiled program are performed by an arithmetic routine which always appears in every compiled Fortran program. It contains a monitor routine which interprets the string of operand addresses and codes for operations which is compiled as a part of the procedure for every arithmetic expression in the source program. It also contains the various subroutines to accomplish the basic operations of add, subtract, multiply, and divide for both fixed-point and floating-point numbers, and a routine to normalize floating-point results.

In addition, the monitor routine will, when a function code in the string is encountered, initiate a transfer of control to one of the various function routines. These are referred to collectively as the relocatable functions, and are individually and selectively loaded by the compiler as required.

The arithmetic routine will also transfer control to a subscript routine, which will calculate the proper operand address when the string indicates the presence in the expression of a subscripted variable in which at least one of the subscripts includes a variable.

Arithmetic Routine

Any expression compiled in the procedure involves one or more groups of serial simple arithmetic or function evaluation operations, each terminated by a store of the result at a location specified in the string.

The result of a single arithmetic or function evaluation operation (within a group) is stored in a *working accumulator* (see below). If the result is the terminal value of a group of operations, it will then be moved to a temporary storage area, whose address, together with the "store" operator code appears in its string, associated with that group. If the result in the working accumulator is the terminal value of the entire expression (i. e., terminal value of the final group in the expression) it will then be moved to the final storage location, also obtained from the compiled string. This location is the address assigned to the variable on the left of the equal sign in an arithmetic statement, and is an available temporary area if the expression appeared in an IF statement.

The working accumulator is used to store the partial result during the course of a group of operations. This location and other work areas necessary to arithmetic operations occupy the arithmetic work area, core storage positions 200-332. The working mantissa precision during a floating-point expression evaluation is $f + 2$ positions, providing 2 extra positions beyond the floating-point precision specified. This provision serves to improve the accuracy of the calculated value of the expression. For fixed-point calculations the working precision is k positions.

The working-accumulator mantissa is thus $f + 2$ positions in length (floating point), or k positions (fixed point), having its leftmost digit at symbolic address ACCHI + 1. Its characteristic (floating point only) is stored in a three-character location whose rightmost position has the symbolic name EXP.

The size and format of the temporary storage locations are the same as those of source program variables, except that the mantissa is $f + 2$, (not f) digits long. The two digits of the characteristic make the total length $f + 4$ positions. Temporary storage areas for fixed point values are k digits long.

During the calculation of an expression, all partial results are truncated to the $f + 2$ digits available in the working accumulator and in the temporary storage. The final value of the expression, however, is rounded in the $f + 1$ position before it is stored to f digits of precision in final storage. Also, any output value is rounded one position to the right of the last position output. In fixed point, all results and output quantities are taken as the integral part of the true result, modulo 10^k.

The floating-point add, subtract, multiply and divide subroutines are designed to handle one operand (the working accumulator) of $f + 2$ digits of precision, and one operand of $f + 2$ *or fewer* digits of precision. The latter operand may be either a variable (f digits), a temporary location ($f + 2$ digits), or a source program constant. Such constants are stored by the compiler only to the precision to which they are written in the source program statement, up to the precision given on the control card.

The analogous fixed-point routines handle one operand (the working accumulator) of k digits of precision, and one operand of k digits or less, again allowing for the possible smaller precision of source program constants.

The basic subroutines in the arithmetic routine are tabulated as follows:

Symbolic Name	*Purpose*
ARITF	Entry point from procedure; monitor and interpret string
FSIZE	Initialize for a floating-point calculation
None	Floating-point add/subtract
FMPY	Floating-point multiply
FDIV	Floating-point divide
NMLZ1	Normalize floating-point result of a single arithmetic operation; place the normalized result in the working accumulator. If exponent overflow is detected, go to ERMSG to print message (NOF); then go to STR99. If exponent underflow is detected, go to STRZE.
XSIZE	Initialize for a fixed-point calculation.
None	Fixed-point add/subtract
XMPY	Fixed-point multiply
XDIV	Fixed-point divide.
STR99	Exponent overflow; set result magnitude equal to largest value possible in floating-point notation; set result sign as appropriate. Go to CLRWK.
STRZE	Exponent underflow, or result equals zero; set floating-point result equal to zero. Go to CLRWK.
DVERR	Division by zero; go to ERMSG to print message (DZE); then go to STR99.
QFUNCT	Linkage to relocatable function transfer control.
ERMSG	Print appropriate error messages, which includes a mnemonic three-character code and the display address in the generated procedure of the source program statement being executed. This subroutine is used to record certain circumstances, occurring during arithmetic operations, which may affect the calculations adversely.
CLRWK	Clear the work area after an individual arithmetic operation. Return to monitor.

Accuracy

In what follows, the terms *absolute error* and *relative error* will be used, and are defined as follows:

The absolute error in the calculation of a function g(x, y) of two arguments x and y (e. g., for addition, g (x, y) = x + y) is equal to: calculated g (x, y) —exact g (x, y).

The relative error is equal to: (calculated g (x, y)— exact g (x, y)) ÷ exact g (x, y).

Add/Subtract. When two numbers of like sign are added, or unlike sign subtracted, the absolute value of the relative error in the result is less than $10^{-(f+1)}$.

When two numbers of unlike sign are added, or like sign subtracted, the absolute value of the *absolute* error in the result is less than $10^{-(f+2)+c}$, where c is the larger of the characteristics of the two numbers. The relative error can be as high as 10.

Multiply. The absolute value of the relative error in the product is less than $10^{-(f+1)}$.

Divide. The absolute value of the relative error in the quotient is less than $10^{-(f+1)}$.

The error limits above do not apply if exponent overflow or underflow occurs. This will be detected during normalization of the result. For overflow, control is transferred to ERMSG, where the code NOF and the statement address are printed, and then to STR99. For underflow, control is transferred to STRZE.

Relocatable Function (Library) Routines

A number of relocatable routines designed to find a specific function of an argument y are included in the Fortran system deck, and are selected by the compiler for inclusion in the object program in accordance with the need for each evidenced by the source program. Some of these routines may be explicitly invoked by the programmer through the use of the function name assigned to the routine. Some are implicitly invoked by the programmer through the use of certain types of arithmetic expressions; for instance, a sub-expression of the form A**B requires both the exponential routine and the logarithm routine for evaluation. Figure 5 tabulates each of these functions, and exhibits the function name available to the programmer (if any). Also shown are the arithmetic meaning of the function, the correct mode of the argument and the mode of the calculated function (for those functions which are named) and the operator code, used in the generated procedure string, which at object time indicates to the arithmetic routine that control is to be transferred to that particular function routine.

Computation Method

The functions square root, exponential, sine, cosine, arc tangent and natural logarithm are computed during evaluation of an arithmetic expression wherever codes for those operations are encountered in the compiled string of addresses and operators corresponding to the source program expression. Control is passed to the proper function evaluation routine with the mantissa and exponent of the floating-point argument in fixed locations. Return from the function routine to the arithmetic routine occurs in various ways, for instance:

1. control is returned to NMLZ1 for normalization of the function value.
2. the value of the argument is found to be such that the result is known, for instance:
   a. cos(x) = 1 if x = 0; control is returned to the routine which will store +1 as the result.
   b. exp(x) is greater than or equal to 10^{99}; control is returned to the exponent overflow routine, etc.

The square-root function is computed by the odd-integer method. The result is calculated from left to right beginning with the most significant digit of the argument.

The basic computation for the exponential, sine, cosine, arc tangent and natural logarithm functions is an evaluation of the appropriate power series, in which the last term used depends upon the precision to which floating-point arithmetic is to be done. The same series evaluation routine is used for all of the functions, although it is used to compute a slightly different series for arc tangent and logarithm than for the other three functions. The routine is initialized by

			Mode of		*Procedure*
Function	*Meaning*	*Name*	*Argument*	*Function*	*Code*
Exponential	exp(y)	EXPF	Floating	Floating	E
Sine	sin(y)	SINF	Floating	Floating	S
Cosine	cos(y)	COSF	Floating	Floating	C
Arctangent	$\tan^{-1}(y)$	ATANF	Floating	Floating	T
Natural logarithm	ln(y)	LOGF	Floating	Floating	G
Float	Float a fixed number	FLOATF	Fixed	Floating	F
Fix	Fix a float number	XFIXF	Floating	Fixed	X
Absolute value	Absolute value of y	ABSF	Floating	Floating	A
Absolute value	Absolute value of y	XABSF	Fixed	Fixed	A
Negate	−y				N
Square root	$\sqrt{y}$	SQRTF	Floating	Floating	Q

Figure 5. IBM 1401 Fortran Functions

the function main line routine to give the proper result. Figure 6 exhibits the series used and shows the initialization quantities necessary to produce the different functions.

The power series routine is written to accept arguments of the form:

$$X \cdot 10^{-r}$$

where $r \geqslant 0$, and the magnitude of X is such that neither the series (partial sum or final sum) nor any of its terms equals or exceeds 10.

Function of Argument = S(arg)

For EXP, SIN, COS:

$$S(\text{arg}) = \sum_{i=0} 10^{-ai} T_i$$

Where: $\text{arg} = X \cdot 10^{-r}$

$$T_i = h(X) \frac{T_{i-1}}{D_i}$$

$$D_i = D_{i-1} + C_i$$

$$C_i = C_{i-1} + B$$

For LOG:

$$S(\text{arg}) = \sum_{i=0} 10^{-ai} \frac{T_i}{D_i}$$

Where: $\text{arg} = X$

$$T_i = h(X)\, T_{i-1}$$

$$D_i = D_{i-1} + C_i$$

$$C_i = C_{i-1} + B$$

Initialization:	EXP	SIN	COS	LOG	ATAN
T_0	1	X	1	$\left(\frac{X-1}{X+1}\right)$	X
a	r	2r	2r	0	2r
h(X)	X	$-X^2$	$-X^2$	$\left(\frac{X-1}{X+1}\right)^2$	$-X^2$
D_0	0	0	0	1	1
C_0	1	−2	−6	2	2
B	0	8	8	0	0

Thus,

for EXP: $S(\text{arg}) = 1 + X \cdot 10^{-r} + \frac{X^2}{2!} \cdot 10^{-2r} + \ldots$

$= \exp(\text{arg})$

for SIN: $S(\text{arg}) = X - \frac{X^3}{3!} \cdot 10^{-2r} + \frac{X^5}{5!} \cdot 10^{-4r} - \ldots$

$= 10^r \sin(\text{arg})$

for COS: $S(\text{arg}) = 1 - \frac{X^2}{2!} \cdot 10^{-2r} + \frac{X^4}{4!} \cdot 10^{-4r} - \ldots$

$= \cos(\text{arg})$

for ATAN: $S(\text{arg}) = X - \frac{X^3}{3} + \frac{X^5}{5} - \ldots$

$= c \pm \tan^{-1}(\text{arg})$

for LOG: $S(\text{arg}) = \left(\frac{X-1}{X+1}\right) + \frac{1}{3} \cdot \left(\frac{X-1}{X+1}\right)^3 + \frac{1}{5}\left(\frac{X-1}{X+1}\right)^5 + \ldots$

$= \frac{1}{2} \log(\text{arg}) + r \cdot \ln 10$

Figure 6. Function Evaluation

To meet these conditions, a quantity for which the sine, cosine, or exponential function is to be found may require a reduction in magnitude. This is accomplished by the main line routines for the functions, and has the following mathematical basis:

For exponential: $\exp(y) = 10^q \exp(x)$

where: $y = q \ln 10 + x$ $\quad$ q integral; $|x| < \ln 10$

For sine: if $y = n\frac{\pi}{2} + x$ $\quad$ n integral; $|x| < \frac{\pi}{2}$,

then: $\sin(y) = \sin(x)$ $\quad n = 0, 4, 8 \ldots$

$\sin(y) = \cos(x)$ $\quad n = 1, 5, 9 \ldots$

$\sin(y) = -\sin(x)$ $\quad n = 2, 6, 10 \ldots$

$\sin(y) = -\cos(x)$ $\quad n = 3, 7, 11 \ldots$

For cosine: if $y = n\frac{\pi}{2} + x$ $\quad$ n integral; $|x| < \frac{\pi}{2}$

then: $\cos(y) = \cos(x)$ $\quad n = 0, 4, 8 \ldots$

$\cos(y) = -\sin(x)$ $\quad n = 1, 5, 9 \ldots$

$\cos(y) = -\cos(x)$ $\quad n = 2, 6, 10 \ldots$

$\cos(y) = \sin(x)$ $\quad n = 3, 7, 11 \ldots$

For arc tangent:

if $y < 0$, $\tan^{-1}(y) = -\tan^{-1}(|y|)$

if $y \geq 1$, $\tan^{-1}(y) = \frac{\pi}{2} - \tan^{-1}\left(\frac{1}{y}\right)$

then if $0 \leq y < \cdot 42$, $\tan^{-1}(y) = S(\text{arg})$, where $X = y$

if $\cdot 42 \leq y \leq 1$, $\tan^{-1}(y) = -S(\text{arg})$,

where $X = \left(\frac{1-y}{1+y}\right)$

the result will be such that: $|\tan^{-1}(y)| < \frac{\pi}{2}$.

The necessary reductions are thus accomplished when necessary by a division routine so programmed as to obtain an integral quotient and remainder. For exponential, the divisor is ln 10; for sine or cosine it is $\frac{\pi}{2}$.

For logarithm:

if $y = x \cdot 10^e$ $\quad .31 \leq x < 3.1$; e integral

then

$$\ln(y) = e \cdot \ln 10 + \ln x$$

Both the series evaluation (CALC) and the division routine (DIVID) are closed subroutines contained in a relocatable program called FORTRAN FUNCTION COMMON DECK. It is made a part of the compiled program only if one or more of the four functions (exponential, sine/cosine, arc tangent and logarithm) is included in the compiled program.

The task of determining the magnitude of the argument of the function and of using the COMMON routines (if necessary to obtain the function value) is left to the individual function main line routines. There are four such routines, since sine and cosine are evaluated by using two different entry points to the same routine. When the routines are entered, the mantissa of the argument y is located in the working accumulator whose leftmost position has the symbolic name ACCHI+1 and whose length is $f + 2$ positions. The characteristic of the argument is located in a three-character location whose rightmost position has the symbolic name EXP.

Accuracy

In what follows, the terms *absolute error* and *relative error* will be used and are defined as follows:

The absolute error in the calculation of a function g(y) of an argument y is equal to: calculated g(y) —exact g(y).

The relative error is equal to: (calculated g(y) — exact g(y)) ÷ exact g(y).

The error specifications refer to the normalized function value stored in the working accumulator.

Exponential Function. For $0 \leq |y| < \ln 10$, the absolute value of the relative error in exp(y) is less than $2 \times 10^{-(f+1)}$.

For $\ln 10 \leq y < 99 \cdot \ln 10$, and for $-100 \cdot \ln 10 \leq y \leq -\ln 10$, the absolute value of the relative error in exp(y) is less than:

$$(q+2) \cdot 10^{-(f+1)}$$

where q is the integral quotient obtained when y is divided by ln 10.

For $y < -100 \cdot \ln 10$, $\exp(y) < 10^{-100}$. Thus, the value of the function is too small to be stored in floating-point notation, a circumstance which is known as exponent underflow. In this case, the value of the function is set equal to zero and the program proceeds to the next calculation.

For $y \geq 99 \cdot \ln 10$, $\exp(y) \geq 10^{99}$. The value of the function is too large to be stored and the exponent overflow routine is invoked. One of two error messages is printed, either NOF (normalize overflow) or EOF (exponential overflow), since the condition will be detected in either the normalization routine (if $y < 100 \ln 10$) or the main line exponential routine (if $y \geq 100 \cdot \ln 10$). In either case, the display address of the statement being executed is also printed, the result mantissa is set equal to a field of nines (positive), the result characteristic is set to +99, and the program proceeds to the next calculation.

Sine Function. For $0 \leq |y| \leq \frac{\pi}{2}$, the absolute value of the relative error in sin(y) is less than $2 \cdot 10^{-f}$.

For angles whose absolute values lie in quadrants other than the first, the absolute value of the *absolute* error in sin(y) is less than:

$$(q+2) \cdot 10^{-(f+1)}$$

where q is the integral number of quadrants in the angle (obtained by taking the integral part of the product $y \cdot \frac{2}{\pi}$). The upper bound on the relative error in these quadrants is equal to this quantity divided by sin(y), and can be very large when $|y|$ is close to $n\pi$, $n = 1, 2, 3, \ldots$

For $|y| > 10^f$, no attempt is made to calculate the sine. The error message SCL (sine-cosine large) is printed together with the display address of the statement being executed, the function is set equal to zero, and the program proceeds to the next calculation.

Cosine Function. For $0 \leq |y| < 1$, the absolute value of the relative error in cos(y) is less than $4 \cdot 10^{-(f+1)}$.

For $1 \leq |y| < \frac{\pi}{2} - .04 \approx 1.53$, the absolute value of the relative error in cos(y) is less than $5 \cdot 10^{-f}$.

