SDS

SCIENTIFIC DATA SYSTEMS

Reference Manual

SDS 940 BASIC

SUMMARY OF BASIC COMMANDS

Command Syntax		Page
step	DATA constant 1[,constant 2]... (RET)	9
step	DEF FN letter 1(letter 2) = expression (RET)	18
*	DEL{step 1 {ALL / [-step 2] / [, step 2]...}} (RET)	21
step	DIM letter 1(expression 1[,expression 2])[,letter 2(expression 3[,expression 4])]... (RET)	11
*	DUMP [step 1 {[-step 2] / [, step 2]...}] (RET)	21
step	END (RET)	20
step	FOR variable = expression 1 TO expression 2 [STEP expression 3] (RET)	10
step 1	GOSUB step 2 (RET)	19
**[step 1]	GO TO step 2 (RET)	8
step 1	IF relational-expression THEN step 2 (RET)	8
step	INPUT[FILE]variable 1[,variable 2]... (RET)	9,12
**[step]	LET variable = expression (RET)	7
*	LIST [step 1 {[-step 2] / [, step 2]...}] (RET)	21
*	LOAD {TELETYPE / /file-name/} (RET)	20
*** step	MAT matrix keywords, operators, and delimiters (RET)	12,13,14
step	NEXT variable (RET)	10
step	OPEN /file-name/, {INPUT / OUTPUT} (RET)	15
**[step]	PRINT {expression 1 / "character-string 1"}[{, / □ / ;}{expression 2 / "character-string 2"}]... (RET)	7,15
step	PRINT FILE expression 1[,expression 2]... (RET)	15
step	READ variable 1[,variable 2]... (RET)	9
step	READ FILE expression 1[,expression 2]... (RET)	16
step	REM character-string (RET)	20
step	RESTORE (RET)	9
step	RETURN (RET)	19
*	RUN (RET)	20
step	STOP (RET)	20
step	WRITE expression 1[,expression 2]... (RET)	16

*System Directives
**Optionally, Statements or System Directives
***For variations of the matrix statement, see inside back cover.

Price: $1.00

BASIC

REFERENCE MANUAL

for

SDS 940 TIME-SHARING COMPUTER SYSTEMS

90 11 11C

August 1968

SDS

SCIENTIFIC DATA SYSTEMS/701 South Aviation Blvd./El Segundo, California 90245

Printed in U.S.A.

REVISION

This publication is a major revision of the SDS BASIC Reference Manual (dated January 1968). A change in the text is indicated by a vertical line in the margin of the page.

RELATED PUBLICATIONS

Title	Publication No.
SDS 940 Computer Reference Manual	90 06 40
SDS 940 Time-Sharing System Technical Manual	90 11 16
SDS 940 Terminal User's Guide	90 11 18

CONTENTS

ILLUSTRATION

1. INTRODUCTION

The BASIC[†] language and compiler were originally developed at Dartmouth College for time-sharing computer users with no previous knowledge of computers, as well as for users with considerable programming experience. Thus, BASIC is as useful to the businessman as it is to the scientist or engineer. A simple, straightforward language, BASIC closely resembles standard mathematical notation. In addition to its powerful arithmetic capability, it also contains editing features, simple input and output procedures, and complete language diagnostics.

This manual is intended to serve as a tutorial guide for the new BASIC user and as a reference source for the experienced user. Material is presented in a sequence that will enable the reader to immediately use the SDS 940 time-sharing computer terminal to write simple programs, to gain confidence in the system, and then to progress to more difficult programs.

For clarity, several typographic conventions have been used throughout this manual. These are explained below.

1. Underscored copy in an example represents copy produced by the system in control of the computer. Unless otherwise indicated, copy that is not underscored in an example must be typed by the user.
2. The (RET) notation appearing after some lines in the examples indicates a carriage return. The carriage return key is labeled RETURN on the Teletype keyboard. The user must depress the carriage return key after each command to inform the computer that the current command is terminated and a new one is to begin. The computer then upspaces the paper automatically.
3. Non-printing control characters are represented in this manual by an alphabetic character and a superscript c (e.g., D^c). The user depresses the alphabetic key and the Control (CTRL) key simultaneously to obtain a non-printing character. For editing purposes some control characters will cause a symbol to be printed, but this symbol does not appear in the final version of an edited line.
4. Other typographic conventions pertain to the method used in this manual to define the syntax of BASIC commands. These conventions are described in the paragraph titled "Basic Syntax Notation" later in this chapter.

OPERATING PROCEDURES

The standard procedure for gaining access to an SDS 940 time-sharing computer center from a Teletype terminal is described in the SDS 940 Terminal User's Guide, which also includes information concerning the 940 Executive System and the calling of the various subsystems available to the terminal user. The following paragraphs summarize the standard procedures as they apply to BASIC users.

[†]The word "BASIC" is an acronym for "Beginner's All-purpose Symbolic Instruction Code".

LOG IN

To gain access to the computer, the following operating sequence is performed:

1. If the FD-HD (Full Duplex-Half Duplex) switch is present, turn the switch to FD. This is a toggle switch with two locking positions. When the Teletype is not connected to the computer (sometimes called the Local Mode), this switch must be in the HD position. Whenever the Teletype is connected to the computer, this switch must be in the FD position.
2. Press the ORIG (originate) key, located at the lower right corner of the console directly under the telephone dial. This key is depressed to obtain a dial tone before dialing the computer center.
3. Dial the computer center number. When the computer accepts your call, the ringing will change to a high-pitched tone. There will then appear on the teletype a request that the user log in:

   PLEASE LOG IN:

4. The user must then type his account number, password, and project code (if he has one) in the following format:

   PLEASE LOG IN: number password;name;project code

   Only persons who know the account number, password, and name may log in under that particular combination. The following examples illustrate acceptable practice.

   PLEASE LOG IN: A1PASS;JONES;REPUB (RET)
   PLEASE LOG IN: B4WORD;BROWN;DEMO (RET)
   PLEASE LOG IN: C6PW;SMITH (RET)

   The optional 1-12 character project code is provided for installations that have several programmers using the same account number. The project code is not checked for validity.

   If the user does not type his correct account number, password, and name within a minute and a half, a message is transmitted instructing him to call the computer center for assistance. The computer will then disconnect the user, and the dial and log-in procedure will have to be repeated.

5. If the account number, password (nonprinting), and name are accepted by the computer, it will print READY, the date, and the time on one line and a dash at the beginning of the following line.

   READY date time

   -

   The dash indicates that the Executive is ready to accept a command.

6. In response to the dash, the user types

   BASIC (RET)

   The Executive will respond with the symbol > at the beginning of the next line, which means that the BASIC subsystem is in control and waiting for a command.

ERROR CORRECTION

If the user makes a mistake while typing and notices it immediately, he can correct the error at once. The BASIC language will accept the following edit characters:

Character	Function
←	The left arrow (located on the letter "O" while the shift key is depressed) is used to delete the most recent character typed. If the user notices that he has just mistyped a letter or a symbol, he strikes the ← key, which tells BASIC to ignore the previous character. Striking this key repeatedly will delete a corresponding number of characters, but only to the start of the current line.

For example, the command

PRE ← INT A+C ← B (RET)

will appear to BASIC as

PRINT A+B (RET)

while

PRINT←←←←←←←←LET X=Y (RET)

will appear to BASIC as

LET X=Y (RET)

A command preceded by a step number (see "Writing Programs in BASIC") may be deleted by retyping the step number immediately followed by a carriage return. A command preceded by a step number may be changed by retyping the step number and the new command, followed by a carriage return.

ESCAPE

The ESCAPE (ESC) key[t] may be used at almost any time. It causes the system in control to abort the current operation and to ask for a new command. Striking the (ESC) key before terminating a command with (RET) aborts the command. Striking the (ESC) key during the execution of a program will abort the program and return control to BASIC. BASIC responds by printing a > at the beginning of the next line.

EXIT AND CONTINUE

Striking the (ESC) key twice in succession causes computer control to return to the Executive. If the user wants to reenter BASIC without losing his program and if he has not called any other subsystem since leaving BASIC, he can type CONTINUE in response to the dash. This will return him to BASIC so that he can resume his work.

LOG OUT

When the user wishes to be disconnected from the computer, he types two consecutive escapes (to return control to the Executive) and then types

LOGOUT (RET)

The computer will respond by printing the amount of computer time and hook-up (line) time charged to the user's account since the previous log-in procedure was completed.

BASIC ARITHMETIC COMPONENTS

The arithmetic components of the BASIC language are constants, variables and expressions.

CONSTANTS

BASIC accepts any decimal number, positive or negative, with or without a decimal point, that can be expressed in eight digits or less. If more than 8 digits are input, the number will be truncated. The numbers may be expressed as integers (whole numbers, for example, 1, -7, 130, 256) or in floating-point form, that is, as numbers with a decimal point, (for example, 1.5, -10.0, 152.55).

To accommodate numbers that are too large or too small to be expressed in eight digits, BASIC accepts constants in scientific notation. Scientific notation consists of a number (whole number or whole number and fraction), followed by the letter E, followed by the exponent. The exponent represents the integral power of ten by which the number to the left of the exponent is to be multiplied. For example, all of the quantities

2.59 0.259E1 25.9E-1 259E-2

express the same number. The largest number that BASIC will accept is 5.789604E76.

BASIC will output a maximum of eight digits with seven digit accuracy. The constants will be printed according to the following rules:

1. A constant will be printed in integer form if it has 6 or fewer significant digits and no fractional part. If the number of digits is greater than or equal to 7, the constant will be expressed in scientific notation.
2. A constant will be printed in floating-point form if the number is less than 1000000.0 but not less than 0.1. Otherwise, the number will be expressed in scientific notation.

The following examples illustrate these rules.

>PRINT 123456 (RET) 123456	Number has fewer than 6 digits and no fractional part.

[t] In some 940 time-sharing system configurations the ALT MODE key is used instead of the ESCAPE key. Where (ESC) appears in this manual, ALT MODE may be substituted.

