Publication Number 29-338R03

BASIC
LANGUAGE REFERENCE MANUAL

INFORMATION CONTAINED IN THIS MANUAL IS SUBJECT TO DESIGN CHANGE OR PRODUCT IMPROVEMENT

INTERDATA®
Subsidiary of PERKIN-ELMER
Oceanport, New Jersey 07757, U.S.A.

January 1975

PREFACE

BASIC is a widely accepted, general purpose, interactive programming language designed to include extensive programming power along with simplicity of use. The INTERDATA 03-055 BASIC Interpreter provides the BASIC language under any existing INTERDATA operating system and at R02 and above for INTERDATA 16-bit and 32-bit processors. INTERDATA BASIC contains all the features of Dartmouth BASIC*, plus various extensions of the language listed below:

Matrix operations
Optional LET statement
Extended IF statement
INPUT and PRINT via logical unit
ON statement
PRINT USING (picture format)
String operations (including string arrays)
Boolean operations
CALL (assembly language subroutine)
File handling (OPEN and CLOSE)
BASIC Batch operations
Rewind, write file mark, backspace
String to floating point conversion
Floating point to string conversion
Program trace

INTERDATA BASIC is compatible for the most part with BASIC written for other systems. Programs to be converted should require only minor modifications.

The INTERDATA BASIC Interpreter is written as a reentrant package and can provide multi-user capability and file handling features within the BASIC language itself when run under the multi-user BASIC Executive (03-058). The BASIC Interpreter requires approximately 11KB.

The INTERDATA BASIC Interpreter supports a single user under BOSS, or a single user with file handling under DOS. It supports single users under OS32ST on the 32-bit processors. It supports multi-user capability under RTOS and OS16MT on the 16-bit processors and under OS32MT on the 32-bit processors. BASIC requires a Model 5, 70, 7/16, 80, 7/32 or 8/32 Processor or equivalent; or a Model 74 or a Basic 7/16 using an OS with floating point trap support.

For information on the availability of related programs or documents, refer to the Software and Documentation Price List, Publication Number 38-076.

*As defined in BASIC PROGRAMMING, (Kemeny and Kurtz, John Wiley and Sons, Inc. #46825 © 1967)

TABLE OF CONTENTS

APPENDICES

CHAPTER 1
INTRODUCTION TO BASIC PROGRAMMING

1.1 BASIC STATEMENTS AND MODES

A user performs operations in the BASIC language by means of statements containing instructions to BASIC. BASIC statements may be up to 72 characters long. In BASIC, there are three modes available to the user:

Immediate Mode

In the Immediate mode, the user issues a statement to BASIC, which is immediately executed. The statement is characterized by the absence of a preceding line number. The Immediate mode is discussed in detail in Chapter 4.

Example: PRINT SIN (.12)

Program Mode

In the Program mode, the user enters BASIC statements to be stored for later execution. A program statement must contain a preceding line number which serves to identify the statement and to indicate the order in which the statement is to be executed.

Example: 100 PRINT X+SIN (.12)

Following the line number, each statement contains a word specifying the type of operation to be performed. All lines present in the user's program area are available for modification, deletion or execution.

Run Mode

The user enters the Run mode to execute program statements entered in the Program mode.

1.2 LINE NUMBERS

Line numbers indicate the order in which statements are evaluated, regardless of the order in which they have been written or entered. They enable the normal order of evaluation to be changed; that is, the execution of the program can branch or loop through designated statements. Line numbers facilitate program debugging by permitting modification of any line without affecting any other lines in the program.

Line numbers must be integers in the range of 0 to 65,535. It is good programming practice to number lines in increments of 5 or 10 (starting with Line 10) when first writing a program, to allow for insertion of additional lines when debugging the program. The system will not allow two lines to have the same line number. The latest line entered replaces the earlier one in its entirety. A line number followed immediately by a carriage return deletes that statement from the program.

When the program is executed with the RUN command, BASIC evaluates the statements in the order of their line numbers starting with the smallest line number and going to the largest.

1.3 BASIC CHARACTER SET

BASIC recognizes the ASCII character set. In addition to the alphanumeric characters, BASIC uses the following special characters for specific functions.

Keys	Function
$	Used to specify string variables.
"	Used to delimit string constants.
< >	Within a string constant string, angle brackets are used to introduce special character codes.
.	Decimal Point.
, (comma)	Used to format output and delimit lists.
;	Used to format output (also acts as a PRINT command).
CARRIAGE RETURN	Used to terminate a line.
()	Used to group arguments in an arithmetic expression and to define subscripts for array elements.
> = <	Relational Operators.
+ - */↑	Arithmetic Operators.

Spaces can be used freely when entering a program to make statements easier to read. They are ignored by BASIC except when contained in string literals; for example:

```
30 LET B = A*2 + 3

30LETB=A*2+3
```

Both of the above statements mean the same to BASIC and are stored the same when entered.

1.4 WRITING A BASIC PROGRAM

Statements may be entered via a Teletype or any other input device. When using a Teletype, the programmer terminates each statement with a carriage return. The following is an example of a BASIC program. This example is explained in the following sections.

```
105 REM - SOLVE A QUADRATIC EQUATION
110 READ A, B, C
120 LET X = B*B-4*A*C
121 IF X  <  0 GOTO 999
122 LET X1 = (-B+SQR (X) )/2*A
130 LET X2 = (-B-SQR (X) )/2*A
240 PRINT X1, X2
400 GOTO 110
500 DATA - 1, 2, 3, + 2, 3, - 4, 2, 1, 6
999 PRINT "NO REAL SOLUTION"
```

In a BASIC program, single letters or single letters followed by a single digit represent program variables. A BASIC program terminates when there are no more program statements, when an END or STOP statement is encountered, or when an error condition occurs. Most programs can be considered to contain three phases.

1. Providing data
2. Performing calculations
3. Printing answers

1.5 PROVIDING DATA

One method of providing data is to write equations that contain the required values. The BASIC statement used for equations is the assignment (LET) statement; for example:

```
120 LET X = 3.1416 * 18.27
```

where: the symbol * means multiplication.

The statement will cause 3.1416 and 18.27 to be multiplied and the result to be stored in a variable named X. However, writing values into equations is not the most efficient means of providing data. Programs are generally used for repetitive computations with a large number of different values. Instead of writing values into an equation, BASIC uses variables that can be assigned different values; for example:

```
120 LET X = 3.1416*Y
```

To provide values, the programmer may use the READ and DATA statements. The READ statement contains the variables that are to be assigned values and the DATA statement provides the values; for example:

```
110 READ Y
120 LET X = 3.1416*Y
130 DATA 40.2, 7.8, 456.41, -.003, 184.6
```

Each time the READ statement is executed, Y assumes a new value in the order of the values in the DATA statement.

1.6 PROGRAM LOOPS

Statements in BASIC programs execute in the sequential order indicated by their statement numbers. However, if a program is completely sequential, it is not possible to perform repetitive calculations on a number of input values.

In the previous example, insertion of the following statement allows the READ and LET statements to be executed more than once:

```
125 GOTO 110
```

The GO TO statement causes a transfer back to statement Number 110. The program reads the second value for Y, (7.8) and executes the LET statement again. The program will continue to loop in this manner until the data is exhausted, causing an error message.

The GO TO statement is a means of transferring control to a part of a program in a non-sequential manner. Another useful statement in transferring control is the IF statement. For example:

```
110 READ Y
115 IF Y < 0 GOTO 110 (skip the LET statement, if Y is negative)
120 LET X = 3.1416*Y
125 GO TO 110
```

Transfer in an IF statement depends upon whether the expression following the word IF is true or false.

1.7 CALCULATIONS

Data provided as input may be computed into answers. A simple arithmetic equation of the calculations to be performed must be written in such a way that the BASIC system can recognize the operations required. The statement used is the assignment (LET) statement.

The LET statement is used to assign the result of a calculation to a variable. The calculation to be evaluated, called an expression, appears on the right side of the equal sign in the LET statement.

The variable to which the expression is assigned appears on the left side of the equal sign. In the previous example, BASIC multiples 3.1416 by the value assigned to Y and then assigns the result to the variable X.

An expression is made up of elements described in Chapter 2 - simple variables, numbers, arrays, array elements, functions, and operators. Parentheses may be used in arithmetic expressions to enclose subexpressions. A subexpression in parentheses is evaluated first. Within each expression or subexpression, arithmetic operations are performed in the following sequence: exponentiation first, multiplication and division next, and addition and subtraction last. When two operations are of equal precedence, such as addition and subtraction, evaluation proceeds from left to right. In addition to arithmetic operations, BASIC provides a number of standard mathematical functions, such as SIN (X). These are described in Chapter 2.

An example of an expression to be evaluated and assigned to a variable is:

```
50 LET A=A-(23*X/SIN(A**2)
```

1.8 PRINTING OUTPUT

The PRINT statement may be used to print out results of calculations. For example:

```
110 READ Y
120 LET X = Y
122 PRINT X
125 GO TO 110
130 DATA 40, 2, 7.8, 456.41, -.003, 184.6
```

The PRINT statement is part of the loop, so that a value for X is printed out each time the LET statement is executed. Each value of X is printed out on a new line. The output would look as follows:

```
40
2
7.8
456.41
-.3E-2
184.6
```

It is also possible to print out a message using the PRINT statement. The user might want an explanation of each value printed. For example, the statement:

```
100 PRINT "VALUES OF X"
```

could be added to print a heading preceding the numeric output.

1.9 PROGRAMMING EXAMPLE

The sample BASIC program shown earlier is explained in detail below:

105 REM - SOLVE A QUADRATIC EQUATION	A program comment or remark
110 READ A, B, C	Read coefficients
120 LET X = B*B-4*A*C	Compute b^2-4ac
121 IF X < 0 GOTO 999	Check for complex solution
122 LET X1 = (-B+SQR (X)) /2*A	Compute one solution
130 LET X2 = (-B-SQR (X)) /2*A	Compute second solution
240 PRINT X1, X2	Print solutions
400 GOTO 110	Loop to get new values
500 DATA -1, 2, 3, +2, 3, -4, 2, 1, 6	Data for coefficients
999 PRINT "NO REAL SOLUTION"	Error message

This program solves the equations

$$-X^2 + 2X + 3 = 0$$
$$2X^2 + 3X - 4 = 0$$
$$2X^2 + X + 6 = 0$$

This program terminates following execution of statement 999 or if the data in the DATA statement is exhausted. The latter condition causes a data error message to be printed.

In the above example the third group of coefficients (2, 1 and 6) will cause the "NO REAL SOLUTION" message. Once the program terminates, the user can enter a new DATA statement and re-execute the program.

```
500 DATA 0, 1, 0, 4, 3, -12
```

1.10 EDITING A PROGRAM

The user controls the contents of his current program by statement number. In effect, every program statement the user types at the terminal must have a statement number. This number is matched by BASIC against statement numbers existing in the current program. By this means, the user can delete, insert, or change any given statement.

If the user enters a statement number followed by a carriage return, BASIC searches the current program for the statement number. If found, the statement is deleted. If not found, no action is taken.

If the user enters a statement number followed by a statement, BASIC searches the current program for the statement number. If not found, the statement is inserted in the current program. If found, the statement in the current program is replaced by the entered statement.

1.11 EXECUTING A PROGRAM

When the programmer has written and edited his program, he can execute it by giving the command:

RUN

The program executes starting at the lowest numbered statement. If no program errors occur (Appendix 1), the BASIC system prints any output from the program and outputs the massage:

BASIC

when execution is complete.

When a RUN command is given without a statement number following it, the user is effectively running the program for the first time. Arrays must be dimensioned, strings must be given lengths, and variables have no associated values.

The programmer has the option to interrupt his program while running either by pressing the ESCAPE key at the keyboard under the Multi-user Executive or by a programmed STOP statement. When a running program is interrupted in this manner, all current string lengths, array dimensions, and variable values are maintained until the programmer issues another RUN command.

The programmer has the option to execute while retaining all current information. To do so, he resumes execution by giving the command:

RUN n

where: n is the number of some statement in the program at which execution is to be resumed.

The programmer can resume execution at the statement at which the execution was interrupted or at any other statement in the program.

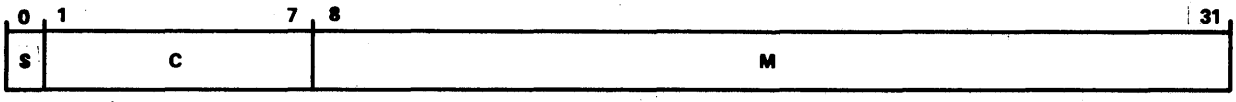
CHAPTER 2
ELEMENTS OF BASIC

2.1 NUMBERS

On output, any number n in the range $.1 \leq |n| \leq 999999$ is printed out without using exponential form. All other numbers except zero are printed in six digit format with a decimal point on the left, followed by the letter E, followed by an exponent.

Example	Output
2,000,000	.2E+7
2,192,000	.2192E+7
20,000,000,000	.2E+11
-0	0
108.999	108.999
.0000256789	.256789E-4
26	26
-3.0	-3
.16	.16
.43E-2	.43E-2
1/16	.625E-1

Internally, BASIC stores numbers in INTERDATA Single Precision Floating Point Format.



where: S is the sign of the mantissa M. (0 = positive, 1 = negative)
M is the mantissa, considered to be a normalized six digit hexadecimal fraction.
C is the integer exponent of 16 in excess 64 code.

The range of floating point numbers is approximately:

$$5.4 * 10^{-79} \text{ through } 7.2 * 10^{75}$$

2.2 ARITHMETIC VARIABLES

The names of arithmetic variables are either a single letter or a single letter followed by a single digit.