For $1.53 \leq |y| < \frac{\pi}{2}$, the absolute value of the *absolute* error in cos(y) is less than $2 \cdot 10^{-(f+1)}$, and for angles in quadrants other than the first, the upper limit of this absolute error is:

$$(q+2) \cdot 10^{-(f+1)}$$

where q is defined as for the sine routine. The upper bound on the relative error in these quadrants, and near $\frac{\pi}{2}$ in the first quadrant, is equal to this quantity divided by cos(y), and can be very large when $|y|$ is close to $(2n-1) \cdot \frac{\pi}{2}$, $n = 1, 2, 3, \ldots$

For $|y| > 10^f$ no attempt is made to calculate the cosine. The error message and procedure (set function equal to zero) is the same as the procedure for the analagous circumstance in the sine routine.

Arc Tangent Function. For arguments less than $10 - \left(\frac{f+3}{2}\right)$ in absolute value, the absolute value of the relative error is less than $10^{-(f+1)}$.

For arguments less than .42 in absolute value, the absolute value of the absolute error is less than $.5 \cdot 10^{-(f+1)}$.

For arguments greater than .42 in absolute value, the absolute value of the absolute error is less than $3.0 \cdot 10^{-(f+1)}$.

For arguments greater than $10\left(\frac{f+3}{3}\right)$ in absolute value, the absolute value of the absolute error is less than $10^{-(f+1)}$.

Logarithm Function. For $0 < y < 0.5$ and for $2 < y$, the absolute value of the relative error in ln(y) is less than $3.5 \cdot 10^{-f}$.

For $0.5 < y < 0.95$ and for $1.05 < y < 2$, the absolute value of the relative error in ln (y) is less than $18 \cdot 10^{-f}$.

For $0.95 < y < 1.05$, the absolute value of the *absolute* error in ln(y) is less than $0.5 \cdot 10^{-f}$. The upper bound on the relative error in this range is equal to this quantity divided by ln(y), and can get very large as y approaches 1.

If y—0, the error message LNZ is printed together with the display address of the statement being executed, the function is set equal to the largest negative number in the floating-point range, and the program proceeds to the next calculation.

If $y < 0$, the error message is LNN and the function calculated is ln $|y|$.

Square Root Function. For $0 \leqslant y < 10^{99}$, the absolute value of the relative error in SQRT (y) is less than $10^{-(f+1)}$.

If *y* is negative, the error message SQN is printed along with the display address of the statement being executed. The square root of the absolute value of *y* is calculated, and the program proceeds.

Input/Output Operations

Input/output operations necessary to the execution of the compiled program are performed by the FORMAT routine.

Format Routine

For each Input-Output statement, an entry to the Format Routine is compiled. Following this appears:

1. a code indicating the appropriate I/O device;
2. the address of the series of instructions (format string) which determines the arrangement of the data (compiled from the referenced format statement); and
3. the address of the specified list of data (list string).

The format string consists of:

1. branches to appropriate closed subroutines of the Format Routine,
2. parameters describing the data which are needed by these subroutines,
3. the data itself (H-conversion fields), and
4. certain register-updating instructions.

When an item of numerical data is called for by the format statement, (GETAD), control temporarily transfers to a list routine, OBLIST (a relocatable and selectively loaded object time subroutine), which supplies the address required by processing the list string. The data is then converted to the appropriate internal (INEFI) or external (EFNTN, INOTN) notation by a Format Routine subroutine.

The H-conversion subroutine (HOLLR) is divided into two sections. On output, H-conversion transfers alphameric information from the format specification to the output area. On input, H-conversion (HOLIN) transfers alphameric data from the input area to the proper location in the format specification.

Logic Flow

1. Initialization: Work areas and index registers are initialized. Counters and switches are reset.
2. Select I/O routine: Test the code indicating the appropriate I/O device and branch to the corresponding subroutine. (Read a card, punch, print, read input tape, write output tape, write tape, read tape.)
3. I/O Routines: The input I/O routines bring in the data and place it in the work area. Control is then transferred to the format specification and the return address is saved. The output routines clear the output area, branch to the format specification and save the return address.
4. Control: Processing is now under control of the format string. This series of instructions branches to appropriate closed subroutines in the Format Routine. The subroutines necessary to process Format specifications are:

   a. OPNPR:
      1. A branch to OPNPR occurs for each left parenthesis.
      2. The OPNPR routine sets up a counter indicating number of times this set of parentheses should be executed. (This number was found as a parameter in the format specification).
      3. Sets the address of the first executable item following the open parenthesis.

   b. CLSPR:
      1. A branch to CLSPR occurs for each right parenthesis except the last.
      2. Adjusts counter set up in OPNPR and determines whether it has been satisfied. If not, control returns to last open parenthesis. If satisfied, control proceeds to next executable instruction in format string.

c. EOJ1: (The rightmost close parenthesis is translated as a branch to EOJ1)

1. For output, data is transferred (via NDLIN) to the I/O unit previously specified. If the list has not been exhausted, control is sent back to the last open parenthesis and its coefficient; otherwise, control is returned to the generated in-line procedure.
2. For input, the list is checked first. If the list is exhausted, an exit from the format routine to the procedure occurs. Otherwise, control is transferred to the I/O subroutine, more data is read into the work area, and control returns to the last open parenthesis.

d. NDLIN:

1. A branch to NDLIN occurs upon encountering a slash (/) in the format specification.
2. Resets address of I/O work area to left end position (thereby spacing a line).
3. Branches to I/O subroutine and either puts out or brings in data.

e. SCALE:

1. A branch to SCALE occurs when the format specification indicates a scale factor.
2. The SCALE subroutine saves the scaling factor for subsequent processing of E- and F-conversion data.

f. GETW:

1. A branch to GETW occurs for each E, F, or I specification in the format statement.
2. Transfers control temporarily to OBLIST for the purpose of obtaining the address of the data to be processed.
3. Upon return, transfers control to INEFI, for input data.

g. INOTN: Processes I-conversion data for output statements, including when necessary, the insertion of a leading minus sign, space permitting.

h. EFOUT:

1. Processes E- and F-conversion output data.
2. Adjusts characteristic of internal data for scaling.
3. Moves data to output area, inserts sign if necessary, positions decimal point and adds E nn as last 4 positions of data for E-type conversion.

i. INEFI:

1. Determines from the format specification the location of rightmost character of E, F, or I input data within the work area.
2. Scans for first significant digit of data.
3. Branches to INI for I-conversion input data.
4. E and F Input data are converted to internal notation and the characteristic adjusted as required by a scaling factor and/or decimal point position.
5. Transfers converted data to storage as specified by the LIST address.

j. INI:

1. Processes I-conversion input data.
2. Converts data to internal notation.
3. Transfers converted data to storage as specified by the list address.

Performance Data

The time required to process a 1401 Fortran program is determined by the following factors:

1. *Overhead.* This involves the time necessary to read and pass through the phases of the compiler. The time required:

   a. for a card system: 2 minutes 56 seconds

   b. for a tape system: 16 seconds.

   (The time difference is because the compiler can be read faster from tape.)

2. *Input/Output Operations.* This involves the need to read the source-program deck, print a core-storage snapshot (dump), and punch the condensed deck.

3. *Resorting.* This involves the time needed to reorder the statements into their final core-storage locations after processing is completed. This time is the most significant part of compilation time, particularly when:

   a. there are a large number of different types of statements, and

   b. core storage is completely filled.

4. *Number of Input Characters* (size of the source program). Compilation time varies directly with the number of input characters.

Minimum and Maximum Compilation Time

The time required to compile a 1401 Fortran program varies from:

1. 16 seconds to 15 minutes for a tape system (i.e., where the compiler is on tape), and
2. 2 minutes 56 seconds to 17 minutes 45 seconds for a card system i.e., where the compiler is read from cards. (The difference is that information can be read faster from tape than from cards.)

The minimum program in this case consists of a single control statement. The suggested maximum program is one that:

1. involves the use of every type of Fortran statement.
2. fills core storage (16000 positions in this case). This would require 400 statements, assuming an average length of 25 characters per statement.

Examples

The following three cases are presented as examples:

Case	*Number of Input Statements*	*Compilation Time*	*Core-Storage Positions Used*
1	42	52 secs.	7,996
2	25	1 min. 50 secs.	7,352
3	424	10 min. 35 secs.	15,856

Case 1 (see Figure 8) is a matrix calculation. Case 2 (see Figure 9) illustrates a use of the library functions. Case 3 calculates characteristics of sort programs.

Input/output operations and sorting and resorting of statements require the most significant part of compilation time. For example, input/output operations required approximately:

1. 32 seconds in Case 1 (includes snapshot).
2. 1 minute 26 seconds in Case 2 (includes snapshot and condensed deck).
3. 4 minutes 35 seconds in Case 3.

In Case 3, more than half of the remaining 6 minutes was used to sort and resort.

Fortran Operating Procedures — IBM 1401

This section contains the information necessary to compile and execute an object program from a source program written in 1401 Fortran. Included also are the diagnostics, halts, and messages that may be encountered when compiling and executing the object program.

Compiling Operation Procedures

Library Tape

The 1401 Fortran system on tape, consists of a self-loading program, blocked printer records, and blocked condensed card records. You may retrieve the data from the tape through the following procedure:

1. Ready the tape on Tape Unit 1.
2. Set the I/O check stop switch up.
3. Reset the system.
4. Press Tape Load. A program halt will occur at 364.
5. a. If the *symbolic listing* is desired, press Start. At the end of the listing a program halt will occur at 600. If *condensed cards* are then desired, press Start. Otherwise, press Start Reset, then Start to rewind the tape.

   b. If *only condensed cards* are desired, press Start Reset, then Start. The tape will be searched past the symbolic listing records and then commence punching. After punching is completed, an automatic tape rewind occurs.

The cards which are produced by this operation constitute the 1401 Fortran Compiler Deck, Utility Deck 3, Utility Deck 2, the two sample problems, and Utility Deck 1 (the compiler tape generator). The decks may then be used as described below.

Compiler Deck Description

The decks comprising the 1401 Fortran system are identified as such by 50 in columns 76 and 77. The version number is punched in column 80. Phase numbers punched in columns 78 and 79 identify the functional segments of the system. From an operational point of view, it is only necessary to locate the following phase boundaries in the deck:

1. The end of phase 02, the loader.
2. The beginning of phase 95, Utility Deck 3, the library tape generator.
3. The beginning of phase 96 (optional, on request), Utility Deck 2, the relocatable condensor deck.
4. The beginning of phase 97, sample problem 1 (matrix arithmetic).
5. The beginning of phase 98, sample problem 2 (trigonometric, logarithmic, exponential, and square root functions).
6. The beginning of phase 99, Utility Deck 1, the compiler tape generator.

Phases 00-63 are continuously numbered (with gaps between some phases) in columns 72-75 and constitute the compiler deck. Set the rest aside. Phases 95-99 are each continuously numbered in columns 74-75.

Addition of Arbitrary Relocatable Library Functions

This section describes the procedures to follow in:

1. assembling the user's 1401 Autocoder function routines
2. including the additional function names in the 1401 Fortran function table, and
3. including the user's assembled function routine in the 1401 Fortran compiler. See *User Functions.*

Assembling the User's Function Routines

The user's function routines are assembled using the 1401 Autocoder (on tape) processor and procedures. No condensed output need be specified in the control card. *If there are no errors:*

1. Place the 1401 Fortran Utility Deck 2 (phase 96) in the 1402 card-read hopper.
2. Press the 1402 load key.
3. Press the start key when the reader stops at the last card.

A condensed deck without clear-storage and bootstrap cards will be produced from the data on the Autocoder tape. This condensed deck will be suitably zoned so that it can be relocated and loaded when it is named in the source program.

Incorporating the User's Function into 1401 Fortran

To incorporate the new function into the 1401 Fortran system, the user must:

1. add the name of the function to the table of valid library functions, and
2. insert the relocatable condensed deck into the system deck.

To add the function name:

1. consult the 1401 Fortran listing, Phase 33 (Arith Phase One), under the comment card TABLE OF FORTRAN FUNCTIONS.
2. commencing at the statement bearing the remark "USER FUNCTIONS", note the column of codes, H, D, M, L, K, etc.
3. choose an unassigned code and note its condensed card number along with the value of *n* in its comment USER FUNCTION *n*.
4. pull the indicated card from the system deck and find the first unassigned code punch. It will be preceded by 8 blanks.
5. in this blank field, if the name has 7 characters, a left parenthesis must be punched, followed by the characters of the name, IN REVERSE ORDER, commencing with F.

   For a six-character function name, the left parenthesis is preceded by one blank. A five-character function name has its left parenthesis preceded by two blanks. A four-character function name, the minimum, has three blanks preceding the left parenthesis.
6. restore the card to the same position in the system deck.

To insert the condensed, relocatable deck in the system deck:

1. List phase 53 of the condensed compiler deck to find the series of cards with the comment USER FUNCTION *n* GOES HERE in columns 1-25.
2. Note the condensed card number of the comment containing the value of *n* selected in phase 33.
3. Pull that card *and* the one following from the system condensed deck and replace them by the condensed relocatable deck.
4. Generate a new system tape, if desired.

Compiling Procedure

Note: Program segments are assembled as though each were a separate program.

As a card system:

1. Place source program deck, preceded by an appropriate control card, between phase 02 and phase 03 of the Fortran compiler deck in the 1402 read hopper.
2. Set sense switch A up. Set all other sense switches down.
3. Set sense switch B up, if the condensed deck is to be listed on printer.
4. Reset the machine.
5. Press Load on the 1402.
6. Press Start when the reader stops at the last card. When the end-of-compilation message prints (see *Compiler Output)* the compiler deck (with inserted source deck) and the condensed object deck (if any) will be in the 1402 stackers.

As a tape system to generate the compiler tape:

1. Place phase 99, the compiler tape generator at the front of the compiler condensed deck in the 1402 read hopper.
2. Ready an unprotected tape on Tape Unit 1.
3. Set sense switch A up. Set all other sense switches down.
4. Reset the machine.
5. Press Load on the 1402.
6. Press Start when the reader stops at the last card. The following message will be printed:
   1401 FORTRAN COMPILER GENERATED
   ON TU1
7. File-protect the compiler tape.

Once the compiler tape is generated, the compiler deck may be filed.

To run the tape system:

1. Ready the compiler tape on Tape Unit 1.
2. Set sense switch A up. Set all other sense switches down.
3. Set sense switch B up, if the condensed deck is to be listed on the printer.
4. Reset the machine.
5. Press Tape Load.
6. Place the source program in the 1402 read hopper, preceded by appropriate control card.
7. Press Start. Press Start again when reader stops at the last card. When the end-of-compilation message prints (see *Compiler Output*), the compiler tape will rewind, and the source deck and condensed object deck (if any) will be in the 1402 stackers.

Compiler Output

The following information is obtained, during compilation, at the 1403 printer unless otherwise indicated:

1. Machine core-storage size, specified and actual.
2. The source program listing including an internal sequence (SEQ) number for each statement. SEQ will be referenced by any error diagnosis at either compile or execute time. Each page listed will be identified by the punches in columns 76-80 of the source program cards and by a page number.
3. The number of input characters.
4. The specified modulus (k), equal to the word size, for fixed point (integer) variables.
5. The specified mantissa length (f) for floating point variables. Two extra positions will be reserved for the characteristic. Word size thus equals $f + 2$.
6. Array storage assignment, naming each array with its decimal and machine language boundaries.
7. Simple variable storage assignment, naming each variable with its decimal and machine language (right-hand) address.
8. Constant storage assignment boundaries.
9. Diagnostic messages. See *Compiler Diagnostics*.
10. For each executable statement: the SEQ number, the object time starting address (machine language and decimal) of the generated procedure, and a display code (related to the starting address) which may be used during diagnosis of execution of the object program (see next section).
11. If requested on the control card, and if there have been no errors that would prevent successful execution of the object program, a core storage snapshot will be printed and a condensed deck in condensed Autocoder format, will be punched. The condensed deck listing will be printed if sense switch B is up.
12. The system will halt after printing the message:

    END OF COMPILATION
    PRESS START TO GO

    At this time data tapes and cards may be loaded and the system tape unloaded. Initial object-time sense-switch settings may be made.

Compilation Checking Aid

The core storage snapshot can be forced at various times during compilation by the use of sense switches. Switches C, D, and E all up will cause the snapshot to print after every compiler phase. Because this is usually undesirable, fewer phases can be printed by the use of switches D and E up. These are: DIMEN2, VARBL5, CONST3, LIST3, STNUM5, ARITH6, DO, RESORT 4.

Sense Switch E up will cause printout DIMEN2, STNUM5, DO.

G on will cause a halt after any snapshot. F on will bypass the printout of any snapshot.