>PRINT 1234567 (RET) 1.234567E+06	Number has more than 6 digits.
>PRINT 123.E2 (RET) 12300	Number has fewer than 6 digits and no fractional part.
>PRINT 123.33412 (RET) 123.33412	Number is less than 1,000,000.
>PRINT 1000021.4 (RET) 1.0000214E+06	Number is greater than 1,000,000.
>PRINT .123 (RET) .123	Fraction is greater than .1
>PRINT .0123 (RET) .123E-01	Fraction is less than .1
>PRINT 9.9E78 (RET) .0738375E-74	Number exceeds limit
>PRINT 123456789 12 (RET) 1.2345679E+10	More than 8 digits were input. The number was rounded to 8 digits and printed in scientific notation.

>

VARIABLES

The numerical value of a constant is always the number itself. BASIC also permits the user to use a symbol to represent a number. Such symbols are called variables because the value of the symbol may be changed. Associated with each variable is a place in computer memory where the defined value of the variable is stored. In BASIC, a variable may be a single letter or a letter followed by a digit. For example, some legal variables are

B
C
A1
Z5

and some illegal variables are

1B (first character is not a letter)
BA (second character is not a digit)
A35 (too many characters)

It is often convenient to keep data in a list. Such a list is called an array. The individual values in the list are called array elements. We refer to an array element by using the name of the array and the position of the element in the array. For example, we can refer to the fourth element in array A by writing A(4). In this example, the 4 is called the subscript.

When the array has only one "dimension", it is often called a vector or a linear array. However, arrays may also have two dimensions. A two-dimensional array, often called a matrix, may be thought of as having columns and rows. There is always one subscript for each dimension; thus, a two-dimensional array is written as A(X,Y) where X represents a row number and Y represents a column number.

Note that the name of an array must be only one letter, while a subscript may be any evaluatable expression, which may also include an array element. (Expressions are discussed below.)

As a further example of matrix notation, consider the following chart, which lists the expenses for a four-day car trip:

Row	Item \ Date	June 5	June 6	June 7	June 8
	Column	1	2	3	4
1	Gas, oil	21.29	20.84	19.42	6.08
2	Tolls	1.32	.86	.40	.07
3	Food	11.18	12.82	14.39	5.06
4	Lodging	10.05	12.79	10.35	.00
5	Misc.	1.35	.90	.44	.10

If we consider the chart to be a two-dimensional array called P, the amount spent for lodging on June 5 would be represented by P(4,1). The first subscript always represents the row number, while the second subscript represents the column number. Thus, the amount spent for food on June 8 would be represented by P(3,4).

EXPRESSIONS

Arithmetic Expressions

Arithmetic expressions are formed by combining variables and/or numbers with arithmetic operators, as in mathematical formulas. There are six arithmetic operators in BASIC:

- Negation	a unary operator
↑ Exponentiation	binary operators
* Multiplication	
/ Division	
+ Addition	
- Subtraction	

The following are examples of legal arithmetic expressions:

Expression	Meaning
F + H	F plus H
D + A + X	D plus A plus X
C - A(3)	C minus the third element of A
Q * - 5	Q times minus 5
X/Y	X divided by Y
Q ↑ 2	Q to the power of 2 (or Q^2)
P ↑ R	P to the power of R (or P^R)
X/Y + 6	X divided by Y, plus 6

In the last example in the preceding table, the order of computation is not clear. The expression might have been interpreted as

X divided by the quantity Y+6

rather than

the ratio X/Y plus 6

To make sure that the computer evaluates the expression the way the user meant it to be evaluated, there is an established rule of precedence:

> Exponentiation is always performed before negation, which is always calculated before multiplication and division, which are always calculated before addition and subtraction. The computer calculates from left to right if operators of the same precedence (for example, multiplication and division) appear in the same line.

To alter this order, parentheses must be used. Thus, to represent

X divided by the quantity Y+6

we must write

X/(Y + 6)

Otherwise, according to the precedence rules it would be interpreted as "the ratio X/Y plus 6" because division is calculated before addition.

The following are examples of precedence in expressions:

Expression	Interpretation
A+B*C	A plus the product B times C.
(A+B)*C	C times the sum A plus B.
Z-Y/X+W	Z minus the ratio Y divided by X, plus W.
(Z-Y)/(X+W)	the difference Z minus Y divided by the sum X plus W.
X↑2+Y	X to the power of 2, plus Y.
X↑-(2+Y)	X to the negative power of the sum 2 plus Y.

Mathematical Functions

The mathematical functions available in BASIC are:

SIN(expression)	sine of expression in radians
COS(expression)	cosine of expression in radians
TAN(expression)	tangent of expression in radians
ATN(expression)	arctangent of expression in radians
EXP(expression)	natural exponent of expression
ABS(expression)	absolute value of expression
LOG(expression)	natural log of expression (base e)
SQR(expression)	square root of expression
LGT(expression)	common log of expression (base 10)
INT(expression)	integer part of expression
RND.	random number

The expression enclosed in parentheses is called the argument. It may be any arithmetic expression, for example, SQR(A*B). The arithmetic expression also may include a function, for example, COS(N*X+SIN(T)). A more complete discussion of functions is contained in Chapter 3.

Relational Expressions

A relational expression consists of two arithmetic expressions separated by one of the relational operators. The relational operators available in BASIC are:

Operator	Relation
=	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
<>	not equal to

In BASIC, a relational expression is either "true" or "false", depending on whether the answer to the question implied by the relational expression is "yes" or "no". For example, each of the following relational expressions implies a different question:

Relational expression	Question
X>5	Is the value of the variable X greater than the constant 5?
A<>B	Are the values of the variables A and B unequal?
Z<=Y↑K	Is the value of the variable Z either less than or equal to the value of the variable Y raised to the power of K?

If the answer to the question is "yes", the relational expression is "true"; if the answer is "no", the relational expression is "false".

Evaluatable Expressions

An evaluatable expression is an arithmetic expression or a relational expression for which the values of all variables contained within the expression are known to BASIC at the time the command in which the expression appears is executed. In order for the value of a variable to be known, the variable must be defined (or "declared"). Variables may be declared by the execution of the following BASIC commands:

LET
READ and DATA (used together)
INPUT

These commands are described in Chapter 2.

BASIC SYNTAX NOTATION

The following paragraphs describe the notation used in this manual to define the syntax of the BASIC language.

1. A "notation variable" is used to represent a variable element of the BASIC language. A notation variable consists of lower-case letters, or lower-case letters in combination with digits (of which the first character is a letter). For example,

   variable

   denotes the occurrence of a BASIC variable.

2. A "notation constant" denotes the literal occurrence of the characters represented. A notation constant consists either of all capital letters or of a special character or symbol. For example,

   LET variable = expression ⏎

   denotes the literal occurrence of the word LET, followed by a BASIC variable, the literal occurrence of an equal sign, a BASIC arithmetic expression, and the literal occurrence of a carriage return.

3. The term "syntactical unit" (used in subsequent rules) is defined as a single notation variable or constant, or as any collection of notation variables, notation constants, BASIC language symbols, and reserved words surrounded by braces or brackets.

4. Braces { } are used to denote a grouping. For example,

   LOAD { TELETYPE / /file-name/ }

   The vertical stacking of syntactical units indicates that a choice is to be made. The above example indicates that the word LOAD must be followed by either the word TELETYPE or a slash character followed by a file name, followed by a second slash character.

5. Brackets [ ] denote options. Anything enclosed in brackets may or may not appear. For example,

   [step 1] GO TO step 2 ⏎

   denotes that the words GO TO may or may not be preceded by a step number; however, the words GO TO must be followed by a step number and a carriage return.

6. Ellipsis marks (. . .) denote the occurrence of the immediately preceding syntactical unit one or more times in succession. For example,

   expression 1 [,expression 2]. . . ⏎

   denotes that the variable "expression 1" must occur; however, the variable "expression 2" may or may not occur since it is surrounded by brackets. If expression 2 does occur, it may be repeated one or more times (with a comma preceding each occurrence), each occurrence may have a unique form, and the last occurrence must be followed by a carriage return.

7. The character "i" is used as a collective reference designator when a syntactical unit may appear any number of times in succession. For example,

   expression i

   denotes any of the expressions in the sample given above for rule 6.

8. The BASIC language ignores all spaces (blanks) except those that are within a message of text output. Any spaces that appear in the description of the commands or in sample problems have been inserted to improve readability.

BASIC NOTATION VARIABLES

The common variables used in this manual to define the syntax of the BASIC language are described in the following table.

Notation Variable	Meaning
constant	a BASIC constant (see "Constants")
variable	a BASIC variable (see "Variables")
element	an element of an array (see "Variables") identified by its position in the array as in letter (expression 1[,expression 2])
function	a mathematical function (see "Mathematical Functions") of the general form { SIN, COS, TAN, ATN, EXP, ABS, LOG, SQR, LGT, INT, RND. } (expression)
expression	an arithmetic expression (see "Arithmetic Expressions") of the general form { constant 1 / variable 1 / element 1 / function 1 } [{ + / - / * / / / ↑ } { constant 2 / variable 2 / element 2 / function 2 }] . . .
relational expression	a relational expression (see "Relational Expressions") of the general form expression 1 { = / > / >= / < / <= / <> } expression 2

BASIC NOTATION CONSTANTS

The notation constants of the BASIC language consist of command keywords, function identifiers, operators, and delimiters.

Command Keywords

The command keywords in the BASIC language are

Keywords	Use
DATA	defines data for a READ command
DEF	defines a nonstandard function
DEL	deletes a portion of a program

Keyword	Use
DIM	defines the dimensions of an array
DUMP	designates portion of a program to be saved
END	identifies the end of a program
FILE	specifies a system file
FN	used with DEF
FOR	part of FOR command
GO TO	transfers control to a designated step
GOSUB	calls a subprogram
IF	conditional GO TO command
INPUT	specifies Teletype or input file with OPEN
LET	arithmetic assignment
LIST	lists portion of program
LOAD	specifies program loading
MAT	specifies a matrix operation
NEXT	terminates FOR command group
OPEN	opens file for input or output
OUTPUT	specifies output file with OPEN command
PRINT	prints data on Teletype or on file
READ	accepts data from DATA command
REM	inserts documentary comments
RETURN	terminates subprogram
RUN	begins execution
STEP	part of FOR command
STOP	terminates program execution
TELETYPE	designates remote terminal
THEN	part of IF command
TO	part of FOR command

Function Identifiers

The following function identifiers are a standard part of the BASIC language.

