



**DATA GENERAL  
CORPORATION**

Southboro,  
Massachusetts 01772  
(617) 485-9100

PROGRAM

Single User BASIC

TAPES

Absolute Binary: 091-000018 - 4

ABSTRACT

Single User BASIC is a dedicated interpretive system that allows conversational entry and execution of programs written in the BASIC language as developed by Dartmouth College. It includes use of all elementary and advanced BASIC statements as defined in Basic Programming by J. G. Kemeny and T. E. Kurtz (copyrighted 1967 by John Wiley & Sons, Inc.), but does not include string or matrix manipulation functions. The Data General implementation also permits execution of certain statements in a "desk calculator" or "keyboard" mode which is most useful in testing or debugging programs as well as in performing simple computations or evaluating complex formulas without the necessity of writing a program. The system will operate in a 4K or larger memory configuration and requires a teletypewriter for input/output.



Introduction

Operating Procedures

Loading, Restarting, and Clearing User's Space

Entering and Editing a Program

Keyboard Commands

Ground Rules on Numbers, Variables, Arrays and Expressions

Summary of BASIC Statements

Appendices

A. Punching and Loading Tapes of User's Programs

B. Programming Hints

C. Short Summary of BASIC Statements

D. Error Messages

## INTRODUCTION

Data General Corporation's single-user BASIC allows programming in the standard BASIC language using a Nova or Supernova central processor with 4K or more memory words and a teletypewriter. It includes use of all the elementary and advanced BASIC statements as defined in BASIC Programming by John G. Kemeny and Thomas E. Kurtz (c. 1967 by John Wiley & Sons, Inc.), but does not include matrix manipulation functions. The BASIC language developed by Dartmouth College allows conversational program entry, editing, execution, and input/output operations. The Data General Corporation implementation also permits execution of certain statements in a 'desk calculator' or 'keyboard' mode which is most useful in testing or debugging programs as well as in performing simple computations or evaluating complex formulas without the necessity of writing a program. The purpose of this write-up is to inform the user of operating procedures, including error messages, and to provide a brief summary of the BASIC statements available. Beginning programmers, or programmers not familiar with the use of BASIC are referred to the above mentioned book by Kemeny and Kurtz.

## OPERATING PROCEDURES

### Loading, Restarting and Clearing User's Space

The binary tape of single user BASIC is loaded like any other absolute binary tape by the binary loader. Once loaded, the BASIC system takes control and the first task it performs is to determine the size of memory it has to operate in. When the system finds itself in a 4K environment it types out the question: "DO YOU WISH TO OVERWRITE THE FUNCTIONS SIN, COS, ATN AND TAN (TYPE Y OR N)? " If the user responds with a Y the core required for these functions is made available for storage of user's program and data, thereby expanding the total storage available for this purpose by sixty percent. The number of spaces represented by a comma in a PRINT statement may now be selected to be either 14 or 15. Once selected, the system will treat the full page as either 70 or 75. Thus, the function TAB will reduce its argument modulo 70 or 75, etc. (See "PRINT and TAB"). The following message will be typed:

"DO YOU WISH COMMA'S TO BE 14 SPACES (TYPE Y), OR 15 (TYPE N)? "

The appropriate key, Y or N may then be typed. To change the number of spaces represented by comma, the system must be reloaded. It then initializes itself and in that process destroys the binary loader, but it leaves intact the bootstrap loader. If it finds memory is larger than 4K words, it will preserve both the binary and bootstrap loaders and will use the additional core to store the user's data and program. Hence, larger memory configurations provide the user with the capability of handling larger programs with larger bodies of data. After it has sized memory and initialized itself, the BASIC system performs a carriage-return/line-feed on the teletypewriter, and waits for the user to respond.

If at some point in working with the system the user wishes to restart the program, due to a power failure for example, he may do so by setting the Nova/Supernova panel data switches to the restart address (000002), and pressing the RESET and START operating switches. This action will place the system in a state of suspended animation until the user strikes the ESC key on the teletypewriter. All other teletypewriter keys will give no response until the ESC key is hit. When struck it will cause the system to do a carriage-return on the teletypewriter and type the message "\*READY".