Compiler Diagnostics

The following messages will be printed, during compilation, when appropriate:

MACHINE SIZE SPECIFIED IS GREATER THAN ACTUAL MACHINE

SYSTEM DOES NOT FOLLOW END CARD

OBJECT PROGRAM TOO LARGE

NO PARAMETER CARD (Control Card)

ERROR 1	UNDETERMINABLE STATEMENT (SEQ number)
ERROR 2	DOUBLY DEFINED ARRAY (NAME)
ERROR 3	DIMENSION SYNTAX, STATEMENT (SEQ number)
ERROR 4	EQUIVALENCE SYNTAX, STATEMENT (SEQ number)
ERROR 5	ILLEGAL MIXING IN EQUIVALENCE, STATEMENT (SEQ number)
ERROR 6	UNDEFINED ARRAY, STATEMENT (SEQ number) (NAME)
ERROR 7	ILLEGAL EQUIVALENCE, STATEMENT (SEQ number)
ERROR 8	REDUNDANT EQUIVALENCE, STATEMENT (SEQ number)
ERROR 9	VARIABLE SYNTAX, STATEMENT (SEQ number)
ERROR 10	UNDEFINED VARIABLE, STATEMENT (SEQ number)
ERROR 11	UNREFERENCED VARIABLE, STATEMENT (SEQ number)
ERROR 12	FLOATING POINT SUBSCRIPT, STATEMENT (SEQ number)
ERROR 13	STATEMENT NUMBER SYNTAX, STATEMENT (SEQ number)
ERROR 14	UNREFERENCED FORMAT, STATEMENT (SEQ number)
ERROR 15	FORMAT SYNTAX, STATEMENT (SEQ number)
ERROR 16	PARENTHESIS ERROR, STATEMENT (SEQ number)
ERROR 17	DOUBLY DEFINED FORMAT, STATEMENT (SEQ number)
ERROR 18	LIST SYNTAX, STATEMENT (SEQ number)
ERROR 19	UNREFERENCED STATEMENT NUMBER, STATEMENT (SEQ number)
ERROR 20	DOUBLY DEFINED STATEMENT NUMBER, STATEMENT (SEQ number)
ERROR 21	nnn UNDEFINED STATEMENT NUMBER(S), STATEMENT (SEQ number)
ERROR 22	UNDEFINED FORMAT, STATEMENT (SEQ number)
ERROR 23	CODING UNINTELLIGIBLE, STATEMENT (SEQ number)
ERROR 24	SYSTEM ERROR, STATEMENT (SEQ number)
ERROR 25	LEFT SIDE INVALID, STATEMENT (SEQ number)

ERROR 26	EXCESS OF – SIGNS, STATEMENT (SEQ number)
ERROR 27	ARITHMETIC SYNTAX ERROR, STATEMENT (SEQ number)
ERROR 28	INCORRECT MODE OF FUNCTION ARGUMENT, STATEMENT (SEQ number)
ERROR 29	UNDEFINED FUNCTION NAME, STATEMENT (SEQ number)
ERROR 30	FIX TO FLOAT POWER, STATEMENT (SEQ number)
ERROR 31	DOUBLE OPERATORS, STATEMENT (SEQ number)
ERROR 32	MULTIPLE EXPONENT, STATEMENT (SEQ number)
ERROR 33	NO TAPE UNIT NUMBER, STATEMENT (SEQ number)
ERROR 34	COMPUTED GO TO SYNTAX, STATEMENT (SEQ number)
ERROR 35	HALT NUMBER NNNNN TO BE DISPLAYED AS NNN, STATEMENT (SEQ number)
ERROR 36	ILLEGAL SENSE LIGHT, STATEMENT (SEQ number)
ERROR 37	ILLEGAL SENSE SWITCH, STATEMENT (SEQ number)
ERROR 38	ILLEGAL RANGE OF DO, STATEMENT (SEQ number)
ERROR 39	ILLEGAL NESTING, STATEMENT (SEQ number)
ERROR 40	DO SYNTAX ERROR, STATEMENT (SEQ number)
ERROR 41	CONSTANT LEFT SIDE OF EQUAL SIGN, STATEMENT (SEQ number)
ERROR 42	MODULUS
ERROR 43	MANTISSA
ERROR 44	CONSTANT SYNTAX, STATEMENT (SEQ number)
ERROR 45	HOLLERITH COUNT, STATEMENT (SEQ number)
ERROR 46	MIXING IN ARITH, STATEMENT (SEQ number)
ERROR 47	BAD LIST, STATEMENT (SEQ number)

Compilation Time Halt

When running as a tape system, a halt will occur with 3333 displayed in the B-address register if a permanent redundancy is detected on the systems tape. Press Start to reread the record.

Object-Program Storage Allocation

The storage allocation of compiled programs is diagrammed in Figure 7.

The following information will be helpful in the estimation of the size of an object program:

	Floating	*Fixed*
Variable storage word-size, including array members	f + 2	k
Temporary storage word-size	f + 4	k

The relocatable library and processing subroutines appear in the object program only if needed.

	Name	*Approx. Size*
1.	SINF/COSF	437
2.	LOGF (or A**B)	320
3.	EXPF (or A**B)	297
4.	COMMON (if 1 or 2 or 3 above)	263
5.	ATANF	471
6.	SQRTF	216
7.	XFIXF (or I = A)	133
8.	FLOATF (or A = I)	59
9.	NEGATE (–B*C, etc.)	8
10.	ABSF/XABSF (also requires NEGATE)	7
11.	DO	92
12.	LIST	404
13.	DO/LIST COMMON (if 11 or 12 above)	50
14.	SUBSCRIPTS (if variable)	220
		2977

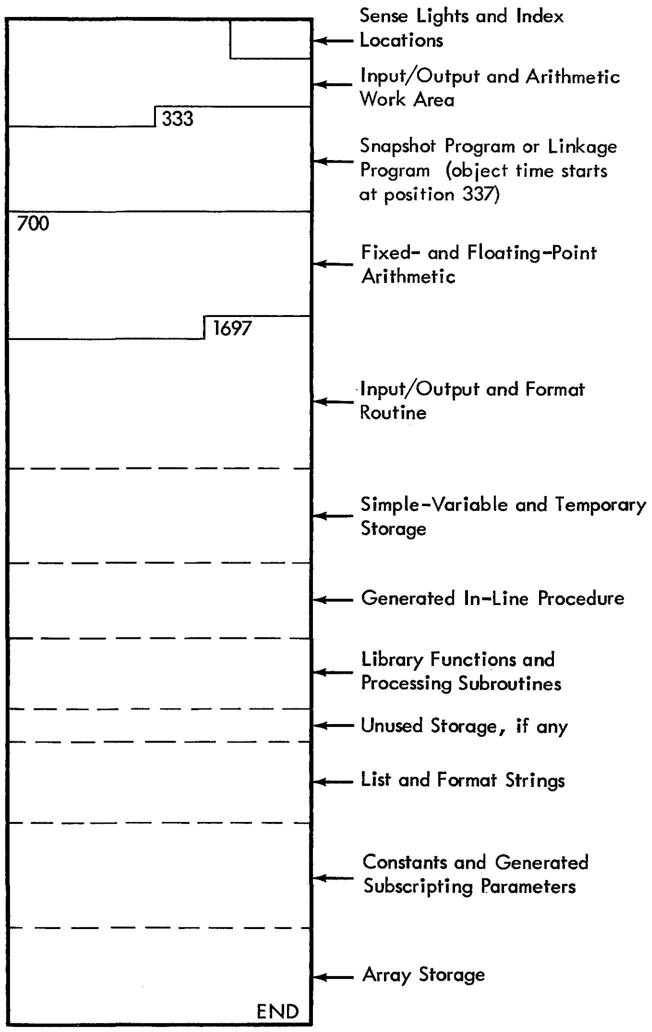

Sense Lights and Index Locations
Input/Output and Arithmetic Work Area
333
Snapshot Program or Linkage Program (object time starts at position 337)
700
Fixed- and Floating-Point Arithmetic
1697
Input/Output and Format Routine
Simple-Variable and Temporary Storage
Generated In-Line Procedure
Library Functions and Processing Subroutines
Unused Storage, if any
List and Format Strings
Constants and Generated Subscripting Parameters
Array Storage
END

Figure 7. Object Program Storage Allocations

The generated in-line procedure, the generated list and format strings, the constants and generated subscripting parameters, all together generate less than twice as many object program characters as are in the source program.

For examples of typical storage allocation, see the sample programs, Figures 8 and 9.

Object Program Operation Procedures

The compiled program may be executed immediately after compilation, while still in core storage, by readying any card or tape problem-data and pressing Start.

Execution of the Condensed Card Deck

To execute the object program at a later time:

1. Ready data tapes, if any.
2. Place the condensed deck in the read hopper.
3. Reset the machine.
4. Set any sense switches required by the source program.
5. Press 1402 Load.
6. Press Start when the reader stops at the last card.
7. Place card data, if any, in the read hopper when required.

Note: Alternatively, card data may be placed in the hopper behind the condensed deck at step 2.

When the compiler is used on tape, data cards, preceded by a card containing a 5 and 8 multipunch in column 1, may follow the source program.

Program Checking Aid

At any point during execution of the object program, a snapshot of core storage can be obtained by executing the snapshot program at position 337. Position 333 of the snapshot routine during compilation is not available at object time. Sense switch F must be off or printing will be suppressed. Programs containing a linkage (XLINK) statement do not have this facility; a transfer of control to 337 causes execution of the linkage program.

When the snapshot program is executed, positions 84-86 will contain one of the display codes printed during compilation. The SEQ number corresponding to this display code identifies the current or most recently executed statement in which an arithmetic expression appeared.

Object Time Halts or Error Conditions

In addition to the halts generated by STOP and PAUSE statements, the object program will contain halts that are invoked by badly coded or positioned data, or by unanticipated values, tape errors, or end of file. When the system halts at object time and the stop light is lit, display the B-address by pressing the B-address register key-light.

One of the following three-digit halt codes may appear:

Code	*Meaning*
581	A permanent read error was encountered on the LIB tape during execution of the linkage routine.
603 (during execution of linkage routine) 342 (during execution of monitor on the LIB tape)	The necessary program segment is not on the LIB tape. Press Start to get the segment from the card reader.
777	A tape error was encountered in the limited input/output format routine.
888	End-of-file was encountered in the limited input/output format routine.
999	Read error on the LIB tape during execution of the monitor program on the LIB tape. Rewind the LIB tape and press TAPE LOAD to try executing the monitor program again. Check to see if the call card is still available in the reader.

One of the following four-digit halt codes may appear on the register:

Code	*Meaning*
1001	The value of the index in a computed GO TO statement exceeds the number of exits.
1111	Parity errors when attempting to read tape, or having skipped and blanked tape 50 times while attempting to write tape. Press Start to continue the attempt.
1121	Data and FORMAT specifications disagree in mode or acceptable characters. Not all disparity is detectable.
2002	The value of a computed subscript is greater than 15,999.
3700	Output record too long because of incorrect FORMAT specifications. Snapshot routine has been destroyed. Press Start to continue execution.
4002	End of file detected while reading tape. Press Start to read next record (or first rewind and unload old tape; load new tape).
4003	End of file while writing tape. A tape mark will be written, and the tape rewound and unloaded. After new tape is mounted, press Start to proceed.

Note: 1. An X will replace an output data field whenever,
a. a fixed-point value has E or F format, *or*
b. insufficient integer space has been allocated to an F format value.
2. If an F format value is negative and the numeric digits exactly fill the output data field, the sign is lost.

A coded message and the display address will be printed in the leftmost positions of the 1403, in case of the following errors. There will be no halt. The program will continue with the indicated result.

Message	*Meaning*	*Result*
NOF	Exponent overflow during normalization	$9\ldots\overset{\pm}{\bar{9}}9\overset{+}{9}$
DZE	Attempted to divide by zero	$9\ldots\overset{\pm}{\bar{9}}9\overset{+}{9}$
LNZ	Attempted to find reciprocal of zero	$9\ldots\overset{\pm}{\bar{9}}9\overset{+}{9}$
EOF	Exponential 10^{99}	$9\ldots\overset{+}{9}9\overset{+}{9}$
LNZ	Logarithm of zero	$9\ldots\bar{9}9\overset{+}{9}$
SCL	Sine or Cosine argument too large	zero
LNN	Logarithm of negative number	ln $\lvert arg \rvert$
ZTZ	Zero to zero power	one
SQN	Square root of negative argument	$\sqrt{\lvert arg \rvert}$

Running Programs Containing Linkage Routine

The following are the procedures for executing a program that consists of more than one section or segment. See *Program Linkage.*

Preparing the Condensed Decks for Execution

The condensed decks of the compiled segments are read into storage (for execution) from cards, tape, or both cards and tape. In the first two cases a single combined deck is required. In the last case the condensed decks to be read from cards are combined into a single deck, and those to be read from tape are combined into a single deck that is loaded on tape.

Reading from Cards

In reading the compiled segments from cards, the condensed decks of the segments are combined into a single deck as follows:

1. Remove the clear-storage cards (first two cards) from all the segments, except the first segment of the program, and any others that require all of core storage to be cleared before being read for execution.
2. Place data cards behind each condensed deck that requires data.
3. Combine the decks (with data cards) into a single deck, in the order they are to be read.

Reading from Tape

In reading the compiled segments from tape for execution, the condensed decks must be combined into a single deck and loaded on tape (LIB tape). The procedures are as follows:

1. Remove the clear-storage cards (first two cards) from each segment.
2. Place the associated title card before the condensed deck of each segment.
3. Combine the condensed decks into a single deck. They need not be in the order they are to be read from tape; however, they should be in that order for efficiency in execution.
4. Load the combined condensed deck on tape using the following procedure:
   a. Place the LIB tape generator (phase 95) in the reader followed by the combined condensed deck (with title cards).
   b. Ready tape unit 1.
   c. Press the load key on the 1402. The LIB tape generator performs the necessary loading operation.

Note: Two halts may occur while writing the LIB tape. When a halt occurs, display the contents of the B-address register. A 195 in the B-address register indicates that a *title card* was searched for in the reader and not found. Put the necessary title card in the reader and press START to read more cards. A 666 in the B-address register indicates that the job is completed.

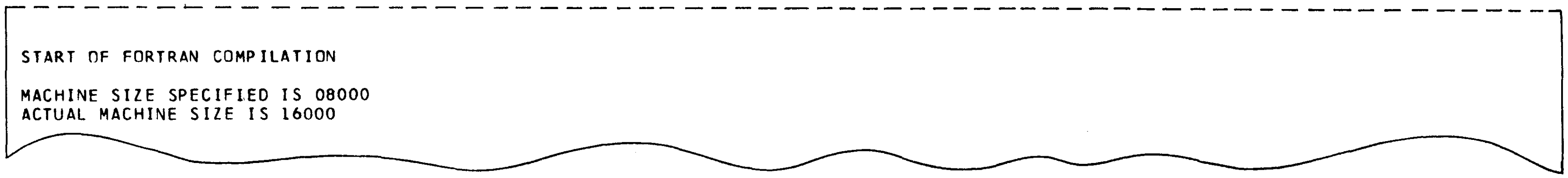
Executing the Segmented Program

To run the segmented object program:

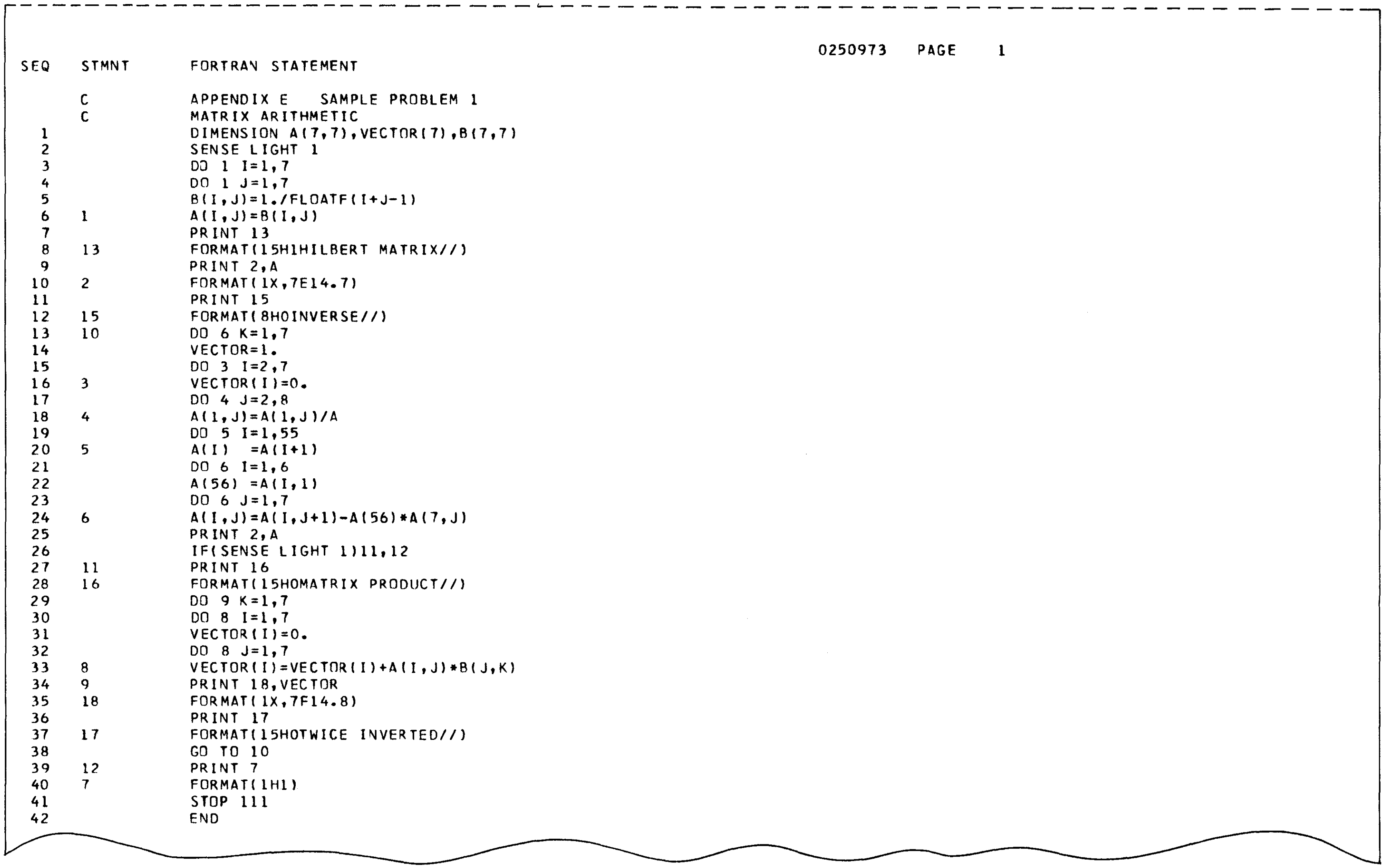
1. If all segments are to be read from cards:
   a. load the 1402 card read-punch with the combined condensed card-decks.
   b. press the load key on the 1402. The program is loaded, and execution begins.
2. If all segments are to be read from tape:
   a. ready the LIB tape on tape unit 1.
   b. clear core storage to blanks unless previously executed routines or data are to remain. This is a precaution against errors resulting from extra group-mark word-marks in core storage.
   c. place the initial call card in the card reader. It should be followed by any card data or call cards required by the entire program, in the order they are to be read.
   d. Press Tape Load.
   e. Turn on sense switch G if a core-storage dump is desired.
   f. Press Start to read the initial call card. The first segment is then read in and execution begins.
3. If segments are to be read from both cards and tape, follow the procedure in item 2, also loading the segments to be read from cards along with the card data (for segments on tape) and call cards, in the order they are to be read.

Sample Programs

Figure 8 illustrates a matrix calculation. Figure 9 illustrates the use of library functions. In each case a core-storage snapshot is specified for the printout. The printout for the second program, however, also includes a listing of the condensed deck.

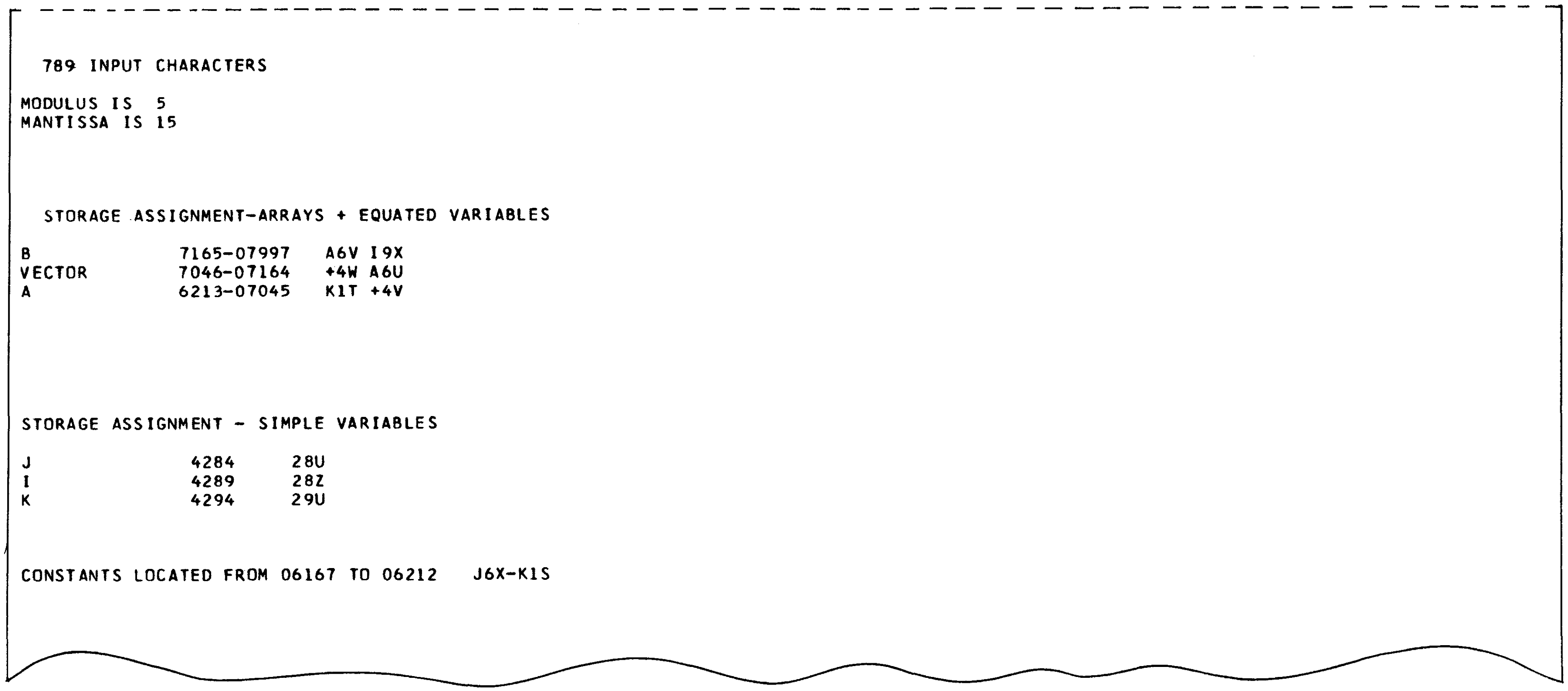


```
START OF FORTRAN COMPILATION

MACHINE SIZE SPECIFIED IS 08000
ACTUAL MACHINE SIZE IS 16000
```



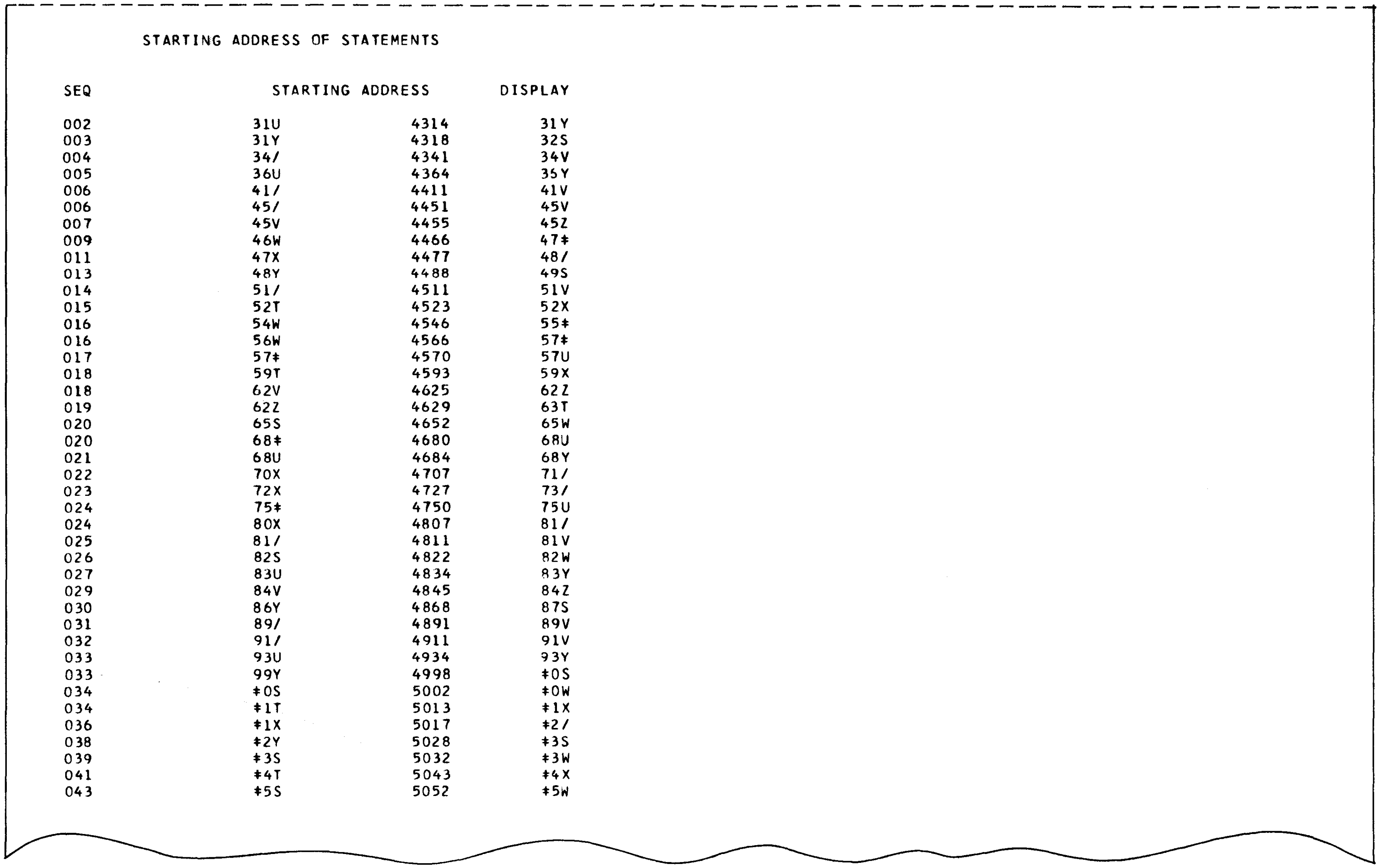
```
                                                                          0250973    PAGE    1
SEQ   STMNT       FORTRAN STATEMENT

      C           APPENDIX E   SAMPLE PROBLEM 1
      C           MATRIX ARITHMETIC
  1               DIMENSION A(7,7),VECTOR(7),B(7,7)
  2               SENSE LIGHT 1
  3               DO 1 I=1,7
  4               DO 1 J=1,7
  5               B(I,J)=1./FLOATF(I+J-1)
  6   1           A(I,J)=B(I,J)
  7               PRINT 13
  8   13          FORMAT(15H1HILBERT MATRIX//)
  9               PRINT 2,A
 10   2           FORMAT(1X,7E14.7)
 11               PRINT 15
 12   15          FORMAT(8H0INVERSE//)
 13   10          DO 6 K=1,7
 14               VECTOR=1.
 15               DO 3 I=2,7
 16   3           VECTOR(I)=0.
 17               DO 4 J=2,8
 18   4           A(1,J)=A(1,J)/A
 19               DO 5 I=1,55
 20   5           A(I)  =A(I+1)
 21               DO 6 I=1,6
 22               A(56) =A(I,1)
 23               DO 6 J=1,7
 24   6           A(I,J)=A(I,J+1)-A(56)*A(7,J)
 25               PRINT 2,A
 26               IF(SENSE LIGHT 1)11,12
 27   11          PRINT 16
 28   16          FORMAT(15H0MATRIX PRODUCT//)
 29               DO 9 K=1,7
 30               DO 8 I=1,7
 31               VECTOR(I)=0.
 32               DO 8 J=1,7
 33   8           VECTOR(I)=VECTOR(I)+A(I,J)*B(J,K)
 34   9           PRINT 18,VECTOR
 35   18          FORMAT(1X,7F14.8)
 36               PRINT 17
 37   17          FORMAT(15H0TWICE INVERTED//)
 38               GO TO 10
 39   12          PRINT 7
 40   7           FORMAT(1H1)
 41               STOP 111
 42               END
```

Figure 8. Matrix Calculation Part 1



```
  789 INPUT CHARACTERS

MODULUS IS  5
MANTISSA IS 15



  STORAGE ASSIGNMENT-ARRAYS + EQUATED VARIABLES

B              7165-07997   A6V I9X
VECTOR         7046-07164   +4W A6U
A              6213-07045   K1T +4V




STORAGE ASSIGNMENT - SIMPLE VARIABLES

J               4284     28U
I               4289     28Z
K               4294     29U



CONSTANTS LOCATED FROM 06167 TO 06212   J6X-K1S
```

Figure 8. Matrix Calculation Part 2



```
          STARTING ADDRESS OF STATEMENTS

SEQ                     STARTING ADDRESS        DISPLAY

002                     31U            4314            31Y
003                     31Y            4318            32S
004                     34/            4341            34V
005                     36U            4364            35Y
006                     41/            4411            41V
006                     45/            4451            45V
007                     45V            4455            45Z
009                     46W            4466            47‡
011                     47X            4477            48/
013                     48Y            4488            49S
014                     51/            4511            51V
015                     52T            4523            52X
016                     54W            4546            55‡
016                     56W            4566            57‡
017                     57‡            4570            57U
018                     59T            4593            59X
018                     62V            4625            62Z
019                     62Z            4629            63T
020                     65S            4652            65W
020                     68‡            4680            68U
021                     68U            4684            68Y
022                     70X            4707            71/
023                     72X            4727            73/
024                     75‡            4750            75U
024                     80X            4807            81/
025                     81/            4811            81V
026                     82S            4822            82W
027                     83U            4834            83Y
029                     84V            4845            84Z
030                     86Y            4868            87S
031                     89/            4891            89V
032                     91/            4911            91V
033                     93U            4934            93Y
033                     99Y            4998            ‡0S
034                     ‡0S            5002            ‡0W
034                     ‡1T            5013            ‡1X
036                     ‡1X            5017            ‡2/
038                     ‡2Y            5028            ‡3S
039                     ‡3S            5032            ‡3W
041                     ‡4T            5043            ‡4X
043                     ‡5S            5052            ‡5W
```