ABS	ATN	COS	EXP	INT	
LGT	LOG	RND.	SIN	SQR	TAN

Other function identifiers may be declared by the programmer, as described in Chapter 3.

Operators

The operators included in the BASIC language are the arithmetic and relational operators. (See "Arithmetic Expressions" and "Relational Expressions".)

Delimiters

Certain special characters are used as separators and as other types of delimiters. The BASIC delimiters are:

Character	Use
,	separates elements of a list
;	separates elements of a formatted line
( )	encloses array subscripts and controls the evaluation of expressions
(RET)	terminates a command
"	denotes the beginning or the end of a text character string

Remember that blanks are ignored except in quoted text. The following example demonstrates the general rule. (For the purpose of this discussion, the symbol for a blank is □).

35□5□PR□INT□"□SAMPLE□TEXT□"□B□8

ignored blanks — these blanks are part of text — ignored blanks

The above line is equivalent to

355PRINT"□SAMPLE□TEXT□"B8

Note that blanks may be eliminated entirely if the programmer is not concerned with the readability of the printed copy. Furthermore, since each BASIC command is limited to a single line of 80 characters, the completion of a complex command may be more important that its readability.

2. BASIC COMMANDS

SINGLE COMMANDS IN BASIC

The BASIC language allows the remote Teletype terminal to be used as an extremely powerful desk calculator to evaluate complicated mathematical expressions. Two commands enable the user to express almost any mathematical expression: PRINT and LET.

PRINT

The PRINT command causes BASIC to print the value of an expression. The user types the word PRINT and the expression he wishes to evaluate (followed by a carriage return). The computer will then issue a line feed and print the value of the expression. The format of the PRINT command used for this operation is

PRINT expression (RET)

Example:

```
>PRINT 7.56*8.73 (RET)
 65.9988
```

Note that underlined copy is that which is generated by the computer. Several expressions can be evaluated with one PRINT command by separating the expression with commas. The format of the PRINT command used for successive evaluation and printing is

PRINT expression 1[,expression 2]. . . (RET)

Example:

```
>PRINT 5*6,  7+8+40, 45/9 (RET)
 30          55          5
```

Text can also be printed by enclosing the characters to be printed in quotation marks according to the format

PRINT "character-string" (RET)

BASIC will cause the computer to print exactly what appears between the quotation marks, for example,

```
>PRINT "THIS MESSAGE INCLUDES BLANKS" (RET)
THIS MESSAGE INCLUDES BLANKS
```

the PRINT command can also be used to type text and the values of expressions, for example,

```
>PRINT "A = " 5+6 (RET)
A = 11
```

More elaborate output formats can be constructed as described in Chapter 4.

LET

If a variable is used as part of an expression in the PRINT command, the user must first assign a value to the variable. The LET command is used in other BASIC systems (notably, the Dartmouth BASIC compiler) to assign the value of an expression to a variable. In 940 BASIC, LET is not required but is accepted to preserve compatibility with these systems.

The format of the replacement (LET) statement, which assigns the value of an expression to a variable, is

variable = expression (RET)

or

LET variable = expression (RET)

For example, the command

X = 2+3 (RET)

assigns the value 5 to the variable X; the command

Y = 10 (RET)

assigns the value 10 to the variable Y; and the command

A(5) = 9 (RET)

assigns the value 9 to the fifth element of array A.

Once the variable has a value, it may be used in a PRINT command, for example,

```
>A = 5 (RET)
>PRINT A (RET)
 5
>B = -4 (RET)
>PRINT B (RET)
-4
>PRINT "A + B =" A + B (RET)
A + B = 1
```

A replacement statement can be used to change the value of a variable at any time, for example,

```
>A = 5 (RET)
>PRINT "A =" A (RET)
A = 5
>A = A+1 (RET)
>PRINT "NOW A =" A (RET)
NOW A = 6
```

WRITING PROGRAMS IN BASIC

The foregoing concerns some of the things BASIC can do when commands are entered and executed one at a time. BASIC may also be used as a means for writing and storing computer programs for future execution.

A program is composed of commands (steps) that are to be used in solving a problem. For example, consider the following steps of a program that calculates the hypotenuse of a right triangle according to the formula:

$$\text{HYPOTENUSE} = \sqrt{(\text{side 1})^2 + (\text{side 2})^2}$$

```
>100 A = 4 (RET)
>110 B = 3 (RET)
>120 C = SQR(A ↑ 2 + B ↑ 2) (RET)
>130 PRINT "A=" A, "B=" B, "C=" C (RET)
>RUN (RET)
```

Note that each step has a unique step number (which may be any integer in the range 1 through 99999). The presence of the step numbers tells BASIC that these steps are not to be immediately executed, but are to make up a program. When the RUN command is given, telling the computer to execute the program, the steps are executed one at a time in ascending numerical sequence by step number. The result printed by the computer would be:

A= 4 B= 3 C= 5

Remember that the = sign in the program means replacement, not equality. Thus, step 100 means "assign the value 4 to A".

The following program calculates and prints the area and the volume of a sphere:

```
>200 P = 3.14 (RET)
>300 R = 2 (RET)
>400 A = 4 * P * R ↑ 2 (RET)
>500 V = (4/3) * P * R ↑ 3 (RET)
>600 PRINT R, A, V (RET)
>RUN (RET)
```

When the RUN command is given, BASIC will print

2 50.24 33.49333

Although BASIC executes commands according to the numerical sequence of step numbers, the steps of a BASIC program need not be prepared in numerical sequence. For example, the above program could have been prepared as

```
>200 P = 3.14 (RET)
>300 A = 4 * P * R ↑ 2 (RET)
>400 V = (4/3) * P * R ↑ 3 (RET)
>500 PRINT R, A, V (RET)
>250 R = 2 (RET)
>RUN (RET)
```

and the results will be identical.

GO TO

As we have seen, BASIC executes the steps of a program in ascending numerical sequence by step number. However, in writing programs it is sometimes necessary to change the normal sequence of execution. This can be accomplished by using the GO TO command, which has the format

[step 1] GO TO step 2 (RET)

where

step 1 is the optional step number of the GO TO command. If step 1 is not specified, BASIC begins executing commands (beginning with step 2) immediately after the carriage return.

step 2 is the step number of the command that is to be executed next (instead of the command with the next step number that is numerically higher than step 1).

Examples:

```
GO TO 100 (RET)
385 GO TO 215 (RET)
```

IF

It is often convenient to go to a step only under certain conditions. This type of statement is called the IF (or conditional GO TO) command, which has the format

step 1 IF relational-expression THEN step 2 (RET)

This command means, "If the relational expression is true, go to step 2 for the next command; otherwise (that is, if the relational expression is false) go to the next step number in numerical sequence after step 1". For example, if we want to say, "If X is greater than 5, go to step 100", we would write

```
70 IF X > 5 THEN 100 (RET)
```

Some other examples are:

```
100  IF A = 10 THEN 500 (RET)
500  IF C(5) > 10 THEN 2300 (RET)
2300 IF D <= E THEN 100 (RET)
```

The following program solves a quadratic equation of the form $ax^2 + bx + c = 0$, by using the formulas:

$$X_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$X_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

```
>100 A = 5 (RET)
>200 B = 6 (RET)
>300 C = 7 (RET)
>350 D = B ↑ 2 - 4 * A * C (RET)
>400 IF D < 0 then 700 (RET)
>450 X1 = (-B + SQR(D)) / (2 * A) (RET)
>500 X2 = (-B - SQR(D)) / (2 * A) (RET)
>550 PRINT X1, X2 (RET)
>600 GO TO 750 (RET)
>700 PRINT "NO REAL ROOTS" (RET)
>750 STOP (RET)
>GO TO 100 (RET)
```

Execution of the above program begins immediately after the carriage return following the second GO TO command. Note that in the example, the second GO TO command could have been used to start execution at any step, whereas the RUN command always causes execution to begin with the lowest-numbered step. Note also that step 400 is a conditional GO TO command. It tells BASIC to skip the intervening steps and execute step 700 only if the discriminant (D) is less than zero. If the discriminant is not less than zero the next command (step 450) is executed after step 400.

DATA, READ, AND RESTORE

Values can be assigned to variables in several ways. The assignment (LET) statement is one method. Another method involves the combined use of the DATA and READ commands.

All the constants that are to be assigned to variables throughout the program are written together in DATA commands, which have the format

step DATA constant 1[,constant 2]. . . (RET)

Examples:

125 DATA 5,10,15 (RET)
150 DATA 100,0,4E2,4.2 (RET)
345 DATA 1.1,1.7, 34902,33.367E-15 (RET)

Similarly, the READ command has the format

step READ variable 1[,variable 2]. . . (RET)

Examples:

40 READ D (RET)
50 READ A1,A2 (RET)
60 READ X(1),X(2),X(3) (RET)

Each time a READ statement appears, the computer automatically assigns each constant in the DATA list to the corresponding variable in that READ statement. For example, the commands

100 READ A,B,C (RET)
200 DATA 1,2,3 (RET)

would be equivalent to the commands

100 A = 1 (RET)
150 B = 2 (RET)
200 C = 3 (RET)

Generally a program uses more than one value for a variable in order to prevent excessive use of constants and assignments. For example, consider the program

```
>10 G = 100 (RET)
>20 P = 20 (RET)
>30 D = G * P * .01 (RET)
>40 A = G - D (RET)
>50 PRINT D,A (RET)
>55 G = 150 (RET)
>60 P = 5 (RET)
>70 D = G * P * .01 (RET)
>80 A = G - D (RET)
>90 PRINT D,A (RET)
```

Another way of writing this program using the READ and DATA commands is

```
> 10 READ G,P (RET)
> 30 D = G * P * .01 (RET)
> 40 A = G - D (RET)
> 50 PRINT D,A (RET)
> 60 READ G,P (RET)
> 70 D = G * P * .01 (RET)
> 80 A = G - D (RET)
> 90 PRINT D,A (RET)
>100 DATA 100,20,150,5 (RET)
```

Note that all the data that is to be assigned to G and P is now located in step 100.

