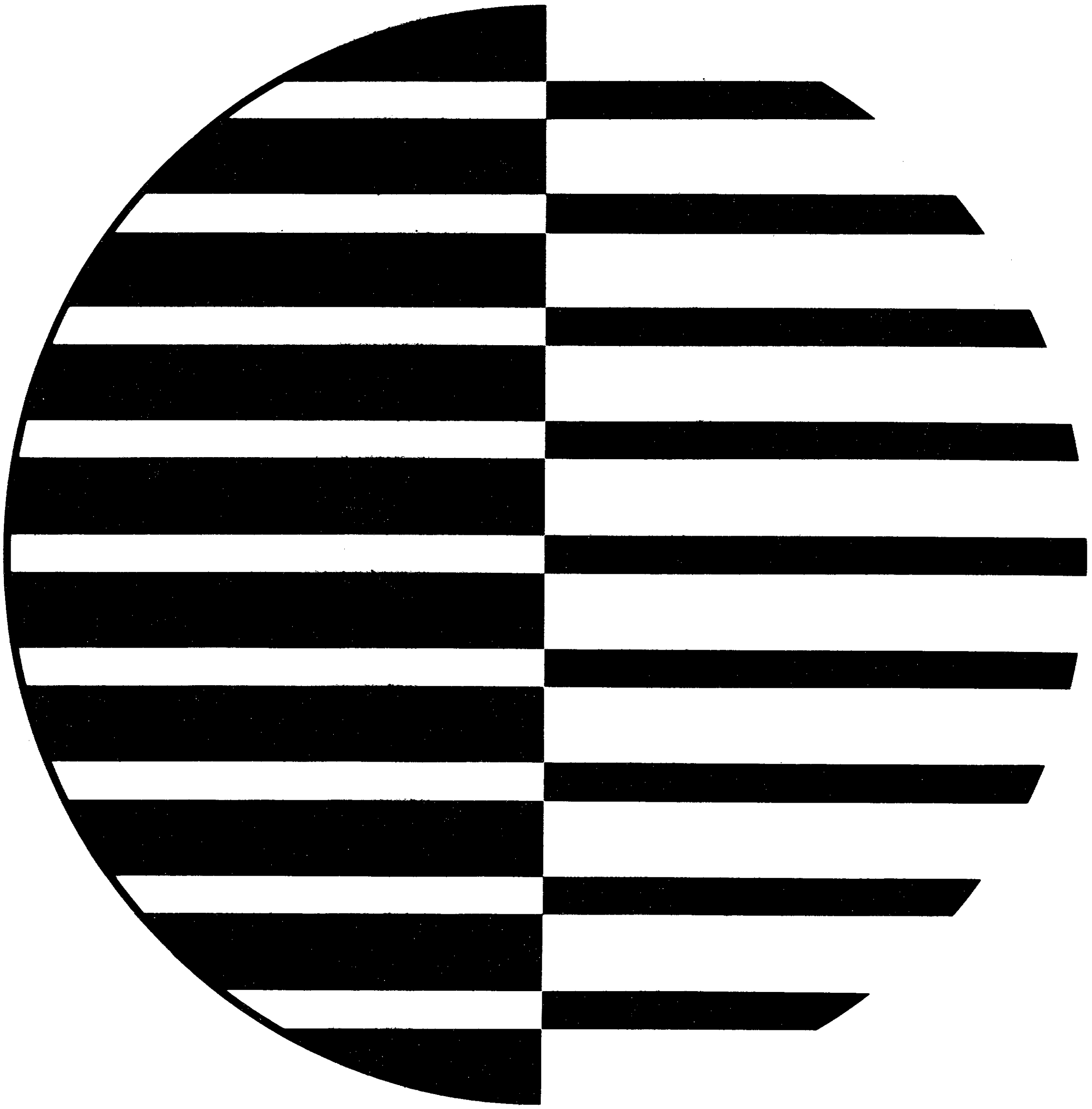
CONTROL DATA®

6000 SERIES COMPUTER SYSTEMS

BASIC Language Reference Manual



REVISION RECORD	
REVISION	DESCRIPTION
A	Original printing.
(2-13-70)	
Publication No. 60305000	

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Documentation Department
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

PREFACE

This manual describes the BASIC programming language for CONTROL DATA® 6000 Series Computers.

Methods employed to write, compile and execute BASIC programs in both remote terminal and local batch environments are discussed in detail. To simplify the user's task, terms have been narrowly defined and are not generally applicable outside the immediate context.

CONTENTS

INTRODUCTION

BASIC (beginner's all-purpose symbolic instruction code) enables the layman to write programs in a language resembling standard mathematical notation. Because of its simplicity and flexibility — as well as its precision — BASIC is used by programmers and nonprogrammers alike.

Designed and implemented at Dartmouth College, under the direction of Professors John G. Kemeny and Thomas E. Kurtz, the original version of BASIC is known as Dartmouth BASIC. CONTROL DATA'S version (BASIC 1.0), developed for use with CONTROL DATA 6000 Series Computers, provides optimum compatibility with the Dartmouth version.

BASIC is commonly used in time sharing environments where a number of users can gain simultaneous computer access; input is generated at a remote terminal, such as an electric typewriter or similar keying device, and is transmitted through telephone lines to a central computer facility.

The language designation, BASIC, refers to both the source language in which a programmer codes his instructions, and to the medium that translates source code into a form recognizable to a particular computer. The translating medium for BASIC is a program called a compiler. During the translation process source code emerges as object code — the form in which the computer executes all instructions.

BASIC 1.0 incorporates a communication feature which enables the user to modify his program or correct compilation errors at the terminal prior to program execution. Terminal mode may be bypassed in favor of batch mode; jobs may be submitted at the terminal and executed in direct sequence at the central computer.

An integral part of every computer system is the operating system. Synonymous with supervisory or executive systems, operating systems generally consist of a collection of program sets which oversee and control the performance of the total system. The operating system for CONTROL DATA 6000 Series Computers is SCOPE (supervisory control of program execution). Remote terminal processing is controlled through INTERCOM I — an adjunct to the SCOPE operating system. Detailed information relative to SCOPE and INTERCOM is presented in the following publications:

6000 Series Computer System Reference Manual (Publication No. 60100000)

SCOPE 3 Reference Manual (Publication No. 60189400)

6000 Series Computer System INTERCOM I Reference Manual (Publication No. 60305300)

ELEMENTS OF THE BASIC LANGUAGE 1

The BASIC language is oriented toward numerical problem solving; as such, it is primarily concerned with the evaluation of expressions. The following paragraphs describe the components and the methods by which they may be combined to form BASIC expressions.

CHARACTER SET

The BASIC language utilizes all alphanumeric characters (A to Z and 0 to 9). Special characters which serve as separators, delimiters, or operators are discussed in sections covering their usage.

The representation of BASIC symbols may vary depending on the character set of different hardware devices.

All BASIC characters acceptable to the 6000 Series and their internal representation (display code) are listed in Appendix A.

OPERANDS

BASIC uses three classes of operands: constant values, variable values and functions. Refer to section 3 for a description of BASIC functions.

CONSTANTS

BASIC recognizes numeric and string constants. Numeric constants represent values which do not change during program execution. String constants represent information to be printed during program execution.

Numeric Constants

Numeric constants, written as strings of signed or unsigned numeric characters, can be expressed with or without decimal points. The following are examples of valid numeric constants:

```
    1      -1      33333     -10003

    34.5      -23.445      6.      -.3      -0.3
```

A numeric constant may contain any number of numeric characters; but the value of the constant must not exceed the capacity of the computer, approximately 1E-308 to 1E337. The computer is capable of representing values with an accuracy of about 14 significant digits.

Very large or small numbers utilize a shorthand form of expression. The numeric constant is followed by an exponent symbol, E. The number .000789 could be expressed as 7.89E-4. The letter E signifies "times 10 to the power of."

7.89E-4 is equivalent to 7.89×10^{-4}.

Examples:

```
-1.45E3      1.0032E+20      0.0234E-12      123456E-11
```

String Constants

String constants serve as comments or messages. Restricted to PRINT statements, their usage is described in section 4.

VARIABLES

A variable represents a quantity having a value that may change during program execution. The size of the value of a variable must not exceed the capacity of the computer, which is capable of representing values from approximately 1E-308 to 1E337. All variables, represented internally as floating point numbers, are preset to zero at the beginning of program execution. BASIC variables are written as identifiers; each consists of a single alphabetic character with or without a succeeding single numeric character. They must not exceed two characters. For example:

```
A      Z3      C9      E
```

The following types are not allowed:

```
B23      49      G*      AA
```

Scalar Variables

A scalar variable has a single value and is written as an identifier without subscripts. Whenever an identifier is used in an expression, the computer substitutes its current value.