Figure 8. Matrix Calculation Part 3

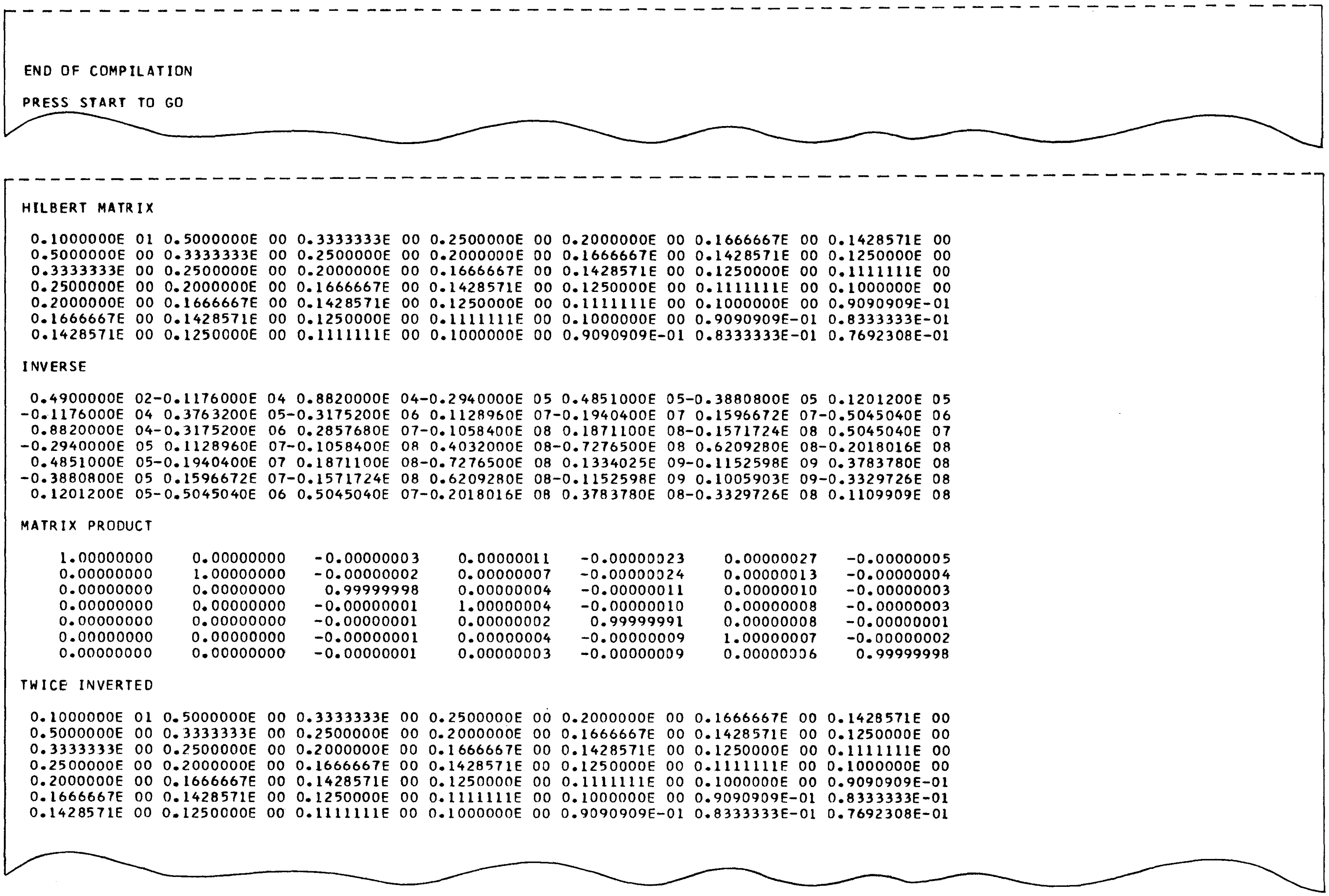
Figure 8. Matrix Calculation Part 4

```
                                  SNAPSHOT OF OBJECT PROGRAM

INPUT/OUTPUT AREAS LOCATED FROM 001-332

FIXED OBJECT TIME ROUTINES LOCATED FROM 333-4279


........09........19........29........39........49........59........69........79........89........99-AREA-04200
9,26XBH610‡4 )26XBH61            5                   0.0 X RW4A281   )0‡1BA3810                     -AREA-04200
 1   1     21    1   1  1  1  1  1 11  1  111  1  1  1  1  111 1   1111  1   1 2

........09........19........29........39........49........59........69........79........89........99-AREA-04300
              )081B‡6/B‡8‡J7TJ7SJ8Y28Z45VB‡6/B‡8‡J7TJ7SJ8Y28U32SB7003JT=2QZ+2QU-JXTF‡$+DV28UJ8S28ZJ8-AREA-04300
              1   1   1   1  1  1  1  1  1   1   1  1  1  1  1  1   1                 1  1  1  1  1

........09........19........29........39........49........59........69........79........89........99-AREA-04400
U$=J7W/31T‡B700$-IT28UJ8S28ZJ8U$=$+DV28UJ8S28ZJ8U$‡B/2‡BW97*Z4SZ4/BW97*Z8‡Z2VBW97*-1SZ4/B‡6/B‡8‡J7TJ-AREA-04400
          11   11  1  1  1  1     1  1  1  1  1   11   1   11  1  1   11  1  1   11  1  1   1   1  1

........09........19........29........39........49........59........69........79........89........99-AREA-04500
7SJ8Y29U81/B700+FS=J7W‡B‡6/B‡8‡J6YJ7SJ8Y28Z57‡B700$+DV28ZJ8U$=J7Z‡B/2‡B‡6/B‡8‡J6YJ6XJ8Y28U62ZB700$JA-AREA-04500
  1  1  1  1   1      11   1   1  1  1  1  1  1   11  1  1       11   1   1   1  1  1  1  1  1   11

........09........19........29........39........49........59........69........79........89........99-AREA-04600
‡28UJ8S$=$JA‡28UJ8S$/KBZ‡B/2‡B‡6/B‡8‡J7TJ7‡J8Y28Z68UB700$KAS28ZJ8U$=$KBZ28ZJ8U$‡B/2‡B‡6/B‡8‡J7TJ7/J8-AREA-04600
 1  1     1  1  1       11   1   1   1  1  1  1  1  1   11  1  1     1  1  1   11   1   1   1  1  1

........09........19........29........39........49........59........69........79........89........99-AREA-04700
Y28Z49SB700AFU=$KAS28ZJ8U$‡B‡6/B‡8‡J7TJ7SJ8Y28U68YB700$-IT28UJ8S28ZJ8U$=AFU*$KAS28UJ8S$N+$KAS28UJ8S2-AREA-04700
 1  1  1   1    1  1  1   11   1   1  1  1  1  1  1   11  1  1  1  1         1  1  1      1  1  1  1

........09........19........29........39........49........59........69........79........89........99-AREA-04800
8ZJ8U$‡B/2‡BW97*Z8‡Z2VV‡3S0811,081BW97*-4TZ4/B‡6/B‡8‡J7TJ7SJ8Y29U‡1XB‡6/B‡8‡J7TJ7SJ8Y28Z‡0SB700$+DV2-AREA-04800
  1   11   1   11  1  1       1   1   11  1  1    1  1  1  1  1  1  1   1   1  1  1  1  1  1   11  1

........09........19........29........39........49........59........69........79........89........99-AREA-04900
8ZJ8U$=J7Z‡B‡6/B‡8‡J7TJ7SJ8Y28U87SB700$+DV28ZJ8U$=$-IT28UJ8S28ZJ8U$*$+DV29UJ8S28UJ8U$+$+DV28ZJ8U$‡B/-AREA-04900
  1       11   1   1  1  1  1  1  1   11  1  1     1  1  1  1  1     1  1  1  1  1     1  1  1   11

........09........19........29........39........49........59........69........79........89........99-AREA-05000
2‡BW97*-8/Z3TB84ZBW97*J1TZ4/B48YBW97*J5/Z4/N111.B‡4TN000.B‡5SHS0SH094H0940-4B/5XH094M0J1/2WMM0-5/3‡M-AREA-05000
  1   11   1 1   1   11  1  1   1   11  1  1   11   1   11   1   1   1      1   1   1      11      1

........09........19........29........39........49........59........69........79........89........99-AREA-05100
0J1/3XM0J4/4UH/5S0J5A000000+000J9TS000J9TV000J9TKB000HS0SM0-2/6X+000J9TM0-8/8/S000J9TM0J1/9YLJ9T000B-AREA-05100
      1      1      1      1      1      1       1   1   1      1      1      1      1      1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05200
000H094HS9W0-2HT0‡0-3M0-2V9XMV9X094VT0X0-01BT2V0-0,BU5W0-0$BU7Y0-0(BV3X0-0)BV0/0-0=MW0‡089MV9X000B00-AREA-05200
   1   1      1      1      1      1       1       1       1       1       1       1      1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05300
0XXXXXXM0-2089HV9X0-3BS9‡YJ36T3ZBU3/W0S2BVT8XT0S2M0-6T0WD690U0‡DVT8‡T0SKD692U0‡DYT8ST0SMT0T089H0890‡-AREA-05300
 1     1      1      1   1      1       11       1      1      11       1      11      1      1
```

```
........09........19........29........39........49........59........69........79........89........99-AREA-05400
0M089T0TCT0WT0TBS9‡/YU2‡T0SBU4VM0-6T0WMW0T089HV9X0-7BS9‡BW0XYU6S088M094V9XBS9‡H0940-1B/5TD0-0H094BV0-AREA-05400
 1      1      1    1      1   1      1      1      1   1   1      1      1   1      1   1   1   1

........09........19........29........39........49........59........69........79........89........99-AREA-05500
YM0-3094M0J2V5‡MM0-6V5UM0J2V6/HW0W0-0MW0W094A000000+000J9TS000J9TVV8UJ9TKHV9X0J6BS2YM0J5V9XBS2Y     -AREA-05500
 1      1      11      1      1      1      1      1      1      1       1      1   1      1   1  1

........09........19........29........39........49........59........69........79........89........99-AREA-05600
 2S.   HX8VM0-3Y0/SY0WBW2XZM0-9W5/M0-6W4U+000Y1/L000Y1W'Y1/Y2SAY2SY0WCY0TY2UBX8WTBX0‡0J0$H0940-6BW2X-AREA-05600
 1 11  1   1      1   1    1      1      1      1      1      1      1      1    1       1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05700
AY2WY0TBX0‡ZYY0SY0W+Y0U090YWZZY0UMY0W089M089X6XYX9WX6WMY0/089H0890‡0YY0‡088H0940J1B000V-02.BX8W2SK3 -AREA-05700
1      1    1      1      1      1      1      1      1      1      1      1      1   1   11   1   1

........09........19........29........39........49........59........69........79........89........99-AREA-05800
                       159F-2G9Y2G9‡87B/34)W87VY5Y0-41BY6VD2G9W84H094250L2G9M692Y8ZH099000+099W82B‡1-AREA-05800
  1    1    1          1 1 1   1      1   1   1       1   1      1      1   1      1      1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05900
0BY2X0-4NBY4S0-4F       ‡,KAS+4V.,+DVA6U..BJ5200ABL281HILBERT MATRIXBK08BK08BK23BJ5200AH0990+1BL85E0-AREA-05900
 1       1       11       1       1       1   1  1   1              1   1   1   1   1  1      1   11

........09........19........29........39........49........59........69........79........89........99-AREA-06000
0G003011BK23BJ5200ABL280INVERSEBK08BK08BK23BJ5200ABL280MATRIX PRODUCTBK08BK08BK23BJ5200AH0990+1BL85F-AREA-06000
  1  1  1   1   1  1   1       1   1   1   1   1  1   1              1   1   1   1   1  1      1   1

........09........19........29........39........49........59........69........79........89........99-AREA-06100
00G006008BK23BJ5200ABL280TWICE INVERTEDBK08BK08BK23BJ5200ABL281BK238255671A0A+0+11917A0+1           -AREA-06100
1  1  1  1   1   1  1   1              1   1   1   1   1  1   11   111 1111  1  1  1 1  11    1

........09........19........29........39........49........59........69........79........89........99-AREA-06200
                                                                                                    -AREA-06200
```

```
........09........19........29........39........49........59........69........79........89........99-AREA-07500
                                                                                                    -AREA-07600

........09........19........29........39........49........59........69........79........89........99-AREA-07700
                                                                                                    -AREA-07700

........09........19........29........39........49........59........69........79........89........99-AREA-07800
                                                                                                    -AREA-07800

........09........19........29........39........49........59........69........79........89........99-AREA-07900
                                                                                                    -AREA-07900
```

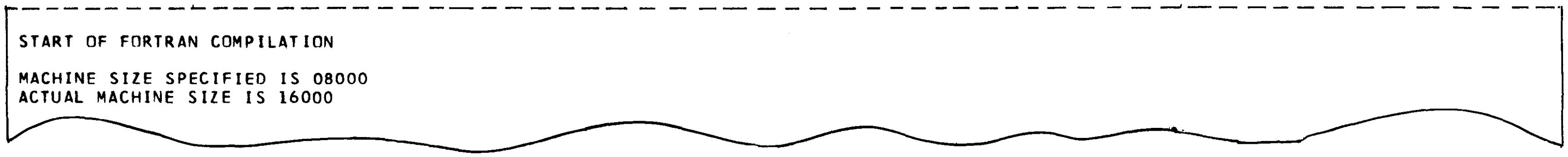
Figure 8. Matrix Calculation Part 5



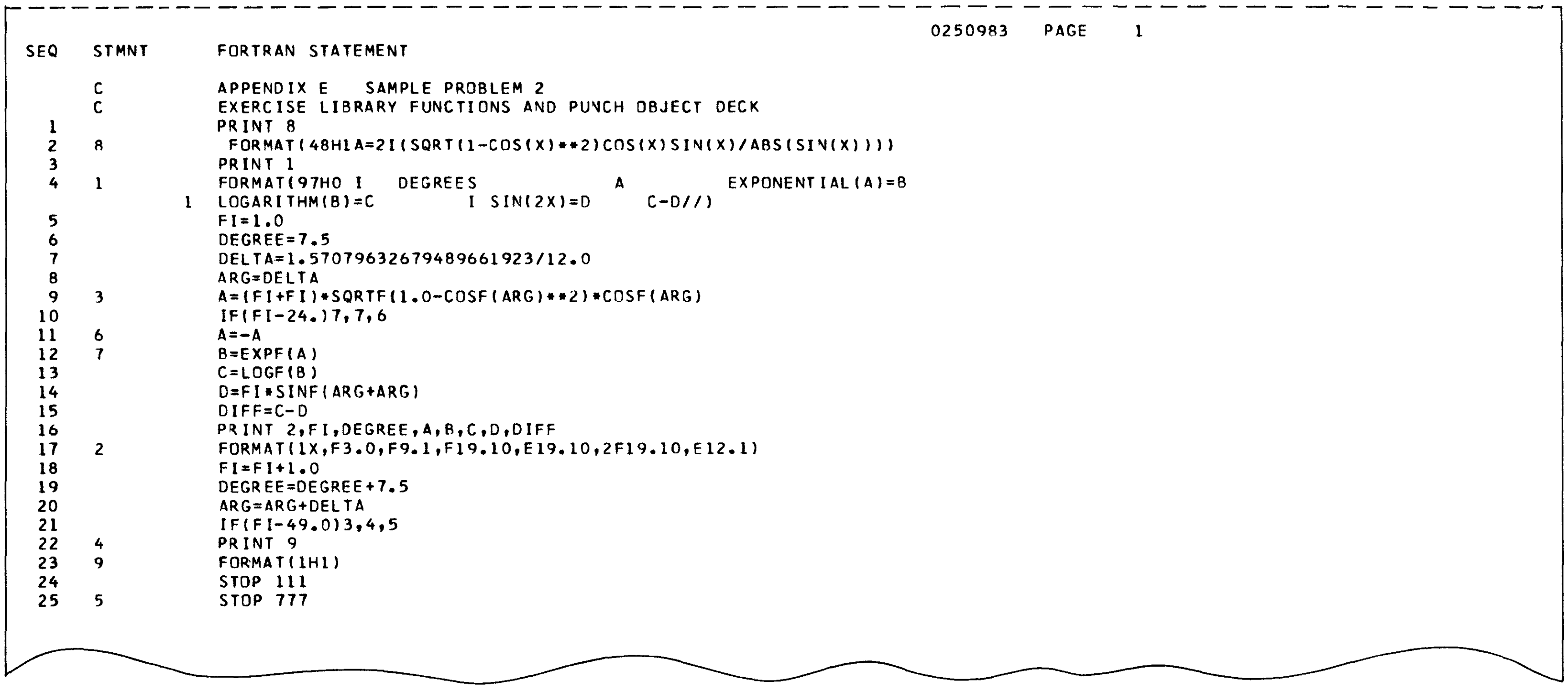
```
END OF COMPILATION

PRESS START TO GO
```

```
HILBERT MATRIX

 0.1000000E 01 0.5000000E 00 0.3333333E 00 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00
 0.5000000E 00 0.3333333E 00 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00
 0.3333333E 00 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00
 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00
 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00 0.9090909E-01
 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00 0.9090909E-01 0.8333333E-01
 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00 0.9090909E-01 0.8333333E-01 0.7692308E-01

INVERSE

 0.4900000E 02-0.1176000E 04 0.8820000E 04-0.2940000E 05 0.4851000E 05-0.3880800E 05 0.1201200E 05
-0.1176000E 04 0.3763200E 05-0.3175200E 06 0.1128960E 07-0.1940400E 07 0.1596672E 07-0.5045040E 06
 0.8820000E 04-0.3175200E 06 0.2857680E 07-0.1058400E 08 0.1871100E 08-0.1571724E 08 0.5045040E 07
-0.2940000E 05 0.1128960E 07-0.1058400E 08 0.4032000E 08-0.7276500E 08 0.6209280E 08-0.2018016E 08
 0.4851000E 05-0.1940400E 07 0.1871100E 08-0.7276500E 08 0.1334025E 09-0.1152598E 09 0.3783780E 08
-0.3880800E 05 0.1596672E 07-0.1571724E 08 0.6209280E 08-0.1152598E 09 0.1005903E 09-0.3329726E 08
 0.1201200E 05-0.5045040E 06 0.5045040E 07-0.2018016E 08 0.3783780E 08-0.3329726E 08 0.1109909E 08

MATRIX PRODUCT

      1.00000000     0.00000000    -0.00000003     0.00000011    -0.00000023     0.00000027    -0.00000005
      0.00000000     1.00000000    -0.00000002     0.00000007    -0.00000024     0.00000013    -0.00000004
      0.00000000     0.00000000     0.99999998     0.00000004    -0.00000011     0.00000010    -0.00000003
      0.00000000     0.00000000    -0.00000001     1.00000004    -0.00000010     0.00000008    -0.00000003
      0.00000000     0.00000000    -0.00000001     0.00000002     0.99999991     0.00000008    -0.00000001
      0.00000000     0.00000000    -0.00000001     0.00000004    -0.00000009     1.00000007    -0.00000002
      0.00000000     0.00000000    -0.00000001     0.00000003    -0.00000009     0.00000006     0.99999998

TWICE INVERTED

 0.1000000E 01 0.5000000E 00 0.3333333E 00 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00
 0.5000000E 00 0.3333333E 00 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00
 0.3333333E 00 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00
 0.2500000E 00 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00
 0.2000000E 00 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00 0.9090909E-01
 0.1666667E 00 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00 0.9090909E-01 0.8333333E-01
 0.1428571E 00 0.1250000E 00 0.1111111E 00 0.1000000E 00 0.9090909E-01 0.8333333E-01 0.7692308E-01
```