The RESTORE command has the form

step RESTORE (RET)

RESTORE instructs the computer to reread DATA values beginning with the first DATA statement. Thus, in the example below, the values for E, F, and G (step 35) are the same as for A, B, and C because "READ E, F, G" follows "RESTORE".

Study the following example:

```
>10 READ A, B, C
>15 PRINT "A="A, "B="B, "C="C
>20 READ D
>25 PRINT "D="D
>30 RESTORE
>35 READ E, F, G
>40 PRINT "E="E, "F="F, "G="G
>45 DATA 1,3
>50 DATA 5, 7, 9, 11, 13
>RUN
A = 1      B = 3      C = 5
D = 7
E = 1      F = 3      G = 5
>
```

INPUT

There is a third method of assigning values to variables in a program. This may be done when the program is executed, contrary to the other methods mentioned. To use the READ and DATA commands or the LET command, the user must assign values to all variables when the program is written. To assign values to variables at execution time the user may use the INPUT command, which has the format

step INPUT variable 1[,variable 2]. . .

Examples:

400 INPUT A, B, C, D, E (RET)
500 INPUT X, Y, Z1, Z2 (RET)

Each time the input command is encountered during execution, the program is halted and a ? is output to the terminal. At that time, numbers corresponding to the variable list, separated by commas or blanks, must be typed in, terminating with a carriage return. The values are automatically assigned to the respective variables and execution is continued.

Now the sample program given for the READ and DATA commands could be written as:

```
>10 INPUT G,P (RET)
>20 D = G * P * .01 (RET)
>30 A = G - D (RET)
>40 PRINT D,A (RET)
>50 INPUT G,P (RET)
>60 D = G * P * .01 (RET)
>70 A = G - D (RET)
>80 PRINT D,A (RET)
>RUN (RET)
```

When the RUN command is given, BASIC executes the first INPUT command and waits for the values of G and P, and a carriage return. Upon receiving these, execution continues until the next input command. Output would appear as

```
? 100,20 (RET)
 20                    80
? 150,5  (RET)
 7.5                   142.5
```

WRITING PROGRAM LOOPS

Note that the first four steps of the sample program for the INPUT command are exactly like the second four steps. This makes it possible to represent the program in the following way:

```
>10 READ G, P (RET)
>20 D = G * P * .01 (RET)
>30 A = G - D (RET)
>40 PRINT D, A (RET)
>50 GO TO 10 (RET)
>60 DATA 100, 20, 150, 5 (RET)
>RUN (RET)
```

The computer will perform steps 10 through 50 in the normal fashion, but after completing step 50 it will go back to step 10 and repeat steps 10 through 50. When the computer comes to step 50 once more, step 50 sends it back to step 10 again. This process is repeated over and over until all data defined in the DATA command (or any higher numbered DATA command) has been used. At this time, the message "OUT OF DATA 10" would be typed. This technique (often called a loop) is perhaps the single most important feature in programming. The following example shows all the steps that are necessary to set up a controlled loop to print all the numbers between 1 and 100.

```
>10 I = 1 (RET)
>15 IF I > 100 THEN 60 (RET)
>20 PRINT I (RET)
>45 I = I + 1 (RET)
>50 GO TO 15 (RET)
>60 PRINT "FINISHED" (RET)
>RUN (RET)
```

First, a variable (I) was selected to be the counter. Second, an initial value (1) was assigned to the counter variable. Third, the value of the counter variable was tested to see if it exceeded the upper limit (I > 100). Fourth, the value of the counter variable was increased each time the loop was repeated (I = I + 1).

FOR, NEXT

A second (and more concise) method of constructing program loops is to use the FOR and NEXT commands. The FOR command, which has the format

step FOR variable = expression 1 TO expression 2 (RET)

assigns the value of expression 1 to the variable and uses expression 2 as an upper limit for the value of the variable. The NEXT command, which has the format

step NEXT variable (RET)

must appear somewhere after the FOR command. The variable must be exactly the same variable given in the FOR command. The purpose of NEXT is to increment the value of the variable (by 1) and to compare its incremented value with the value that expression 2 had when the previous FOR command was first encountered. If the incremented variable is less than or equal to that value, BASIC interprets the NEXT command as, "GO TO the step after the previous FOR command". However, if the incremented value of the variable is greater than the initial value of expression 2 given in the FOR command, BASIC interprets the NEXT command as, "GO TO the next step in numerical sequence after the NEXT command".

The FOR loop is always executed at least once, even when expression 1 is initially greater than expression 2.

Thus, using the FOR and NEXT commands, the program given above could also be written as:

```
>10 FOR I = 1 TO 100 (RET)
>20 PRINT I (RET)
>30 NEXT I (RET)
>40 PRINT "FINISHED" (RET)
>RUN (RET)
```

Exactly the same looping procedure is followed; however, it happens automatically by using the FOR command, where

I is the counter variable
1 is the initial value of I
I > 100 is the test for completion
1 is the increment to be added to I.

In this sample program the "body" of the loop consists of one step (step 20). The body of the loop may be any number of steps, but it is always terminated by the NEXT command. When the loop is finished, the next step executed is the step following the NEXT command.

In some program loops it is necessary to increment the counter variable by a value other than 1. This can be accomplished by adjusting the value of the counter variable before the NEXT command is executed. For example, to find and print all even numbers in the range 50 through 76, the following program could be used:

```
>10 FOR X = 50 TO 76 (RET)
>20 PRINT X (RET)
>25 X = X + 2 (RET)
>30 NEXT X (RET)
>40 PRINT "FINISHED" (RET)
>RUN (RET)
```

A simpler way of doing this is to use a STEP clause in the FOR command, as in the format

step FOR variable = expression 1 TO expression 2
[STEP expression 3] (RET)

where expression 3 is the increment to be added to the variable when the NEXT command is executed and where the keyword STEP is in no way related to the idea of a step number. Thus, the above program can also be written as

```
>10 FOR X = 50 TO 76 STEP 2 (RET)
>20 PRINT X (RET)
>30 NEXT X (RET)
>40 PRINT "FINISHED" (RET)
>RUN (RET)
```

According to the looping procedure

X is the counter variable
50 is the initial value of X
X > 76 is the test for completion
2 is the increment to be added to X

Note that the increment is assumed to be 1 unless a STEP clause is added to the FOR command.

It is often useful to have loops within loops. These "nested" loops can be expressed with FOR and NEXT commands. In the following skeleton examples, the enlarged brackets mark the body of the loop.

Legal	Illegal
FOR X	FOR X
FOR Y	FOR Y
NEXT Y	NEXT X
NEXT X	NEXT Y
FOR X	FOR Z
FOR Y	FOR Z
FOR Z	NEXT Z
NEXT Z	NEXT Z
FOR W	FOR X
NEXT W	FOR Y
NEXT Y	FOR X
FOR Z	NEXT X
NEXT Z	NEXT Y
NEXT X	NEXT X

USE OF SUBSCRIPTS

The concept of subscripting and arrays becomes extremely useful in relation to programming loops. Consider the following table, which lists the quantity of each type of item sold by each of five salesmen in one week.

	Salesman				
	Jones	Smith	Brown	Doe	White
Item 1	40	20	37	29	42
Item 2	10	16	3	21	8
Item 3	35	47	29	16	33

The price of each item is listed in the following table:

Item	Price
1	$1.25
2	$4.30
3	$2.50

In the following discussion, the quantities of items in the first table are regarded as the two-dimensional array Q(I,S) where I is the item number (row reference) and S is the salesman(column reference). The prices of the items are regarded as the one-dimensional array P(I) where I is the item number.

The following program calculates the total sales in dollars for each salesman using data from the preceding tables:

```
>10 FOR I = 1 TO 3 (RET)
>20 READ P(I) (RET)
>30 NEXT I (RET)
>40 FOR I = 1 TO 3 (RET)
>50 FOR S = 1 TO 5 (RET)
>60 READ Q(I,S) (RET)
>70 NEXT S (RET)
>80 NEXT I (RET)
>90 FOR S = 1 TO 5 (RET)
>100 T = 0 (RET)
>110 FOR I = 1 TO 3 (RET)
>120 T = T + P(I) * Q(I,S) (RET)
>130 NEXT I (RET)
>140 PRINT"TOTAL SALES FOR SALESMAN"S,"$"T (RET)
>150 NEXT S (RET)
>200 DATA 1.25, 4.30, 2.50 (RET)
>210 DATA 40,20,37,29,42 (RET)
>220 DATA 10,16,3,21,8 (RET)
>230 DATA 35,47,29,16,33 (RET)
>RUN (RET)
```

Steps 10 through 30 read in the values of the list P. Steps 40 through 80 read in the values of the table Q. Steps 90 through 150 compute T (the total sales for each of the five salesmen) and print each answer as it is computed. The calculation for a single salesman takes place in steps 100 through 130. In steps 90 through 150, the letter I stands for the item and the letter S stands for the salesman.

DIM

BASIC automatically provides 11 locations in memory for a one-dimensional array, so that the subscript may vary from 0 to 10. If the user wants to allow for a longer array, he must specify its subscript range with the DIM (dimension) command, which has the format

step DIM array 1[,array 2]. . . (RET)

where each array has the form

letter (expression)

Each expression must evaluate as zero or a positive integer value, which specifies the upper subscript limit of the array as the value of the expression. (The lower subscript limit is 0).