Array Variables

An array variable designates the elements of an array such as a vector or a matrix; it is written as an identifier with subscripts. The general form of the array variable is:

```
A (s1,s2,s3)
```

Up to three subscripts are allowed. They are separated by commas and enclosed in parentheses. Each subscript may be an expression of any complexity. The following array variables are acceptable:

```
A(1)     B(2,5)     G3(X*5+A3↑9,7*(A+1))     X(A5↑6*4,3*(Y+2),C)
```

The individual items in an array, called elements, indicate the position of a value within the array. If the subscript is an arithmetic expression, it will be evaluated and converted to an integer by truncating before it is used as a subscript.

Variables in BASIC are normally declared implicitly. Unless otherwise specified, each subscript of an array variable is assumed to have bounds of 0 and 10. Thus, an array variable with one subscript has 11 elements, two subscripts has 121 elements, and three subscripts has 1331 elements. With a DIM statement (refer to section 2), variables need be declared only if a subscript value greater than 10 is required, or if the programmer wishes to save space by dimensioning an array to have an upper bound of less than 10.

The same identifier may be used to denote both a scalar variable and an array variable within the same BASIC program. However, two or more array variables may not have the same identifier if the number of subscripts varies.

OPERATORS

Arithmetic operators, used with constants, variables, and parentheses to form expressions, are listed below:

Operator	Definition
↑ or **	exponentiation
*	multiplication
/	division
+	addition
-	subtract/negation

Relational operators, used to compare two expressions, are as follows:

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

ARITHMETIC EXPRESSIONS

An arithmetic expression (or formula) is a rule for computing a value. Expressions are composed of alternating operands and arithmetic operators, beginning and ending with an operand. During evaluation of the expression, the current values of the operands are used to compute a single value according to the rules for executing the arithmetic operators.

Arithmetic operators may not appear in sequence and must be explicitly stated. The following are invalid arithmetic expressions:

```
    X++Y     (X+1)(Y+2)
```

Rule for precedence of arithmetic operations: Exponentiation is performed first, followed by multiplication or division, and finally addition or subtraction is performed. Expressions are evaluated from left to right.

Example:

```
    X+(Y+(A+3))
```

This expression is evaluated in this order:

1. A + 3

2. Y + the result of 1

3. X + the result of 2

Left and right parentheses must balance within an expression. Redundant parentheses have no effect; for example, (X), X and ((X)) are all representations of the same expression.

Any expression may be preceded by a + or - arithmetic (unary) operator. They are called unary because they affect only one variable or constant. To maintain compatibility with Dartmouth BASIC, unary operations are performed before other arithmetic operations. For example, the value of -2 ↑ 2 is +4. The following are examples of valid arithmetic expressions:

```
    A+B*C/D↑E

    A1(3,I+4)↑2.6-G3/Z

    A+B↑C

    (A(I.J.K)+3.95)*(G3+B↑(C2+3))

    G2(A(1,4)+3,I2)↑(D4↑(X+3))

    -3.14*R↑2

    C*(-G2(1,3)-D4)

    A8(3)↑(-49)+B(2,-X+15)
```

RELATIONAL EXPRESSIONS

Relational expressions are formed by operands, arithmetic expressions and/or mathematical functions joined by relational operators. When evaluated, relational expressions may have one of two values—true or false. A relational expression may have only one relational operator. The following are legal relational expressions:

```
    A ≥ B

    X > COS(B)

    (F+I 6) ≤ B/5
```

Relational expressions compare two values or expressions. They may appear only in IF statements (section 2).

BASIC STATEMENTS 2

BASIC PROGRAMMING

A BASIC program is composed of statements formed by the components of the BASIC language. Each statement is a step in the solution of the program's problem and tells the computer how to evaluate an expression and assign its results. Each statement occupies one line and cannot be continued to another line.

LINE NUMBERS

Each BASIC statement must begin with a unique line number of one to five numeric characters. These numbers identify statements in the program and provide a means for editing the program at the terminal.

The line number is assumed to terminate at the first non-numeric character. Embedded blanks are ignored and leading zeros are not significant; 1, 01, and 001 are all considered the same line number. Line numbers must be integers within the range 0 to 99999.

The BASIC compiler checks for missing or duplicated line numbers. Further, the compiler ascertains whether line numbers are in ascending order. When a duplicate line number is encountered in a program entered from a terminal, the INTERCOM I system will arrange the statements in ascending order and interpret the duplicate line as a replacement for the first line of the same number. When a program is submitted to the SCOPE batch of jobs, the programmer must insure that line numbers appear in ascending order; as statements are processed as they are received.

Throughout this manual a line number is assumed for each BASIC statement.

STATEMENT FORMAT

A statement consists of up to 72 characters of information written in columns 1-72 of a punched card or typed at a terminal keyboard for direct transmission to the computer. A statement on a punched card terminates at column 72. A statement typed in at a terminal terminates when the carriage return key is pressed; in any case, no more than 72 characters are translated by the BASIC compiler.

A BASIC statement may begin and end at any character position. Blanks have no significance, except within string constants; and they may be included or omitted without changing the meaning of the statement.

BASIC STATEMENTS

After the line number, each BASIC statement begins with a word which indicates the statement type. The following are the simple BASIC statements. Functions and subroutines, matrix statements and input/output statements are discussed in subsequent chapters.

DIM STATEMENT

The DIM statement declares the dimensions of an array variable. Variables need a DIM statement if a subscript value greater than 10 is required. If the programmer wishes to save space, he also may use the DIM statement to dimension an array so that the upper bound is less than 10.

Format:

```
    DIM v1 (i1), v2(i2), v3(i3),...vn(in)
```

v	Name of array variable
i	Upper limit of array, must be an unsigned integer

Example:

To declare a two dimensional array variable, A, with elements 0 to 20 and 0 to 5:

```
DIM A(20,5)
```

This statement reserves space for array A with 21*6 or 126 elements. Each time A is used in the program, it must be used with two subscripts.

DIM statements may appear anywhere in a program. If the same array variable is declared more than once in the same program, the last declaration is used for the entire program.

Examples of acceptable DIM statements:

DIM X(5,5), B3(1,2), X1(50) reserves space for a two dimensional array X with 36 elements, a two dimensional array B3 with 6 elements, and a one dimensional array X1 with 51 elements.

DIM G2(5,6,7),A0(9,2) reserves space for a three dimensional array G2 with 336 elements and a two dimensional array A0 with 30 elements.

LET STATEMENT

The LET statement assigns a value to a variable during the execution of a BASIC program.

Formats:

```
    LET v1=v2=v3...=vn=e

        v1=v2=v3...=vn=e

         vi   Variables

         e    Expression
```

Use of the word LET is optional.

Example:

```
    LET X=2 assigns the value 2 to the variable X

    X=Y=2 assigns the value 2 to both X and Y

    LET X=A*B assigns the value of the expression A*B to X
```

The order of computation is left to right. Lefthand-side subscript expressions are evaluated before righthand subscript expressions (the latter are positioned after the final equal sign).

Example:

```
    LET I=1

    LET A(I) = I = I+1

    is equivalent to

    I=1

    A(I) = I+1

    I = I+1
```

Examples:

```
    LET X = (A+B)*(C+D)

    Y(1,A(3)) = Q = 2*3.14*R ↑ 2

    A1(G+99*X ↑ 2) = 2*(X3+B4)*X(Q2,23↑2) ↑(-8)+6
```

GO TO STATEMENT

The GO TO statement transfers control from one point in a program to another, and interrupts the normal sequence of instructions.

Format:

```
    GO TO n

          n    Line number of statement to which control transfers
```

This statement causes the statement at the referenced line number to be executed. Normal sequential execution will follow from that point. If the GO TO references a non-executable statement, such as a DIM statement, the system will go to the next executable statement after the referenced statement and resume execution from there.

IF STATEMENT

The IF statement specifies conditions to control the sequence of operations.

Format:

```
    IF r THEN n

          r    Relational expression

          n    Line number to which control transfers conditionally
```

If the relational expression is TRUE, program control transfers to the specified line number of the statement. If the relational expression is FALSE, the next sequential statement is executed.

Example:

If I=8 and J=4, the statement:

```
IF 2*I  ≥  J ↑ 2-1 THEN 165
```

Causes the value 16 to be compared to 15. Since it is true that 16 is greater than 15, the next statement executed is at line number 165.

FOR AND NEXT STATEMENTS

The FOR and NEXT statements allow repetition of specified portions of a program (loop).

Formats:

```
    FOR sv = e1 TO e2

    FOR sv = e1 TO e2 STEP e3
```

sv	Any scalar variable
e1	Initial value assigned to sv
e2	Maximum value sv can reach
e3	Value by which sv is incremented when the NEXT statement is executed; if not specified, a step value of +1 is assumed.

Format:

```
        NEXT  sv
```

sv	Scalar variable used in the companion FOR statement

The scalar variable must be the same in the FOR and NEXT statements; it is known as the control variable.