Examples: X, A, A1

2.3 ARITHMETIC EXPRESSIONS

Arithmetic expressions are composed of:

1. Simple variables
2. Array elements
3. System functions
4. User defined functions
5. Constants
6. Arithmetic operators
7. Relational operators
8. Boolean operators

The operators for arithmetic expressions are:

SYMBOL	TYPE	EXAMPLE	MEANING
+	ARITHMETIC	A+B	Add A to B
(unary) +	ARITHMETIC	+A	Positive A
-	ARITHMETIC	A-B	Subtract B from A
(unary) -	ARITHMETIC	-A	Negative A
↑	ARITHMETIC	A ↑ B	A raised to the B power
*	ARITHMETIC	A*B	Multiply A times B
/	ARITHMETIC	A/B	Divide A by B
OR	BOOLEAN	A OR B	0 if A and B both 0 1 if A or B not 0
AND	BOOLEAN	A AND B	0 if either A or B are 0 1 if A and B both not 0
=	RELATIONAL	A = B	A equal to B
<	RELATIONAL	A < B	A less than B
>	RELATIONAL	A > B	A greater than B
< =	RELATIONAL	A < = B	A less than or equal to B
> =	RELATIONAL	A > = B	A greater than or equal to B
< >	RELATIONAL	A < > B	A not equal to B
			(result is 1 if RELATIONAL is true; 0 if RELATIONAL is false)

The order in which operations are evaluated affects the result. In BASIC, unary minus or plus is evaluated first, then exponentiation, then multiplication and division, then addition and subtraction, then relational operators, then Boolean AND, and finally Boolean OR. When two operators are of equal precedence (* and /), evaluation proceeds from left to right. For example:

```
20-A+B*C ↑ D
```

1. 20-A is evaluated.
2. C ↑ D is evaluated.
3. B is multiplied by the value from 2.
4. The value from 1 is added to the value from 3.

Unary minus and unary plus may appear only at the beginning of an expression or immediately following a left parenthesis.

Examples: -3+A

```
A+B* (-C/2)
```

The programmer can alter the order of evaluation by enclosing subexpressions in parentheses. A parenthesized subexpression is evaluated first. Parentheses can be nested, and the innermost parenthesized operation is always evaluated first. For example:

```
20-( (A+B)*C) ↑ D
```

1. A+B is evaluated.
2. The value from 1. is multiplied by C
3. The value from 2. is raised to the power D.
4. The value from 3. is subtracted from 20.

Some examples of expressions are:

```
A1-2
SIN (X+.02)
A(I+1, J+1) * A/2
SIN (COS (X) ) * (-D)
1.5 * .0356
```

2.4 ARRAYS

An array represents an ordered set of numeric values. Each member of an array is called an array element. Names of arrays are written the same as variable names. The name of an array may be the same as the name of a variable in a program, and are considered independent of each other.

2.5 ARRAY DEFINITION

Most arrays are declared in a DIM statement, which specifies the name of the array and its dimensions. An array can have either one or two dimensions. The lower bound of a dimension is always zero; the upper bound is given in the DIM statement and is limited only by the amount of available user storage. If the upper bound exceeds user storage, an error message will result.

Dimensioning information is enclosed in parentheses immediately following the name of the array in the DIM statement:

```
DIM X (15), X1 (2, 8) ←— X is a one-dimensional array of 16 elements (0-15).
                         X1 is a two-dimensional array of 27 elements.
```

If the programmer uses an array but does not declare it in a DIM statement, BASIC will define an array of 11 elements for each dimension. An undeclared one-dimensional array cannot have more than 11 elements. If the programmer does not need 11 or 121 elements for a given array and wishes to optimize space, he should declare the array in a DIM statement with the desired number of elements.

2.6 ARRAY ELEMENTS

Each element of an array is identified by the name of the array followed by a parenthesized subscript. The elements of array X (4) would be:

X (0), X (1), X (2), X (3), X (4)

For a two-dimensional array, the first number gives the number of the row and the second gives the number of the column for each element. The elements of array A(2, 3) would be:

A (0, 0)	A (0, 1)	A (0, 2)	A (0, 3)
A (1, 0)	A (1, 1)	A (1, 2)	A (1, 3)
A (2, 0)	A (2, 1)	A (2, 2)	A (2, 3)

An array element can be referenced with expression subscripts. Any variable or expression that is used as a subscript must evaluate such that:

$0 \leq$ value $\leq$ upper bound declared in the DIM statement

If the variable or expression does not evaluate to an integer, BASIC will trucate to convert it to an integer i.e., A (1.1) and A (1.7) are truncated to A (1). For example, some elements of array A (8, 8) might be:

A (I+3, J)

A (I, 5)

A (ABS (I), I*J)

If a subscript evaluates to an integer larger than the upper limit of the array's dimension, an error message is printed.

2.7 REDIMENSIONING ARRAYS

It is possible to redimension a previously defined array during execution of a program. Redimensioning does not affect the amount of storage originally defined for the array nor the current contents of the array.

Redimensioning may be used to change the subscripting of two-dimensional arrays. Suppose the user originally defines a 3 x 4 array A with the statement DIM A (2, 3).

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

A (0, 0) contains 1, A (0 , 1) contains 2, . . . , A (2, 2) contains 11, and A (2, 3) contains 12.

Array A may be redimensioned with the statement DIM A (3, 2) which transposes the dimensions of A.

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

The values in A remain the same but the subscripts referring to those values have changed.

Value	New Subscript	Old Subscript
4	A (1, 0)	A (0, 3)
6	A (1, 2)	A (1, 1)
8	A (2, 1)	A (1, 3)
11	A (3, 1)	A (2, 2)

An array can be redimensioned only in such a way that it has the same or fewer elements than the original definition. For example, redimensioning a 3 x 5 array as a 4 x 4 array causes an error.

Subscript references outside the currently declared range of subscripts cause errors. For example, once array A above is redefined as A (3, 2). The use of the 3 as a column subscript, e. g., A (2, 3) causes an error.

Redimensioning an array to have fewer elements (e.g., redimensioning B (3, 5) as B (2, 2) or redimensioning A (6) as A (3)) makes referencing the unused locations impossible and does not free these locations for other storage.

2.8 FUNCTIONS

Certain standard mathematical functions are supplied as part of BASIC. They are:

SIN (X)	The sine of X where X is in radians.
COS (X)	The cosine of X where X is in radians.
TAN (X)	The tangent of X where X is in radians.
ATN (X)	The arctangent of X where ($-\pi/2 \leq$ ATN (X) $\leq \pi/2$).
LOG (X)	The natural logarithm of X. (X > 0).
EXP (X)	The exponential of X (i.e., e^X).
SQR (X)	The square root of X. (X $\geq$ 0).
ABS (X)	The absolute value of X.

The arguments of functions SIN, COS, TAN, ATN, and ABS may be any real number.

A negative or zero argument in the LOG function or a negative argument in the SQR function causes BASIC to print an error message.

The argument of the EXP function should be within the range of values that will generate the largest and smallest possible real numbers, (-178 < X < 175).

In addition to the standard mathematical functions, the following functions are supplied as part of BASIC.

INT (X)	The greatest integer not larger than X.
RND (X)	Generate a random number between 0 and 1.
NOT (X)	The Boolean complement of X.
SGN (X)	The algebraic sign of X.
LEN (S)	The current length of string variable S.
EOF (X)	Test if file mark encountered on last I/O statement.
VAL (S)	Convert string variable S to a numeric value.
STR$ (X)	Convert X to a string value.
ERR$ (X)	Generate the error code on execution errors.
ERL (X)	Generate the line number on execution errors.

For the RND, EOF, ERR$, and ERL functions, there must be a predefined variable or a constant as an argument, although the argument has no significance. A list of error messages that may be generated while executing a program can be found in Appendix 1.

The INT function yields the largest integer less than or equal to the absolute value of its argument.

```
INT (7.25) = 7
INT (-7.25) = -7
INT (12)    = 12
INT (-.1)   = 0
INT (1.5)   = 1
```

INT may be used to round a number to the nearest integer. To round the value, add .5 to the argument:

```
INT (X+.5)
```

The RND function yields a random number having a value in the range: $0 \leq$ value < 1. The function requires an argument, although the argument does not affect the resulting random number. The argument can be any constant or previously defined variable.

```
RND (1)      might yield     .303561
RND (X)      might yield     .911674
```

The NOT function generates a result of 0 if the argument is not 0, and a result of 1 if the argument is 0.

```
NOT (-1)  = 0
NOT (2000) = 0
NOT (0) = 1
```

The SGN function generates a result of +1 if the argument is positive, 0 if the argument is 0, and -1 if the argument is negative.

```
SGN (.37) = 1
SGN (0.0) = 0
SGN (-2.5) = -1
```

The LEN function produces an integer representing the current length of its string variable argument. The argument of the LEN function must be a single unsubscripted string variable. If string variable A$ contains the string: "TOTAL Ø SUM" Then:

```
LEN (A$) = 9        (the space between TOTAL and SUM counts as a character.)
```

The EOF function yields a 1 if the last I/O statement detected a file mark, and a 0 if not.

```
INPUT  ON (7) X, Y, Z
IF EOF(0) = 1 THEN PRINT "EOF"
```

The VAL function converts a string variable to its numeric value. If string variable A$ contains the string "123.45" then:

```
X = VAL (A$)
```

would assign X the floating point value 123.45.

The STR$ function converts a numeric value to its character string value. If variable X contains the value 67.891 then:

```
A$ = STR$ (X)
```

would assign string A$ the value "67.891".

The ERR$ function generates the two character error code when BASIC has encountered an execution error in a program.

The ERL function generates the line number where an execution error has occurred.

Example:

```
 10 ON ERROR GO TO 450
    :
    .
100 PRINT ON (3) "THE RESULTS ARE", X, Y
    :
    .
450 Z=ERL (X)

460 A$=ERR$ (X)

470 PRINT A$, "ERROR AT", Z

480 PRINT "RUN", Z, "TO CONTINUE"

490 STOP
```

If Device 3 is not correctly assigned to an output device, or if an error occurs in the attempted output BASIC will transfer to statement 450.

2.9 STRING LITERALS

A string literal is written as a character string enclosed in quotation marks.

```
"WORDS FOR PRINT"
```

All blank spaces within the quotation marks are significant. The quotation marks are not printed if the literal is output. If the user wishes to insert any eight bit character into a literal string he encloses the decimal equivalent of the character in angle brackets, i.e., < n > . where n is an integer from 0 to 255. A useful application is the placing of quotes within the quotation mark delimiters.

```
" < 34 > I AM IN QUOTES < 34 > "
```

If n is not within the specified range, then the brackets and n are treated as string literal characters.

See Appendix 3 for a list of all ASCII characters and their decimal values.

2.10 STRING VARIABLES

An extension to BASIC permits use of string variables as well as literals. String variables are denoted by a variable name followed by a dollar sign.

```
A$
A1$
```

String names may duplicate the name of numeric variables and arrays. For example; A, A(1) and A$ are all valid in the same program. String variables must be declared in DIM statements with a single dimension giving the maximum number of characters the string can contain, where the range is:

$$1 \leq \text{Dimension} \leq 255$$

Once the maximum length of a string has been declared in a DIM statement, it cannot be changed.

Example: 20 DIM A$ (3), C1$ (20)

2.11 STRING ARRAYS

A string array is a list of string variables. String arrays are defined in a DIM statement with two dimensions specified. The first is the maximum number of characters in each string, (as in the string variable DIM statement), and the second is the number of strings in the list.

Example: 20 DIM D$ (8, 4), D1$ (10, 10)

where: D$ contains 4 strings each up to 8 characters.

D1$ contains 10 strings each up to 10 characters.

2.12 STRING EXPRESSIONS AND SUBSCRIPTING

String expressions are composed of string literals, string variables or string array elements. Valid references to these items are described below:

	Number of Subscripts in the DIM statement	Number of Subscripts when referenced
String literal	None	None
String variable	1	None or 2
String array	2	1

String variables may be unsubscripted which causes reference to the entire contents of the string, or may have 2 subscripts referring to inclusive characters in the string.

String arrays require a single subscript which refers to a specific string in the list. Subscripts may be any valid numeric expression, and follow the same rules as numeric array subscripts, with the exception that a subscript of 0 is not allowed.

Strings may not be redimensioned. An attempt to redimension a string will cause an error message. Strings may contain a variable number of characters. BASIC keeps track of the current number. If a 10 character string is loaded with a 3 character string, and the 10 character string is printed, the result is 3 characters output.

Examples:

DIM A$ (10), A1$ (9, 9)

A$	References the entire string variable A$.
A1$ (2)	References the second string in string array A1$.
A1$ (I)	References the Ith string in the string array A1$.
A$ (3, 7)	References characters occupying positions 3 through 7 inclusive in string variables A$
A$ (I, J)	References characters occupying positions I through J inclusive in string variable A$, where I and J are integerized and I ≤ J.
A$ (1, 1)	References only the first character in string variable A$.

Examples of Illegal Subscripting:

A$ (2)	A string variable must have 2 or no subscripts.
A$ (2, 1)	Subscript 1 must be ≤ to subscript 2.
A$ (0, 20)	Both subscripts out of bounds.
A1$ (2, 2)	Arrays must have one subscript.
A1$	Arrays must have one subscript.
A1$ (0)	Subscript out of bounds.

A double-subscripted variable allows the programmer to reference a subset of one or more characters within a string. String expressions can be used in assignment (LET) statements, PRINT statements, DATA statements, and in relational expressions of IF statements.

```
20 PRINT A$ (1,4)                    Print the first 4 characters of A$.
30 LET B$ = "RESULTS ARE:"           Assign a string literal to B$.
40 IF A$ (I,I) = B$ (J,J) GO TO 100  If the Ith character of A$ is equal
                                     to the Jth character of B$, trans-
                                     fer to statement 100.
50 INPUT C$, D$ (2,2)                Input one or more characters for C$
                                     and a single character for D$ (2,2).
60 LET A1$ (2) = "COLD"              Assign a string literal to the second
                                     string of array A1$.
```

On the right hand side of an assignment statement, and in IF statements, string expressions may be concatenated by the operator (+).

```
100 DIM A$ (50), B$ (50)
110 LET A$ = "@2.50 EACH, THE PROFIT MARGIN IS 15.8%"
120 LET B$ = A$ (1,3) + "25" + A$ (6,33) + "11.2%"
```

B$ contains the following, after statement 120 is executed:

@2.25 EACH, THE PROFIT MARGIN IS 11.2%.

Following are some string assignment considerations:

20 LET A$ = B$	A$ is replaced by the contents of B$.
25 LET A$ = ""	A$ is replaced by the null string (the null string contains no characters and has a length of zero).
30 LET A$ = A$+B$	Contents of B$ are appended to current contents of A$.
40 LET A1$ (3) = A1$ (2) + "AB"	A1$ (2) is combined with "AB" and is placed in A1$ (3).

When characters are assigned to a string or part of a string, the number of characters available on the right of the equal sign determines how many will be stored. All other characters remain unchanged. Assigning a string equal to the null string affects the length of the string only and does not change any characters in the string.

```
100 LET A$ = "ABCDEF"
110 LET B$ = "1"
```

```
120 LET A$ (3,3) = B$
130 LET A$ (3,6) = B$
```

120 and 130 both produce the same result: A$ = AB1 (note: A$ (1,6) = AB1DEF)

```
150 LET A$ (3,6) = B$+B$+B$
```

150 produces AB111, since the expression to the right of the equals sign contains three characters. (note: A$ (1,6) = AB111F).