Figure 8. Matrix Calculation Part 6

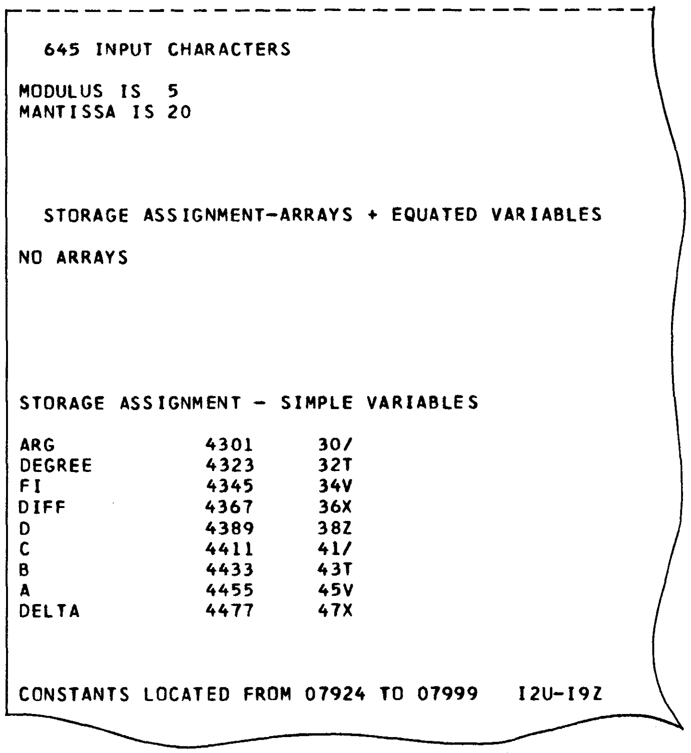


```
START OF FORTRAN COMPILATION

MACHINE SIZE SPECIFIED IS 08000
ACTUAL MACHINE SIZE IS 16000
```

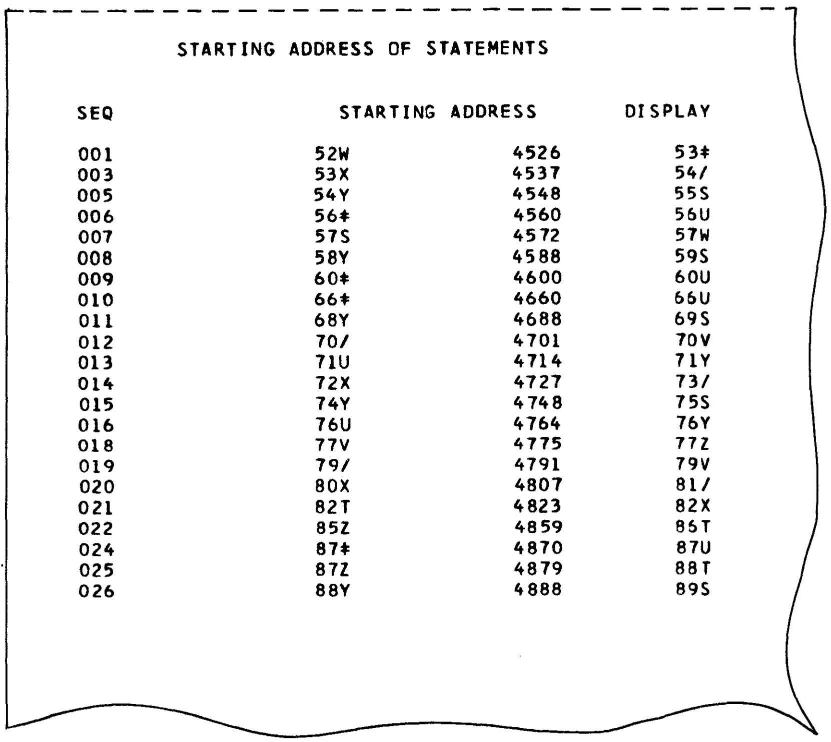


```
                                                                          0250983    PAGE    1
 SEQ   STMNT      FORTRAN STATEMENT

        C         APPENDIX E   SAMPLE PROBLEM 2
        C         EXERCISE LIBRARY FUNCTIONS AND PUNCH OBJECT DECK
  1               PRINT 8
  2     8          FORMAT(48H1A=2I(SQRT(1-COS(X)**2)COS(X)SIN(X)/ABS(SIN(X))))
  3               PRINT 1
  4     1         FORMAT(97H0 I   DEGREES             A          EXPONENTIAL(A)=B
               1  LOGARITHM(B)=C         I SIN(2X)=D     C-D//)
  5               FI=1.0
  6               DEGREE=7.5
  7               DELTA=1.57079632679489661923/12.0
  8               ARG=DELTA
  9     3         A=(FI+FI)*SQRTF(1.0-COSF(ARG)**2)*COSF(ARG)
 10               IF(FI-24.)7,7,6
 11     6         A=-A
 12     7         B=EXPF(A)
 13               C=LOGF(B)
 14               D=FI*SINF(ARG+ARG)
 15               DIFF=C-D
 16               PRINT 2,FI,DEGREE,A,B,C,D,DIFF
 17     2         FORMAT(1X,F3.0,F9.1,F19.10,E19.10,2F19.10,E12.1)
 18               FI=FI+1.0
 19               DEGREE=DEGREE+7.5
 20               ARG=ARG+DELTA
 21               IF(FI-49.0)3,4,5
 22     4         PRINT 9
 23     9         FORMAT(1H1)
 24               STOP 111
 25     5         STOP 777
```

Figure 9. Use of Library Functions Part 1



```
  645 INPUT CHARACTERS

MODULUS IS  5
MANTISSA IS 20



  STORAGE ASSIGNMENT-ARRAYS + EQUATED VARIABLES

NO ARRAYS




STORAGE ASSIGNMENT - SIMPLE VARIABLES

ARG             4301     30/
DEGREE          4323     32T
FI              4345     34V
DIFF            4367     36X
D               4389     38Z
C               4411     41/
B               4433     43T
A               4455     45V
DELTA           4477     47X


CONSTANTS LOCATED FROM 07924 TO 07999   I2U-I9Z
```



```
              STARTING ADDRESS OF STATEMENTS


   SEQ                STARTING ADDRESS        DISPLAY

   001               52W            4526          53‡
   003               53X            4537          54/
   005               54Y            4548          55S
   006               56‡            4560          56U
   007               57S            4572          57W
   008               58Y            4588          59S
   009               60‡            4600          60U
   010               66‡            4660          66U
   011               68Y            4688          69S
   012               70/            4701          70V
   013               71U            4714          71Y
   014               72X            4727          73/
   015               74Y            4748          75S
   016               76U            4764          76Y
   018               77V            4775          77Z
   019               79/            4791          79V
   020               80X            4807          81/
   021               82T            4823          82X
   022               85Z            4859          86T
   024               87‡            4870          87U
   025               87Z            4879          88T
   026               88Y            4888          89S
```

Figure 9. Use of Library Functions Part 2

```
                                      SNAPSHOT OF OBJECT PROGRAM

INPUT/OUTPUT AREAS LOCATED FROM 001-332

FIXED OBJECT TIME ROUTINES LOCATED FROM 333-4279


........09........19........29........39........49........59........69........79........89........99-AREA-04200
9,26XBH610‡4 )26XBH61            5                   0.0 X RW4A281   )0‡1BA3810                     -AREA-04200
 1   1      21   1   1  1  1  1  1 11  1  111  1  1  1  1  111 1  1111   1   1 2

........09........19........29........39........49........59........69........79........89........99-AREA-04300
                                                                                                    -AREA-04300


........09........19........29........39........49........59........69........79........89........99-AREA-04400
                                                                                                    -AREA-04400


........09........19........29........39........49........59........69........79........89........99-AREA-04500
                          BW97*F2TF2SBW97*F8WF2SB7003DV=I6S‡B7003BT=I6W‡B7004GX=I5Y/I3V‡B7003+/=4GX‡-AREA-04500
                          1   11  1  1   11  1  1   1      11   1      11   1          11   1      1

........09........19........29........39........49........59........69........79........89........99-AREA-04600
B7005+/=3+/C‡52V=50/*50/N+I6SQ‡5+/=3+/C‡4EV=3DV+3DV*52V*50/‡B70050/=3DV-I2X‡V68Y2G7BB70/B7004EV=4EVN-AREA-04600
1   1                                                      11   1          11       1   1   1

........09........19........29........39........49........59........69........79........89........99-AREA-04700
‡B7004CT=4EVE‡B7004A/=4CTG‡B7003HZ=3+/+3+/S*3DV‡B7003FX=4A/-3HZ‡BW97*HOWF0‡B7003DV=3DV+I6S‡B7003BT=3-AREA-04700
11   1       11   1       11   1               11   1          11   11  1  1   1          11   1

........09........19........29........39........49........59........69........79........89........99-AREA-04800
BT+I6W‡B7003+/=3+/+4GX‡B70050/=3DV-I3/‡B85Z2800V87Z2G7BB60‡BW97*I0YF2SN111.B87‡N777.B87ZN000.B88YH94-AREA-04800
      11   1          11   1          11       1       1   1   11  1  1   11   1   11   1   11   1

........09........19........29........39........49........59........69........79........89........99-AREA-04900
WM0-291/+000I7VM0-892VS000I7VM0J194SLI7V000B000H094H‡4‡0-2H‡4U0-3M0-2T4/MT4/094V‡5/0-01B‡6Z0-0,BS0‡0-AREA-04900
 1      1      1      1      1      1      1   1   1      1      1      1      1       1       1

........09........19........29........39........49........59........69........79........89........99-AREA-05000
-0$BS2S0-0(BS8/0-0)BS4V0-0=MT4U089MT4/000B000XXXXXXM0-2089HT4/0-3B‡3UYJ36‡8TB/7VT4W2BV/3/‡4W2M0-6‡5‡-AREA-05000
   1       1       1       1      1      1   1     1      1      1   1      1       11       1

........09........19........29........39........49........59........69........79........89........99-AREA-05100
D690/4UDV/2U‡4WKD692/4UDY/2W‡4WM‡4X089H0890‡0M089‡4XC‡5‡‡4XB‡3U/Y/6U‡4WB/8ZM0-6‡5‡MT4X089HT4/0-7B‡3U-AREA-05100
1      11       1      11      1      1      1      1      1    1      1   1      1      1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05200
B   YS0W088M094T4/B‡3UH0940-1B89XD0-0H094BS5SM0-3094M0J2S9UMM0-6S9YM0J2T0VHT5‡0-0MT5‡094A000000+000I-AREA-05200
1   1      1      1   1      1   1   1   1   1      1      11      1      1      1      1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05300
7VS000I7VVT2YI7VKHT4/0J6B97SM0J5T4/B97S      2S.   HT9XM2G9250DLW1SSW1S2M9(0‡0250D2M92G9MDQ089B000HV-AREA-05300
  1      1       1      1   1      1   1  1  1 11  1   1      11   1      1      1      111   1   1
```

Figure 9. Use of Library Functions Part 3

```
........09........19........29........39........49........59........69........79........89........99-AREA-05400
4/)V0‡V2SS2+2)280),S096H094250Z250,0‡0BV3U0-0 Y2510-1A0-12+2AW1‡095C094089BV3UUAW0/W0UAW0UW0XN000251-AREA-05400
  1      1   1   111   1      1   1   1       1      1      1      1      1    1      1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05500
+'2G92E4Y2E4255N255000+(W0X0‡4BU3‡,280B000S2G9DW1T280YW1/2G9YT9YW1/+W1TW79B7552302585092994045684017-AREA-05500
11      1      1      11      1   1   1   1   1      1      1      1      1   1

........09........19........29........39........49........59........69........79........89........99-AREA-05600
9          A+ABW3Y924CB/42SY2G9Y3XBW5‡BV4SSY-4SY3XM-4T-4X+W79W82S-4SW79VZ1ZW79<A-4UW79S099W79V-0UW79-AREA-05600
 11  1  1  1111       1    1      1   1    1      1      1      1      1       1      1      1

........09........19........29........39........49........59........69........79........89........99-AREA-05700
B+W82W79H089-AZ+W80095BT5/+0‡1095S-4W095VX3T095BBX6T924CH0940-1Y-JU‡87D-JU-4XSW1‡SW79BY5S-4X2+2G92+2-AREA-05700
 1      1      1      1   1      1      1       1       1      1      1      1   1   1       1

........09........19........29........39........49........59........69........79........89........99-AREA-05800
BY7Y+2+2249+251--4UW0U+-4YW0/SW0XBT9Y+‡87H0942+1B‡24BY7YD-4S0‡0--4ZW0USW79BY2SHZ1YM2G9250H089L-5‡'2G-AREA-05800
1   1      1   1      1      1   1   1   1      1   1   1      1      1   1   1   1      1   1   1

........09........19........29........39........49........59........69........79........89........99-AREA-05900
92E1-2522G9S251B000A099W79VZ8VW79K+W82W79Y2G92G8+2G82G9AW82-W83W1‡Y-4S‡87BY5S924CBX9TBV4S924C+W82W79-AREA-05900
    1      1   1   1      1       1      1      1      1   1      1      1       1   1       1

........09........19........29........39........49........59........69........79........89........99-AREA-06000
B/34BU71SCLB/42AKJBA1570796326794896619231A B4+ HF0Y2G9W84BL0/SVK7V2G9KC2482G9Q089,0‡0DL6W0‡2C281L6Y-AREA-06000
1   1   1  1   1    1                     1111 11111      1    1       1      1   1   1      1

........09........19........29........39........49........59........69........79........89........99-AREA-06100
BJ1ZU+2G82G9AL6WW79S2G9249-249L6VSL6WW79AL6W280SL6Z2E0(2G9249L249L6U'L6U2E1+2492G9+L6U251+L3YW1‡+++,-AREA-06100
1    1      1      1      1      1      1      1      1      1      1      1      1      1      1111

........09........19........29........39........49........59........69........79........89........99-AREA-06200
V0‡V2SHU9WL6UHV2/HV4/K2YBU0ZA2+2YL6V2+1+VGY248'W79252A2+1252H094252SW82B‡10VK9‡0-41BU71LNNYL3Y2G9B-7-AREA-06200
      1      1   1      1   1   1      1      1      1      1      1   1   1       1   1  1      1

........09........19........29........39........49........59........69........79........89........99-AREA-06300
/VL2X0-41BU71LNZYL7‡2G9B/15)W84BV4S+BA+                           A31+-Y06/‡87DW84L9SBM1/07‡0BBBBYW8-AREA-06300
 1       1   1  1      1   1   1   11111                         111 111      1      1       11111

........09........19........29........39........49........59........69........79........89........99-AREA-06400
4‡87D07/W84VM5‡W841,W84-2G9BM9S/BU71ZTZY‡87W1/BV4SBM3ZS+W79W82S06/W82V01WW82KS06VW82VN1/W82KV/422G9K-AREA-06400
    1      1       1   1   1    1   1  1      1   1    1      1      1       1      1       1

........09........19........29........39........49........59........69........79........89........99-AREA-06500
BU71E0FB/15H089VGX+W80095BT5/C0‡006UBM9ST+0‡0W79Y2G9W79SW1‡C2502G9Q089,0‡0S251D06/0‡0+06SW0X++BT9YY2-AREA-06500
1   1  1   1      1      1   1      1    1      1      1   1      1   1   1   1      1      111   1

........09........19........29........39........49........59........69........79........89........99-AREA-06600
+22+1H0942+1B‡24A099W82VM3ZW82K-W80W1‡SW79Y2G92G8+2G82G9BN5Z0A09IC024688-2G9Y2G9‡87B/34VQ7U2G9KD2G92-AREA-06600
     1      1   1      1       1      1   1      1      1   111  11    11   1      1   1       1

........09........19........29........39........49........59........69........79........89........99-AREA-06700
B2M,H089LQ9W201H094+W80283AQ9W282'Q9Y285D283W79DBP6X2840H0890‡1BP8SVP8SW79BAQ9WW79S2G9SR0‡0-2,0K1)H0-AREA-06700
  111   1      1   1      1      1      1      11       1      1   1       1      1   1      1   11

........09........19........29........39........49........59........69........79........89........99-AREA-06800
940-1-Q9WR0/AQ9WR0/AR0S0-1S0-10‡2VQ1S0‡2BA0-10‡2DR0/0P9H0890‡2VP8W0P92B/34BU71SQNYQ9W2G9/278B09VA5+1-AREA-06800
     1      1      1      1      1       1      1      1      1       1   1   1  1      1   1   11 1
```