When the FOR statement is executed, the expressions are evaluated; and their values are saved as initial, step, and final values of the loop. The control variable is assigned the initial value, and the statements between the FOR and NEXT statements are executed repeatedly. Each time the NEXT statement is reached, the value of the control variable is incremented or decremented by the step value (one if not specified). The control value is compared to the final value. If the step value is positive, the loop is continued until the control value becomes greater than the final value. If the step value is negative, the loop is continued until the control value becomes less than the final value. In either case, the loop is complete; and execution continues in normal sequence, beginning with the statement following NEXT.

Example of a BASIC loop routine:

```
    10  FOR X=1 TO 11 STEP 2
    15  LET Y=X+1
    20  PRINT Y
    30  NEXT X
    40  END
```

The successive values of X (1,3,5,7,9, and 11) will determine the value of Y, which will be output as the results of the routine. Statements 10 through 30 will be repeated six times, once for each value assigned to X.

The expressions in the FOR statement are evaluated only once — at the start of the loop. These values are not changed during execution of the loop, even though the values of the variables in the expressions may change. The value of the control variable, however, may be changed by statements within the loop; its latest value is always updated by the step value and is used in comparison to the final value.

Example:

```
    10  FOR X=1 TO 10
    20  LET X=X+1
    30  PRINT X
    40  NEXT X
```

The FOR statement specifies that X will be incremented by a step value of +1 until it exceeds 10; however, the next statement also adds +1 to X which causes the counter variable X to be incremented by +2 each time it passes through the loop.

If the initial value is greater than the final value and the step value is positive when the FOR statement is first entered, the loop is not executed; and control passes to the statement following the companion NEXT statement. Similarly, if the initial value is less than the final value and the step value is negative, the loop is not executed. However, in both cases the control variable retains the value it had before entrance to the loop.

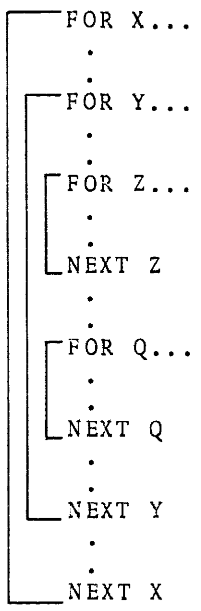
The following examples show the effect of FOR statements on control variables:

Statement	Values of Control Variable
FOR X=-4 TO-2 STEP .5	-4.,-3.5,-3.0,-2.5,-2.0
FOR G= 6 TO 3 STEP -1	6, 5, 4, 3
FOR Z= 5 TO 10 STEP -1	loop is not executed

Nested Loops

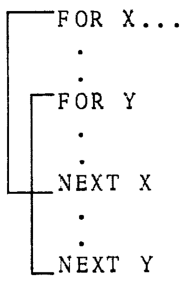
As illustrated below, loops may contain loops; however, they must not intersect each other.

Correct:



```
FOR X...
 .
 .
FOR Y...
 .
 .
FOR Z...
 .
 .
NEXT Z
 .
 .
FOR Q...
 .
 .
NEXT Q
 .
 .
NEXT Y
 .
 .
NEXT X
```

Incorrect:



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

GO TO Statements in Loops

Loops may contain GO TO and other statements that jump outside the range of the loop. In this case, the loop terminates prematurely and the control variable retains its latest value. It is possible also for a GO TO statement to jump to a statement within a FOR loop; but since no check is made at compile or execution time to determine that a NEXT statement is executed before its corresponding FOR statement, the results are unpredictable.

END STATEMENT

The END statement signals termination of a program. Every program must have an END statement as the last and highest line number. When it is executed, the program stops and control returns to the operating system.

Format:

```
    END
```

STOP STATEMENT

The STOP statement halts program execution and returns control to the operating system. Unlike the END statement, the STOP statement may appear at any point in a program.

Format:

```
    STOP
```

STOP is equivalent to a GO TO line number statement that is the program's END statement.

REM STATEMENT

A programmer uses the REM statement to insert comments in the body of the program.

Format:

```
    REM   any string of valid 6000 series display code characters
```

REM is non-executable and has no effect on the execution and results of a program.

FUNCTIONS AND SUBROUTINES 3

STANDARD FUNCTIONS

The standard mathematical functions evaluated by BASIC are described below:

Function	Meaning
SIN(x)	Find the sine of x expressed as an angle in radians
COX(s)	Find the cosine of x expressed in radians
TAN(x)	Find the tangent of x
ATN(x)	Find the arctangent of x in the principal value range $-\pi/2$ to $+\pi/2$
EXP(x)	Find the value of e^x
LOG(x)	Find the natural logarithm of x
ABS(x)	Find the absolute value of x
SQR(x)	Find the square root of x
INT(x)	Find the largest integer not greater than x Example: INT(5.95) = 5 and INT(-5.95) = -6
SGN(x)	Assign a value of 1 if x is positive; 0 if it is 0; or -1 if x is negative
RDN(x)	Find a random number between 0 and 1. If the value of x is 0, a standard random number sequence is used. If x is positive, its value is used to initiate a random number sequence. If negative, a random number is generated to initiate the random number sequence.

In all the above functions, x can be any expression.

A function is an operand and may be used in any expression wherever a variable or numeric constant may be used. Like a scalar variable, a function has a single value; unlike a variable, some calculation may be required to produce that value. When a function is written in a program, it is essentially a request for a procedure or routine to compute a value.

In the above list, the quantity x is called the argument or parameter of the function. The value determined by the function may be directly dependent on this argument. A function is called in an expression by writing the name of the function followed by the argument in parentheses. The parameter may be an arithmetic expression of any complexity and may include other functions; however, no function call may contain more than one parameter.

The following are acceptable expressions using mathematical functions:

```
SIN(A-b)

ABS(48*2.3↑8/44)

COS(12+SIN(Y))
```

USER FUNCTIONS

In addition to the standard functions provided by BASIC, a programmer may define new functions with the following statement.

DEF STATEMENT

To avoid repetition of coding, a programmer can use the DEF statement to define a new function to be used throughout a program. The function name is restricted to three alphabetic characters, the first two of which must be FN; therefore, 26 user function names (FNA through FNZ) are possible.

Format:

```
DEF FNa (sv) = e
```

a	Alphabetic character completing user function name
sv	Scalar variable
e	Expression to be evaluated by function

The variable sv is the formal parameter of the user function. It reserves a place for the variable to be used when the function is called in the program. The value of the argument in the function call is used in evaluating the expression. Every user function must have one parameter. The formal parameter however, need not appear in the expression on the righthand side of the DEF statement. In cases where it does appear in this position, it represents a dummy argument and has no effect on computation.

Each time the user function is needed in a program, the expression on the righthand side of the DEF statement is evaluated and the resultant value is used at that point in the program. This value may vary depending on the value of the function argument. The expression may be any mathematical formula or function or combination of both; however, its length is restricted to the remainder of the statement line.

A DEF statement may appear anywhere in a program, not necessarily before its corresponding function call. As the DEF statement may contain other function calls, functions can call each other, but they are not recursive; that is, they cannot call themselves. A function may be redefined within a program; its most recent definition will be used for each reference.

Example:

The following statement defines a function to calculate the area of a circle:

```
DEF FNC(D) = (3.14159*D↑2)/4
```

The function FNC may be used anywhere in the same program:

```
Y(5) = X3+FNC(SIN(X2)*H4)
```

SUBROUTINES

Subroutines represent complex sets of repetitious operations performed at various points throughout a program. Methods for subroutine use are described below:

GOSUB AND RETURN STATEMENTS

The GOSUB statement directs program control to the first line of a subroutine. The RETURN statement signals the end of the subroutine and returns control to the line following the GOSUB statement.

Formats:

```
GOSUB n

    n    First line number of the subroutine

RETURN
```

Example:

```
  5 GOSUB 100
 10 ---------
  . ---------
  . ---------
  . ---------
100 ---------
  . ---------
  . ---------
  . ---------
150 RETURN
```

In the above example, statement 5 diverts program control to the subroutine beginning at line number 100. Statement 150, at the end of the subroutine, returns control to the main program, resuming execution at statement 10.

Subroutines also may contain GOSUB statements, and thus call other subroutines. Recursion is allowed; subroutines may call themselves. GOSUB statements may nest subroutines to a maximum depth of 40.

BASIC INPUT/OUTPUT STATEMENTS 4

BASIC provides the means to specify internal data files and to access the associated data. Input statements enable the programmer to submit raw data at a terminal; output statements permit him to receive printed results at the terminal. File input/output statements are also available.

INTERNAL DATA FILE INPUT/OUTPUT

DATA STATEMENT

The DATA statement creates a file of data internal to a BASIC program. In conjunction with a READ statement, it performs essentially as a LET statement; a value is assigned to a variable.

Format:

```
    DATA c1,c2,c3,...cn

         ci    Signed or unsigned numeric constants
```

Any number of DATA statements may appear anywhere in a program; the BASIC compiler considers them contiguous and places them in one data block in sequential order.

Examples:

```
    DATA 2, 3, 4, 4, 5, 6,

    DATA -3.1,4.5,-5.678,2.31

    DATA -3.596E-2, 0, 0, 987
```

Like the DIM, REM and DEF statements, DATA statements are non-executable and have no effect on the results of a program if they are encountered in normal sequence.