String assignments can cause the internal length of a string to be changed. After 100, the LEN (A$) = 6. After 120 and 130, LEN (A$) = 3. After 150, LEN (A$) = 5.

When strings appear in the relational expression of an IF statement, the strings are compared character by character left to right on the basis of the ASCII collating sequence until a difference is found. If a character in a given position in string A has a higher ASCII code than the character in that position in string B, then string A is greater. If the characters in the same positions are identical but one string has more characters than the other, the longer string is the greater of the two. Use of strings in relational expressions is described in detail in the IF statement.

20 LET A$ = "ABCDEF"	
30 LET B$="25 ABCDEFG"	
31 IF A$ > B$ GOTO 50	True. Transfer occurs. (A is greater than 2).
32 IF A$ > B$ (4,10) GOTO 50	False. No transfer. (B(4,10) has one more character than A$).
33 IF A$ (1,4) = B$ (4,7) GO TO 50	True. Transfer occurs.

1. If the right side of a string assignment statement contains more characters than will fit in the destination variable the excess characters are ignored.

```
DIM A$ (6)
LET A$ = "ABCDEFGHIJ"  (A$ contains ABCDEF)
```

2. When a double-subscripted variable appears on the left side of a string assignment, the new length of the variable is determined by the second subscript or the amount of data entered.

```
DIM A$ (6)
LET A$ = "AAABBB"                  (LEN (A$) = 6)
LET A$ (2,4) = "CCC"               (LEN (A$) = 4)   (A$ = ACCC)
LET A$ (2,4) = "DD"                (LEN (A$) = 3)   (A$ = ADD)
LET A$ (2,6) = "" (Null String)    (LEN (A$) = 1)   (A$ = A)
```

Note that an explicit reference to A$ (1,6) at this point ignores the length of A$ and yields ADDCBB.

This characteristic may be used to delete trailing spaces from a string. Assume A$ was dimensioned as A$ (20).

```
20 X = 20
30 IF A$ (X,X)  < >   " "GOTO 70
40 A$ (X,X) ="""
50 X = X-1
60 GOTO 30
70  .
    .
    .
```

CHAPTER 3
BASIC STATEMENTS

3.1 INTRODUCTION

The statements available in INTERDATA BASIC allow the user to write programs using more advanced programming techniques as he increases his knowledge of the BASIC language. The statements listed below are described in detail in this chapter. They constitute the statements of INTERDATA Extended BASIC.

The statements described in this section are:

Statement	Meaning
BSP	Backspace a device.
CALL	Call an assembly language subroutine.
DATA	Define a block of user data values.
DEF	Define a user function.
DIM	Dimension arrays, string variables, and string arrays.
END	Optional terminator of program.
ENDTRACE	End trace of a running program.
FOR	Set up a programming loop.
GOSUB	Transfer to an internal subroutine.
GO TO	Transfer control to a program statement.
IF	Conditional transfer to another part of the program; or conditional execution of a statement.
INPUT	Request data from an input device.
LET	Assign values to variables.
MAT	Matrix operations.
NEXT	Terminate programming loop.
ON	Provide a series of possible transfer points.
PRINT	Write data to an output device.
RANDOM	Reinitialize random number generator.
READ	Input data from the DATA block.
REM	Comment.
RESTORE	Reinitialize data block.
RETURN	Return from an internal subroutine.
REW	Rewind a device.
SETTRACE	Start trace of a running program.
STOP	Halt program execution and switch to keyboard mode.
WFM	Write a file mark to a device.

3.2 LET

Syntax: variable = expression

LET variable = expression

Use: To evaluate expression and assign the value to variable. Variable can be a numeric or string variable, or a numeric or string array element. Use of the word LET is optional.

String expressions may be assigned to string variables, and arithmetic expressions may be assigned to arithmetic variables.

The variable may be subscripted in the case of strings and numeric arrays.

Examples:
```
12 LET B=C+2.1417
30 A=A+1
51 LET W1=( (A+B) *J/3)*COS (M)
90 X=0
101 LET A$="NOW"
102 LET B$="IS THE"
103B$=A$+B$+"TIME"
104 LET B$ (1,2) = A$ (3,5) +"LET"
105 E1$ (3) = E1$ (2) +A$
```

3.3 DEF

Syntax: DEF FNa (d) = expression

where: a is a single character.
d is a single letter dummy variable that may appear in expression.

Use: To permit a user to define a numeric function that can be referenced during a program. The function returns a value to the point of reference.

When a function is referenced, the constant, variable or expression appearing in the reference argument replaces dummy argument d in the expression. The reference argument may contain other user defined functions.

In the function definition, expression can be any legal numeric expression including one containing other user-defined functions. There is no limit to function nesting, except that a user defined function may not call itself.

Definition of a function is limited to a single line of text. For longer formulas, subroutines should be used. The DEF statement is not executable and acts merely as a definition.

Examples:

```
10 DEF FNA (X) = EXP (X ↑ 2)          :Definition of function FNA
20 LET Y = Y*FNA (.1)                 :Function reference; argument = .1
30 IF FNA (A+3) > Y THEN 150          :Function reference; argument = A+3
130 LET P = 3.1416
140 DEF FNB (X) = X*P/180
150 DEF FNS (X) = SIN (FNB (X) )      :Function FNB is nested within FNS
160 DEF FNO (X) = COS (FNB (X) )       and FNO
170 FOR X=0 TO 45 STEP 5
180 PRINT X, FNS (X), FNO (X)         :FNS and FNC are referenced with X
190 NEXT X                             having values 0, 5, 10, . . . , 45
200 LET X=FNS (FNO (X) ) +1           :Function FNO is the reference argument of
                                       FNS
```

3.4 DIM

Syntax: DIM array (dim_1), array (dim_1, dim_2), string (dim_1), string array (dim_1, dim_2)

Use: To define, (1) the dimensions of one and two dimensional numeric arrays, (2) the maximum number of characters in string variables, and (3) the maximum number of characters (dim_1) and number of elements (dim_2) of string arrays. The DIM statement is not executable and is used only to allocate storage. It may appear anywhere in a BASIC program, but must precede the usage of a variable or array it dimensions.

Numeric arrays are dimensioned as follows:

1. The upper bound is given in parentheses following the array name.

2. For two dimensional arrays there are two upper bounds, separated by a comma.

3. The lower bound of a dimension is always 0 and does not appear in the DIM statement.

String variable names are followed by a single dimension in parentheses. This gives the upper limit of the number of characters that the string may contain. The upper limit must be in the range:

$$1 \leq \underline{\text{limit}} \leq 255$$

String arrays have two dimensions, (1) the maximum number of characters in each string, and (2) the number of elements in the array.

Numeric arrays and strings may appear in any order in a DIM statement. Any dimension may appear as a variable, constant, or expression.

Example: 2 DIM A (5,6), J (20), X (17), B$ (25), P$ (Z+1), Y(14,10), A1$ (20,10)

where: A is a 6 x 7 element two dimensional array.
J is a 21 element one dimensional array.
X is a 18 element one dimensional array.
B$ is a string with a maximum of 25 characters.
P$ is a string with a maximum of Z+1 characters.
Y is a 15 x 11 element two dimensional array.
A1$ is a string array containing 10 strings of 20 characters each.

Once a numeric array has been dimensioned, it may be redimensioned to an equal or smaller size. See Section 2.7 for example.

Once a string variable or array has been dimensioned it may not be redimensioned. If a numeric array is referenced before being dimensioned it is assumed to have dimensions of 10. All string variables and arrays must be dimensioned with a DIM statement before being used.

3.5 END

Syntax: END

Use: INTERDATA BASIC does not require an END statement as the last program statement. Programs terminate at the last logically executed statement in the program (if an END statement or STOP statement is not encountered). However, BASIC allows END statements for compatibility with BASIC programs written for other systems, and for use as a terminating statement in a LOAD or batch operation (see Chapter 4).

FOR

Syntax: FOR control variable = $expression_1$ TO $expression_2$

FOR control variable = $expression_1$ TO $expression_2$ STEP $expression_3$

Use: To establish the beginning, terminating, and incremental values for control variable in a programmed loop. Control variable determines the number of times the statements contained in the loop are executed.

The loop consists of statements following the FOR statement up to a NEXT statement that contains the name of control variable. The control variable in a FOR statement must be numeric and cannot be subscripted.

$expression_1$ is the first value of the variable.

$expression_2$ is the terminating value of the variable.

$expression_3$ is the increment value added to the variable each time the loop is executed. If not given, the increment value is assumed to be +1.

When the NEXT statement containing the variable name is encountered, the loop is executed again beginning with the statement following the FOR statement. The looping ends when the control variable is greater than the terminating value, $expression_2$ (less than the terminating value in the case of a negative increment value). When the loop terminates, control is passed to the statement following the NEXT statement. FOR loops may be nested to a depth of six. The FOR statement and its terminating NEXT statement must be completely nested. For example:

Legal

```
FOR X
  FOR Y
    FOR Z
    NEXT Z
  NEXT Y
NEXT X
```

Illegal

```
FOR X
  FOR Y
NEXT X
NEXT Y
```

A FOR loop may call subroutines without interfering with the FOR loop.

```
FOR X
GOSUB  <==========>  RETURN
NEXT X
```

NEXT

Syntax: NEXT control variable

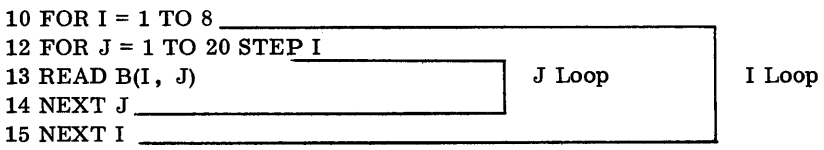
Use: To terminate the loop beginning with a FOR statement. The control variable contained in the NEXT statement must match a control variable contained in an uncompleted FOR statement.

When the FOR statement conditions have been fulfilled, execution continues at the statement following the NEXT statement.

Examples:
```
5 FOR X = .1 TO -.5 STEP -.1
10 LET Y = X*EXP (X)
20 NEXT X

10 FOR I = 1 TO 10
20 PRINT 2 ↑ I
30 NEXT I

10 DIM A (25)
30 FOR I=1 TO 25
40 READ A (I)
50 NEXT I
```



Although FOR loops must be completely nested, early termination of loops is allowed. Execution of a NEXT statement cancels all FOR loops nested within it.

```
10 FOR X = 1 TO 100
20 FOR Y = 1 TO 50
30 LET Z = A (Y)
40 PRINT Z+X
50 IF Z < 0 GOTO 70    : Cancel Y Loop
60 NEXT Y
70 NEXT X
```

The use of non-integer STEP values that cannot be represented exactly as hexadecimal floating point numbers could cause early or late termination of some FOR loops due to possible rounding errors. The user may modify the control variable within a FOR loop but he should insure that its value remains meaningful with respect to loop control.

3.7 GOSUB and RETURN

GOSUB

Syntax: GOSUB statement number

Use: To transfer control to the statement number, the first statement in a subroutine. The statement number may be in the form of a constant, variable or numeric expression.

If the statement number is not an integer it is integerized by the INT function. GOSUB statements may be nested to a depth of six. A subroutine may call another subroutine or it may call itself.

Examples:
```
GOSUB 100
GOSUB X
```

RETURN

Syntax: RETURN

Use: To exit a subroutine, returning to the first statement after the GOSUB statement that caused the subroutine to be entered.

A subroutine may contain more than one RETURN statement when logic might cause the subroutine to terminate at a number of different points.

Examples: In the following example, RETURN causes the return to statement number 12 when the subroutine is entered from statement 11; return is to statement 16 when the subroutine is entered from statement 13.

```
10 LET X = 5
11 GOSUB 50
12 LET X = 7
13 GOSUB 51
16 STOP
  .
  .
  .
50 LET Y = 3*X
51 LET Z = 1.2*EXP (Y)
53 PRINT X, Y
54 RETURN
```

Note that in the second call to the subroutine (statement 13) the value of Y is not changed.

3.8 GOTO

Syntax: GO TO statement number

Use: To transfer control to a statement that is not the next sequential statement. If control is transferred to an executable statement, that statement and those following are executed. If control is transferred to a non-executable statement (e.g., DATA), the first executable statement following the one to which transfer was made is executed. The statement number may be in the form of a constant, variable or numeric expression. If the statement number is not an integer it is integerized by the INT function.

Example:
```
190 DATA 10, -5, -2, 6, -6, 21, -9
200 READ X
220 LET A = SQR (X ↑ 2) + Y*X *FNC (X)
230 PRINT X, A
240 GO TO 200 ←———— Control will transfer back to statement 200 until all values of
                     X have been read.
```

Syntax: IF expression { GO TO / THEN } statement number

IF expression THEN statement

Use: To transfer control or execute a statement conditionally on the basis of whether expression is true or false.

The first format causes control to be passed to the statement whose number appears following GO TO or THEN if the expression is true. If the expression is not true, control is passed to the next statement following the IF statement.

The second format is a generalized form of the IF statement. Any statement, including an IF statement or a GO TO statement may follow THEN.

Expression: expression may be an arithmetic expression or two string expressions separated by a relational operator.

The relational operators are listed in Section 2.3. For the conventions used in comparing string expressions, see Section 2.12.

A numeric expression without a relational operator is considered false if it has a value of 0 and is considered true in all other cases.

Examples:

100 IF X + Z = 0 THEN 10 150 IF .01 > = SQR (X) GOTO 410 200 IF A$ < > "YES" GOTO 85	Expressions containing relational operators.
15 IF ABS (X) GO TO 410 90 IF A+B THEN 27	Expressions that evaluate to zero or non-zero.
10 IF X+Y = 0 THEN LET I = X+Y	If X+Y=0 is true, the LET statement is executed and control passes the next statement in the program; if X+Y = 0 is false, the LET statement is not executed and control passes immediately to the next statement.
160 IF X THEN IF SIN (X) < .1 GOTO 200	The first IF checks the value of X. If it is zero, control passes to the next statement in the program. If it is not zero, the IF statement following THEN is executed and control passes to the next statement in the program or to the statement 200, depending upon the value of the sine ot X.
176 IF A > B OR C > D AND E > F THEN STOP	Multiple relational expression using Boolean operators. If A > B or if both C > D and E > F then the program stops. If not, control passes to the next statement.