Examples:

Statement 10, below, declares an array of 60 elements.

```
10 DIM Q(59) (RET)
20 DIM B(A + X),Z(A(X)) (RET)
```

Similarly, the subscripts of a two-dimensional array may each vary from 0 to 10. This allows for 11 x 11 or 121 elements. If a larger table is desired, the user may use the DIM command with each array having the form

letter (expression,expression)

Each expression must evaluate as zero or a positive integer, which specifies the upper subscript limit for the corresponding dimension of the array.

Examples:

Statement 1, below declares an array of 41 x 51 elements.

```
1 DIM Y(40,50) (RET)
2 DIM X(Y,Z)A(1,1) (RET)
3 DIM A(X(1),X(2)) (RET)
4 DIM L(M+ N * N) (RET)
```

The effect of a DIM statement on the subscripted variable whose dimensions are defined is similar to the effect of assignment statements on the variable that is assigned a value. The major differences are: (1) The DIM statement changes more than one value. In fact, all values of all the variables in a DIM statement are changed. (2) The new value generated by a DIM statement is "no value". Variables that appear in DIM statements have no defined values. They should not be used until a value is assigned to them subsequent to their appearance in the DIM statement.

```
>5 DIM X(4)
>10 X(3) = 3
>15 PRINT X(3)
>20 DIM X(4)
>30 PRINT X(3)

>RUN
 3
UNDEFINED SUBSCRIPTED VARIABLE 30
```

MATRIX OPERATIONS

Just as BASIC has a number of predefined standard functions, there are a number of operations with subscripted variables or matrixes for which BASIC has provided abbreviated forms.

INPUT OF MATRIX DATA

The MAT READ statement will take values from DATA statements to fill the subscripted variables named in the MAT READ statement. The format of the command is:

step MAT READ array 1[,array 2]... (RET)

The dimensions of the variables must be specified either in a previous DIM statement or in the MAT READ statement itself.

```
10 DIMENSION A(3),B(4)
15 MAT READ A,B
20 DATA 1, 2, 3, 1, 7, 8, 9
```

Statements 10 and 15 could have been replaced with

```
10 MAT READ A(3),B(4)
```

The MAT READ statement can be used to dimension an array or to respecify an array that has previously been dimensioned.

The MAT READ statement fills a two-dimensional subscripted variable, or matrix, row by row. For example,

```
10 MAT READ A(I,J)
```

is equivalent to

```
10 DIM A(I,J)
11 FOR I1=0 TO I
12 FOR J1=0 TO J
13 READ A(I1,J1)
15 NEXT J1
16 NEXT I1
```

The command MAT INPUT is available and functions similarly to the MAT READ command, except that the data values are taken from the Teletype rather than a DATA block. The format for the command is

step MAT INPUT array 1[,array 2]... (RET)

The user must supply data in the same format that is required by the INPUT command. The data values must be separated by commas, blanks, or carriage returns.

OUTPUT OF MATRIX DATA

An array can be output by the command MAT PRINT

step MAT PRINT array 1[,array 2]... (RET)

which causes a matrix to be printed row by row in zoned format. If the variables are separated by semicolons rather than commas, the matrix will be printed in packed format. (See the PRINT command for a discussion of zoned and packed formats.) The command

```
10 MAT PRINT  A(3,3),B (RET)
```

will first print the matrix A and then the matrix B in zoned format. Alternatively, the matrixes could have been output by

```
10 MAT PRINT  A(3,3);B; (RET)
```

A vector such as V(N) is considered to be a column vector, and is printed vertically. In order to print a row vector, it would have to be dimensioned as V(0,N).

MATHEMATICAL OPERATIONS WITH MATRIXES

RESETTING A MATRIX

All the elements of an array can be reset to 0 by the command:

step MAT array = ZER (expression 1[,expression 2]) (RET)

The dimension of the array must have been previously defined or specified in the ZER statement itself. Example:

```
10 MAT C = ZER
15 MAT A = ZER(3)
```

Statement 10 will reset an array named C. Statement 15 will generate an array of zeros (A(0)=0, A(1)=0, A(2)=0, A(3)=0).

SETTING A MATRIX TO ALL 1s

The CON (constant) command is similar to ZER, except that it sets all of the elements of the array to 1. The command has the form

step MAT array = CON [(expression 1[,expression 2])] (RET)

It can also be used to specify and dimension (or redimension) a matrix.

SETTING IDENTITY MATRIX

This command requires a two-dimensional array where the two dimensions are the same size (square matrix). It also may deal with a previously dimensioned array, or define (or redefine) the dimension of the array. The command

step MAT array = IDN [(expression 1,expression 2)] (RET)

defines an array where the diagonal elements (that is, those elements whose subscript numbers are the same) are 1, and all other elements of the array are 0. For example,

```
10 MAT D = IDN(K,K)
```

is equivalent to

```
10 MAT D = ZER(K,K)
11 FOR I = 0 TO K
12 D(I,I) = 1
13 NEXT I
```

MATRIX ADDITION AND SUBTRACTION

Two arrays may be added or subtracted by the statements

step MAT array 1 = array 2 + array 3 (RET)
step MAT array 1 = array 2 - array 3 (RET)

The three arrays must have been dimensioned previously and must be of the same dimensions. The statements

```
 10 DIM  A(I,J),B(I,J),C(I,J)
100 MAT  A = B + C
```

are equivalent to

```
 10 DIM  A(I,J),B(I,J),C(I,J)
 :
 :
100 FOR K = 0 TO I
101 FOR L = 0 TO J
102 A(K,L) = B(K,L) + C(K,L)
103 NEXT L
104 NEXT K
```

Since the matrix operations are only convenient short forms, and not operations integrated into BASIC, they cannot be combined. For example, to add a third array to the difference of two others requires

```
10 DIM  A(L),B(L),C(L),D(L)
 :
 :
50 MAT  A = B - C
60 MAT  A = A + D
```

Note: This means that the form MAT A=B-C+D is illegal.

MATRIX MULTIPLICATION

An array can be multiplied by a scalar value via the statement

step MAT array 1 = (expression) * array 2 (RET)

The parentheses around the expression must appear, and the two arrays must be of the same dimension.

Though there is no command of the form MAT C = A, the statement

```
10 MAT C = (1) * A
```

performs the desired operation.

An array can be multiplied by another array via the statement

step MAT array 1 = array 2 * array 3 (RET)

In order for this statement to execute properly, the arrays must be conformable; that is, the number of rows of the first is equal to the number of rows of the second, the number of columns of the first is equal to the number of columns of the third, and the number of columns of the second is equal to the number of rows of the third. The statements

```
10 DIM  A(I,J),B(I,K),C(K,J)
 :
 :
75 MAT  A = B * C
```

are equivalent to

```
10 DIM  A(I,J),B(I,K),C(K,J)
 :
 :
75 MAT  A = ZER (I,J)
76 FOR I1 = 0 TO I
77 FOR J1 = 0 TO J
78 FOR K1 = 0 TO K
79 A(I1,J1) = A(I1,J1) + B(I1,K1)*C(K1,J1)
80 NEXT K1
81 NEXT J1
82 NEXT I1
```

MATRIX TRANSPOSING AND INVERTING

To use the transpose function

step MAT array 1 = TRN(array 2) (RET)

each array must have as many rows as the other array has columns. The statements

```
 10 DIM  A(I,J),B(J,I)
 :
 :
100 MAT  A = TRN(B)
```

are equivalent to

```
110 DIM  A(I,J),B(J,I)
 :
 :
100 FOR I1 = 0 TO I
101 FOR J1 = 0 TO J
102 A(I1,J1) = B(J1,I1)
103 NEXT J1
104 NEXT I1
```

This command will work for vectors as well; thus, the following will transform the column vector V into the row vector W.

```
10 DIM V(N),W(0,N)
15 MAT  W = TRN(V)
```

The inverse function operates on identically dimensioned square matrixes. This function produces array 1 which, when multiplied by array 2, yields the identity array.

```
step  MAT  array 1 = INV(array 2) (RET)
```

Thus,

```
 10 DIM  A(N,N),B(N,N),C(N,N)
 :
 :
100 MAT  A = INV(B)
110 MAT  C = A * B
120 MAT  B = IDN
130 MAT  C = C - B
```

would yield approximately the same array C as would

```
10  MAT  C  =  ZER(N,N)
```

LIMITATIONS ON MATRIX OPERATORS

The expressions that can be formed by using the matrix operators are limited. Some samples are shown below.

Incorrect	Correct
10 MAT C = A	10 MAT C = (1)*A
15 MAT C = A+B+D	15 MAT C = A+B 16 MAT C = C+D
30 MAT C = ZER+A	30 MAT C = ZER 31 MAT C = C+A
40 MAT PRINT TRN(A)	40 MAT B = TRN(A) 41 MAT PRINT B

All array variables become undefined when a transfer is made from the command to the execution mode of BASIC. The scalar variables, however, retain whatever value they acquired the last time the program was executed. Study the sample program shown below. Notice that the statement "GO TO 70" represents a transfer from the command to the execution mode.

```
>10 DIM G(0,9)
>20 A=3
>22 PRINT A
>25 FOR I= 0 TO 9
>30 G(0,I) = I
>40 NEXT I
>45 MAT H = ZER(0,9)
>60 MAT H = (1) * G
>70 MAT PRINT H;
>RUN
 3
 0     1     2     3     4     5     6     7     8     9


>PRINT H(0,7)
 7

>GO TO 70
UNDEFINED SUBSCRIPTED VARIABLE 70
>PRINT H(0,7)
UNDEFINED SUBSCRIPTED VARIABLE
>PRINT G(0,8)
UNDEFINED SUBSCRIPTED VARIABLE
>GO TO 22
 3
MATRICES NOT SAME SIZE 60
>GO TO 10
 3
 0     1     2     3     4     5     6     7     8     9
```

INPUT/OUTPUT COMMANDS

All data permanently stored at the computer installation is kept in files on a large disc memory system. Each user has his own files and file directory (for further description of files see the SDS 940 Terminal User's Guide).