READ STATEMENT

The READ statement uses values created by DATA statements to assign values to certain variables. The READ statement always is used in conjunction with a DATA statement.

Format:

```
    READ v1,v2,v3,...vn

         vi    Variables
```

The values of the constants in the DATA statements are assigned in sequence to the variables listed with the READ statement.

Examples:

```
    READ X

    READ X(I), B4, Y, N, O, P, E
```

The BASIC system maintains a pointer to the data block created by all DATA statements within a given program. Each time a READ statement lists a variable, a value from the data block is assigned to it; and the pointer moves forward through the list of values. The values of constants in DATA statements are assigned in sequence to the corresponding variables listed in the READ statement.

Example:

```
    DATA 2,3.4,1,0.12,0

    READ A,C,Y,X

    DATA 6, 7, 8, 9, 110

    READ B(K), J4
```

The above four lines of coding are equivalent to:

```
    LET A = 2

    LET C = 3.4

    LET Y = 1

    LET X = 0.12

    LET B(K) = 0

    LET J4 = 6
```

Extra constants not matched by variables in the READ statements are inconsequential. However, if the READ statement specifies more variables than there are constants, constituting an attempt to read past the end of a data block, the program will be terminated.

RESTORE STATEMENT

The RESTORE statement returns the data block pointer to the first value in the block.

Format:

```
    RESTORE
```

Whenever RESTORE is encountered, the next unsatisfied variable in a READ list is assigned as the first constant of the data block.

Example:

```
    DATA 1,2,3

    READ A,B,C

    READ D
```

This combination of READ and DATA statements alone would cause the program to terminate. If the second READ were preceded by a RESTORE statement, however, the previously unsatisfied variable, D, would be assigned the value 1; and program execution would continue.

NODATA STATEMENT

The NODATA statement should precede a READ statement. It determines whether the data block pointer has been moved beyond the end of the data block.

Format:

```
    NODATA n

    n    Statement line number
```

If data in the block is exhausted, program control will transfer to the specified line number, otherwise the next sequential statement is executed.

TERMINAL INPUT/OUTPUT

Printed output is received at the terminal by using the PRINT statement. Data is input at a terminal by using the INPUT statement. (INPUT statements are not submitted in batch mode.)

PRINT STATEMENT

The PRINT statement controls output of expressions and constants at a user terminal. All variables in the PRINT statement must have had values assigned before execution of the statement.

Format:

```
    PRINT x1d x2d x3d...xnd

         xi    Expressions or string constant

         d     Delimiter (comma or semicolon).
               A final delimiter is optional.
```

String Constants

A string constant is any number of valid 6000 series display code characters enclosed in quotation marks. Blanks may be included; quotation marks may not. The following are acceptable string constants:

```
    "THIS IS A BASIC STRING CONSTANT"

    "123.45678"
```

The following are incorrect:

```
    "TOO MANY QUOTATION "MARKS"

    "NO CLOSING MARKS
```

String constants serve as comments or messages and are used only in conjunction with PRINT statements. For example:

```
    LET X=Y=Z=2

    PRINT "ANSWER","X AND Z="Z, "X*Y*Z="X*Y*Z
```

This coding would produce the response:

```
    ANSWER         X AND Z=2    X*Y*Z=8
```

Number Formats

Expressions are evaluated and printed out in sequence at a terminal. Their values are printed in one of three standard number formats, depending on the size of the number. String constants are printed without quotation marks, otherwise, they are printed exactly as they appear in PRINT statements.

In the following standard number formats, n signifies a numeric character.

For an integral number of less than nine digits, this format is used:

Δnnnnnnnn

Leading zeros are suppressed and the number is printed left justified.

If the number is not an integer and can be expressed in six digits, the following format is used:

Δnnnnnn (where one n is a decimal point)

The decimal point is floating; leading zeros before and trailing zeros after the decimal point are suppressed. The number is left justified.

For all other numbers, the following format is used:

Δn.nnnnnΔEΔnnn

Leading zeros in the exponent are suppressed, and the exponent is left justified.

Print Line Zoning

The print line normally is divided into five zones of 15 positions each. The comma in the PRINT statement is a signal to move to the next print zone; or if the fifth zone has been filled, the move is to the first zone on the next line.

Example:

```
PRINT 4000, 303, 0051,432, 1.000, 5678.4
```

Output format:

```
4000           303            51             432            1
5678.4
```

A semicolon is used in the PRINT statement instead of a comma to reduce the size of print zones. The print zone is shortened to 6 spaces if the previously printed number is 1 to 3 characters; 9 spaces for numbers of 4 to 6 characters; 12 spaces for numbers of 7 to 9 characters; and the normal 15 spaces for other numbers. If a string constant terminates with a semicolon, the next zone is assumed to begin immediately thereafter. If no delimiter follows a string constant, a semicolon is assumed. Commas and semicolons may be intermixed in a PRINT statement.

A delimiter at the end of a PRINT statement causes the next PRINT statement to continue printing at the same line, or next available zone.

Example:

```
PRINT 300,400,500,600,
PRINT 700
```

Produces:

```
300            400            500            600            700
```

If a PRINT statement terminates without a delimiter, the next PRINT statement will begin its output on a new line.

Example:

```
PRINT 200,300,400,500
PRINT 600,700
```

Produces:

```
200            300            400            500
600            700
```

The statement, PRINT, by itself either will cause a whole line to be skipped, or it will nullify the effect of a delimiter on the previous PRINT statement line.

Examples:

```
    FOR I = 1 to 15
    PRINT I
    NEXT I
```

Produces (prints):

```
1
2
3
etc.
```

If the second statement were PRINT I,

Result:

```
1               2               3               4               5
6               7               8               9               10
11              12              13              14              15
```

If it were PRINT I;

Result:

```
1    2    3    4    5    6    7    8    9    10
11   12   13   14   15
```

The program:

```
    FOR X = 1 TO 100
    PRINT "X = ";X
    PRINT "X2 = ";X2
    NEXT X
```

would print:

```
    X = 1
    X2 = 1
    etc.
```

However, if the first PRINT statement were terminated by a comma it would print:

```
X = 1           X2 = 1          X = 2           X2 = 4          etc.
```

INPUT STATEMENT

The INPUT statement enables the programmer to assign values to variables from the terminal during program execution.

Format:

```
    INPUT v1,v2,v3,...vn

         vi    Variables
```

When a program in execution encounters this statement, a question mark is printed at the terminal. To satisfy the input request, the operator or user must enter data in exactly the same format at it appears in a DATA statement. The correct response to an input request is a string of optionally signed numeric constants, separated by commas, and terminated by a carriage return.

If the user supplies insufficient data, the question mark is iterated until the request is satisfied. If the user makes a typing error, or supplies too much data, a diagnostic message is issued. The user should retype the entire response to the input request.

Delimiters in an input response may be any BASIC character except a digit, period, + or - or the letter E. Redundant delimiters are ignored.

Generally a PRINT statement is used with an INPUT request, to inform the user of the response required.

Example:

```
    PRINT "WHAT IS THE VALUE OF X";

    INPUT X
```

would produce at the terminal:

```
    WHAT IS THE VALUE OF X?
```

The user should respond by typing in one number immediately after the question mark.

INPUT statements generally are used for entering only small amounts of data, as entering data from the terminal is time consuming.

FILE INPUT/OUTPUT

BASIC mass storage file input and output statements correspond to the READ and DATA statements for internal data files, and INPUT and OUTPUT statements for terminal input/output. The file I/O statements are READ FILE and WRITE FILE, INPUT FILE, and PRINT FILE. These statements must be used in pairs. An input statement must be used to read back data written by a print statement; and a read statement must be used for accessing data written by a write statement. Any attempt to read a file produced by a print statement will cause unpredictable results.

A maximum of three different file names may be used in a BASIC program. Files referenced by BASIC programs must be assigned to a mass storage device.

READ FILE AND WRITE FILE STATEMENTS

The READ FILE and WRITE FILE statements are analogous to the READ and DATA STATEMENTS for internal data files.

Formats:

```
    READ FILE (file name) v1,v2,v3,...vn

         (file name)    Any valid SCOPE file

         vi             Variables

    WRITE FILE (file name) x1,x2,x3,...xn

         (file name)    Any valid SCOPE file

         xi             Expressions
```

The WRITE FILE statement writes out the values of the expressions sequentially onto the file specified by the file name. The values are written in internal format and string constants are not allowed. As with DATA statements, consecutive WRITE FILE statements generate one contiguous block of data on the file.

The data can then be read by the READ FILE statement as if the file were an internal data file.

INPUT FILE AND PRINT FILE STATEMENTS

The INPUT FILE and PRINT FILE statements are identical to the INPUT and PRINT statements for terminal input/output.