3.10 INPUT

Syntax: INPUT variable-list

INPUT ON (expression$_1$) variable-list

INPUT ON (expression$_1$, expression$_2$) variable-list

where: variable-list may contain numeric variables and array elements, string variables and array elements

Use: To input values for numeric and string variables at run time from a specified input device.

The value of expression$_1$, integerized, refers to the logical unit of the input device. If none is specified, the default logical unit is 5 (normally assigned to the user's terminal). The value of expression$_2$, integerized, refers to the logical record number of a random access file from which we are reading. If expression$_1$ is not assigned to a random access file, then expression$_2$ is ignored. Each element of data read is delimited from the next by a comma or a single blank (numeric data only). The last data value must terminate with a carriage return or two consecutive blanks (numeric data only). A minus sign is recognized preceding numeric items in the input data.

Arithmetic and string variables may be intermixed in the variable list of the INPUT statement. The data input must match the variable list in both type of data and number of data items.

Character strings in the data list may optionally be enclosed in quotation marks. Character strings may include any characters including digits. Since data is delimited by commas, a comma cannot be part of a character string unless it is a string enclosed in quotation marks.

25.34, THE RESULT IS: ,0 — The second item in the data list is a string of 16 characters including the blanks.

25.34, "THE RESULT IS:" ,0 — The second item in the data list is a string of 14 characters, including the blanks but excluding the quotation marks.

If the data list does not contain enough values to fill the input list, or values are found to be invalid, an error message is printed and BASIC enters the command mode.

During typing of a data list, the programmer may use the line erase (#) or character erase (←) keys to correct errors in the list.

The user may wish to precede an INPUT statement with a PRINT statement that will indicate at the terminal which variables are to be input.

```
40 PRINT "INPUT A, B, AND C"
50 INPUT A, B, C
```

Input Examples:

Basic Statement	Response
INPUT X, Y, Z$ (3)	3,2,DOG
	-.12E-20, 0, "DOG"
	3,2 (invalid: not enough data)
	3,DOG,2 (invalid: data does not match variable type)
INPUT ON (1) A$, B$, C$, D$ (X)	JOHN, SMITH, "301 1ST AVE", MAILMAN
INPUT ON (1, X*Y+Z) A, B, C(X)	.3,.2,.407
PRINT ON (7,X) A;B;C	31 64 32 (write values on a disc file)
INPUT ON (7,X) A,B,C	(read values from the disc file)

Syntax: ON expression { GOTO / THEN / GOSUB } statement number list

ON ERROR { GOTO / THEN } statement number

Use: To provide a series of branch points. The statement number to which transfer is made depends upon the evaluation of expression. For a transfer to occur the value of expression must correspond to the sequence number of one of the statement numbers in the list.

If expression evaluates to an integer less than one or greater than the sequence number of the last statement number in the list, the ON statement is ignored and control passes to the next statement.

If expression does not evaluate to an integer, it is truncated to an integer by the INT function.

ON-GO TO and ON-THEN are equivalent forms.

ON-GOSUB begins execution at the subroutine specified in statement number list and continues until a RETURN statement is encountered. At this time control is returned to the statement following the ON-GOSUB statement.

The ON ERROR form of the ON statement does not cause immediate transfer to occur but provides a transfer line number for BASIC if an execution error is generated in a program.

Examples: 38 ON X-6 GO TO 500, 175, 100

If X-6 does not evaluate to 1, 2, or 3, the statement is ignored.

If X-6 evaluates to 1, transfer is made to statement 500; if the value is 2, transfer is made to statement 175, and if the value is 3, transfer is made to statement 100.

21 ON X GO SUB 100, 200, 300, 400

X must have a value from 1 to 4 to cause transfer.

33 ON ERROR GO TO 150

If an execution error is generated anywhere in the program following this statement transfer will be to statement 150. If another ON ERROR GO TO statement is encountered, execution errors will cause transfer to the new statement number.

3.12 PRINT

Syntax: PRINT expression list

PRINT ON (expression$_1$) expression list

PRINT ON (expression$_1$, expression$_2$) expression list

where: expression list is a list of numeric variables or array elements, string variables or array elements, expressions, and string literals.

Use: To output on a specified output device the current values of any expressions, variables, arrays or the text of any string literals in the expression list.

The value of expression$_1$, integerized, refers to the logical unit of the output device. The ON clause is optional. If none is specified, the default logical unit is 5 (normally assigned to the Teletype). The value of expression$_2$ integerized refers to the logical record number of a random file to which we are writing. If expression$_1$ is not assigned to a random access file, then expression$_2$ is ignored.

Output Formatting: The PRINT statement allows the user to control output formatting or accept default formatting.

Number Representation

Any real or integer number that can be represented as six digits and a decimal point is printed out without using exponential form. A minus sign is printed if the number is negative. Trailing zeros and decimal points are deleted.

All other numbers are printed in the format:

(-). nnnnnnE±e(e)

where: n is a digit.
E indicates exponentiation.
e is a digit of the exponent.
( ) parentheses indicate optional parts of the number.

Number	Printed Output
.00000002	.2E-7
-.0002	-.0002
200	200
-200.002	-200.002
2,000,000	.2E+7
-20,000,000,000	-.2E+11
-2.000	-2
00.0	0

The print line is divided into nine 14 character zones. Zones begin at print positions 1, 15, 29, 43, 57, 71, 85, 99, and 113. A comma between items in the expression list of the PRINT statement indicates "space to the next zone". Once the last print zone has been used, the next value is printed in the first print zone of the next line. If a PRINT statement ends with a comma, it is considered to continue on to the next PRINT statement. Only a PRINT statement ending in a carriage return or a filled up print line can cause the actual output of data.

```
10 LET X=5
30 PRINT X, (X*2),X ↑ 2,
60 PRINT X ↑ 4, X+25, (X/2), X-100
```

Note terminating comma on first PRINT statement controls the output of the first value of the next PRINT statement.

```
                    1       15      29      43      57      71      85   (PRINT position)
                    ↓       ↓       ↓       ↓       ↓       ↓       ↓
PRINT LINE 1        5       10      25      625     30      2.5     95
```

When an output value is longer than a single zone, for example, a long character string, the printing is spaced to the next free zone to print the next value. A numeric value will not be truncated or continued on the following line. If the complete value cannot fit in the final zone, it is printed on the next line.

```
10 LET X = 25
20 PRINT "THE SQUARE ROOT OF X IS: ", SQR (X)

1               15              29  ←—— Position
↓               ↓               ↓
THE SQUARE ROOT OF X IS         5   ←—— Value
```

When print lines are generated by more than one PRINT statement, the logical unit is determined from the PRINT statement causing the output.

Example:
```
PRINT Y, A,
PRINT ON (3) B
```

Y, A and B are written to logical unit 3.

Compact Zone Spacing

The user can obtain a more compact output by use of the semi-colon between list items. The semi-colon inhibits spacing to a print zone, leaving only a single space between values output. Note that like the comma, a semi-colon at the end of a PRINT statement determines the position of the first value of the next PRINT statement.

```
5 LET A = 4
10 LET B$ = "THE RESULTS ARE"
15 LET C$ = "FINAL"
20 PRINT A; A*A; B$; A ↑ A
25 PRINT "END OF THIS JOB"; A+A; A-10; B$; C$

1  3   6                20 22  ← Position
↓  ↓   ↓                ↓  ↓
4  16  THE RESULTS ARE  256    ← Value

1              15 17  19 22                          ←——Position
↓              ↓  ↓   ↓  ↓
END OF THIS JOB  8  -6  THE RESULTS ARE FINAL  ←——Value
```

Commas and semi-colons may exist in the same PRINT statement. The line formatting rules for each are applied.

```
5 LET A = 17.6
10 LET B = 5
15 LET A$ = "THIS IS A TEST"
20 PRINT A,B;A;B,A$; A+B;A-B
25 PRINT "THIS LINE EXCEEDS ONE ZONE", A;B,A*5,A$

1          15    17    22    29             44    49   ← Position
↓          ↓     ↓     ↓     ↓              ↓     ↓
17.6       5     17.6  5     THIS IS A TEST 22.6  12.6 ← Value

1                              29    34    43    56     ← Position
↓                              ↓     ↓     ↓     ↓
THIS LINE EXCEEDS ONE ZONE     17.6  5     88    THIS IS A TEST
```

Spacing to the Next Line

If there is no comma or semi-colon terminating the last item of a PRINT statement, the edited text is output on the next line.

```
10 LET X=5
20 PRINT X, (X*2)
30 PRINT X*3
40 PRINT X-25; (X*2)
50 PRINT X-100
```

```
1          5          15          <---- Position
|          |          |
5                     10          <---- Values
15
-20        10
-95
```

Tabulation

It is possible to tabulate to a particular print position using the TAB function:

TAB (expression)

where: expression evaluates to an integer representing the character position of the next list item following the TAB function. The TAB function tabulates up to 132 character positions. If the expression in the TAB function evaluates to a number greater than 132, the TAB is ignored. The semi-colon or comma delimiter following a list item is ignored when a list item is TABed.

```
10 DATA 6, -7, 9, -11
20 READ A, B, C, D
30 PRINT TAB (5), A; TAB (10), B; TAB (15), C; TAB (20), D
```

```
1    5    10   15   20        <---- Position
|    |    |    |    |
     6    -7   9    -11       <---- Value
```

Printing with a (;) Semi-Colon

The word PRINT may be substituted by a semi-colon (;) for speed of entry.

Example: ; 3*LOG (17) is equivalent to

PRINT 3*LOG (17)

Other PRINT Examples:

```
10 FOR K = 1 TO 10
20 PRINT K
30 NEXT K

1
2
3
4
5
6
7
8
9
10
```

3.13 PRINT USING

Syntax: PRINT USING string, expression list

PRINT ON (exp_1) USING string, expression list

PRINT ON (exp_1, exp_2) USING string, expression list

where: Expression list is a list of numeric or string variables, numeric or string array elements, expressions, and string literals.

String specifies formats of the fields in which the value of each of the expressions in the list is to be output. The format of string is described below. String may be a string literal or string variable. The ON clause is described under Section 3.12 PRINT.

Use: To output current values of any expressions, variables or array elements appearing in the expression list in conjunction with the field formats specified by string.

Formatting Rules and Examples

1. Since the output field formats are specified by string, all formatting conventions used in the PRINT statement (TAB function, comma, and semi-colon) are ignored within the expression list.

2. Within string, a number of format fields and string literal characters for output may appear. A format field is made up of combinations of the following characters.

+ - @ , . $ ↑

The format field characters may appear within string as format field definition characters or as part of a string literal. BASIC differentiates format fields from string literals by the syntax of format fields.

For example:		
	"TWO FOR $2.75"	$2.75 are characters of a string literal.
	"TWO FOR $$$.@@"	$$$.@@ is a field format (a $ followed by an appropriate field format character — another $ in this case).
	"ANSWER IS -85"	-85 are characters of a string literal.
	"ANSWER IS -@@@"	-@@@ is a field format (a - followed by an appropriate field format character — an @ in this case).

3. Format fields are separated by the appearance of any non-format character in the string.

```
"@@@ △ PLUS △ @@@"
format   literal   format
field    data      field
```

(△ = blank)

4. String expressions may appear in the expression list of the PRINT USING statement and are superimposed on a field format in the following manner:

   - Each character of the string replaces a single format field character. (any one of the format field characters).

   - Strings are left justified in the format field, with a fill of spaces when required.

   - When the character string is longer than the format field, the string is truncated.

```
PRINT USING "-$$@@,@@+@", "FIVE", "THIS IS A TEST", "12379.821943"

        results in: FIVE  THIS IS A 12379.8219
```

   Note that the last string contains numbers, but it is treated as a string literal, because it is enclosed in quotes.

5. When there are more expressions in the expression list than field formats in string, the existing formats are used repetitively.

```
"@@@@ AT $@@@.@@ PER @@@"
```

   The first, fourth, seventh, etc., expressions in the list are formatted using the field format @@@@. The second, fifth, eighth, etc., expressions in the list are formatted using the field format $@@@.@@. The third, sixth, ninth, etc., expressions in the list is formatted using the field format @@@. The embedded blanks, AT, and PER are string literals.

```
5 PRINT USING "@@@.@@ △ ", A, B, C,

△ △ 1.40 △ △ 18.90 △ △ 26.70
```

possible output; number of expressions in the list exceeds the number of field formats. (△ = blank)

6. The special characters:

+ - @ , . $ ♦

are used in formatting numeric output as follows:

Digit Representation (@)

For each @ in the field format, a digit (0-9) or a space is substituted.

Field Format	Data	Output	Remarks
@@@@@	15	△△△ 15	Right justify digits in field with leading blanks.
	-399	△△ 399	Signs and other non-digits are ignored.
	1.95	△△△△ 2	Only integers are represented; the number is rounded to an integer.
	9999999	*****	If the data is too large for the field, all asterisks are output.

Decimal Point (.)

The decimal point indicator (.) places a decimal point within the string of digits in the fixed character position in which it appears. Digit positions (@) following the decimal point are filled; no blank spaces are left in these digit positions. When the value to be output contains more fractional digits than the field format decimal indicator allows, the fraction is rounded.

Field Format	Data	Output	Remarks
@@@@@.@@	18	△△△ 18.00	Fractional positions are filled with zeros.
	29.347 0.079	△△△ 29.35 △△△△ 0.08	Rounding occurs on fractions.
	998037.06	********	When the value is too large a field of all asterisks is output.
	.0006	????????	When the value is too small to be represented, a field of all question marks is output.

Fixed Plus or Minus Sign (+ or -)

A sign character may appear as a single plus (+) or minus (-) sign in either the first character position in the format field or the last character position in the format field. The signs have the following effect:

+ prints a + in the given field position if the data is positive and prints a - in the given field position if the data is negative.

- prints a - in the given field position if the data is negative and leaves a blank space in that field position if the data is positive.

When a sign character is used, any leading zeros appearing in the data are replaced by blanks, except for a single leading zero immediately preceding a decimal point.

Field Format	Data	Output	Remarks
+@@.@@	26.5	+26.50	Fixed sign inserted.
	4.01	+△4.01	Blanks precede the value.
	-5.786	-△5.79	Fixed + allows minus sign to be inserted.
	-884.0	******	
@@@.@@-	8.7	△△8.70△	Extra positions filled by a space
	000.06	△△0.06△	The last leading zero before the decimal point is not suppressed.
	-1.487	△△1.49-	
	-234.0	234.00-	

Floating Sign (++ . . . or -- . . .)