Restarting BASIC does *not* destroy the program or data that the user had entered previously. To accomplish this the user must issue the keyboard command "NEW", which causes the user's program area to be cleared in preparation for a new program. The keyboard command NEW, always typed without a preceding line number, can be given at any point, and is not necessarily connected with restarting the system. Thus, if during the entry and editing of a program, or following the execution of a program, a NEW command was given, it would also clear the user's program area.

### Entering and Editing a Program

The BASIC system is ready to accept program statements from the teletypewriter at any point except during execution of a program. Even during execution, it is possible to halt a program by hitting the ESC key, and then to enter additional statements or alterations to statements and to re-enter or re-start the interrupted program. Each line of a program is entered by typing a statement number followed by the statement. When statements are not preceded by a statement number, the BASIC system will attempt to execute the statement in keyboard mode if that is possible, and if it is not, the system will return an error message (usually error 12). Statement or line numbers must be integers, and they are limited to a maximum of four digits. Lines are terminated by typing a carriage-return, and spaces are permitted at any point on a line, including before the line number. Spaces have no significance in BASIC, except in messages enclosed in quotation marks which are to be printed out.

To correct errors made while typing a statement, two keyboard commands are available: "control A" and "control X". The user can delete (rub out) the preceding character in a line by pressing down the CTRL and the A keys ("control A") which will print on a teletypewriter as a backwards arrow "←". He can then type in the correct character. Pressing "control A" a number of times will erase from the current line the characters in that number of preceding spaces. To delete all of the present line, the user should press the CTRL key and the X key ("control X") which will print as a backwards slash "↖", and automatically cause a carriage-return and a line-feed on a teletypewriter.

If, after having entered a program, the user wishes a listing of his program, he can obtain one simply by typing the keyboard command "LIST" followed by a carriage-return. The BASIC system will then produce a listing of the program as it currently stands in line number sequence.

The command LIST has two forms:

LIST	List entire program.
nnnnLIST	List entire program starting at statement number nnnn.

If the statement number cannot be found, the message

ERR 13

will be typed.

In either form, the listing may be stopped at any time with the use of the ESC key. If the user wishes a paper tape copy of his program he may obtain one by turning on the teletypewriter punch unit just before he types the carriage-return at the end of his LIST command. The tape produced may then be reloaded at a later time by mounting it on the teletypewriter reader and moving the control switch to START (See Appendix A). After reviewing his listed program, the user may desire to alter some line or lines. This can be done by typing the line number of the statement to be changed followed by the new or revised statement. To eliminate entirely a statement in his program, the user need only type the line number of the statement followed by a carriage-return. Additional lines may be inserted in a program by typing statements with intermediate line numbers, *ie*, if a statement is to be inserted between lines 100 and 110 the user simply types a line number of say 105 followed by the statement he wishes to insert.

#### Keyboard Mode

Data General Corporation's BASIC system will accept certain statements not preceded by line numbers and execute them immediately. These keyboard commands are: RUN, NEW, LIST, PRINT, GOTO, DIM, LET, REM, RESTORE, and the ESC key, on the teletypewriter. They are useful in testing and debugging programs, and they also allow the system to be used as a simple desk calculator.

After typing a complete program, the user causes sequential execution of his program to begin starting at the lowest statement number by typing the keyboard command "RUN" followed by a carriage-return. If the program is correct, it will be run and will cause whatever output is specified in the program to be printed on the teletypewriter. Errors detected after RUN are structural (incorrect nesting of loops, GOSUBs without matching returns) or arithmetic errors. Grammatical errors or errors of form are not possible at this point, because they are previously detected during the typing of each

statement. If it is obvious that the program running is giving the wrong answers, the user can hit the ESC key on the teletypewriter which will cause computation to cease and the BASIC system to type "STOP @ XXXX" where XXXX is the line number of the statement about to be executed when execution was interrupted. The user can then inquire what the values of program variables are by means of the PRINT statement. For example, if A is a variable in the user's program, he can find its value at the time of interruption by simply typing "PRINT A" and the BASIC system will reply with the values. When using PRINT (or ;) in the keyboard or desk calculator mode, only the expression up to the first comma, semi-colon, or carriage-return will be printed. For example,