Formats:

```
    INPUT FILE (file name) v1,v2,v3,...vn

         (file name)    Any valid SCOPE file

         vi             Variables

    PRINT FILE (file name) x1dx2dx3d...xn(d)

         (file name)    Any valid SCOPE file

         xi             Expressions or string constants

         d              Comma or semicolon; optional at end of list
```

These statements function exactly as for terminal input/output except that an error in the data terminates the program.

RESTORE FILE STATEMENT

The RESTORE FILE statement operates exactly as it does for internal data files.

Format:

```
    RESTORE FILE (file name)
```

The effect of this statement is to set the file named to its beginning of information. When the type of input/output on a file is to be mixed, the RESTORE FILE statement is required.

NODATA FILE STATEMENT

The NODATA FILE statement is identical to the NODATA statement for internal data files.

Format:

```
    NODATA FILE (file name) line number
```

If the file named is positioned at its end of information, this statement transfers program control to the statement that appears at the specified line number.

MATRIX OPERATIONS 5

Although it is possible to construct programs to perform matrix operations with the ordinary statements of the language, BASIC also provides a set of statements explicitly for matrix operations.

Matrix statements are restricted to one and two dimensional arrays. A one dimensional array is always treated as a column vector. To obtain a row vector, a two dimensional array must be specified with only one row.

Example:

```
DIM X(19), Y(0,19), Z(19,19)
```

This statement introduces matrix X with a column of 20 elements, matrix Y with one row of 20 elements, and matrix Z with 20 rows and 20 columns of elements.

If a variable used in a matrix statement has not appeared in a DIM statement, it is assumed to be two dimensional with element ranges from 0 to 10.

MATRIX ARITHMETIC

Matrix Statement	Arithmetic Operation
MAT M1 = M2+M3	addition
MAT M1 = M2-M3	subtraction
MAT M1 = M2*M3	multiplication
MAT M1 = (expression)*M2	scalar multiplication by value of the expression

In each of the above statements M may be any identifier; each identifier represents an array variable. The dimensions of the arrays must conform for each operation, and none of the operands of a matrix multiplication may be used as the result of that matrix multiplication.

MATRIX FUNCTIONS

Matrices can be inverted or transposed by using the INV (inversion) and TRN (transposition) functions.

Formats:

```
MAT M1 = INV(M2)

MAT M1 = TRN(M2)

     M    Array variable
```

The two matrices must conform for inversion or transposition, and inversion and transposition in place is not allowed.

Example:

```
    40 DIM A(5,2), B(8,4), C(5,6)

    50 MAT A = INV(B)

    80 MAT C = TRN(C)
```

Statement 50 is incorrect as the dimensions of matrices A and B do not conform; statement 80 is incorrect as the operand of a matrix statement may not be used as a result of the same statement.

The maximum size of an inversion matrix is 50 by 50.

The following three matrix statements may be used to generate a matrix of all zeros, all ones, or to assign ones to the elements along the principal diagonal and zeros elsewhere (identity matrix):

```
    MAT M = ZER[(x1[,x2])]

    MAT M = CON[(x1[,x2])]

    MAT M = IDN[(x1[,x2])]
```

M	Array variable
x	Expressions

Items enclosed in brackets are optional.

Parenthesized expressions can be used to redimension the matrix at execution time. Expressions are evaluated and used to reset the values of the upper limits of the number of elements in the matrix. However, a matrix must not be redimensioned to be larger than its initial value, whether established by default or a DIM statement or redimensioned to change its number of dimensions. A matrix used in an identity statement must be a square.

Examples:

```
    MAT Z = ZER(Y*4,6)

    MAT B = IDN(X↑2)

    MAT A = CON
```

MATRIX INPUT/OUTPUT

MAT READ STATEMENT

The MAT READ statement causes information on the internal data file to be read into a matrix.

Format:

```
    MAT READ A[(x1 [,x2])],B[(x3[,x4])],C...

         x    Expressions
```

Items in brackets are optional.

Expressions in parentheses indicate the matrix may be redimensioned at execution time. Evaluated expressions are used to reset the values of the upper limits as with the ZER, CON and IDN functions.

The MAT READ statement completely fills the matrices specified from the internal data file generated by the DATA statements. Matrices are read in row order.

```
    Example:  matrix A (3,3) is read as follows:

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

              A(1,1), A(1,2)etc.
```

MAT PRINT STATEMENT

The MAT PRINT statement is used to write matrices.

Format:

```
    MAT PRINT A d B d C d ...

         d    Comma or semicolon
```

Matrices are printed out in row order using the same rules as for the normal PRINT statement. The delimiter specifies the print zone for each element of the matrix. BASIC automatically generates a line of blanks between each row.

BASIC BATCH OPERATION 6

BASIC programs may be submitted to the SCOPE input queue for processing; under this mode of processing, all terminal I/O is inhibited.

BASIC CONTROL CARD

Programs submitted for batch processing must include a BASIC control card to call the BASIC compiler from the SCOPE library.

Format:

```
    BASIC(parameters)
```

Parameters on the BASIC control card indicate the following options:

L	Source listing, diagnostics and execution output will be on the file OUTPUT.
L=filename	Source listing, diagnostics and execution output will be on file indicated.
K	Diagnostics and execution output will be on the file OUTPUT; obtains if K is omitted.
K=filename	Diagnostics and execution output on file specified.
I	Source input will be from the file INPUT; obtains if I is not specified.
I=filename	Source input will be from the file specified.
B	Relocatable code will be on the LGO file; if B is not specified, no relocatable code is produced.
B=filename	Relocatable code will be on the file specified.
A	Assembly listing will be on the file OUTPUT; if omitted, no listing is produced.
A=filename	Assembly listing will be on the file specified.
N	Suppresses program execution; if omitted, program will be executed.

If both the K and L options are used on the same control card, the L option overrides the K option.

Examples:

BASIC.

This control card will direct a BASIC program to be compiled and executed and generates a listing of the diagnostics on the file OUTPUT. Output from the execution of the program will also go to OUTPUT.

BASIC(L,B,N)

This control card will generate relocatable code on LGO, with source and diagnostic listings on OUTPUT. The LGO control card will cause this program to be executed.

SAMPLE BATCH JOBS

The following three jobs illustrate how BASIC is used in batch mode. They show how some of the more common features of BASIC are used and are not necessarily models for programming or mathematical techniques in problem solving.

Example 1:

This job illustrates the use of the DEF and GOSUB statements for calculating the value of PI by the evaluation of a series.

```
10 PRINT ''CALCULATE A VALUE FOR PI''

20 PRINT

25 Z = 100000

26 PRINT ''NUMBER OF ITERATIONS'';Z

27 PRINT

30 A = 1

40 B = 3

50 DEF FNA (D) = (1/D)

60 DEF FNB (D) = (D - FNA(B))

70 DEF FNC (D) = (D + FNA (B))

80 FOR I = 1 TO Z

90 A = FNB (A)

100 GOSUB 150

110 A = FNC (A)

120 GOSUB 150

130 NEXT I
```

```
140 GO TO 170
150 B=B+2
160 RETURN
170 PRINT ''PI='';4*A
200 END
```

The output from this job is:

```
CALCULATE A VALUE FOR PI
NUMBER OF ITERATIONS 100000
PI = 3.1416
```

Example 2:

This job calculates a table of factorials using a looping technique.

```
10 A = 1
50 Z = 20
60 FOR I =1 TO Z
70 A=A*I
75 PRINT ''FACTORIAL '',I,A
80 NEXT I
100 END
```

The output from this job is:

```
FACTORIAL 1        1
FACTORIAL 2        2
FACTORIAL 3        6
FACTORIAL 4        24
FACTORIAL 5        120
FACTORIAL 6        720
FACTORIAL 7        5040
FACTORIAL 8        40320
FACTORIAL 9        362880
FACTORIAL 10       3628800
FACTORIAL 11       39916800
FACTORIAL 12        479001600
FACTORIAL 13        6227020800
FACTORIAL 14        8.71783 E 10
FACTORIAL 15        1.30767 E 12
FACTORIAL 16        2.09228 E 13
FACTORIAL 17        3.55687 E 14
FACTORIAL 18        6.40237 E 15
FACTORIAL 19        1.21645 E 17
FACTORIAL 20        2.43290 E 18
```

Example 3:

This job calculates a value for PI using some of BASIC's standard functions.

```
10 PRINT "CALCULATE A VALUE FOR PI"
20 PRINT
30 A = -1/2
50 R = 100
60 PRINT "RADIUS OF DARTBOARD "; R
70 Z = 100000
75 PRINT ''NUMBER OF DARTS THROWN '';Z
76 PRINT
80 N=(R*2)+1
90 M=(N*N)-1
100 H=R*R
110 DEF FNA (D) = ABS (D-R)
120 FOR I=1 TO Z
130 A=RND (A)
140 B=INT (A*M)
150 Y=INT (B/N)
160 X=INT (B-(Y*N))
170 Y=FNA (Y)
```

```
180 X=FNA (X)
190 J= X ↑ 2+Y**2
200 IF J > H THEN 220
210 K=K+1
220 NEXT I
230 P=4*K/Z
240 PRINT ''VALUE OF PI = '';P
250 END
```

The output from this job is:

```
CALCULATE A VALUE FOR PI
RADIUS OF DARTBOARD 100
NUMBER OF DARTS THROWN 100000
VALUE OF PI = 3.10988
```

BASIC TERMINAL OPERATION 7

All BASIC programs submitted for processing on a CONTROL DATA 6000 Series computer from a terminal may be executed under the control of the INTERCOM I system. This chapter describes a subset of the INTERCOM I commands—those used most frequently by the BASIC programmer. The capabilities of the INTERCOM I system are detailed in the INTERCOM I Reference Manual.