OPEN

Often the values to be assigned to program variables are located in a permanent file. To read a permanent file, it must be in the user's file directory and it must be opened using the OPEN command. Similarly, program output may be written on a file rather than on the Teletype. Again, the file must appear in the user's file directory and must be opened using the OPEN command. The OPEN command has the format

step OPEN/file-name/, {INPUT / OUTPUT} (RET)

where file-name is the name of a file in the user's file directory.

Examples:

```
10 OPEN/DATA1/, INPUT (RET)
20 OPEN/XYZ/, INPUT (RET)
30 OPEN/MASTER/, OUTPUT (RET)
40 OPEN/F1/, OUTPUT (RET)
```

The OPEN command resets the file's location counter to its beginning so that the first value subsequently input from or output to the file will be the first value of the file. A file cannot be open for input and output simultaneously. A user can have a maximum of two files opened at the same time. However, one of the files must be opened for input, while the second file is opened for output. It is not possible to have two files open for input (or output) at the same time.

If a user has one file open for input (or output), and opens a second file for input (output), the first file is automatically closed. When a file is closed, it is no longer available to the program for either input or output until it is opened again.

It is important to remember that a disc file must be named in a user's file directory before it is used in a BASIC program.

INPUT FILE

Once the file is opened for input, the user may read it by using the INPUT FILE command, which has the format

step INPUT FILE variable 1[,variable 2]... (RET)

Each time the INPUT FILE command is encountered during execution, the next value appearing on the file most recently opened for input is read and assigned to the next variable in the list of variables.

Examples:

```
50 INPUT FILE X (RET)
60 INPUT FILE A1,B,Z (RET)
```

Only legal BASIC numbers can be read from a file. The numbers in the data file can be separated by commas (same format as the DATA statement) or they can be separated by spaces. The only non-numeric characters allowed are: comma, space, E (for exponential notation), plus sign, minus sign, and carriage return. A carriage return ends a line. A line containing nothing but a space (or spaces) and a carriage return is incorrect and cannot be read.

The following technique can be used to open a file for input (output) which had been opened for output (input).

```
 5 OPEN /A/, INPUT (RET)  Opens /A/ for input
10 INPUT FILE X, Y, Z (RET)  Inputs data from /A/
15 OPEN /B/, INPUT (RET)  Closes /A/, Opens /B/
                          for input
20 INPUT FILE Q, R, S (RET)  Inputs data from /B/
25 OPEN /C/, INPUT (RET)  Closes /B/, Opens /C/
                          (a dummy) for input
30 OPEN /B/, OUTPUT (RET) B now available for output
35 PRINT FILE G, H, I, J (RET) Outputs data to file /B/
```

PRINT FILE

Once the file is opened for output, the user may write on it using the PRINT FILE command, which has the format

step PRINT FILE expression 1 ,expression 2 ...

Each time the PRINT FILE command is encountered during execution, the value of each expression appearing in the list of expressions is appended to the file most recently opened for output in the same order as given in the expression list.

Examples:

```
70 PRINT FILE X, Y (RET)
80 PRINT FILE X+Z/100, I (RET)
```

The INPUT FILE command describes the format that the numbers output by the PRINT FILE command must have. In addition, the PRINT FILE command can be used to output text on a file. The text must be enclosed by quotation marks in the same manner as the PRINT command. However, these files cannot be read by the INPUT FILE command.

The following small programs illustrate one method of utilizing data files. Briefly, the first program creates a master file consisting of an employee number, hourly rate, and the number of hours worked for each company employee.

The second program reads the master file, calculates and prints the weekly salary for each employee (time and a half for overtime), and calculates total payroll.

The third program reads the master file and calculates company overtime.

```
>110 OPEN/MASTER/, OUTPUT (RET)
>111 INPUT E, R, H (RET)
>112 PRINT FILE E, R, H (RET)
>113 IF E > 0 THEN 111 (RET)
>114 END (RET)
```

Note that the above program will input values for E, R, and H and output these values to the opened file until a value of zero is input for E. When the user inputs a zero for E, it indicates that all the data has been input.

```
>209 S = 0 (RET)
>210 OPEN/MASTER/,INPUT (RET)
>211 INPUT FILE E,R,H (RET)
>212 IF E = 0 THEN 220 (RET)
>213 IF H > 40 THEN 218 (RET)
>214 P = R * H (RET)
>215 PRINT E,P (RET)
>216 S = S + P (RET)
>217 GO TO 211 (RET)
>218 P = 40 * R + (H-40) * R * 1.5 (RET)
>219 GO TO 215 (RET)
>220 PRINT "TOTAL PAYROLL",S (RET)
>221 END (RET)

>309 S = 0 (RET)
>310 OPEN/MASTER/,INPUT (RET)
>311 INPUT FILE E,R,H (RET)
>312 IF E = 0 THEN 316 (RET)
>313 IF H < 40 THEN 311 (RET)
>314 S = S + (H-40) (RET)
>315 GO TO 311 (RET)
>316 PRINT "COMPANY OVERTIME",S (RET)
>317 END (RET)
```

BINARY FILE INPUT/OUTPUT

The data files handled by the INPUT FILE and PRINT FILE commands are symbolically coded decimal files. Another type of file, called a binary file, is available. The numbers are not coded in decimal, but are in binary form which requires less space. Binary files are useful if a program creates large amounts of data that are to be input to another program.

READ, FILE, AND WRITE

The format for the read and write commands is:

```
step  WRITE expression 1[,expression 2]... (RET)
step  READ FILE expression 1[,expression 2]... (RET)
```

The list of expressions to be input (output) must be separated by commas. The list must end with a carriage return (semicolons or commas cannot be used).

Files created by the WRITE command can only be read by the READ FILE command.

A file must be opened before it can be processed by the WRITE or READ FILE commands. The same rules that govern the opening of symbolic files apply to binary files as well.

PRINT

The PRINT command may be used simply and directly, as previously described, or it may be used in conjunction with output format options for the programmer who wants formatted output. For this purpose, there are three format types: zoned, packed, and compressed.

Zoned Format

The Teletype line is divided by BASIC into five zones of fifteen spaces each, which allows for the printing of up to five items per line. A comma is a signal to BASIC to move to the next print zone, or to the first print zone of the next line if it has just filled the fifth print zone. The termination of a PRINT statement signals a new line (unless a comma is the last symbol). Each number occupies one zone, whereas text occupies an integer number of zones; that is, if text occupies part of a zone, the rest of the zone is filled with blanks. If text runs through the fifth zone, part of it may be lost.

Packed Format

The user may specify that output is to be printed in packed format by separating the expressions with the semicolon instead of the comma. Whereas the comma causes BASIC to move to the next zone to print the next item, the semicolon causes BASIC to move to the beginning of the next multiple of three characters to print the next answer. The termination of a PRINT statement signals a new line (unless a semicolon is the last symbol). Thus, with packed output, the user can print eleven three-digit numbers per line, eight six-digit numbers per line, or six nine-digit numbers per line. A more extensive demonstration of packed format is shown in Figure 1 below.

Compressed Format

If two parts of text are separated only by the usual quotes, they will be printed without spaces. Similarly, if text is followed by an expression without a comma or semicolon, the value will be printed immediately following the text (a blank replaces the plus sign for non-negative numbers).

Examples:

Zoned Format

```
>1 FOR X=1 TO 3 (RET)
>2 PRINT "X=" X, "X2=" X↑2, (RET)
>3 NEXT X (RET)
>4 END (RET)
>RUN (RET)
X = 1       X2 = 1       X = 2       X2 = 4       X = 3
X2=9
```

Packed Format

```
>2 PRINT "X=" X; "X2=" X↑2; (RET)
>RUN (RET)
X = 1 X2 = 1 X = 2 X2 = 4 X = 3 X2 = 9
```

Compressed Format

```
>2 PRINT "X="X"X2="X↑2 (RET)
>RUN (RET)
X=1X2=1
X=2X2=4
X=3X2=9
```

Printing a Blank Line

A blank line may be output by

```
step  PRINT (RET)
```

```
>100 FOR X = 100 TO 125 (RET)
>110 PRINT X;(RET)
>120 NEXT X(RET)
>130 END(RET)
>RUN (RET)
 100   101   102   103   104   105   106   107   108   109   110
 111   112   113   114   115   116   117   118   119   120   121
 122   123   124   125
>100 FOR X = 100000 TO 100025 (RET)
>110 PRINT X; (RET)
>120 NEXT X (RET)
>130 END (RET)
>RUN (RET)
 100000   100001   100002   100003   100004   100005   100006   100007
 100008   100009   100010   100011   100012   100013   100014   100015
 100016   100017   100018   100019   100020   100021   100022   100023
 100024   100025
>100  FOR X = 1000000 TO 1000013(RET)
>110  PRINT X; (RET)
>120  NEXT X (RET)
>130  END (RET)
>RUN (RET)
 1.E+06    1.000001E+06    1.000002E+06    1.000003E+06    1.000004E+06
 1.000005E+06    1.000006E+06    1.000007E+06    1.000008E+06    1.000009E+06
 1.00001E+06    1.000011E+06    1.000012E+06    1.000013E+06
>
```

Figure 1. Example of BASIC Packed Format

3. FUNCTION$ AND SUBPROGRAMS

FUNCTIONS

The standard functions that BASIC provides are listed under "Mathematical Functions" in Chapter 1. The manner in which they are used is very simple. To compute $y = \sqrt{1+x^2}$ the programmer would write

```
Y = SQR (1 + X ↑ 2) (RET)
```

The expression enclosed in parentheses is called the argument. The other standard functions are used in the same way; that is, the function name is followed by the argument enclosed by parentheses, as shown by the format

function-name (expression)

Examples:

```
LOG(Y)
SIN(X ↑ 2)
ABS(X + Y + Z)
```

Two additional functions that are in the BASIC repertory but which have not been described are INT and RND.

INT

The INT (integer) function is used to determine the integer part of a number that might not be a whole number. Thus INT (7.8) is equal to 7. As with the other functions, the argument of INT may be any expression. INT always operates by truncating the fractional part, whether the number is positive or negative.