Figure 9. Use of Library Functions Part 4

```
........09........19........29........39........49........59........69........79........89........99-AREA-06900
A BBW1UO-4SBW1UO-4CB-5/0-4GBL7/0-4EBD7SO-4NBD8XO-4Q                                                 -AREA-06900
 111       1       1       1       1       1       11

........09........19........29........39........49........59........69........79........89........99-AREA-07000
                                                                                                    -AREA-07000

........09........19........29........39........49........59........69........79........89........99-AREA-07100
                                                                                                    -AREA-07100

........09........19........29........39........49........59........69........79........89........99-AREA-07200
                                                                                                    -AREA-07200

........09........19........29........39........49........59........69........79........89........99-AREA-07300
                                                                                                    -AREA-07300

........09........19........29........39........49........59........69........79........89........99-AREA-07400
                                                                                                    -AREA-07400

........09........19........29........39........49........59........69........79........89........99-AREA-07500
                                                                                                   ‡-AREA-07500

........09........19........29........39........49........59........69........79........89........99-AREA-07600
3DV3BT4EV4CT4A/3HZ3FX..BJ5200ABL281A=2I(SQRT(1-COS(X)**2)COS(X)SIN(X)/ABS(SIN(X)))BK23BJ5200ABL280 I-AREA-07600
1  1  1  1  1  1  1    1   1  1   1                                               1   1   1  1   1

........09........19........29........39........49........59........69........79........89........99-AREA-07700
   DEGREES            A         EXPONENTIAL(A)=B     LOGARITHM(B)=C        I SIN(2X)=D     C-DBK08BK-AREA-07700
                                                                                              1   1

........09........19........29........39........49........59........69........79........89........99-AREA-07800
08BK23BJ5200AH0990+1BL85F00A003000BL85F00A008001BL85F00A009010BL85E00A005014BL85F00B009010BL85E00A00-AREA-07800
  1   1   1  1      1   11  1  1  1   11  1  1  1   11  1  1  1   11  1  1  1   11  1  1  1   11  1

........09........19........29........39........49........59........69........79........89........99-AREA-07900
7005BK23BJ5200ABL281BK232D0B4I0B1B0B15707963267948966192C0A2A0A7E0AA0+1                             -AREA-07900
 1  1   1   1  1   11   1   1   1   1                      11  1   1  11    1                     3
```

Figure 9. Use of Library Functions Part 5

```
                                        CONDENSED DECK

,008015,022026,030037,044,049,053034,035036N00001026                   000150983
L068116,105106,110117B101/I9ZH029NNNC029056B026/B001/0991,001/00111710+000250983
,008015,022029,036040,047054,061068,072/061039              ,0010011040000350983
0000000000000                          L014100,092097,081082,0830841040000450983
I9Z0522                                L008693,689691,693040,0400401040000550983
H567H408M661656M099415M089422H089001   L036368,337341,348355,3620401040000650983
H099202/332/N110210B621FF1M094250      L033401,376380,381388,3933951040000750983
H216000H256000H2440002FK+662664/332    L035436,409416,423424,4264331040000850983
/FJM658306MH465M665668M668000          L029465,438440,447448,4524591040000950983
M651H465A667669V4596682A6706562,0+0    L035500,470474,481489,4964971040001050983
M0‡00+0V5200‡01)0+0C089688B568/22)     L034534,508516,520527,5325331040001150983
M422089M415099/332/B563GB564.          L029563,542549,553554,5595631040001250983
.000H0890‡1B6320972H09920122)A670664   L036599,568575,583590,5915931040001350983
C664672B433/S664F4331M6802202535A670099L039638,607512,616621,6286321040001450983
B4979........9-00000000B               L026664,643652,654659,6626631040001550983
9I0  A1EEXECUTED                       L016680,666668,670671,6730401040001650983
0000000000000                          L014100,092097,040040,0400401040001750983
H094H086HV06M0-2089Q765HS27BS060-0$    L035734,704708,712719,7237271040001850983
HT750‡0/303//LW85280S091H094000        L031765,742746,747748,7557591040001950983
C0-4W86M0-4924,201BT05TH8740-4B/990-5$ L038803,773780,784789,7960401040002050983
M0-7089Q765VV30088KVV30088SH099000)W87 L038841,811815,823831,8380401040002150983
M0‡0W82Q089M0‡0250H094LW85V8830001     L034875,849853,860864,8680401040002250983
Y250‡87SW852E2C0-1W85A099094BS33924/   L036911,883890,897904,0400401040002350983
BS62924*S924+‡87B‡172800B/34SSW82W79   L036947,920924,928936,9410401040002450983
+W80090C099089V/65W79KB/88UAW79W82     L034981,955962,970975,0400401040002550983
+2502V0+100090Y‡870-0A2X90-0Y0-0‡87    L035‡16,989996,‡03‡10,0400401040002650983
+W82W79MW750-1YYADH089                 L022‡38,‡24‡31,‡32‡33,‡34‡351040002750983
S2H1B/420‡2‡H089B‡430‡10P0‡1280S099094)L039‡77,‡43‡51,‡55‡63,‡70‡771040002850983
)SS089W79+2G9,B/34W770V/42W79K         L030/07,‡79‡80,‡87‡91,‡92/001040002950983
BU71NOF+W89W79DW892G9MM2G8/278         L030/37,/12/15,/22/29,/30/341040003050983
B755SW79S2G9B/34BU71DZEB/15            L027/64,/42/46,/50/54,/58/611040003150983
B‡17US100090Y2G92X9B996AW82W79B/34     L034/98,/70/77,/84/88,/950401040003250983
H0940-5B000D0-0DDDQ765                 L022S20,S06S10,S14S15,S16S171040003350983
B723000$B815B/54SD2G90-1MD(0‡0251      L033S53,S29S33,S38S45,S46S471040003450983
-W82BS83'2G92E1H0940-3SW90W79AW82W79   L036S89,S58S62,S69S76,S830401040003550983
Y2G9S97+‡87B‡24BT310-4‡H7650-1C280W85  L037T26,S97T01,T05T13,T200401040003650983
B000BT692800VT69W871VT920-41H0990+2    L035T61,T31T39,T47T55,0400401040003750983
PW782G8L2G9000V0-50-41Q094B712AW912G8  L037T98,T69T76,T84T88,T920401040003850983
VU18280SY2G92G7BT62AW92W79BU48W771S2G9 L038U36,U07U14,U18U25,U330401040003950983
LW93280BU07DW892G9MM2G8SW92W79BU07     L034U70,U44U48,U55U56,U60U671040004050983
HU92/2+2HV250+0H099000M0+2212H2170002  L037V07,U75U79,U86U93,V00V071040004150983
,201HV290+3H099000B000H099000,W87      L033V40,V12V19,V26V30,V370401040004250983
Z0‡0250BW23924/BV98924*VV87924KA0‡02G9 L038V78,V48V56,V64V72,0400401040004350983
+2G9B/34S0‡02G9BV79L0‡0250'2G92E1      L033W11,V83V87,V94V98,W050401040004450983
M2E12G9B/34B/54250 M0‡02E0DHW64L2G9    L035W46,W19W23,W31W38,W39W431040004550983
+2G92E0(0‡0251M2492G9B/34000‡0         L030W76,W54W61,W68W72,W75W761040004650983
000‡00080= 9I                          L013W89,W81W84,W85W86,W87W881040004750983
BEA1                                   L004W93,W91W92,W93040,0400401040004850983
0                                      L002W96,040040,040040,0400401040004950983
05                                     M002V36,040040,040040,0400401040005050983
22                                     M002837,040040,040040,0400401040005150983
ROT                                    M003T30,040040,040040,0400401040005250983
                                       M003S09,040040,040040,0400401040005350983
BW97*F2TF2SBW97*F8W                    L01954U,53‡53/,53U53X,54/54S1040005450983
```

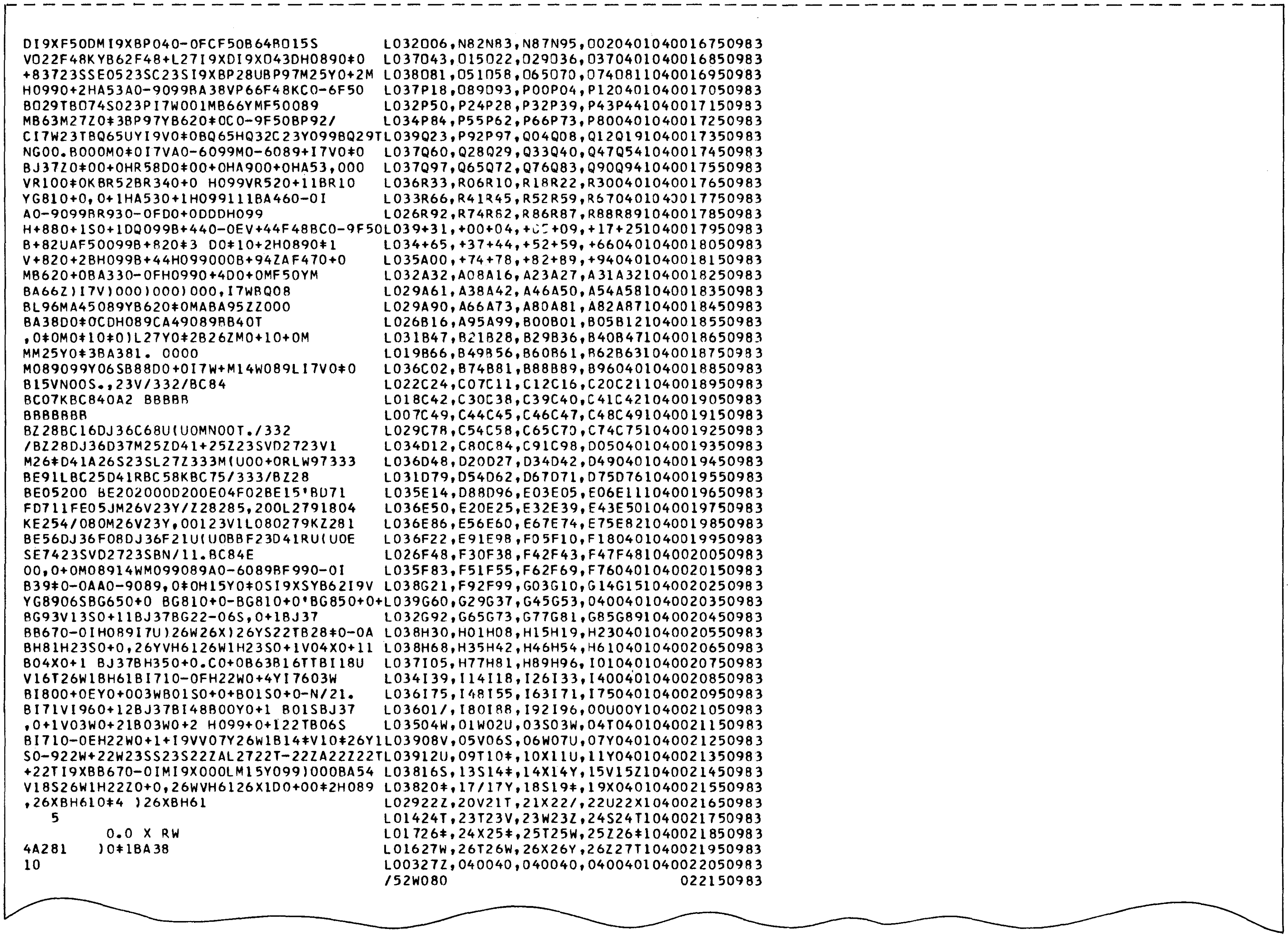
Figure 9. Use of Library Functions Part 6

```
F2SB7003DV=I6S‡B7003BT=I6W‡           L02757/,54Y55S,55Z56‡,56U57/104000555O983
B7004GX=I5Y/I3V‡B7003+/=4GX‡B700      L03260T,57W58X,58Y59S,59Z60‡104000565O983
5+/=3+/C‡52V=50/*50/N+I6SQ‡5+/=3+/C‡4EVL03964S,040040,040040,040040104000575O983
=3DV+3DV*52V*50/‡B70050/=3DV-I2X‡     L03367V)64T64T,65Z66‡,66U67V104000585O983
V68Y2G7BB70/B7004EV=4EVN‡B7004CT=4EVE  L03771S,68U68Y,69S70‡,70/70V104000595O983
‡B7004A/=4CTG‡B7003HZ=3+/+3+/S*3DV‡    L03574X,71U71Y,72W72X,73/74X104000605O983
B7003FX=4A/-3HZ‡BW97*HOWFO‡           L02777U,75S76T,76U76Y,76Z77S104000615O983
B7003DV=3DV+I6S‡B7003BT=3BT+I6W‡B700  L03681‡,77Z79‡,79/79V,80W80X104000625O983
3+/=3+/+4GX‡B70050/=3DV-I3/‡B85Z2800  L03684W,82S82T,82X83Y,83Z040104000635O983
V87Z2G7BB60‡BW97*IOYF2SN111           L02787T,85V85Z,86T86U,86X87‡104000645O983
.B87‡N777.B87ZN000.                   L01989S,87V87Z,88T88U,88Y89S104000655O983
B88YH94WMO-291/+000I7VMO-892VS000I7V  L03692Y,89X90/,90Y91V,92S040104000665O983
MOJ194SLI7V000B000H094H‡4‡0-2H‡4U0-3  L03696U,93W94T,95X95/,95Y040104000675O983
MO-2T4/MT4/094V‡5/0-01B‡6Z0-0,BS0‡0-0$ L038‡0S,97S97Z,98X99V,‡0T040104000685O983
BS2S0-0(BS8/0-0)BS4V0-0=MT4U089MT4/000 L038‡4‡,‡1/‡1Z,‡2X‡3U,‡4/040104000695O983
B000XXXXXXMO-2089HT4/0-3B‡3UYJ36‡8T   L035‡7V,‡4V‡5/,‡5Y‡6V,‡6Z040104000705O983
B/7VT4W2BV/3/‡4W2MO-6‡5‡D690/4UD      L032/0X,‡8U‡8V,‡9T/0‡,/0X040104000715O983
V/2U‡4WKD692/4UDY/2W‡4WM‡4X089H0890‡0  L037/4U,/1W/2T,/2U/3/,/3Y040104000725O983
M089‡4XC‡5‡‡4XB‡3U/Y/6U‡4WB/8ZMO-6‡5‡  L037/8/,/5S/5Z,/6U/7/,/7V040104000735O983
MT4X089HT4/0-7B‡3UB    YSOW088M094T4/  L036S1X,/8Z/9W,S0‡S0U,S1/040104000745O983
B‡3UH0940-1B89XD0-0H094BS5SMO-3094    L034S5/,S2SS2Z,S3TS3X,S4/S4V104000755O983
MOJ2S9UMMO-6S9YMOJ2TOVHT5‡0-OMT5‡094  L036S8X,S5ZS6‡,S6XS7U,S8/040104000765O983
A000000+000I7VS000I7VVT2YI7VKHT4/0J6  L036T2T,S9VTOS,TOZT1X,T2U040104000775O983
B97SMOJ5T4/B97S       2S.             L024T4X,T2YT3V,T3ZT4S,T4VT4X104000785O983
   HT9XM2G9250DLW1SSW1S2M9(0‡0250     L033T8‡,T5/T5V,T6ST6T,T6XT7U104000795O983
D2M92G9MDQ089B000HV4/)V0‡V2S          L028U0Y,T8YT8Z,T9‡T9U,T9YU0S104000805O983
S2+2)280),S096H094250Z250             L025U3T,U1TU1X,U1YU1Z,U2TU3‡104000815O983
,0‡0BV3U0-0 Y2510-1A0-12+2AW1‡095     L033U6W,U3YU4W,U5TU6‡,J6X040104000825O983
C094089BV3UUAW0/W0UAW0UW0XN000251+    L034V0‡,U7UU7Z,U8WU9T,V0‡040104000835O983
'2G92E4Y2E4255N255000+(W0X0‡4BU3‡,280  L037V3X,V0YV1V,V2SV2T,V3‡V3U104000845O983
B000S2G9DW1T280YW1/2G9YT9YW1/+W1TW79  L036V7T,V4SV4W,V5TV6‡,V6X040104000855O983
B755230258509299404568401 79         A L038W1/,V7YW0/,W0SW0V,W0YW1/104000865O983
+ABW3Y924CB/42SY2G9Y3XBW5‡BV4SS       L031W4S,W1TW1U,W2SW2X,W3UW3Y104000875O983
Y-4SY3XM-4T-4X+W79W82S-4SW79VZ1ZW79K  L036W7Y,W5‡W5X,W6UW7/,W7Z040104000885O983
A-4UW79S099W79V-0UW79B+W82W79H089-AZ  L036X1U,W8WW9T,X0/X0Y,X1V040104000895O983
+W8Q095BT5/+0‡1095S-4W095VX3T095B     L033X4X,X2SX2W,X3TX4‡,X4Y040104000905O983
BX6T924CH0940-1Y-JU‡87D-JU-4XSW1‡SW79  L037X8U,X5WX6T,X7‡X7X,X8/040104000915O983
BY5S-4X2+2G92+2BY7Y+2+2249+251--4UW0U  L037Y2/,X9TY0‡,Y0UY1/,Y1V040104000925O983
+-4YW0/SW0XBT9Y+‡87H0942+1B‡24BY7Y    L034Y5V,Y2ZY3T,Y3XY4/,Y4YY5S104000935O983
D-4S0‡0--4ZW0USW79BY2SHZ1YM2G9250H089  L037Y9S,Y6TY7‡,Y7UY7Y,Y8SY8Z104000945O983
L-5‡'2G92E1-2522G9S251B000A099W79     L033Z2V,Y9XZ0U,Z1/Z1V,Z1Z040104000955O983
VZ8VW79K+W82W79Y2G92G8+2G82G9AW82     L033Z5Y,Z3UZ4/,Z4YZ5V,Z5Z040104000965O983
-W83W1‡Y-4S‡87BY5S924CBX9TBV4S924C    L034Z9S,Z6WZ7T,Z8/Z8V,Z9T040104000975O983
+W82W79B/34BU71SCLB/42AKJBA           L027-1Z,-0‡-0U,-0Y-1/,-1V040104000985O983
157079632679489661923 1A B4+ H        L029-4Y,-4S-4T,-4U-4V,-4X-4Y104000995O983
F0Y2G9W84BL0/SVK7V2G9KC2482G9Q089     L033-8/,-5‡-5/,-5Y-6T,-7/-7Y104001005O983
,0‡0DL6W0‡2C281L6YBJ1ZU+2G82G9AL6WW79  L037J1Y,-8W-9T,J0‡J0V,J1S040104001015O983
S2G9249-249L6VSL6WW79AL6W280SL6Z2E0   L035J5T,J2WJ3T,J4‡J4X,J5U040104001025O983
(2G9249L249L6U'L6U2E1+2492G9+L6U251   L035J8Y,J6/J6Y,J7VJ8S,J8Z040104001035O983
+L3YW1‡+++,V0‡V2SHU9WL6UHV2/          L028K1W,J9WJ9X,J9YJ9Z,K0WK1T104001045O983
HV4/K2YBU0ZA2+2YL6V2+1+VGY248'W79252  L036K5S,K2UK2Y,K3SK3Z,K4W040104001055O983
A2+1252H094252SW82B‡10VK9‡0-41BU71LNN  L037K8Z,K6‡K6X,K7/K7V,K8TK8X104001065O983
YL3Y2G9B-7/VL2X0-41BU71LNZYL7‡2G9B/15  L037L2W,K9XL0/,L0ZL1T,L1WL2T104001075O983
)W84BV4S+BA+                          L038L6U,L3/L3V,L3WL3X,L3YL3Z104001085O983
 A31+-Y06/‡87DW84L9S                  L020L8U,L6WL6X,L6ZL7‡,L7/L7Y104001095O983
BM1/07‡0BBBBYW84‡87D07/W84            L026M1‡,L9TL9U,L9VL9W,L9XM0U104001105O983
```