A floating sign appears as the first two (or more) signs in the field format. Floating positive (++) outputs either a plus or minus sign immediately preceding the data; floating negative (--) outputs either a blank space or minus sign immediately preceding the data.

Positions occupied in the field format by the second sign and any additional signs can be used for numeric positions without field overflow occurring.

Field Format	Data	Output	Remarks
---.@@	-80	-80.00	Second and third signs are treated, as digit positions (@) on output.
	987	******	
	8	8.00	

Fixed Dollar Sign ($)

A fixed $ sign may appear as either the first character or second character in the string, causing a $ to be placed in that character position. The $ may appear as the second character if it is preceded by a fixed sign. A fixed $ causes suppression of leading zeros in the data value.

Field Format	Data	Output
-$@@@.@@	80.54	△ $ △ 80.54
$@@@.@@+	-80.54	$ △ 80.54-

Floating Dollar Sign ($$ . . .)

A floating $ consists of at least two $ characters beginning at either the first or second character position in the string, and causes a $ sign to be placed in the character position immediately preceding the first digit.

Only one floating character (sign or $) is permitted in a given format field.

Field Format	Data	Output	Remarks
+$$$$@.@@	48.20	+ △ △ $48.20	$ sign may be replaced by digits like floating + and - signs.
$$@@.@@-	-9.0	△ $09.00-	Leading zeros are not suppressed in the @ part of the field.

Separator Comma (,)

The separator (,) places a comma within a string of digits in the fixed character position in which it appears in the field format. However, if the comma would be positioned in a field of suppressed zeros (blanks), a space is output in the comma's position.

Field Format	Data	Output	Remarks
+$@,@@@.@@	70.6	+$ △△△ 70.60	Space printed in place of comma.
	6000	+$6,000.00	
++@@,@@@	00033	△ +00,033	Comma is printed when leading zeros are not suppressed.

Exponent Indicator (↑)

Four consecutive up arrows (↑↑↑↑) are required to indicate an exponent field and are replaced by E+nn, where each n is a digit. In E format output, the first significant digit of the data value will start in the first numeric position of the format field.

Field Format	Data	Output
+@@.@@ ↑↑↑↑	170.35	+17.04E+01
	-.2	-20.00E-02
	40	+40.00E+00

3.14 RANDOM

Syntax: RANDOM

Use: To permit reuse of the set of random numbers. The RANDOM statement re-initializes the random number generator to its initial starting point. This facilitates debugging of programs that use the random number generator.

3.15 READ and DATA

READ

Syntax: READ variable list

where: Variable list can contain arithmetic variables and array elements, and string variables and array elements.

Use: To read values from a data block into the variables or array elements listed in the READ statements.

The order in which variables appear in the READ statement is the order in which values are read from the data block. Values appearing in all DATA statements in a program can be considered as a continuous single data block. Normally, READ statements are placed in the program at those points at which data is to be manipulated, while DATA statements may be placed anywhere. A pointer is moved to each consecutive DATA value as a value is retrieved for variables in READ statements. If the number of variables in the READ statement exceeds the number of values in the data block, an error message is printed. The RESTORE statement can be used to reset the pointer to the beginning of the first DATA statement.

The type of variable in the list of the READ statements must match the corresponding value type in the DATA statement. An attempt to mix arithmetic and string values will result in an error message.

DATA

Syntax: DATA data list

Use: To provide values to be read into variables or array elements appearing in READ statements. Numbers, string literals, numeric and string expressions may appear in DATA statements. Each data element is separated from the next item by a comma. DATA is a non-executable statement. If a running program encounters a DATA statement, it is ignored.

Examples:

```
15 READ X, Y, Z
      .
      .
20 READ N
      .
      .
24 FOR I = 0 TO 10
25 READ B (I)
26 NEXT I
      .
      .
40 DATA 4.2, 7.5, 25.1, -1, .1, .01, .001, .0001
45 DATA .2, .02, .002, .0002, .015, .025, .3, .03, .003
```

The first three data values are read into X, Y, and Z respectively. The value -1 is read into N. The next 11 values, .1 through .3, are read into the 11 elements of array B.

```
      .
      .
10 READ A, B, C$, E
      .
      .
30 GO TO 10
      .
      .
50 DATA 1, 10, "JACK", .21
51 DATA -1, 1, "JILL", .22
52 DATA X*Y+2, SIN (X), D$+ "WATER", .23
```

Each series of data values, contained in the three DATA statements will, in turn, be read into variables A, B, C$ and E.

Note that the numeric values in the DATA statement correspond to numeric variables in the READ statement and the string value corresponds to the string variable.

3.16 REM

Syntax: REM text comment

Use: To insert comments within a program. The text following REM is stored and is re-produced as it appears when a listing of the program is printed. Although the REM statement is non-executable, storage space is required for the text.

Example: 90 REM - SUBROUTINE TO FIX EXTENTS

3.17 RESTORE

Syntax: RESTORE

Use: To permit reuse of the data block. RESTORE sets the data block pointer to the first value in the first DATA statement.

The next READ statement following execution of a RESTORE statement will begin reading values from the first DATA statement.

Example:

```
20 FOR I = 0 TO 10
30 READ B (I)              Data values 1 to 11 are read into elements of Array B.
40 NEXT I
50 RESTORE
60 READ A, B, C            Data values 1, 2, 3 are read into A, B, and C respectively.
   .
   .
   .
500 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
```

3.18 STOP

Syntax: STOP

Use: To halt the execution of a program at some point within the program. When a STOP statement is encountered, BASIC will cease execution and print the message:

STOP n

where: n is the line number of the STOP statement. The system will wait for a keyboard command.

Example:
```
10 IF X > 0 GOTO 40
20 PRINT "ERROR"
30 STOP
```

The message "STOP 30" is printed on the terminal.

3.19 CALL

Syntax: CALL sub#, P_1, ... , P_n

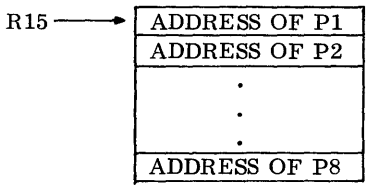
where: sub# is a positive integer from 1 to 32,767 identifying an assembly language subroutine. P_1, . . . P_n are optional parameters to the subroutine where ($0 \leq n \leq 8$)

Parameters may be arithmetic or string variables, or array elements. Expressions are not permitted as parameters.

Use: **To call a subroutine written in INTERDATA Common Assembly Language from a BASIC program.**

When BASIC enters a CALLed assembly language subroutine:

1. Register 15 points to a stack containing the parameter addresses.



2. Register 14 contains the return address to BASIC.

3. Register 1 contains the user's BASIC work area pointer and must not be used by a subroutine at any time.

For BASIC to establish the linkage from the subroutine number to the subroutine entry point, the user must assemble a subroutine table with and in front of his subroutines. For single-user BASIC, a relocatable program containing the table and all the subroutines must be loaded immediately following the OS under which BASIC is to run (before the BASIC interpreter is loaded). For Multi-user BASIC the program must be loaded immediately after the Multi-user Executive (03-058) and before the BASIC interpreter.

The subroutine table is a list of all assembly language subroutines available to BASIC programs. For each subroutine a list of three address length constants is required, containing the following:

- Subroutine number
- Subroutine starting address
- Number of parameters expected

The table is terminated by any negative subroutine number.

Example of a subroutine table and its subroutines:

```
SBRTB   DC      7       SUBROUTINE NUMBER
        DC      ASUB    SUBROUTINE ENTRY POINT
        DC      4       NUMBER OF PARAMETERS
        DC      4       SUBROUTINE NUMBER
        DC      BSUB    SUBROUTINE ENTRY POINT
        DC      0       NUMBER OF PARAMETERS
        DC      -1      END OF SBRTB

ASUB    (coding for subroutine A)
        .
        .
        .
        BR 14           RETURN

BSUB    (coding for subroutine B)
        .
        .
        .
        BR 14           RETURN
        END
```

Legal calls from BASIC to the above subroutines are:

```
CALL 7, A, B, C(10), D
CALL 4
```

Illegal calls which would result in an error message would be:

```
CALL 7, Q, B        Not enough parameters.
CALL 4, Q           Too many parameters.
CALL 2, A, B        No subroutine number 2.
```

3.20 REW, WFM and BSP

Syntax: REW logical unit

WFM logical unit

BSP logical unit

where: Logical unit is an integer logical unit number previously assigned to magnetic tape, cassette tape, disc or drum.

Use: To provide utility control to the above devices.

REW - rewind the logical unit.
WFM - write a file mark on the logical unit.
BSP - backspace the logical unit one record.

Logical unit may be an expression evaluating to an integer. Any non-integer value is integerized with the INT function. If logical unit is not assigned to any of the above devices, the command is ignored.

Example: Skip forward to the first file mark on logical unit 7 and backspace over it.

```
10 X = 7
20 REW X
30 INPUT ON (7) Y
40 IF NOT (EOF (0) ) GOTO 30
50 BSP X
```

3.21 MATRIX OPERATIONS

Matrix statements allow the user to manipulate two-dimensional arrays as matrices. In BASIC, matrix statements begin with the word MAT. Following is a list of the matrix statements available in BASIC.

Statement	Use
MAT READ A, B, ...	Read DATA values for previously dimensioned arrays.
MAT INPUT A, B, ...	Input values for previously dimensioned arrays.
MAT PRINT A, B, ...	Print current values of previously dimensioned arrays. (The semi-colon print delimiter can also be used.)
MAT A = B	Matrix A is dimensioned to the dimensions of matrix B and the values of B are stored into A.
MAT A = B + C MAT A = B - C	Add or subtract matrices B and C. The dimensions of B and C must be identical. Dimension A to the dimensions of B and C and store the result into A.
MAT A = B *C	Matrix multiply B and C. Dimension A to the dimensions of the resulting matrix and store the values into A. The dimensions of B and C must be compatible as defined later in the section on matrix multiplication.
MAT A = (expression)*B	Scalar multiply matrix B by the parenthesized expression. Dimension A to the dimensions of B and store the values into A
MAT A = INV (B)	Invert matrix B. Dimension A to the dimensions of B and store the values of the inverse matrix into A. B must be a square matrix.
MAT A = TRN (B)	Transpose matrix B. Dimension A to the dimensions of the resulting matrix and store the values into A. A and B must be two distinct arrays.
MAT A = (expression)	Store a constant value in all elements of A.
MAT A = IDN	Store the identity matrix in A.
MAT A = DET	Store the determinant of matrix A into element A (0, 0) of array A.

MATRIX SUBSCRIPTING

Arrays to be operated on as matrices must be dimensioned as two dimensional arrays; that is, a row matrix should be defined as A (1,n), not A (n).

It should be noted when manipulating arrays as matrices, that matrices do not have zero subscripts. That portion of an array that has zero subscripts is ignored. For example, the following coding examples will produce identical printouts.

```
10 DIM A (4,4)
20 FOR I = 0 TO 4 ←—values stored in zero subscript elements
30 FOR J = 0 TO 4
40 READ A (I, J)
50 NEXT J
60 NEXT I
70 MAT PRINT A
80 DATA 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5,
```

25 values for array A

```
10 DIM A (4,4)
20 FOR I = 1 TO 4 ←—no values stored in zero subscript elements
30 FOR J = 1 TO 4
40 READ A (I, J)
50 NEXT J
60 NEXT I
70 MAT PRINT A
80 DATA 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5,
```

16 values for matrix A

In the first case, data is stored into all locations of array A; in the second example, data is stored only into those locations with non-zero subscripts. When the MAT PRINT statement is executed the following is the result in both cases:

```
2    2    2    2

3    3    3    3

4    4    4    4

5    5    5    5
```

Like arrays, matrix elements are stored by row in ascending locations in memory. A matrix dimensioned as:

```
20 DIM A (3, 3)        (first dimension represents rows and second dimension represents columns)
```

is stored as:

Columns / Rows	0	1	2	3
0	A(0, 0)	A(0, 1)	A(0, 2)	A(0, 3)
1	A(1, 0)	A(1, 1)	A(1, 2)	A(1, 3)
2	A(2, 0)	A(2, 1)	A(2, 2)	A(2, 3)
3	A(3, 0)	A(3, 1)	A(3, 2)	A(3, 3)

The elements would be stored in the following order:

Element Position	Element
1	A (1, 1)
2	A (1, 2)
3	A (1, 3)
4	A (2, 1)
5	A (2, 2)
6	A (2, 3)
7	A (3, 1)
8	A (3, 2)
9	A (3, 3)

The 0 row and 0 column are both unused.

REDIMENSIONING MATRICES

Matrices, like arrays, may be redimensioned at any time by a DIM statement as long as the new dimensions do not exceed the size of the matrix given in the original DIM statement.

```
10 DIM A (15, 14)   <-- 210 elements in matrix (240 in array A)
20 DIM A (20, 7)    <-- 140 elements
30 DIM A (10, 10)   <-- 100 elements
40 DIM A (20, 8)    <-- 160 elements
```

MATRIX MATHEMATICS

Matrix Assignment

```
MAT A = B
```

The elements of matrix B are stored in matrix A. 'A' must have already been dimensioned. Given the statement:

```
10 MAT A = B
```

where: B is the matrix:

$$\begin{pmatrix} 2 & 4 & 6 & 8 \\ 0 & 3 & 9 & 7 \end{pmatrix}$$

Matrix A will assume the dimensions of B and the values:

$$\begin{pmatrix} 2 & 4 & 6 & 8 \\ 0 & 3 & 9 & 7 \end{pmatrix}$$

A and B may have different dimensions. If A is not large enough to be dimensioned as B, an error message will result.

Addition and Subtraction

```
MAT A = B + C
MAT A = B - C
```

Matrices B and C must have the same dimensions. Only a single arithmetic operation is permitted in one statement. The operands of the addition and subtraction operation may be the same as the matrix appearing on the left hand side of the = sign.

```
A = B + C - D   <-------- illegal
A = B + C      }
A = A - D      } <-------- legal
A = A + A      }
```

Matrix addition and subtraction is scalar arithmetic performed element by element. Given the statement:

```
10 MAT A = B + C
```

If B and C are matrices having the values:

$$\begin{pmatrix} -2 & -5 \\ 3 & 4 \\ .5 & .1 \end{pmatrix} \text{ and } \begin{pmatrix} 6 & 4 \\ -2 & 0 \\ 1.5 & 4 \end{pmatrix}$$

then the resultant value for A is:

$$\begin{pmatrix} 4 & -1 \\ 1 & 4 \\ 2 & 4.1 \end{pmatrix}$$

Scalar Multiplication

```
MAT A = (expression) * B
```

where: Expression may be any numeric expression and must be enclosed in parentheses.