; 1, 2 (CR)

will respond with 1. He may also change the value of a variable or variables before returning to program control by means of the LET command. If he wishes to change the value of A to 15 the user may give the keyboard command "LET A = 15".

In much the same way it is possible following such an interruption to redimension a previously defined array by entering a DIM statement with the same array name. This redimensioning does not in any way effect the amount of storage or the contents of storage previously allocated by a DIM program statement. It neither expands nor contracts the storage available, but in the two-dimensional case it changes the subscript address which refers to a given entry. For example, if the user had defined a 3 x 4 array A in his program by means of a DIM A(2, 3) program statement, and, after interrupting execution, gave the keyboard command DIM A(3, 2) to redimension A as a 4 x 3 array, the array would be transformed as illustrated:

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

Thus, entry A(1, 0) which contained a 5 must subsequently be referenced as A(1, 1); entry A(0, 3) must be referenced as A(1, 0); entry A(2, 2) must be referenced as A(3, 1); etc. It should be noted that redimensioning is confined to transforming an array into one with the same or fewer numbers of entries, and that subscript references outside



the newly defined range (s) of the subscript (s) will cause errors. For example, re-dimensioning a 3 x 5 array as a 2 x 8 array is not permitted (error 28), and in the above example (transforming a 3 x 4 to a 4 x 3) it would cause an error to use a 3 as a column subscript once the re-dimensioning had taken place (A(2, 3) would generate error 31).

During an interrupt, a user may also restore the data block pointer to the top of the data block by issuing the keyboard command "RESTORE". He may insert a line of comments at any point in keyboard mode simply by preceding the comments with "REM". If a line number is typed, the comments following REM will be stored as a program statement, but as a keyboard command the comments will not be retained.

To resume execution of a program following a programmed STOP or keyboard interruption, the user must type "GOTO XXXX" where XXXX is the line number of the statement at which the program should be re-entered. If the user types "RUN" all variables and arrays in the program will be re-initialized to zero, a RESTORE will be executed on the data block, and the program will be started again from the lowest statement number.

#### Ground Rules on Numbers, Variables, Arrays, and Expressions

Numbers handled by Data General Corporation's single-user BASIC are constrained as follows:

Input	-	7 significant digits
Output	-	6 significant digits
Internal	-	23 significant bits
Range	-	Positive Powers: $\pm (1-2^{-23}) (2^{127}) \approx \pm 1.70141 \times 10^{38}$ Negative Powers: $\pm 1/2 (2^{-127}) \approx \pm 2.0 \times 10^{-39}$

Variable names in BASIC must consist of a single letter or a single letter followed by a single digit. Some acceptable variable names would be: A, A7, Q, Q3, etc. Variable names such as 1X, SR, and Y23 would be regarded as errors.

In addition to ordinary variables BASIC allows the use of array variables to designate elements in an array. Arrays may be of one or two dimensions and array names must consist of a single letter. Array names and simple variable names must be distinct, the single letter used to denote an array name may not be used as a simple variable name. It is most efficient to declare an array by means of a dimension (DIM)

statement in a program, but this is not necessary if the subscript (s) will not exceed the range of 0 to 10. If no DIM statement for an array is given in a program the BASIC system will, upon encountering a reference to the array in some statement, automatically allocate storage for 11 entries (1 x 11) if the array reference showed only a single dimension, or for 121 entries (11 x 11) if the array reference showed two dimensions. In light of this it behooves the user to conserve program storage space by declaring small arrays with a DIM statement, particularly if he is working in a 4K environment. If the user requires arrays larger than 1 x 11 or 11 x 11 he must declare them with a DIM statement. Dimensions are limited to the range  $0 \leq \text{dimension} \leq 255$ . An array may not have more than 1024 entries.