INTERCOM I TERMINALS

INTERCOM I provides interface between users at terminals and a central site 6000 series computer. The terminals may be Teletype Model 33 or 35 or Model 217-2 CRT (cathode ray tube).

TELETYPE

The programmer types lines of information to the INTERCOM I system on the Teletype. INTERCOM I responds on the Teletype at the terminal. The Teletype keyboard is similar to an ordinary electric typewriter with a few additional keys as described below:

RETURN Key

The user must terminate every line of information by pressing the RETURN key. This signals to INTERCOM I that the message is complete and returns the Teletype carriage to the beginning of the line just typed. INTERCOM I responds by sending a line feed to the terminal. This moves the carriage to the next line at which point the user may enter input.

Backspace Arrow (←)

Entry errors can be corrected by using the backspace arrow and typing over the erroneous character.

Example:

BAX ← SJK ← ← IC will be interpreted by INTERCOM I as BASIC

CTRL and X Keys

Entire lines may be deleted by simultaneously pressing the X and CTRL keys. The entire line will be ignored by INTERCOM I, and the Teletype carriage will be positioned at the beginning of a new line.

Key

Once the RETURN key is pressed, the command is accepted by INTERCOM I. However, execution of the command can be aborted by typing a # (pound) on the Teletype. INTERCOM I will interrupt processing of the current command, re-enter command mode, and accept a new command.

CRT TERMINAL

The operation of the 217-2 CRT terminal is similar to the Teletype terminal; except that as messages are typed in, they are displayed on the CRT. Successive messages appear on successive lines of the CRT until the bottom of the screen is reached. The display then overwrites information at the top of the screen. Three markers indicate where a message is positioned on the screen. The following special symbols and keys are used to operate the 217-2 CRT:

Beginning-of-Line Symbol (▬)

The solid bar indicates the beginning of the line where the current input message begins.

SEND Key and End-of-Line Symbol (Δ)

The SEND key is equivalent to the RETURN key on the Teletype. It indicates the end of a message and displays the end-of-line symbol (Δ) on the screen. The information between the ▬ and the Δ is transmitted to INTERCOM I.

Underline Marker

An underline marker indicates to the user where the next input character will appear. Each character position of the current message line has an underline marker. As characters are entered and displayed this marker disappears.

BKSP Key

The BKSP key backspaces the entry marker one character position so that the user can write over information.

CLEAR Key

The CLEAR key erases the contents of the entire screen and positions the entry marker and beginning-of-line indicator at the top of the screen.

% Key

The % character causes an interrupt of the currently executing command.

ENTERING INTERCOM I

To access a 6000 series computer from a terminal the user must link up with the INTERCOM I system. The method of establishing the connection between the terminal and the central site computer will vary depending on the type of terminal equipment and the connection provided by the telephone company. Once the connection is established, INTERCOM I can be accessed by the following:

1. User indicates that the terminal requires INTERCOM I by typing:

   LOGIN.

2. INTERCOM I responds with the message:

   TYPE VALID USER NAME-

3. User responds with a valid user name of no more than 10 alphanumeric characters.

4. INTERCOM I responds:

   TYPE PASSWORD-

5. User responds with a password of up to 10 alphanumeric characters.

6. INTERCOM I checks that the user name and password are valid; and if so, displays the message:

   COMMAND-

7. The user/terminal is in command mode and INTERCOM I is waiting for input.

COMMAND MODE

Under command mode, each message sent to INTERCOM I is interpreted as a request to load and execute a program. In general, all SCOPE control cards are also INTERCOM I commands.

After a command is entered, control is given to the program which processes that command; and it remains with that program until processing terminates or the user voluntarily leaves the program. Command mode is re-established, and INTERCOM I is ready to accept another command.

INTERCOM I performs two kinds of tasks for the user. Under command mode INTERCOM I loads a library (utility) program that processes the given command. Subsequent commands are transferred to the utility until the latter is terminated. The system then reverts to command mode.

As almost any SCOPE control card is also an INTERCOM I command, such messages as REWIND (filename) may be sent from the terminal and executed.

Primarily, the BASIC user will be concerned with the operation of only one INTERCOM I command, SETUP.

SETUP

SETUP is an INTERCOM I utility routine which enables the user to create and modify files of information and submit them for execution. Primarily, SETUP is a text editor designed for manipulation of program files with the added capability to direct the execution of these files. The user can access SETUP by entering the command:

```
    SETUP.
```

The SETUP routine is then loaded and the user is provided with a buffer within which he may manipulate his files. SETUP will indicate readiness with the message:

```
    ON AT *hh.mm.ss

    SYSTEM-FORTRAN

    NEW OR OLD FILE
```

hh.mm.ss is the time of day in hours, minutes and seconds when SETUP is loaded.

ENTERING BASIC

After SETUP is loaded, the BASIC user should type:

```
    SYSTEM
```

SETUP will reply:

```
    NEW SYSTEM-
```

BASIC can be entered by typing:

```
    BASIC
```

An alternative form of entering BASIC is:

```
    SYSTEM/BASIC
```

The slash must be used to separate SETUP directives. The user can anticipate questions from SETUP and give several answers, separated by slashes, thereby saving time at the terminal.

Once BASIC mode is entered, the system will type:

```
    NEW OR OLD FILE-
```

If the user wants to continue work on a prevously saved file he should respond:

```
    OLD
```

The terminal will answer:

```
    OLD FILE NAME
```

The user should type a valid SCOPE file name; or alternatively, a new file may be created by typing:

```
    NEW / filename
```

SETUP replies:

```
    READY
```

From this point, the file is created or edited.

BASIC OPERATIONS

After the terminal is in BASIC mode, BASIC statements may be entered into the text buffer as described in the preceding sections of this manual. Each statement must have a line number indicating its order in the sequence statements. A line may be edited or replaced by entering the same line number with a new or altered statement. A line may be deleted by typing the line number only. A new line may be inserted by typing a statement with a line numbered between the line numbers where the insertion is to be made.

SETUP DIRECTIVES

While the terminal is in SETUP mode and before program execution, the user can manipulate the contents of the text buffer with the following directives:

LIST

The contents of the text buffer may be examined at any time by typing the directive LIST. Listing of the text buffer may be terminated before completion by typing S. A portion of the text buffer may be listed by typing LIST (line number) for only one line, or LIST (line number1,line number2) to indicate the range of the list.

DELETE

Portions of the text buffer may be deleted with the DELETE directive:

```
    DELETE (line number) or DELETE (line number1,line number2)
```

SAVE

This directive saves contents of the text buffer for later use. If SAVE is not used, the file will be lost when the user leaves the SETUP routine.

Format:

```
    SAVE/filename
```

If a file already exists with the same name, the new file replaces the old.

UNSAVE

A file may be deleted by typing:

```
    UNSAVE/filename
```

RENAME

The name of the file in the text buffer may be changed by typing:

```
    RENAME/filename
```

When the BASIC program is ready for execution, it may be submitted to the BASIC compiler by typing RUN. INTERCOM I will respond with:

```
    PROGRAM TRANSFERRED TO COMPILER
```

Errors in the program will be listed at the terminal along with the message:

```
    BASIC COMPILATION ERRORS
```

The user then can modify the program in the text buffer by correcting the errors. When a program compiles correctly, it is executed immediately; and the results are returned to the terminal. If an execution error occurs, a diagnostic is printed outfollowed by:

```
    BASIC EXECUTION ERROR
```

The user can modify the program in the text buffer and again resubmit it for BASIC compilation and execution until the program is debugged and the user is satisfied with the results. Since, at this point, the BASIC program is in the text buffer; it may be saved and recalled at any time to be run again.

SCRATCH

The SCRATCH directive clears the contents of the text buffer which enables the user to work on a new program. Contents of saved files are not altered.

BYE

BYE directs exit from the SETUP utility. SETUP will reply:

```
    OFF AT *hh.mm.ss
```

The command mode is re-established.

TRANS

A BASIC program read into INTERCOM I from the central site, or created in any way without using SETUP, can be transferred to the text buffer by the directives:

```
    OLD/filename/TRANS
```

This directive causes the BASIC program to be transferred to SETUP so that it may be edited by using BASIC line numbers.