One use of INT is to round numbers to the nearest integer. If the value of X, for example, is positive, it may be rounded by using the statement INT(X+.5). If the value of X is negative, however, the statement INT(X-.5) must be used because a number like -7.8, rounded, is -8 not -7. INT can be used to round to any number of decimal places. For positive values of X, for example, the statement INT(100*X+.5)/100 will round X to two decimal places.

RND.

The RND. function is a pseudo-random number generator. When called, it will produce a number between zero and one. For example, the command

```
PRINT    RND.(RET)
```

would cause BASIC to print

```
.502793
```

When the function is called repeatedly, it will produce a sequence of pseudo-random numbers. For example, the command

```
PRINT    RND., RND., RND. (RET)
```

would cause BASIC to print

```
.502793      .2311643      .3898417
```

RND may be called with a meaningless argument, as in PRINT RND(X), but must be followed by either an argument or a period.

The same sequence of pseudo-random numbers will occur in every program that uses the RND function. This feature is useful for debugging programs.

DEF

The DEF command permits the user to define an abbreviation so he will not have to repeat an arithmetic expression each time he uses it in his program. The name of an abbreviated expression must be three letters, the first two of which are FN. Thus the user may define up to 26 abbreviations. DEF commands have the format

step DEF FN letter 1(letter 2) =expression (RET)

where

letter 1 is the third letter of the user-defined abbreviation,

letter 2 denotes an unsubscripted variable that is initially set to the value of the argument used in the call for the user-defined function, and

expression may be any expression that can fit into one line. It may not include another user-defined function, but may include standard functions (like SIN and SQR) and may involve other variables besides the one denoting the argument of the function. However, the variables used in the expression must not be subscripted.

Examples:

```
25 DEF FNF(Z) = (Z*3.14159265/180) (RET)
40 DEF FNL(X) = LOG (X) / LOG (10) (RET)
```

Thus, step 25 defines FNF as the function "sine of Z degrees" and step 40 defines FNL as the function "log-to-the-base-ten of X".

The DEF command must occur before it is used in the program. In a program containing FNF as defined above, the variable Z takes on a new value each time the function FNF is called.

As another example:

```
60 DEF FNX(X) = SQR (X*X + Y*Y) (RET)
```

may be used to set up an expression for the square root of the sum of the squares of X and Y. To use FNX, one might write the following:

```
>10 Y = 30 (RET)
>20 S1 = FNX(40) (RET)
```

In this case, S1 is set to the value 50 as the result of step 20.

The safest practice is to avoid using, elsewhere in a program, the same variable that was used in a DEF command to define an expression.

When a statement containing an abbreviated expression is executed, the result is equivalent to what would have occurred had the expression been written out in full and had the argument variable been redefined. For example,

```
>10 DEF  FNF(X) = X + 1
>20 X = 5
>30 Y = 8
>40 Y = FNF(Y)
```

is identical in effect to

```
>20 X = 5
>30 Y = 8
>35 X = Y
>40 Y = X + 1
```

where lines 35 and 40, here, are identical in effect to line 40 in the former program.

Each program, when run, would leave X = 8 and Y = 9.

It should be noted that one does not need DEF unless the abbreviated expression must appear at two or more locations in the program. Thus,

```
>10 DEF  FNF(Z) = SIN(Z*P) (RET)
>20 P = 3.14159265/180 (RET)
>30 FOR X = 0 TO 90 (RET)
>40 PRINT X, FNF(X) (RET)
>50 NEXT X (RET)
>60 END (RET)
```

might be more efficiently written as

```
>20 P = 3.14159265/180 (RET)
>30 FOR X = 0 TO 90 (RET)
>40 PRINT X, SIN(X*P) (RET)
>50 NEXT X (RET)
>60 END (RET)
```

to compute a table of values of the sine function in degrees.

SUBPROGRAMS

The use of DEF is limited to those cases where the value of the expression can be computed within a single BASIC statement. Often much more complicated functions, or perhaps even sections of a program that are not functions, must be calculated at several places within the program. For this, the GOSUB command may be useful.

GOSUB/RETURN

The GOSUB command has the format

step 1 GOSUB step 2

Example:

200 GOSUB 400 (RET)

The effect of the GOSUB command is exactly the same as a GOTO command except that BASIC notes where the GOSUB command is in the program.

The RETURN statement complements the GOSUB statement.

step RETURN (RET)

As soon as a RETURN command is encountered, the computer automatically goes back to the command immediately following the most recently executed GOSUB command. As a skeleton example, the following program will input N numbers. The subroutine starting at 200 will sort the numbers into ascending sequence and return to the main program. The main program will then output the sorted array.

```
10 INPUT N
11 DIM A(N),B(N)
12 IF N=0 THEN 100
15 FOR I = 1 TO N
20 INPUT A(I)
25 NEXT I
30 GOSUB 200
35 FOR I= 1 TO N
40 PRINT A(I);
45 NEXT I
50 GO TO 10
100 STOP
200 FOR J= 1 TO N
205 X=A(J)
210 FOR I=J TO N
215 IF A(I) > X THEN 225
216 T=X
220 X=A(I)
221 A(I)=T
225 NEXT I
230 A(J)=X
235 NEXT J
240 RETURN
```

The following program will calculate the factorial of any number X that is input. Note that the subroutine is recursive.

```
>10 INPUT X
>20 IF X<1 THEN 10
>30 X=INT(X)
>40 MAT F=CON(X)
>50 MAT F = (-1)*F
>60 F(0)=1
>70 F(1)=1
>80 GO SUB 110
>90 PRINT X;F(X)
>100 GO TO 10
>110 IF F(X) >-1 THEN 180
>120 X=X-1
>130 GO SUB 110
>160 X=X+1
>170 F(X)=F(X-1)*X
>180 RETURN
>

 RUN
? 1
 1          1
? 2
 2          2
? 3
 3          6
? 4
 4          24
? 5
 5          120
? 6
 6          720
?
```

4. PROGRAM PREPARATION AND EXECUTION

PROGRAM INPUT FROM THE TELETYPE

There are several methods for preparing a BASIC program for execution. The first method is to type the program directly into BASIC. This is usually the procedure used for small and medium sized programs. For example:

```
-BASIC (RET)
>100 PRINT "THIS IS A SORTING PROGRAM" (RET)
>110 INPUT N (RET)
  :
```

PROGRAM INPUT FROM PAPER TAPE

It is often convenient as well as economical to type longer programs on paper tape while off line. To do this, the Teletype is placed in (1) LOCAL MODE, (2) HALF DUPLEX MODE, and (3) PAPER TAPE PUNCH ON. The statements are typed just as though the user were connected to the computer, with one exception. Following each line, the user must type a carriage return and a line feed. When connected to the computer, the user types only a carriage return and the computer performs the line feed.

If the user has punched his program on a paper tape, he can enter the text into BASIC with the following procedure:

```
-BASIC (RET)
>LOAD TELETYPE (RET)
```

After this, BASIC is waiting for the program; when the user turns on the paper tape reader, the program reads in.

BASIC is unable to distinguish typed characters from those that are read from the paper tape reader. Therefore, an alternate way of entering a paper tape is to turn on the paper tape reader without using the LOAD command:

```
-BASIC
(turn on reader)
>100 PRINT "THIS IS A SORTING PROGRAM"
>110 INPUT N
  :
```

PROGRAM ON FILE

If the program to be run has been previously prepared and is now located in a file on the disc, the user may type the command LOAD to bring his program into memory. The format of the LOAD command is

```
LOAD (RET)
```

In response, BASIC types

```
FROM: /file-name/ (RET)
```

Examples:

```
LOAD (RET)
FROM: /PROG/(RET)
```

MISCELLANEOUS BASIC COMMANDS

REM

An important part of any computer program is the description of what it does, and what data should be supplied. One way of documenting a program is to supply remarks along with the program itself. BASIC provides this capability with the REM (remark) command. For example, consider the difficulty of identifying the program for the following equation

$$\text{Mean} = \frac{\Sigma x_i}{N} \qquad SD = \sqrt{\frac{\Sigma(x_i)^2 - \frac{(\Sigma x_i)^2}{N}}{N-1}}$$

if it were not labeled with REM statement 100.

```
>100 REM MEAN AND STANDARD DEVIATION (RET)
>100 S = 0 (RET)
>120 Q = 0 (RET)
>125 INPUT N (RET)
>130 FOR I = 1 TO N (RET)
>140 INPUT A(I) (RET)
>150 S = S + A(I) (RET)
>160 Q = Q + A(I) * A(I) (RET)
>170 NEXT I (RET)
>180 PRINT "MEAN=" S/N (RET)
>190 D = SQR(((Q-(S * S) / N)) / (N-1)) (RET)
>200 PRINT "SD=" D (RET)
>210 END (RET)
```

END

An END command indicates the termination point of a program. When the END command is encountered, it causes BASIC to stop executing the program and to await further commands.[t]

RUN

The user types RUN to begin execution. The program always begins with the smallest step number and executes according to ascending step numbers. To begin execution in the middle of the program, the user can use the GO TO command as a direct command.

STOP

The STOP command is used to stop program execution. When the program execution halts at the STOP command, the user may examine relevant variables. He may then, if he so desires, issue a command GO TO s, where s is the step number following the STOP command. The GO TO command will then cause execution to continue at step number s.

[t] A program will always stop execution and will return control to BASIC when it has executed the last executable statement.