Scalar multiplication is performed element by element. The matrix in the expression may be the same as the matrix variable on the left hand side of the = sign. Matrix A will assume the dimensions of matrix B. Given the statement:

```
10 MAT A = (LOG (X) ) *B
```

LOG (X) is evaluated first. If LOG (X) evaluates to 2.5 and B is the matrix:

$$\begin{pmatrix} 2 & -3 \\ .3 & 0 \end{pmatrix}$$

then A is the matrix:

$$\begin{pmatrix} 5 & -7.5 \\ .75 & 0 \end{pmatrix}$$

Constant Matrix

MAT A = (expression)

Each element of matrix A, except for values in row zero and column zero, is set to the value of expression by this statement. Expression must be numeric and enclosed in parentheses.

For example:

Assume A is a 3 x 3 array as follows:

2	3	6.5	7
1	7	18	0
3	1	1.5	-2
4	2	7	8

The statement MAT A = (0) will have the following results:

2	3	6.5	7
1	0	0	0
3	0	0	0
4	0	0	0

The statement MAT A = (X-X+1) will have the following results:

2	3	6.5	7
1	1	1	1
3	1	1	1
4	1	1	1

Identity Matrix

MAT A = IDN

The major diagonal of matrix A is set equal to ones and the remaining elements of the matrix are zeroed by this statement.

The major diagonal is the diagonal that starts at the final element of the array and runs diagonally upward from the last element until the first row is encountered.

Assume DIM A (4, 4), B(4, 1), C(2, 3)

Examples of the identity matrix are:

```
10 MAT A = IDN

1     0     0     0
0     1     0     0
0     0     1     0
0     0     0     1
```

In a square matrix, the major diagonal terminates at the first element of the matrix.

```
20 MAT B = IDN

0
0
0
1
```

If a matrix contains only one column, only the last element of the matrix is considered on the major diagonal.

```
30 MAT C = IDN

0     1     0
0     0     1
```

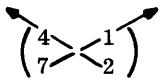
If a matrix is two-dimensional but not square, the major diagonal does not terminate at the first element of the matrix.

Matrix Determinant

MAT A = DET

The determinant of matrix A is computed and stored in element A (0, 0) of array A. Matrix A is unchanged. Assume A is a matrix as follows:

MAJOR OPERAND MINOR OPERAND



MAJ MIN
DET = (2 x y) - (7 x 1) = 8 - 7 = 1

```
10 MAT A = DET
20 PRINT A (0, 0)
```

The output produced is: 1

NOTE

The determinant of a 2 x 2 matrix is defined in the section Inverse Matrix.

Matrix Transposition

```
MAT A = TRN (B)
```

A matrix is transposed by **reversing** its rows and columns. A matrix cannot be transposed into itself.

```
200 MAT A = TRN (B)
```

where: $B = \begin{pmatrix} 9 & 5 & 5 & 9 \\ 0 & 0 & 0 & 0 \\ 1 & 3 & 8 & 7 \end{pmatrix}$

When the statement is executed, A assumes the transposed dimensions of B and

$$A = \begin{pmatrix} 9 & 0 & 1 \\ 5 & 0 & 3 \\ 5 & 0 & 8 \\ 9 & 0 & 7 \end{pmatrix}$$

Matrix Multiplication

```
MAT A = B*C
```

In matrix multiplication, the number of columns of the first matrix (B) must match the number of rows of the second matrix (C). The resultant matrix A is re-dimensioned to have the same number of rows as B and the same number of columns as C. If A is not large enough to be re-dimensioned in such a manner, an error message will result. For example:

```
10 DIM B (3,5), C (5,4), A (6,6)
   .
   .
   .
50 MAT A = B*C
```

A will become a 3 x 4 matrix.

The matrix appearing on the left hand side of the equals sign cannot appear as a matrix within the expression. Since the columns of B must match the rows of C, a statement of the form:

```
60 MAT A = B*B
```

requires that B must be a square matrix

To obtain the matrix product of B*C, each row of B is multiplied by each column of C. Each row/column set is added together to find the resultant matrix element. For example, given the following two matrices, B (3, 2) and C(2, 2):

$$B = \begin{pmatrix} 10 & 3 \\ 1 & 5 \\ 0 & 4 \end{pmatrix} \qquad C = \begin{pmatrix} -1 & -2 \\ 7 & 8 \end{pmatrix}$$

then:

```
130 MAT A = B*C
```

$$\begin{pmatrix} B(1,1)*C(1,1)+B(1,2)*C(2,1) & B(1,1)*C(1,2)+B(1,2)*C(2,2) \\ B(2,1)*C(1,1)+B(2,2)*C(2,1) & B(2,1)*C(1,2)+B(2,2)*C(2,2) \\ B(3,1)*C(1,1)+B(3,2)*C(2,1) & B(3,1)*C(1,2)+B(3,2)*C(2,2) \end{pmatrix}$$

$$\begin{pmatrix} 10*(-1)+3*7 & 10*(-2)+3*8 \\ 1*(-1)+5*7 & 1*(-2)+5*8 \\ 0*(-1)+4*7 & 0*(-2)+4*8 \end{pmatrix} = \begin{pmatrix} 11 & 4 \\ 34 & 38 \\ 28 & 32 \end{pmatrix}$$

Matrix multiplication is not associative. For example, an attempt to execute the statement, MAT A = C*B, using the matrices B (3, 2) and C (2, 2) defined above, will result in an error message since the number of columns of C do not match the number of rows of B. As another example, given the following two square matrices:

$$B = \begin{pmatrix} 10 & 3 \\ 1 & 5 \end{pmatrix} \qquad C = \begin{pmatrix} 0 & -1 \\ 4 & 6 \end{pmatrix}$$

then:

130 MAT A = B*C $\qquad A = \begin{pmatrix} (10*0+3*4) & (10*(-1)+3*6) \\ (1*0+5*4) & (1*(-1)+5*6) \end{pmatrix} = \begin{pmatrix} 12 & 8 \\ 20 & 29 \end{pmatrix}$

If the expression is reversed:

140 MAT A = C*B $\qquad A = \begin{pmatrix} (0*10 + (-1) * 1) & (0*3+(-1)*5) \\ (4*10 + 6*1) & (4*3+6*5) \end{pmatrix} = \begin{pmatrix} -1 & -5 \\ 46 & 42 \end{pmatrix}$

Inverse Matrix

MAT A = INV (B)

The matrix appearing in the expression must be a square matrix (at least 2 x 2). The matrix appearing on the left hand side of the statement may appear on the right hand side, i.e., matrices may be inverted into themselves.

Matrix inversion requires a knowledge of matrix determinants and of cofactors of matrix elements. Determinants and cofactors for 2 x 2 matrices are described here. For larger matrices, consult a mathematics text.

The determinant of a 2 x 2 matrix is obtained by multiplying along the diagonals and subtracting the second diagonal from the major diagonal;

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = (1*4) - (2*3) = -2$$

$$\begin{vmatrix} 1 & 5 \\ 3 & 20 \end{vmatrix} = (1*20) - (5*3) = 5$$

An inverse of a matrix is defined such that the product of a matrix and its inverse (via matrix multiplication) is always the identity matrix i.e., the product of the determinants of the matrix and its inverse is always one. The two matrices above would have inverse matrices whose determinants were -.5 and .2 respectively.

Cofactors of matrix elements of a 2 x 2 matrix are obtained by:

1. Reversing the elements along the major diagonal.
2. Changing the signs of the elements along the other diagonal.

To obtain the inverse matrix, scalar multiply the cofactors by the determinant of the inverse matrix:

If:

$$\text{MAT A} = \begin{pmatrix} 6 & 2 \\ 3 & 4 \end{pmatrix}$$

then:

$$\text{INV (A)} = (1/18)* \begin{pmatrix} 4 & -2 \\ -3 & 6 \end{pmatrix} = \begin{pmatrix} 2/9 & -1/9 \\ -1/6 & 1/3 \end{pmatrix}$$

If:

$$\text{MAT B} = \begin{pmatrix} 2 & 5 \\ 3 & 10 \end{pmatrix}$$

then:

$$\text{INV (B)} = (.2) * \begin{pmatrix} 10 & -5 \\ -3 & 2 \end{pmatrix} = \begin{pmatrix} 2 & -1 \\ -.6 & .4 \end{pmatrix}$$

By multiplying the matrix and its inverse we can verify the procedure:

$$\begin{pmatrix} 2 & 5 \\ 3 & 10 \end{pmatrix} * \begin{pmatrix} 2 & -1 \\ -.6 & .4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

BASIC will invert any square matrix except one having a determinant of zero, i.e., a singular matrix.

Because of inaccuracies inherent in matrix inversion, an inverse of a matrix may not be as accurate as desired. Accuracy may be improved by an error reduction approximation.

```
10   DIM A(N, N), B(N, N), C(N, N), D(N, N)
.
.
.
100  MAT A = INV (B)          Calculate the inverse of B.
110  MAT C = A*B
120  MAT D = IDN
130  MAT D = D+D
140  MAT C = D-C
150  MAT D = A*C              D is now an improved version of A.
160  MAT A = D
```

This process may be repeated as many times as necessary until the elements of C in 110 are as close to the identity matrix as desired.

A matrix to be inverted must have at least one element whose absolute value is greater than one. To invert a matrix not satisfying this requirement, multiply it be some constant (C) large enough to increase any element's absolute value to greater than one. After computing the inverse multiply the inverse matrix by 1/C.

MAT READ Statement

MAT READ list of matrices

The MAT READ statement is used to read values from DATA statements into the elements of a previously dimensioned matrix or list of matrices separated by commas.

```
10 DIM A(2,3), B (2,2)
20 MAT READ A, B
   .
   .
50 DATA 1, 2, 3, 4, 5, 6, -1, -2, -3, -4
```

Values from the data block are read into A and B by rows as follows:

Element	Contents
A(1,1)	1
A(1,2)	2
A(1,3)	3
A(2,1)	4
A(2,2)	5
A(2,3)	6
B(1,1)	-1
B(1,2)	-2
B(2,1)	-3
B(2,2)	-4

MAT INPUT Statement

MAT INPUT list of matrices

MAT INPUT ON (expression) list of matrices

The MAT INPUT statement is used to read values from an input device into the elements of a matrix or a list of matrices. The logical unit of the input device is determined by expression. The ON clause is optional and defaults to logical unit 5.

Example:
```
10 DIM A (2,3), B (2,2)
20 MAT INPUT ON (1) A, B
```

INPUT DATA is:
```
1, 2, 3
4, 5, 6
-1,-2,-3,-4
```

Values input are assigned to the matrix elements in the same manner as in the MAT READ statement.

The user inputs the data values for each element of the matrix, delimited by commas, on one or more input records.

The user must begin the values for each matrix on a new input record. A line of data containing too many values will cause an error message to print. BASIC will continue to INPUT until all matrix elements have been filled.

MAT PRINT Statement

MAT PRINT list of matrices

MAT PRINT ON (expression) list of matrices

The MAT PRINT statement is used to output the elements of a matrix or a list of matrices, by row on a list device whose logical unit is determined by expression. Each row of a matrix is printed on a new print line. If a row of values is too large to fit on a single print line, the excess columns are continued on the next line. Full print zone spacing is used if a matrix in the list is followed by a comma or carriage return. If a matrix is followed by a semi-colon compact zone spacing is used. The ON clause is optional and defaults to logical unit 5.

Examples:

$$\text{MATRIX A} = \begin{pmatrix} 1 & 2 & 3.5 & 4 \\ 0 & -1 & 3 & 2 \\ 2 & 4 & 5 & 6 \\ 3 & 7 & 0 & 1 \end{pmatrix} \quad \text{MATRIX B} = \begin{pmatrix} 1 & 5 & 6 & 7 & 3 & 2 \\ 1 & 7 & 4 & 3 & 1 & 4 \end{pmatrix}$$

$$\text{MATRIX C} = \begin{pmatrix} 1 & -3 \\ 4 & 0 \end{pmatrix} \quad \text{MATRIX D} = \begin{pmatrix} 1 & 2 & 4 & 11 & 13 & 15 \\ 20 & 21 & 22 & 23 & 24 & 25 \end{pmatrix}$$

```
30 MAT PRINT A, B; C, D

 1             2            3.5          4
 0            -1            3            2
 2             4            5            6
 3             7            0            1
 1 5 6 7 3 2
 1 7 4 3 1 4
 1            -3
 4             0
 1             2            4           11           13
 15
 20            21           22           23           24
 25
```

3.22 SETTRACE and ENDTRACE

Syntax: SETTRACE

ENDTRACE

Use: To cause the statement numbers of each BASIC statement being executed to be printed on the user's console.

After a SETTRACE statement, all statement numbers are printed until an ENDTRACE statement is encountered.

CHAPTER 4
COMMAND MODE OPERATIONS

4.1 INTRODUCTION

In command mode operation, the user can:

- Execute programs.
- Request information about the contents of his program and variables.
- Edit programs.
- Perform dynamic debugging.
- Perform desk calculator operations.
- Make assignments for file I/O and perform file directory maintenance under the Multi-user Executive only, (03-058).

These functions are carried out by issuing keyboard commands. Keyboard commands start with a command word, which may be followed by arguments, and terminate with a carriage return. Some of the commands are keyboard versions of certain BASIC statements; BASIC can recognize such a command since it is not preceded by a line number.

4.2 CONTROL KEYS

Escape key or Data switch 15

Pressing the Escape key (control, shift K) under the Multi-user Executive or depressing Data switch 15 under BOSS or DOS, essentially means "interrupt the current operation".

If a program is being executed, execution is suspended and the message:

STOP n

is printed, where n is the statement number before which execution ceased. The system reverts to command mode. If already in the command mode, either of the above actions is ignored. Neither of these features is available under other systems. If the program is running under OS/32ST or OS/32MT, pressing the escape key causes those systems to PAUSE the BASIC Interpreter.

#

When the user is writing and editing BASIC programs at the keyboard or when he is responding to an INPUT request, pressing the # key results in deletion of the line he is currently typing. He may then retype the line.

BASIC outputs a carriage return/line feed and the user may replace the deleted line, as shown in the following example:

```
80 PRINT "TATOL #
80 PRINT "TOTAL = ", X
```

←

When the user is writing and editing BASIC programs at the keyboard or when he is responding to an INPUT request, depressing the ← key results in the deletion of the last character in the current line. He may then retype the character.