INTERCOM I COMMANDS

INTERCOM I commands of particular relevance to the BASIC user follow:

FILES

The user can obtain a list of files available to him by typing:

```
    FILES.
```

Only private files and attached permanent files are listed; SCOPE common or permanent files accessible through the permission feature are not listed.

ETL and EFL

The system default time limit (in octal seconds) and the field length (in octal words) for a job can be altered by typing:

```
    ETL, time limit, and/or EFL, field length
```

BASIC

The BASIC control card enables the user to run a BASIC program without using SETUP. If filename1 contains a BASIC source program, the latter can be compiled and executed by entering the normal BASIC control card:

```
    BASIC (I=filename1)
```

The results of a program entered in this manner will not be returned to the terminal automatically but will appear on the file named OUTPUT. This file can be examined by using the LIST directive in SETUP.

CONNECT

This command connects any file to the terminal.

To direct any output from filename2 to the terminal:

```
    CONNECT (filename2)
```

To obtain printed output at the terminal:

```
    BASIC (I=filename1,L=filename2)
```

To obtain error listing and execution output only:

```
    BASIC (I=filename1,K=filename2)
```

A user may create his own object program with the command:

```
    BASIC (I=filename1, K=filename1, B=filename3)
```

The object program can be executed at any time with the commands:

```
    CONNECT (filename1, filename2,...)
```

DISCONT

A file may be disconnected from a terminal with this statement:

```
    DISCONT (filename)
```

Any subsequent use of the file named in this statement will refer to mass storage.

LEAVING INTERCOM I

The user can exit from the INTERCOM I system by entering the command:

```
    LOGOUT.
```

The system will respond:

```
    YOU NO LONGER OWN ANY FILES
```

Finally the system will print the date, time of day, and time consumed during the session with the system.

SAMPLE TERMINAL SESSION

The following sample BASIC job run under the INTERCOM I system illustrates some of the INTERCOM I commands section and the creation of a BASIC program using the SETUP utility.

```
LOGIN.
TYPE VALID USER NAME - BROWN
TYPE PASSWORD - CHARLIE
COMMAND - SETUP.
ON AT *10.30.00
SYSTEM FORTRAN
NEW OR OLD FILE - SYSTEM/BASIC/NEW/BASPROG
READY
10  PRINT "TYPE A NUMBER";
20  INPUT X
30  LET X=1
40  FOR I=1 to N←X
50  F=F*I
30  F=1
60  PRINT "FACTORIAL"X;"IS"F
70  GO TO 10
80  END
RUN
PROGRAM TRANSFERRED TO COMPILER
FOR WITHOUT NEXT AT 40
BASIC COMPILATION ERRORS
READY
55  NEXT I
25  IF X=0 THEN 80
RUN
PROGRAM TRANSFERRED TO COMPILER
TYPE A NUMBER ? 3
FACTORIAL 3 is 6
TYPE A NUMBER ? 0
READY
LIST
10  PRINT "TYPE A NUMBER";
20  INPUT X
25  IF X=0 THEN 80
30  F=1
40  FOR I=1 TO X
50  F=F*I
55  NEXT I
60  PRINT "FACTORIAL"X;"IS"F
70  GO TO 10
80  END
READY
SAVE
READY
BYE
OFF AT *10.50.33
COMMAND - CONNECT (BASOUT)
COMMAND - BASIC (I=BASPROG,K=BASOUT)
TYPE A NUMBER ? 6
FACTORIAL 6 IS 720
TYPE A NUMBER ? 0
COMMAND - LOGOUT.
YOU NO LONGER OWN ANY FILES
CP TIME - 3.103
PP TIME - 6.002
10/27/69 OFF AT * 11.01.16
```

CHARACTER SET A

BASIC Char.	Display Code	Card Code	Printer	TTY	217-2
	00				
A	01	12-1	A	A	A
B	02	12-2	B	B	B
C	03	12-3	C	C	C
D	04	12-4	D	D	D
E	05	12-5	E	E	E
F	06	12-6	F	F	F
G	07	12-7	G	G	G
H	10	12-8	H	H	H
I	11	12-9	I	I	I
J	12	11-1	J	J	J
K	13	11-2	K	K	K
L	14	11-3	L	L	L
M	15	11-4	M	M	M
N	16	11-5	N	N	N
O	17	11-6	O	O	O
P	20	11-7	P	P	P
Q	21	11-8	Q	Q	Q
R	22	11-9	R	R	R
S	23	0-2	S	S	S
T	24	0-3	T	T	T
U	25	0-4	U	U	U
V	26	0-5	V	V	V
W	27	0-6	W	W	W
X	30	0-7	X	X	X
Y	31	0-8	Y	Y	Y
Z	32	0-9	Z	Z	Z
0	33	0	0	0	0
1	34	1	1	1	1
2	35	2	2	2	2
3	36	3	3	3	3
4	37	4	4	4	4
5	40	5	5	5	5
6	41	6	6	6	6
7	42	7	7	7	7
8	43	8	8	8	8
9	44	9	9	9	9
+	45	12	+	+	+
-	46	11	-	-	-
*	47	11-8-4	*	*	*
/	50	0-1	/	/	/
(	51	0-8-4	(	(	(
)	52	12-8-4	)	)	)
unused	53	11-8-3	$	$	$
=	54	8-3	=	=	=
△	55	space	△	△	△
, ↓	56	0-8-3 or 6-8	, ↓	, ↓	, ↓
.	57	12-8-3	.	.	.

"	60	0-8-6	≡	"	≡
unused	61	8-7	[	[	[
unused	62	0-8-2	]	]	]
unused	63	8-2	:	:	:
unused	64	8-4	≠	'	≠
unused	65	0-8-5	→	reserved	Δ
unused	66	11-0 or 11-8-2	∨	\	∨
unused	67	0-8-7	∧	&	∧
	70	11-8-5	↑	↑	↑
unused	71	11-8-6	↓	reserved	↓
	72	12-0 or 12-8-2	<	<	<
	73	11-8-7	>	>	>
unused	74	8-5	≤	reserved	≤
unused	75	12-8-5	≥	@	≥
?	76	12-8-6	¬	?	%
;	77	12-8-7	;	;	;

Display code characters not used by the BASIC system may be used within string constants or remarks.

Display code characters 65, 71 and 74 are used by the INTERCOM I system for special functions and cannot be used at a Teletype terminal.

Display code character 76 is interpreted as an interrupt character from a 217-2 and cannot be input from a CRT terminal.

DIAGNOSTICS B

All diagnostics produced by the BASIC system at compile and execution time are printed in the following format:

```
    message AT line-number
```

The line number indicates the statement in error.

COMPILE TIME DIAGNOSTICS

The following messages may be produced during program compilation. All compile time diagnostics inhibit execution of the program and BASIC writes a dayfile message: BASIC COMPILATION ERRORS.

Message	Error
DUPLICATE LINE NO	Same line number used twice
END NOT LAST	END statement incorrectly placed
FOR NEST TOO DEEP	More than 10 nested for statements
FOR WITHOUT NEXT	FOR statement has no balancing NEXT statement
ILLEGAL BOUND	DIM statement declared variable >131071
ILLEGAL CHARACTER	Unrecognizable character
ILLEGAL FN NAME	User function name not of the form FNx, x is any alphabetic character
ILLEGAL FILE NAME	Name is not allowed as a SCOPE file name
ILLEGAL LINE NO	Line number greater than 99999
ILLEGAL LINE REF	Incorrectly written line number, or line number referenced greater than 99999
ILLEGAL NUMBER	Numeric constant incorrectly written
ILLEGAL STATEMENT	Statement does not begin with a recognizable word or is written incorrectly
ILLEGAL STRING	String constant incorrectly written
ILLEGAL WORD	Unrecognizable word

LINES OUT OF ORDER	Line numbers not in ascending order
MISSING LINE NO	Statement written without a line number
NEXT WITHOUT FOR	NEXT statement has no balancing FOR statement
NO END STATEMENT	Program does not have END statement
OUT OF SPACE*	Program too large to continue checking source statements; compilation stops
PROGRAM TOO LARGE*	Compiled program too large to execute in the field length given to the BASIC compiler
PROGRAM TOO LONG*	Program too long to compile, but the compiler continues to check the source statements
READ WITHOUT DATA	Program containing a READ statement has no DATA statements
RECURSIVE FN	Recursive DEF statement
TOO MANY FILES	More than three file names
UNDEFINED FN REF	Undefined user function
UNDEFINED LINE REF	Line number referenced does not exist

*If any of these errors occur, the program should be recompiled with a larger field length. No maximum size is defined for a BASIC program, as the limits depend entirely on the field length given to the BASIC system. If, at the beginning of a compilation, BASIC determines that the field length is too small to attempt compilation, the dayfile message BASIC FIELD LENGTH TOO SHORT is written and the compilation is terminated.

EXECUTION TIME DIAGNOSTICS

The following diagnostics may occur during execution of a program. All terminate execution, and BASIC writes a dayfile message BASIC EXECUTION ERROR.