Figure 9. Use of Library Functions Part 7

Figure 9. Use of Library Functions Part 8

```
VM5‡W841,W84-2G9BM9S/BU71ZTZY‡87W1/    L035M4V,M1ZM2T,M2XM3S,M3WM3Z1040011150983
BV4SBM3ZS+W79W82S06/W82VO1WW82KS06VW82 L038M8T,M5‡M5V,M6SM6Z,M7X0401040011250983
VN1/W82KV/422G9KBU71EOFB/15H089VGX     L034N1X,M9SN0‡,NOUNOX,N1/0401040011350983
+W80095BT5/C0‡0D6UBM9ST+0‡0W79Y2G9W79  L037N5U,N2VN2Z,N3WN4/,N4Y0401040011450983
SW1‡C2502G9Q089,0‡0S251D06/0‡0+06SWOX  L037N9/,N5ZN6W,N7‡N7U,N7YN8V1040011550983
++BT9YY2+22+1H0942+1B‡24A099W82        L031O2S,N9TN9U,N9YO0V,O1SO1W1040011650983
VM3ZW82K-W80W1‡SW79Y2G92G8+2G82G9BN5Z0 L038O6‡,O3/O3Y,O4SO4Z,O5WO6‡1040011750983
A09IC024688-2G9Y2G9‡87                 L022O8S,O6SO6V,O6WO7/,O7SO7W1040011850983
B/34VQ7U2G9KD2G92B2M,H089LQ9W201       L032P1U,O8XO9V,POSPOT,POUPOY1040011950983
H094+W80283AQ9W282'Q9Y285D283W79D      L033P4X,P1ZP2W,P3TP4‡,P4X0401040012050983
BP6X2840H0890‡1BP8SVP8SW79BAQ9WW79S2G9 L038P8V,P5WP6T,P6XP7V,P8S0401040012150983
SR0‡0-2,0K1)H0940-1-Q9WR0/AQ9WR0/      L033Q1Y,P9TP9X,P9YQ0V,Q1S0401040012250983
AR0S0-1S0-10‡2VQ1S0‡2BA0-10‡2DR0/0P9   L036Q5U,Q2WQ3T,Q4/Q4Y,Q5V0401040012350983
H0890‡2VP8W0P92B/34BU71SQNYQ9W2G9/278  L037Q9/,Q6SQ7‡,Q7UQ7Y,Q8/Q8Y1040012450983
BO9VA5+1A BBW1U0-4S                    L019R1‡,Q9WQ9X,Q9ZR0/,R0SR0T1040012550983
BW1U0-4CB-5/0-4GBL7/0-4EB07S0-4N       L032R4S,R1ZR2X,R3VR4T,0400401040012650983
B08X0-4Q                               L010R5S,R5/R5S,040040,0400401040012750983
3DV3BT4EV4CT4A/3HZ3FX..                L023F2S,F0TF0W,F0ZF1S,F1VF1Y1040012850983
BJ5200ABL281A=2I(SQRT(1-COS(X)**2)COS(XL039F6/,F2XF3‡,F3U040,0400401040012950983
)SIN(X)/ABS(SIN(X)))BK23BJ5200ABL28    L035F9W)F6SF6S,F8SF8W,F9‡F9T1040013050983
0 I   DEGREES             A           EXPOL039G3V,040040,040040,0400401040013150983
NENTIAL(A)=B      LOGARITHM(B)=C       L039G7U)G3WG3W,040040,0400401040013250983
I SIN(2X)=D      C-DBK08BK08BK23BJ52   L035H0Z)G7VG7V,G9UG9Y,H0SH0W1040013350983
00AH0990+1BL85F00A003000               L024H3T,H1TH2‡,H2UH2V,H2YH3/1040013450983
BL85F00A008001BL85F                    L019H5S,H3YH3Z,H4SH4V,H4YH5S1040013550983
00A009010BL85E00A005                   L020H7S,H5WH5Z,H6SH6W,H6XH7‡1040013650983
014BL85F00B009010BL85                  L021H9T,H7WH8‡,H8/H8U,H8XH9‡1040013750983
E00A007005BK23BJ5200A                  L021I1U,H9VH9Y,I0/I0U,I0YI1S1040013850983
BL281BK232D0B4I0B1B0B                  L021I3V,I1ZI2‡,I2UI2Y,I3S0401040013950983
1570796326794896619 2C0A2A0A7E0AA0+1   L035I7‡,I5ZI6‡,I6TI6X,I7‡0401040014050983
                                       L028I9Y,I7WI9Y,040040,0400401040014150983
H089M0‡0J36MX08D34HJ350‡7M0‡6Z68+X29L27L039X35,X01X08,X15X22,X290401040014250983
)23V,27ZM0‡3094H099200H23Y334BE560‡0+  L037X72,X40X44,X51X58,X650401040014350983
BE250‡0-BD710‡0*VC120‡0KVC750‡0BH094Z16L039Y11,X81X89,X97Y05,0400401040014450983
/332//H099100M-79D34,0+0BZ43           L028Y39,Y16Y17,Y18Y25,Y32Y361040014550983
VC12J362H24/100M0‡00+0Q094BJ37VY820-11 L038Y77,Y48Y55,Y62Y66,Y700401040014650983
BY55BQ08HK22C84BY32BQ08L0+00‡0BZ43     L034Z11,Y82Y86,Y93Y97,Z01Z081040014750983
BJ37VY970+11BZ12HK22M09924/B0-0H-06    L035Z46,Z16Z24,Z28Z32,Z39Z431040014850983
M094-02V-3123U1B94X    H094YB62088     L033Z79,Z54Z62,Z66Z69,Z730401040014950983
B-46089.B-07089 H094000B000VJ3223V1    L035-14,Z88Z96,-03-07,-150401040015050983
C24/099BK08/BJ32)23UM24Z089BZ73M0-2099 L038-52,-22-27,-31-35,-42-461040015150983
M0-5089M0‡124SV-790‡11)24TL27Z0‡1BJ37  L037-89,-60-67,-75-79,-860401040015250983
H094J09VC12J362BC84M24S0‡1VJ3224T1)0‡1 L038J27,-97J05,J09J16,J240401040015350983
,24TB000 HJ51H0990+1B000H094           L028J55,J32J36,J37J41,J48J521040015450983
D0-0DDDQL09M0-224WHK070-3              L025J80,J60J61,J62J63,J67J741040015550983
BJ89H094SE7424WV0-024WKB000H094M24/099 L038K18,J85J89,J96K04,K08K121040015650983
B000VK4323V1C24/099BK08/MZ68K56BK98000,L039K57,K23K31,K38K43,K500401040015750983
MZ68K71B94X    BJ32089 MK71Z68M08924Z  L036K93,K65K69,K72K80,K870401040015850983
,23UVK0823V1B000H094+0-2L27B0-300+     L034L27,K98L06,L10L14,L21L251040015950983
H094VL4723V1M0-00+0M0+00-0BJ37H0940-1  L037L64,L32L40,L47L54,L580401040016050983
VL770-01BL32BQ08B0-0H094M0-325SSE7425S L038M02,L73L77,L81L85,L89L961040016150983
VM3125SBB0-70-0IB0-70-0AB0J0BZ43SI7V   L036M38,M11M19,M27M31,M350401040016250983
,I9WVF5123V1/024,0+0DH099HQ970+2       L032M70,M43M51,M55M59,M60M641040016350983
HA49,001BQ330-0IB41Z0-0AM0‡0I9XMH089   L036N06,M75M79,M87M95,N02N031040016450983
,0‡0A0-6099HA450+2M25V,0+2BN490‡10     L034N40,N11N18,N25N29,N330401040016550983
V074I9V2BN680-0EAL27I9XBN75SL27I9X     L034N74,N49N57,N64N68,N750401040016650983
```



```
DI9XF50DMI9XBP040-0FCF50B64B015S      L032006,N82N83,N87N95,002040104001675098 3
```

Figure 9. Use of Library Functions Part 9

Figure 9. Use of Library Functions Part 10

```
END OF COMPILATION

PRESS START TO GO
```

```
A=2I(SQRT(1-COS(X)**2)COS(X)SIN(X)/ABS(SIN(X)))

 I   DEGREES            A           EXPONENTIAL(A)=B      LOGARITHM(B)=C          I SIN(2X)=D      C-D

 1.      7.5      0.2588190451    0.1295399375E 01        0.2588190451        0.2588190451     0.4E-19
 2.     15.0      1.0000000000    0.2718281828E 01        1.0000000000        1.0000000000     0.0E 00
 3.     22.5      2.1213203436    0.8342144716E 01        2.1213203436        2.1213203436     0.0E 00
 4.     30.0      3.4641016151    0.3194774551E 02        3.4641016151        3.4641016151     0.0E 00
 5.     37.5      4.8296291314    0.1251645325E 03        4.8296291314        4.8296291314     0.0E 00
 6.     45.0      6.0000000000    0.4034287935E 03        6.0000000000        6.0000000000     0.0E 00
 7.     52.5      6.7614807840    0.8639205288E 03        6.7614807840        6.7614807840     0.0E 00
 8.     60.0      6.9282032303    0.1020658443E 04        6.9282032303        6.9282032303     0.0E 00
 9.     67.5      6.3639610307    0.5805413502E 03        6.3639610307        6.3639610307     0.0E 00
10.     75.0      5.0000000000    0.1484131591E 03        5.0000000000        5.0000000000     0.0E 00
11.     82.5      2.8470094961    0.1723615989E 02        2.8470094961        2.8470094961     0.0E 00
12.     90.0      0.0000000000    1.0000000000E 00        0.0000000000        0.0000000000    -0.4E-20
13.     97.5     -3.3646475863    0.3457419839E-01       -3.3646475863       -3.3646475863     0.0E 00
14.    105.0     -7.0000000000    0.9118819656E-03       -7.0000000000       -7.0000000000     0.0E 00
15.    112.5    -10.6066017178    0.2475206303E-04      -10.6066017178      -10.6066017178     0.0E 00
16.    120.0    -13.8564064606    0.9599290509E-06      -13.8564064606      -13.8564064606     0.0E 00
17.    127.5    -16.4207390469    0.7388625308E-07      -16.4207390469      -16.4207390469     0.0E 00
18.    135.0    -18.0000000000    0.1522997974E-07      -18.0000000000      -18.0000000000     0.0E 00
19.    142.5    -18.3525906995    0.1070461693E-07      -18.3525906995      -18.3525906995     0.0E 00
20.    150.0    -17.3205080757    0.3004684793E-07      -17.3205080757      -17.3205080757     0.0E 00
21.    157.5    -14.8492424049    0.3556771481E-06      -14.8492424049      -14.8492424049     0.0E 00
22.    165.0    -11.0000000000    0.1670170079E-04      -11.0000000000      -11.0000000000     0.0E 00
23.    172.5     -5.9528380374    0.2598455530E-02       -5.9528380374       -5.9528380374     0.0E 00
24.    180.0      0.0000000000    0.1000000000E 01        0.0000000000        0.0000000000    -0.3E-16
25.    187.5      6.4704761276    0.6457911327E 03        6.4704761276        6.4704761276     0.1E-18
26.    195.0     13.0000000000    0.4424133920E 06       13.0000000000       13.0000000000     0.0E 00
27.    202.5     19.0918830920    0.1956588407E 09       19.0918830920       19.0918830920     0.0E 00
28.    210.0     24.2487113060    0.3396890234E 11       24.2487113060       24.2487113060     0.0E 00
29.    217.5     28.0118489624    0.1463495638E 13       28.0118489624       28.0118489624     0.0E 00
30.    225.0     30.0000000000    0.1068647458E 14       30.0000000000       30.0000000000     0.0E 00
31.    232.5     29.9437006150    0.1010145526E 14       29.9437006150       29.9437006150     0.0E 00
32.    240.0     27.7128129211    0.1085229847E 13       27.7128129211       27.7128129211     0.0E 00
33.    247.5     23.3345237792    0.1361616844E 11       23.3345237792       23.3345237792     0.0E 00
34.    255.0     17.0000000000    0.2415495275E 08       17.0000000000       17.0000000000     0.0E 00
35.    262.5      9.0586665786    0.8592685341E 04        9.0586665786        9.0586665786     0.0E 00
36.    270.0      0.0000000000    1.0000000000E 00        0.0000000000        0.0000000000     0.4E-20
37.    277.5     -9.5763046688    0.6935275619E-04       -9.5763046688       -9.5763046688     0.0E 00
38.    285.0    -19.0000000000    0.5602796438E-08      -19.0000000000      -19.0000000000     0.0E 00
39.    292.5    -27.5771644663    0.1055333309E-11      -27.5771644663      -27.5771644663     0.0E 00
40.    300.0    -34.6410161514    0.9028130704E-15      -34.6410161514      -34.6410161514     0.0E 00
41.    307.5    -39.6029588779    0.6319074743E-17      -39.6029588779      -39.6029588779     0.0E 00
42.    315.0    -42.0000000000    0.5749522264E-18      -42.0000000000      -42.0000000000     0.0E 00
43.    322.5    -41.5348105304    0.9155055464E-18      -41.5348105304      -41.5348105304     0.0E 00
44.    330.0    -38.1051177665    0.2825905416E-16      -38.1051177665      -38.1051177665     0.0E 00
45.    337.5    -31.8198051534    0.1516471339E-13      -31.8198051534      -31.8198051534     0.0E 00
46.    345.0    -23.0000000000    0.1026187963E-09      -23.0000000000      -23.0000000000     0.0E 00
47.    352.5    -12.1644951198    0.5212269879E-05      -12.1644951198      -12.1644951198     0.0E 00
48.    360.0      0.0000000000    0.1000000000E 01        0.0000000000        0.0000000000    -0.1E-15
```

Index

R016512MPM

READER'S COMMENT FORM

Fortran Specifications and Operating Procedures IBM 1401, Form C24-1455-2

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. All comments will be handled on a non-confidential basis.

	Yes	No
● Does this publication meet your needs?	☐	☐
● Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? ____________________.
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other ____________________.

- Please give specific page and line references with your comments when appropriate. If you wish a reply, be sure to include your name and address.

COMMENTS:

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

C24-1455-2

fold

fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Systems Development Division
Development Laboratory
Rochester, Minnesota 55901

Attention: Product Publications, Dept. 245

fold

fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

C24-1455-2

IBM 1401 Printed in U.S.A. C24-1455-2

IBM®

International Business Machines Corporation

Data Processing Division

112 East Post Road, White Plains, N.Y. 10601