The following example shows character deletion and replacement:

```
80 PRINT "TA ← OTO ← AL = ", X
```

The statement will appear in the program as:

```
80 PRINT "TOTAL = ", X
```

4.3 KEYBOARD COMMANDS

Keyboard commands begin with a keyword, recognized as a command by BASIC. Some commands include one or more arguments following the keyword. A keyboard command is terminated by pressing carriage return (←) and is immediately executed by BASIC.

All BASIC statements and File I/O statements may be written as keyboard commands except those that are only understood as part of the current program. The statements that cannot be used as commands are DEF, END, FOR, NEXT, ON, DATA, REM, STOP, GOTO, GOSUB, and RETURN. All other statements may be written as commands with their appropriate arguments. If a command does not begin with a keyword, BASIC expects an assignment (LET) command.

In addition to BASIC statements, there are a number of additional commands that are recognized by BASIC:

PAUSE	Terminate user-BASIC interaction and return to the operating system.
LOAD	Enter program statements from a file into the current program.
LIST	List statements of the current program.
NEW	Clear user area of all statements and variables.
RENUM	Renumber the statements in the current program.
RUN	Execute the current program.
SIZE	Print the size of the current program and the space still available.
DELETE	Delete a disc file name from the disc directory of files.
FILES	List file names in the disc directory.

4.4 PAUSE

Command: PAUSE

Use: This command is valid only for the BASIC Interpreter running under BOSS, DOS or RTOS. The PAUSE command gives control to OS at the keyboard. To continue BASIC, use the OS CONTINUE command.

Example:
```
PAUSE
AS 3,62      Assign the line printer under BOSS
CO           Continue BASIC
```

4.5 NEW

Command: NEW

Use: This command clears all currently loaded statements and variables. This command should be given before beginning input processing of a new current program.

Example: NEW

4.6 LOAD

The user can read a BASIC program into memory in several ways. It may be loaded from a disc file, input from the Teletype, or input from another device such as the paper tape or card reader. Once it is read in, it is called the current program. A current program can be listed, modified if necessary, and executed. It can also be saved as a file on disc or on another device.

Command: LOAD logical unit number

where: logical unit number refers to a device or file containing BASIC statements in ASCII format.

Use: This command causes the BASIC statements contained in the file to be read into the current program.

If statements in the file have the same statement number as a line in the current program, the new line replaces the current line. Statements in the file having statement numbers differing from those in the current program, are inserted in their proper sequence in the current program. Thus the user can write or edit lines in the current program using an ASCII file as input in the same way as he would input new program statements on the Teletype.

The file to be LOADed may have been created by a LIST command (see LIST command description) or could have been written off-line. Any ASCII format input device can be used for LOADing program lines.

The LOAD command terminates upon recognition of an END statement or abnormal device status (i.e., file mark).

Examples:
```
LOAD 2
LOAD
```

Batch Operation

The LOAD command permits batch BASIC operation from any sequential file. To perform a batch load and go operation, assign the desired logical unit and enter a LOAD logical unit number command. BASIC will continue to read both program statements and immediate mode commands from the input device until an-END statement is encountered.

LOAD 1 Entered from the keyboard (Logical Unit 1 assigned to the card reader).

The following is a list of batched card input to load and execute two programs.

```
99 REM PROGRAM 1
100 X = 1.235
101 PRINT ON (3) X, X+.03
RUN                          run first program
NEW                          clear first program
100 REM PROGRAM 2
200 PRINT "ALL IS WELL"
202 PRINT SIN (.356)
204 PRINT COS (.821)

LIST ON (3)                  list second program
RUN                          run second program
NEW                          clear second program
END                          return to keyboard control
```

4.7 LIST

Command: LIST

LIST ON (number)

LIST ON (number) statement-no$_1$

LIST ON (number) statement-no$_1$ TO statement-no$_2$

where: statement-no$_1$ is the first statement to be listed.

statement-no$_2$ is the last statement to be listed.

(number) refers to the logical unit number of the device to be listed on. The ON (number) clause is optional and defaults to logical unit 5.

Use: This command causes all or part of the current program to be listed in ASCII on the specified device.

LIST - List the entire program starting at the lowest numbered statement.

LIST n_1 - List only the single statement numbered n_1.

LIST n_1 TO n_2 - List from the statement numbered n_1 through statement n_2.

A file created by the LIST command can be read by BASIC using the LOAD command.

Examples:

```
LIST 200
LIST 600 TO 900
LIST
LIST ON (3)
LIST ON (3) 700 TO 9999
```

Programs that are listed after being entered from a keyboard or prepared off-line may appear slightly different from the original text: The following is a list of possible changes:

1. All unnecessary spaces are deleted.

2. Unnormalized E format numbers are normalized.

3. Leading zeros are deleted.

4. Spaces are inserted around reserved words that are not functions.

Examples:

Entered	Listed
100 PRINT X, Y	100 PRINT X, Y
100 LET X=. 03E-8	100 LET X=. 3E-9
010 NEXTX	10 NEXT X
100 LET X=1. 000	100 LET X=1

4.8 RUN

Command: RUN

RUN statement-no

where: Statement-no is the number of the statement in the current program where execution is to begin.

Use: This command causes all or part of a current program to be executed. The effects of the RUN commands are as follows:

RUN - Clears all variables, un-dimensions all arrays and strings, does a RESTORE, and then starts the current program at the lowest numbered statement.

RUN n - All existing information (variable values, dimensioning, etc.) resulting from a previous execution of the current program is retained and the current program is started at the statement numbered n. This form of the RUN command allows the user to examine variables of his program, modify statements and to retain current values when resuming execution.

Examples:
```
RUN
RUN 250
```

4.9 RENUM

Command: RENUM

RENUM $number_1$

RENUM $number_1$, $number_2$

Where: $number_1$ is the number given to the first statement.

$number_2$ is the statement number increment.

Use: The RENUM command causes all statements in the current program to be renumbered by assigning the initial statement a value of $number_1$ and incrementing by $number_2$ for each succeeding statement. If $number_1$ and $number_2$ are omitted their default values are assumed to be 10.

Examples:

RENUM	Renumber starting the first statement with 10 and incrementing by 10.
RENUM 500	Renumber starting the first statement with 500 and incrementing by 10.
RENUM 500, 5	Renumber starting the first statement with 500 and incrementing by 5.

If an IF, ON or GOSUB statement conatins a reference to a non-existent statement number, an error message (LN-ERR) results and the RENUM operation is aborted.

NOTE

If this condition occurs some line numbers may have been changed and others may not.

4.10 SIZE

Command: SIZE

USE: This command causes a printout at the terminal of the decimal number of bytes used by the program (including data areas if the program has been run) followed by the total number of bytes that are still available.

Example: SIZE

560 3750

560 bytes are used, 3750 bytes are remaining.

4.11 IMMEDIATE MODE BASIC STATEMENTS

Any BASIC statement that can meaningfully be written as a keyboard command can be used in that mode. Certain statements having meaning only within the context of a program cannot be used as keyboard commands; these are DEF, ON, END, FOR, NEXT, DATA, STOP, GOTO, GOSUB, RETURN, and REM. All other statements can be used as Immediate mode commands. Some uses of these statements are:

Desk Calculator. Forms of the PRINT command can be used to request values for expressions.

```
PRINT EXP (SIN (3.4/8) +.51032

;LOG (2.71)
```

Desk Calculator - Using Program Values. Besides numeric operands, the user can include program variables. The user can STOP a running program and examine values of his program variables.

```
10 LET X = 1.03          USER PROGRAM ENTRY
20 STOP                  USER PROGRAM ENTRY
RUN                      USER COMMAND ENTRY
STOP 20                  BASIC MESSAGE
PRINT X                  USER COMMAND ENTRY
1.03                     BASIC PRINT OUTPUT
PRINT X*X+3              USER COMMAND ENTRY
4.0609                   BASIC PRINT OUTPUT
```

Dynamic Program Debugging. A running program can be interrupted (using the ESCAPE key, Data switch 15 or by programmed STOP statements) at a number of different program points. The current values of the variables can then be checked at those points and corrections made in the program, either to the statements or variables, as necessary. The programmer can use the RUN command to re-execute or, use the RUN statement-no command to restart the interrupted program at the point of interruption.

Example:

```
.
.
.
(ESCAPE)          halt the program
STOP 300
PRINT X;Y         examine variables
.05 .03
LET X = .08       make modification
RUN 300           restart
```

4.12 ERASE

Command: ERASE

ERASE statement - no_1

ERASE statement - no_1 TO

ERASE statement - no_1 TO statement - no_2

Where: statement - no_1 is the first statement to be erased

statement - no_2 is the last statement to be erased

Use: This command causes all statements between and including the specified limits to be erased from the current program.

Examples:

ERASE	- Erase the entire program starting at the lowest numbered statement. This form is equivalent to NEW.
ERASE n_1	- Erase only the single statement numbered n_1.
ERASE n_1 TO	- Erase from statement n_1 to the end of the program.
ERASE n_1 TO n_2	- Erase from statement n_1 to statement number n_2.

CHAPTER 5
OPERATING INSTRUCTIONS

5.1 SINGLE-USER BASIC UNDER BOSS AND DOS

1. Load BOSS or DOSS with the 06-024 Relocating Loader, the 06-025 General Loader, or when loading from disc or drum, the 07-046 Bulk Storage Bootstrap Loader.

2. If there are any assembly language subroutines, they must be loaded now with the BOSS or DOS resident loader without setting any loader BIAS.

Example: LO 13 Load from the paper tape reader under BOSS.

3. Load the BASIC Interpreter with the BOSS or DOS resident loader. To obtain the maximum amount of BASIC work area, the user should not set a loader BIAS.

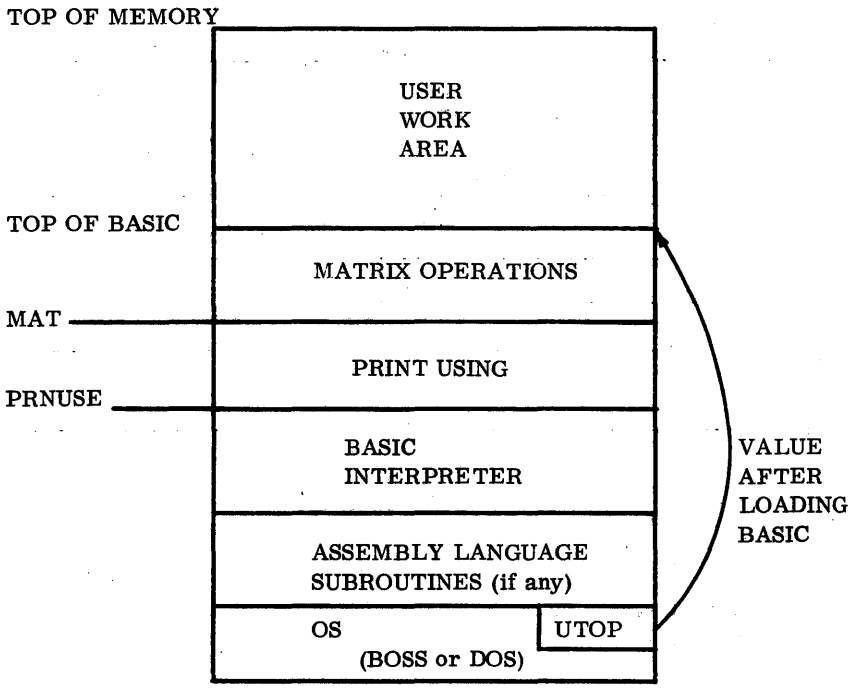
4. When loading is complete the user should issue a ST instruction with no operand to the OS. BASIC will print the message:

BASIC

and enter the command mode, ready for user statements.

If assembly language subroutines were loaded, the ST command should contain, as its operand, the BIAS of BASIC which can be found in the BIAS XXXX message that was printed by the OS when BASIC was loaded.

5. BASIC uses storage from the top of BASIC to the top of memory as the user's work area. If the user desires more work area and does not use MATRIX operations or the PRINT USING statement, he may allows his work area to overlay these sections of BASIC. After loading BASIC, location UTOP in DOS or BOSS points to the first location of the user work area. To gain added work area, UTOP should be set to point to location MAT in BASIC to overlay the MATRIX operations or to location PRNUSE to overlay both the MATRIX operations and the PRINT USING function. Note the following map:



See the BOSS or DOS listing for the address of location UTOP in the OS. See the BASIC Interpreter listing for the relative addresses of MAT and PRNUSE, and use the value in the BIAS XXXX message (printed when loading BASIC) to compute the absolute addresses of MAT and PRNUSE. After loading BASIC and before issuing the ST command, the user may reset UTOP using the OP and RE command in the OS.

(XXXX + relative address of MAT or PRNUSE) ⟶ UTOP

5.2 SINGLE-USER BASIC UNDER RTOS

The BASIC Interpreter may run under RTOS after being established as a "task" using the RTOS Task Establisher Program (TET).

1. Load RTOS from disc or drum with the 07-046 Bulk Storage Bootstrap Loader. RTOS will type the following on the system console:

   *SUPER HH:MM:SS RTOS

2. Load TET (03-042) from paper tape or the system library using one of the following commands:

   tape - LOAD TET , 13

   library - LOAD TET

   RTOS will respond with:

   *LODER HH:MM:SS TET: LOADED

3. Allocate scratch area for TET by using one of the following:

   ALLO C6, 20, 50 (Disc System, Device C6)

   or

   REWIND 5C6 (Drum System, Device 86)

4. Start TET by entering:

   START TET

   on the console. TET will reply:

   ENTER DATA

   after a series of RTOS system messages.

5. The following commands should now be entered to establish BASIC.

BASIC A	BASIC B	BASIC C
PRIO 2 OPT		
ASS1 5, 02	ASS1 5, 22	ASS1 5, 12

ESTA BASIC - names the task being created.

PRIO X - where X may be a number from 2 to F.

OPTI 0010 1000 0000 0000 - (makes BASIC core resident and allows floating point).

ASSI 5, X - where X is the physical device number of the BASIC terminal (e.g, X = 2 for a Teletype).

GET XXXX - where XXXX is the amount of space in hex to be allowed for the user work area.

At this time user written assembly language subroutines may be loaded. Load the subroutine table and subroutines (as described in Section 3.19) using the command LOAD XX where XX is the load device address.

LOAD XX - Load the BASIC Interpreter from some sequential device. (XX is the address of the device). If assembly language subroutine were loaded, enter the command MAP.