Message	Error
ARGUMENT IS POLE IN TAN	
ARGUMENT NEGATIVE IN LOG	
ARGUMENT NEGATIVE IN SQR	
ARGUMENT TOO LARGE IN COS	
ARGUMENT TOO LARGE IN EXP	
ARGUMENT TOO LARGE IN SIN	
ARGUMENT TOO LARGE IN TAN	
ARGUMENT ZERO IN LOG	
BASIC SYSTEM ERROR	Malfunction of BASIC system. Please report the problem
DIVISION BY ZERO	
END OF DATA	READ statement executed when all the DATA statements are exhausted
END OF DATA ON FILE	READ FILE or INPUT FILE statement executed after file data exhausted
GOSUB NEST TOO DEEP	More than 40 GOSUB's nested
ILLEGAL DEVICE FOR FILE	File assigned to a non-mass storage device
ILLEGAL INPUT ON FILE	Type of input/output statement changed without repositioning file to beginning of information, or an input request follows an output request on the same file
ILLEGAL NUMBER ON FILE	INPUT FILE statement attempted to read a file in incorrect format
ILLEGAL OUTPUT ON FILE	Type of input/output statement changed without repositioning the file to beginning of information
INFINITE VALUE	
MATRIX DIMENSION ERROR	Dimension inconsistency in one of the MAT statements

NEARLY SINGULAR MATRIX	Attempt to invert a singular or nearly singular matrix
NEGATIVE NUMBER TO A POWER	
POWER TOO LARGE	
RETURN BEFORE GOSUB	RETURN statement has been executed and no GOSUB is in effect
SUBSCRIPT ERROR	Attempt to reference an element outside bounds of an array
TIME EXCEEDED	Program exceeded its time limit
TRANSMISSION ERROR ON FILE	BASIC system unable to complete an input/output request
UNDEFINED VALUE	

The following messages may occur after the data typed in response to an INPUT request from the terminal has been checked. The BASIC system will print a question mark and the user should retype the data.

ILLEGAL NUMBER, RETYPE INPUT

TOO MUCH DATA, RETYPE INPUT

TRANSMISSION ERROR, RETYPE INPUT

INDEX OF BASIC STATEMENTS C

The following alphabetical list of BASIC statements gives the formats, functions, and pages on which they appear. Throughout this appendix the following abbreviations are used:

c	numeric constant	m	matrix
e	expression	r	relational operator
d	delimiter	sv	scalar variable
fn	file name	ufn	user function name
i	identifier	v	variable
ln	line number		

Items enclosed in brackets are optional.

Statement Format	Function	Page No.
DATA c1,c2,c3,...cn	Creates a file of data internal to the BASIC program	4-1
DEF ufn(i) = e	Defines a new function to be used within a BASIC program	3-3
DIM v1,v2,v3,...vn	Declares the dimensions of an array variable	2-2
END	Terminates a program	2-7
FOR sv = e1 to e2 [STEP e3]	Begins a program loop; this statement must be saved with a NEXT statement	2-4
GO TO ln	Interrupts the normal sequence of program execution and transfers program control to indicated line number	2-4
GOSUB ln	Transfers program control to a subroutine beginning at line number indicated	3-3
IF e1r e2 THEN ln	Transfers program control to indicated line number if certain conditions are met	2-4
INPUT v1,v2,v3,...vn	Enters data from a terminal	4-7
INPUT FILE (fn) v1,v2,v3,...vn	Enters a file from a terminal	4-8
[LET] v1 = v2 = v3...vn = e	Assigns a value to a variable during program execution	2-3
MAT m1 = m2 + m3	Matrix addition	5-1
MAT m1 = m2 - m3	Matrix subtraction	5-1
MAT m1 = m2 * m3	Multiplies one matrix by another matrix	5-1
MAT m1 = (e) * m2	Multiplies a matrix by an expression	5-1
MAT m = CON [(e1[,e2])]	Creates a matrix of all ones	5-2
MAT m = IDN(e)	Creates an identity matrix	5-2
MAT m1 = INV(m2)	Inverts a matrix	5-1
MAT m1 = TRN(m2)	Transposes a matrix	5-1
MAT m = ZER(e)	Creates a matrix of zeros	5-2

Statement Format	Function	Page No.
MAT READ m1 [(e1[,e2])], m2 [(e3[,e4])],m3...	Reads matrices from the internal data file	5-2
MAT PRINT m1d m2d m3...[d]	Prints matrices on the terminal	5-3
NEXT sv	Terminates a program loop or increments the value tested by the loop	2-4
NODATA ln	Tests data pointer for increment beyond end of data block	4-3
NODATA FILE (fn)ln	Transfers program control to specified line number if named file is positioned at end of information	4-9
PRINT e1d e2d e3d...en[d]	Prints data at terminal	4-3
PRINT FILE (fn) e1d e2d e3[d]...	Prints data on specified file	4-8
READ v1, v2, v3,...vn	Accesses data created by DATA statements	4-1
READ FILE (fn) v1,v2,v3,...vn	Accesses data created by WRITE FILE statements	4-8
REM any string of valid 6000 series display code characters	Inserts explanatory remarks into a program	2-7
RESTORE	Reinitializes data pointer to the first word of the data block	4-2
RESTORE FILE (fn)	Sets named file to beginning of information	4-9
RETURN	Resumes execution at statement following companion GOSUB statement	3-3
STOP	Terminates program execution at places other than the END statement	2-7
WRITE FILE (fn) e1,e2,e3,...en	Writes expression values onto named file	4-8

INDEX OF INTERCOM I COMMANDS AND DIRECTIVES D

Alphabetical list of SCOPE 3 and INTERCOM I commands as well as SETUP directives used in processing BASIC programs:

Command/Directive	Function	Page No.
Backarrow key	Deletes last character of current input line	7-1
BKSP key	Deletes last character of current input line	7-2
BASIC	Call for BASIC compiler	7-4, 7-7
BYE	Exit from SETUP routine	7-7
CLEAR key	Erases entire contents of display screen	7-2
CONNECT (fn)	Connects file named to user terminal	7-8
DELETE(1n1 [,1n2])	Erases portions of the text buffer	7-5
DISCONT (fn)	Disconnects file named from the terminal	7-8
EFL,field length	Changes system default field length	7-7
ETL,time limit	Changes the system default time limit	7-7
FILES	Obtains a list of files available to the user	7-7
LIST[1n1 [,1n2])]	Displays current contents of text buffer	7-5
LOGIN.	Calls for INTERCOM I system	7-3
LOGOUT.	Exits INTERCOM I system	7-8
NEW	Creates a new file	7-5
Percent (%) key	Inhibits the execution of the current command	7-2
Pound (#) key	Inhibits the execution of the current command	7-2
OLD	Retrieves a previously saved file	7-5
RENAME	Changes name of file in text buffer	7-6

RETURN key	Terminates a line of input and causes the system to act upon input received	7-1
RUN	Causes current program to be compiled and executed	7-6
SAVE	Saves contents of text buffer	7-6
SCRATCH	Clears text buffer	7-6
SEND key	Terminates a line of input causing system to interpret and act upon input received	7-2
SETUP	Calls for SETUP utility routine	7-4
SYSTEM	Indicates change of language	7-4
TRANS	Transfers a program created without SETUP into the text buffer	7-7
UNSAVE	Deletes file currently in text buffer	7-6
X and CTRL keys	Deletes the entire current input line when used simultaneously	7-1

System requests for information:

System Request	User Response	Page No.
Commands:	INTERCOM I Command:	
NEW OR OLD FILE-	OLD to continue work on a previously saved file; NEW to create a new file	7-4
NEW FILE NAME-	Name of file to be created	7-5
OLD FILE NAME-	Name of previously saved file	7-5
NEW SYSTEM-	Name of the programming language	7-4
TYPE VALID USER NAME-	Valid user name of no more than 10 alphanumeric characters	7-3
TYPE PASSWORD-	User's password of up to 10 alpha-numeric characters	7-3

System messages other than diagnostics, which will appear at the terminal;

Message	Significance	Page No.
ON AT *hh.mm.ss SYSTEM -system name	SETUP is ready for use. Time of day in hours, minutes, seconds. System name is that of the system currently in control.	7-4
READY	SETUP is ready to handle the creation or manipulation of a file in the text buffer.	7-5
PROGRAM TRANSFERRED TO COMPILER	Program in the text buffer is transferred to the appropriate compiler	7-6
OFF at *hh.mm.ss	User is returned to command mode.	7-7

INDEX

COMMENT SHEET

CONTROL DATA
CORPORATION

TITLE: 6000 Series Computer Systems
BASIC Language Reference Manual

February 1970

PUBLICATION NO. 60305000 REVISION A

Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

CUT ON THIS LINE

FROM	NAME: ______	POSITION: ______	
	BUSINESS ADDRESS: ______		
	______		

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ON THIS LINE

FOLD FOLD

STAPLE STAPLE

6000 SERIES BASIC LANGUAGE REFERENCE MANUAL

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN, 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60305000

Litho in U.S.A.