LIST

To list all or some of the steps of a program, the user types a LIST command, which has the format

LIST [step 1 {[-step 2] / [, step 2]}...] (RET)

Examples:

Statement	Meaning
LIST (RET)	list entire program
LIST 100 (RET)	list step 100
LIST 100-500 (RET)	list steps 100 through 500
LIST 100, 200, 300 (RET)	list steps 100, 200, and 300

DEL

To delete one or more steps of a program, the user types a DEL command, which has the format

DEL {step 1 {ALL / [-step 2] / [, step 2]...}} (RET)

Examples:

Statement	Meaning
DEL ALL (RET)	delete entire program
DEL 100 (RET)	delete step 100
DEL 100-500 (RET)	delete steps 100 through 500
DEL 100, 200, 300 (RET)	delete steps 100, 200, and 300

To delete one step, the user may simply type the step number followed by a carriage return, for example,

>125 (RET)

DUMP

To save a program on a permanent file, the user types a DUMP command, which has the format

DUMP [step 1 {[-step 2] / [, step 2]...}] (RET)

Examples:

Statement	Meaning
DUMP (RET)	dump entire program
DUMP 100-210 (RET)	dump steps 100 through 210
DUMP 100, 200, 221 (RET)	dump steps 100, 200, and 221

In response to the DUMP command, BASIC will print

ON:

on the line following the DUMP command. The user then types the name of the file on which he wishes to dump the desired portion of the program. The appropriate file name must appear between slashes, as follows:

/file-name/ (RET)

In response to the user's designation of the file, BASIC determines whether the file name currently exists in the user's file directory. If the name does exist, BASIC will print

OLD FILE

on the following line; however, if the name does not exist, BASIC will add the name to the user's file directory and print

NEW FILE

on the following line.

Examples:

```
>DUMP (RET)
ON:/SAVE/ (RET)
OLD FILE

>DUMP 100-120 (RET)
ON:/PGM/ (RET)
NEW FILE

>DUMP 100, 200, 221 (RET)
ON:/P1/ (RET)
OLD FILE
```

The user must type a carriage return, after BASIC responds with NEW FILE or OLD FILE, to confirm that he wishes the dump to occur. If the user does not want the dump to occur, he responds with ESCAPE and the DUMP command will be aborted.

SAMPLE SESSION AT THE TELETYPE

The following is a demonstration of a sample session at the Teletype using BASIC. The user writes a program that will calculate the area and circumference of a circle. He saves this program on a file named /CIRCLE/. He also uses a previously written program, named /FC/ that will convert any farenheit temperature to centigrade.

```
SDS 940 TIME SHARING SERVICES 2.1
PLEASE LOG IN: A7;100;BET
```

User typed his password immediately after typing A7 and before typing the semi-colon.

```
READY   6/20   11:44

-BASIC

>10 PRINT
>15 PRINT "RADIUS=
MISSING "  15
>15 PRINT "RADIUS=";
>20 INNPUT R
N ILLEGAL  20
>20 INPUT R
>25 P=3.1415926
>30 C=2*P*E←R
>35 A=C*R
>40 PRINT"CIRCUM="R, "AREA="A
>50 GO TO 20
>22 IF R=0 THEN 60
>60 STOP
>
```

Prints a blank line
User forgot terminal "

BASIC did not recognize the command

Shift 0 used to delete the E (shift O prints ←)

```
 50 GO TO 10
>RUN

RADIUS=? 2
CIRCUM= 2        AREA= 25.13274
```

User replaces original statement 50

Error in circumference

```
RADIUS=? (ESC)
>40 PRINT"CIRCUM="C,"AREA="A
>RUN

RADIUS=? 2
CIRCUM= 12.56637              AREA= 25.13274

RADIUS=? 4
CIRCUM= 25.13274              AREA= 100.53096

RADIUS=? 0
```

Error in the PRINT statement

Program terminates when R=0

```
>LIST
```

User lists corrected version of program

```
10 PRINT
15 PRINT "RADIUS=";
20 INPUT R
22 IF R=0 THEN 60
25 P=3.1415926
30 C=2*P*R
35 A=C*R
40 PRINT"CIRCUM="C,"AREA="A
50 GO TO 10
60 STOP


>DUMP
ON: /CIRCLE/
 NEW FILE
```

User saves the program

```
>LOAD
FROM: /FC/


>RUN

DO YOU NEED DIRECTIONS FOR USING THIS PROGRAM?
TYPE A 1 FOR YES AND A 2 FOR NO.
? 1
THE PROGRAM WILL ASK YOU TO INPUT A VALUE OF TEMPERATURE
IN FARENHEIT.  IT WILL THEN OUTPUT THE EQUIVALENT TEMP-
ERATURE IN CENTIGRADE.  THIS SEQUENCE WILL BE REPEATED
UNTIL A VALUE OF 7777 IS INPUT.  THE VALUE OF 7777
WILL TERMINATE THE PROGRAM.

FARENHEIT=? 0
CENTIGRADE=-17.777777

FARENHEIT=? 212
CENTIGRADE= 100

FARENHEIT=? 7777

>LIST                                          User wished to see the program


10 PRINT
20 PRINT "DO YOU NEED DIRECTIONS FOR USING THIS PROGRAM?"
25 PRINT "TYPE A 1 FOR YES AND A 2 FOR NO."
30 INPUT A
35 IF A=2 THEN 70
40 PRINT "THE PROGRAM WILL ASK YOU TO INPUT A VALUE OF TEMPERATURE"
45 PRINT "IN FARENHEIT.  IT WILL THEN OUTPUT THE EQUIVALENT TEMP-"
50 PRINT "ERATURE IN CENTIGRADE.  THIS SEQUENCE WILL BE REPEATED"
60 PRINT "UNTIL A VALUE OF 7777 IS INPUT.  THE VALUE OF 7777"
65 PRINT "WILL TERMINATE THE PROGRAM."
70 PRINT
72 PRINT "FARENHEIT=";
75 INPUT F
77 IF F=7777 THEN 100
80 C=5/9*(F-32)
85 PRINT "CENTIGRADE="C
90 GO TO 70
100 STOP

> (ESC)
> (ESC)
-LOGOUT

TIME USED 0:0:46 IN 0:32:17
```

INDEX

R

S

T

V

W

SUMMARY OF BASIC COMMANDS

Matrix Command Syntax	Page
step MAT {READ / INPUT / PRINT} array 1[,array 2]... (RET)	12
step MAT array = {ZER / CON} [(expression 1[,expression 2])] (RET)	12, 13
step MAT array = IDN [(expression 1,expression 2)] (RET)	13
step MAT array 1 = array 2 {+ / - / *} array 3 (RET)	13
step MAT array 1 = (expression) * array 2 (RET)	13
step MAT array 1 = {TRN / INV} (array 2) (RET)	13, 14

SDS 701 South Aviation Blvd./El Segundo, California 90245

EXECUTIVE OFFICES
AND OPERATING DIVISIONS
701 South Aviation Blvd.
El Segundo, Calif. 90245
(213) 679-4511

EASTERN SYSTEMS DEPT.
12150 Parklawn Drive
Rockville, Maryland 20852
(301) 933-5900

PRINTED CIRCUITS DEPT.
600 East Bonita Avenue
Pomona, Calif. 91767
(714) 628-7371

TECHNICAL TRAINING
5250 West Century Blvd.
Los Angeles, California 90045
(213) 679-4511

SALES OFFICES

Eastern

Maryland Engineering Center
12150 Parklawn Drive
Rockville, Maryland 20852
(301) 933-5900

69 Hickory Drive
Waltham, Mass. 02154
(617) 899-4700

Brearley Office Building
190 Moore Street
Hackensack, New Jersey 07601
(201) 489-0100

1301 Avenue of the Americas
New York City, N.Y. 10019
(212) 765-1230

673 Panorama Trail West
Rochester, New York 14625
(716) 586-1500

P.O. Box 168
1260 Virginia Drive
Fort Washington Industrial Park
Fort Washington, Pa. 19034
(215) 643-2130

Southern

Suite 620
State National Bank Bldg.
200 W. Court Square
Huntsville, Alabama 35801
(205) 539-5131

Orlando Executive Center
1080 Woodcock Road
Orlando, Florida 32803
(305) 841-6371

2964 Peachtree Road, N.W.
Suite 350
Atlanta, Georgia 30305
(404) 261-5323

Suite 311-B
First National Bank Office Bldg.
7809 Airline Highway
Metairie, Louisiana 70003
(504) 721-9172

8383 Stemmons Freeway
Suite 233
Dallas, Texas 75247
(214) 637-4340

3411 Richmond Avenue
Suite 202
Houston, Texas 77027
(713) 621-0220

Midwest

2720 Des Plaines Avenue
Des Plaines, Illinois 60018
(312) 824-8147

17500 W. Eight Mile Road
Southfield, Michigan 48076
(313) 353-7360

4367 Woodson Road
St. Louis, Missouri 63134
(314) 423-6200

Seven Parkway Center
Suite 238
Pittsburgh, Pa. 15220
(412) 921-3640

Western

1360 So. Anaheim Blvd.
Anaheim, Calif. 92805
(213) 865-5293

5250 West Century Blvd.
Los Angeles, California 90045
(213) 679-4511

505 W. Olive Avenue
Suite 300
Sunnyvale, Calif. 94086
(408) 736-9193

World Savings Bldg.
Suite 401
1111 So. Colorado Blvd.
Denver, Colo. 80222
(303) 756-3683

Fountain Professional Bldg.
9004 Menaul Blvd., N.E.
Albuquerque, N.M. 87112
(505) 298-7683

Dravo Bldg., Suite 501
225 108th Street, N.E.
Bellevue, Wash. 98004
(206) 434-3991

Canada

864 Lady Ellen Place
Ottawa 3, Ontario
(613) 722-8387

England

London Branch
I.L.I. House
Olympic Way
Wembley Park
Middlesex

INTERNATIONAL
REPRESENTATIVES

France

Compagnie Internationale
pour l'Informatique, C.I.I.

EXECUTIVE AND
SALES OFFICES
66-68 Route de Versailles
78 - Louveciennes
951 86 00 (Paris area)

MANUFACTURING
AND ENGINEERING
Rue Jean Jaures
Les Clayes-sous-Bois
Seine et Oise
950 94 00 (Paris area)

Israel

Elbit Computers Ltd.
Subsidiary of Elron
Electronic Industries Ltd.
88 Hagiborim Street
Haifa
6 4613

Japan

F. Kanematsu & Co. I
Central P.O. Box 141
New Kaijo Building
Marunouchi, Chiyoda-Ku
Tokyo

90 11 11C