REWIND - GO TO RTOS

TASK XX - Output the entire task where XX is the physical device # of output device to contain the established BASIC.

END - End TET operation.

REWIND FILE

6. At this time the BASIC task is ready to be loaded by RTOS. Enter the RTOS command.

FORMULTI EXCL BASIC

LOAD BASIC, XX

where: XX is the physical device containing the BASIC task.

BASIC will load, and RTOS will respond with:

*LODER XX:XX:XX BASIC: LOADED

7. If no assembly language subroutines were loaded, BASIC may be started by issuing the RTOS command.

START BASIC

If subroutines were loaded, refer to the MAP output by TET. Under the heading ENTRYS is a hex address followed by the name BASIC. To start BASIC enter the command

START BASIC, , XXXX

where: XXXX is the MAP address of BASIC minus X'6E'

8. If the PRINT USING or MATRIX functions are not desired, their code in BASIC may be allocated for user work area by the following operation.

   Issue the RTOS command:

   ```
   MAP
   ```

   The system will respond with:

   ```
   BASIC XXXX -- YYYY
   ```

   where: XXXX and YYYY are the task limits in memory. The XXXX address plus four is the address of "UTOP", where an adjustment can be made to allow overlay of the unneeded BASIC functions.

   Calculate the new UTOP value by adding to X'6E' the relocatable address of MAT (to overlay MATRIX operations) or PRNUSE (to overlay PRINT USING and MATRIX operations) from the BASIC listing. That value should, in turn, be added to the 'XXXX' address obtained from the MAP command. The sum should then placed in the 'XXXX'+4 (UTOP) location by the REPL command.

   $$\begin{pmatrix}\text{MAT or}\\ \text{PRNUSE}\end{pmatrix} + X'6E' + XXXX \longrightarrow XXXX+4$$

5.3 MULTI-USER BASIC UNDER THE MULTI-USER EXECUTIVE (03-058)

Multi-user BASIC, with up to 32 terminals and file handling capability, may be obtained by using the Multi-user Executive with the BASIC Interpreter. For detailed system generation procedures and operating instructions, see the Multi-user Executive document 03-058A15, provided in the Multi-User Basic Operating System Documentation Package, S90-203M99.

5.4 MULTI-USER BASIC UNDER RTOS

The BASIC Interpreter may be used to create a multi-user terminal system under RTOS. To accomplish this, the BASIC Interpreter must be made part of the RTOS re-entrant library and individual dummy tasks must be created for each user. These tasks exist only to provide a user work area for each user and to allow the assignment of a unique device number for each user terminal. The number of users is limited only by the amount of memory on the system.

The 07-045F06 RTOS Re-entrant Library program, RLSTAB, must be modified and reassembled as follows:

1. Before the statement RLSEND EQU *, insert the following statements:

   ```
   EXTRN BASIC
   DC  C 'BASIC'
   DC  A (BASIC)
   ```

2. Re-assemble RLSTAB with the above changes and create a new RTOS load module following instructions in Chapter 6 of the RTOS Reference Manual, Publication Number 29-240. The BASIC object program must be linked into the load module before Initialize.

3. Each BASIC user requires a unique task (program listing below) that will provide the storage for the user work area and allow a unique terminal device number to be assigned to Logical Unit 5 (LU5). This task is established, loaded and started using the same procedure described in Section 5.2. Assembly language subroutines for each task may be loaded in front of each task. The following conditions also must be met.

   - Each task must have a different ID (ESTA Command).
   - The physical device assigned to LU5 must be different for each task.
   - LU 5 must be both an input and output device, i.e., a TTY or CRT
   - The TET command EXCL BASIC must be entered prior to the LOAD command.

The required task for each user is listed below:

```
        EXTRN   BASIC
START   B       BASIC       EXECUTE BASIC
        END


        OPEN    BASIC ie 0000
        REPL    start of BASIC
```

5.5 SINGLE-USER BASIC UNDER OS/32-ST

To operate BASIC under OS/32ST load the 32-bit relocatable object program (03-055) with the OS LOAD command, specifying a bias if desired. (See the OS/32-ST Program Reference Manual, Publication Number 29-380.)

Example: LOAD PTRP:, 3000 - Load BASIC from the paper tape reader.

When loading is complete the user should issue the OS START command. BASIC will print the message:

BASIC

and enter the command mode, ready for user statements.

For operation under OS/16MT and OS/32MT refer to the appropriate operating system manual for task establishment and loading procedures.

OS/16-MT Reference Manual	29-367
OS/32-MT Program Reference Manual	29-390
OS/32-MT Task Establisher Task (TET/32) User's Guide	29-412

APPENDIX 1

ERROR MESSAGES

When errors are encountered in BASIC, a message is printed on the terminal.

1. Errors while inputting a program.

   The Message XX-ERR is printed where XX are two alpha characters describing the error.

XX	Meaning
OF	Storage overflow
LN	No such line number
PF	Statement too complex
SY	Syntax error
WD	Unrecognizable word

2. I/O Errors

   The Message IO-ERR XXDD is printed on detection of an I/O error where XX is the device status and DD is the device number causing the error.

XX	Meaning
C0	Illegal function
A0	Device Unavailable
90	End-Of-Medium
88	End-Of-File
84	Unrecoverable Error

See the appropriate operating system document or the Multi-user Executive document 03-058A15 for details on device status.

3. Errors while executing a program.

The Message XX-ERR n is printed where XX are two alpha characters describing the error and n is the line number at which the error occurred.

XX	Meaning
AR	Arithmetic overflow or underflow or division by zero.
CA	Call to assembly language subroutine is undefined or contains the wrong number of parameters.
DA	DATA is exhausted or does not match variable type.
EX	Expression too complex for evaluation in available memory.
FN	Argument error in the LOG or SQR function or in the ↑ operation.
FR	FOR statement nested too deep.
GS	GOSUB's nested too deep, RETURN without GOSUB.
IN	Input data does not match INPUT statement variable list.
LN	No such line number.
MD	Matrix operation attempted within incompatible dimensions, or a resulting matrix of insufficient size.
MV	Matrix is singular and cannot be inverted.
NX	NEXT without FOR
OF	Out of storage while allocating a variable or array, or array re-dimensioning exceeds original maximum size.
PR	Missing parenthesis.
PU	Invalid PRINT USING format field.
RF	Reference to undefined variable or array.
SB	Illegal subscript.
SU	String undefined or missing.
SY	Syntax Error.
UF	User function undefined.

APPENDIX 2

PROGRAM SIZE AND TIME ESTIMATES

PROGRAM SIZE:

1. In 16-bit BASIC, each user program requires 426 bytes for pointers, buffers, temporary storage, etc. BASIC uses the first 426 bytes of the users work area for this purpose.

2. In 32-bit BASIC, each user program requires 558 bytes for pointers, buffers, temporary storage, etc. BASIC uses the first 558 bytes of the users work area for this purpose.

3. Each character in a program statement requires a single byte of user work area, in both 16-bit and 32-bit BASIC versions with the following exceptions:

   - All reserved words require only 1 byte.
   - All integer constants between 01 and 63 require only one byte.
   - Constants that are not integers from 0 to 63 require 5 bytes of user work area.
   - All line numbers require 2 bytes plus a one byte character count.

4. User variables and arrays require various amount of storage.

	16-bit BASIC	32-bit BASIC
- simple numeric variable	6 bytes	8 bytes
- 1D numeric array A(N)	4*(N+1) +8 bytes	4*(N+1)+8 bytes
- 2D numeric array A(N, M)	4*(N+1)*(M+1)+8 bytes	4*(N+1)*(M+1)+8 bytes
- string variable A$(N)	N+5 bytes, N odd N+6 bytes, N even	$\left[\text{Integer of } \left(\frac{N+5}{4}\right)\right]$*4 bytes
- string array A$(N, M)	M*(N+1) +4 bytes	$\left[\text{Integer of} \left(\frac{M*(N+1)+4}{4}\right)\right]$ *4 bytes

TIME ESTIMATES:

The following programs provide execution time estimate for various BASIC statements.

PROGRAM	16-bit BASIC RUN TIME (Model 70)	32-bit BASIC RUN TIME (Model 7/32)
FOR X=1 TO 1000 NEXT X	.2 seconds	.27 seconds
FOR X=1 TO 1000 Y=1 NEXT X	.9 seconds	.83 seconds
FOR X=1 TO 1000 Y=SIN (X) NEXT X	2 seconds	1.8 seconds
FOR X=1 TO 1000 Y=LOG (X) NEXT X	2 seconds	1.8 seconds
FOR X=1 TO 1000 Y=EXP (X) NEXT X	8 seconds	8 seconds
FOR X=1 TO 1000 Y=ATN (X) NEXT X	2 seconds	2 seconds
DIM A (20, 20), B(20, 20), C(20, 20) MAT A=INV (B)	7.5 seconds	7.5 seconds
MAT A=DET (A)	2.5 seconds	2.5 seconds
MAT C=A*B	2 seconds	2 seconds

APPENDIX 3

ASCII CODE CONVERSION TABLE

CHARACTER	DECIMAL	7-BIT ASCII CODE
NULL	0	0
SOM	1	1
EOA	2	2
EOM	3	3
EOT	4	4
WRU	5	5
RU	6	6
BELL	7	7
FE_0	8	8
HT/SK	9	9
LF	10	A
VT	11	B
FF	12	C
CR	13	D
SO	14	E
SI	15	F
DC_0	16	10
X-ON	17	11
TAPE-ON	18	12
X-OFF	19	13
TAPE-OFF	20	14
ERR	21	15
SYNC	22	16
LEM	23	17
S_0	24	18
S_1	25	19
S_2	26	1A
S_3	27	1B
S_4	28	1C
S_5	29	1D
S_6	30	1E
S_7	31	1F

CHARACTER	DECIMAL	7-BIT ASCII CODE	CARD CODE	CHARACTER	DECIMAL	7-BIT ASCII CODE	CARD CODE
SPACE	32	20	BLANK	@	64	40	8-4
!	33	21	12-8-7	A	65	41	12-1
"	34	22	8-7	B	66	42	12-2
#	35	23	8-3	C	67	43	12-3
$	36	24	11-8-3	D	68	44	12-4
%	37	25	0-8-4	E	69	45	12-5
&	38	26	12	F	70	46	12-6
'	39	27	8-5	G	71	47	12-7
(	40	28	12-8-5	H	72	48	12-8
)	41	29	11-8-5	I	73	49	12-9
*	42	2A	11-8-4	J	74	4A	11-1
+	43	2B	12-8-6	K	75	4B	11-2
,	44	2C	0-8-3	L	76	4C	11-3
-	45	2D	11	M	77	4D	11-4
.	46	2E	12-8-3	N	78	4E	11-5
/	47	2F	0-1	O	79	4F	11-6
0	48	30	0	P	80	50	11-7
1	49	31	1	Q	81	51	11-8
2	50	32	2	R	82	52	11-9
3	51	33	3	S	83	53	0-2
4	52	34	4	T	84	54	0-3
5	53	35	5	U	85	55	0-4
6	54	36	6	V	86	56	0-5
7	55	37	7	W	87	57	0-6
8	56	38	8	X	88	58	0-7
9	57	39	9	Y	89	59	0-8
:	58	3A	8-2	Z	90	5A	0-9
;	59	3B	11-8-6	[	91	5B	12-8-2
<	60	3C	12-8-4	\	92	5C	11-8-1
=	61	3D	8-6	]	93	5D	11-8-2
>	62	3E	0-8-6	↑	94	5E	11-8-7
?	63	3F	0-8-7	←	95	5F	0-8-5

TABLE OF MATHEMATICAL CONSTANTS

Constant	Decimal Value
π	3.141593
π -1	0.3183099
$\sqrt{\pi}$	1.772454
Ln π	1.14473
e	2.718282
e^{-1}	0.3678794
$\sqrt{e}$	1.648721
$\log_{10}e$	0.4342945
$\log_{2}e$	1.442695
γ	0.5772157
Ln γ	-0.5495393
$\sqrt{2}$	1.4142114
Ln2	0.6931472
$\log_{10}2$	0.30103
$\sqrt{10}$	3.162278
Ln10	2.302585

APPENDIX 4

REVISION INFORMATION

The 'BASIC' entry point to the Interpreter has been added and the DIM statement has been modified to zero the length bytes in string arrays.

The following BASIC commands, statements, and functions are added to the BASIC Interpreter, R01.

BASIC Commands

RENUM	Renumber user program in core starting
RENUM n_1	the first line with n_1 and incrementing
RENUM n_1,n_2	by n_2. Default values will be 10 and 10.
ERASE	Erase entire program. (Same as NEW)
ERASE n_1	Erase only line n_1.
ERASE n_1 TO	Erase all lines from n_1 to end of program.
ERASE n_1 TO n_2	Erase all lines from n_1 to n_2.

BASIC Statements

SETTRACE	Print the line numbers of all statements
ENDTRACE	executed until encountering ENDTRACE.
ON ERROR GO TO n_1	Go to line n when
ON ERROR THEN n_1	an execution error occurs.

ON expression GO SUB n_1,n_2 .....

If expression evaluates to 1 transfer will be to subroutine n_1, if expression evaluates to 2 transfer will be to subroutine n_2.

If expression evaluates to an integer less than one or greater than the sequence number of the last statement number in the list, the ON statement is ignored and control passes to the next statement.

If expression does not evaluate to an integer, it is truncated to an integer by the INT function.

BASIC Functions

ERR$ (X)	After an execution error has occurred these functions
ERL (X)	return a two character error code (ERR$) or the line number where the execution error occurred (ERL).
VAL (S)	Converts character string S to a numeric variable.
STR$ (X)	Converts numeric variable X to a character string.

The BASIC Interpreter R02 has been rewritten in common mode CAL and the object program is available in two forms:

03-055Mx6R02	For 16 bit processors
03-055Mx1R02	For 32 bit processors

where x is the media designation

x = 1 paper tape
2 cassettes
3 magtape

The PRINT statement has been changed to print 132 character output.

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

Error (Page No. ___, Drawing No. _____)
Addition (Page No. ___, Drawing No. _____)
Other (Page No. ___, Drawing No. _____)

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple
No postage necessary if mailed in U.S.A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT No. 22
OCEANPORT, N.J.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY:

INTERDATA®

2 Crescent Place, Oceanport, New Jersey 07757

TECH PUBLICATIONS DEPT. MS 53

FOLD FOLD

STAPLE STAPLE