Expressions in BASIC can be constructed using simple variables, arrays, and functions linked together by parentheses and the following five arithmetic operators:

SYMBOL	EXAMPLE	MEANING
+	X+Y	Addition (add X to Y)
-	X-Y	Subtraction (subtract Y from X)
*	X*Y	Multiplication (multiply Y by X)
/	X/Y	Division (divide X by Y)
↑	X↑3.2	Raise to the power (find $X^{3.2}$ )

(Single User BASIC computes  $A \uparrow B$  by means of the identity  $A \uparrow B = \text{EXP}(B * \text{LOG}(A))$ . This may result in some slight arithmetic errors in the sixth decimal position of the result. In addition, an arithmetic error (ERR 16) will result if A is negative. To raise variables and constants to integer exponents, multiplication should be used rather than exponentiation if this slight arithmetic error is objectionable.)

Parentheses are used to enclose subexpressions which are to be treated as entities within the larger expression. The computer evaluates an expression beginning with the lowest subexpressions and moves upwards until the whole is evaluated. Within each sub-expression arithmetic operations are performed in the usual precedence adopted for computer languages: first exponentiations, then multiplications and/or divisions, and lastly additions and/or subtractions. In the absence of parentheses in an expression involving operations of the same priority, the operations are performed from left to right.

## SUMMARY OF BASIC COMMANDS

### LET

This statement is used to instruct the BASIC system to perform a computation and assign the value obtained to a variable. Each LET statement is of the form:  $\text{LET } (\text{variable}) = (\text{expression})$ . The statement can be executed as a program statement, or, when not preceded by a line number, it can be executed immediately as a keyboard command. For example,

```
100 LET X = X*X + X + 3
```

```
255 LET Y = Z - 24 + X* 3.32
```

```
LET X = 2 + 5 * A
```

### READ, DATA, and RESTORE

The READ statement is used to retrieve values stored in DATA statements and to assign these values to the variables specified. Neither the READ statement nor the DATA statement can be used without the other. A READ statement causes the variables listed in it to be assigned, in order, the next available numbers from the whole body of DATA statements. It should be mentioned here that before running a program, the BASIC system assembles all the DATA statements, in the order which they appear in the program (line number sequence), into a single large data block. Thus, whenever a READ statement is encountered during execution, the BASIC system removes the next available unused entry or entries from the data block and assigns these values to the variable or variables specified in the READ statement. Normally, READ statements are placed in the program at those points where the data is to be manipulated, but the placement of DATA statements is arbitrary so long as they are in the correct order and the programmer keeps in mind that the data will be collected into a single block. For this reason, it is a common practice to collect all data statements and place them together at the beginning or end of a program. It should be remembered that only numbers are permitted in DATA statements and that formulas such as  $\text{SQR}(3)$ ,  $17\ 3$ ,  $16*7$ , and  $15/9$  will be rejected as errors. Also, READ and DATA statements are only acceptable to the system as program statements and may not be executed as keyboard commands.

If it is necessary to use data stored by DATA statements more than once in a program, the user may reset the data block pointer to the first number in the data block by means of the RESTORE statement. Thus, whenever a RESTORE statement is

encountered in a program, a subsequent READ statement will begin reading from the top of the block created by the DATA statements. If the data which the programmer wishes to reuse will not be stored at the top of the data block, he must take care to insert the correct number of dummy READ statements to move the pointer down the block to the data he desires. The RESTORE statement may also be used as a keyboard command.

## INPUT

The INPUT statement is used to permit entry of data from the teletypewriter during the running of a program. It serves the same function as a READ statement, but it does not require a DATA statement from which to draw numbers. Further, data entered with an INPUT statement is not saved within the program. The statement is acceptable only within a program and may not be used as a keyboard command.

When an INPUT statement is encountered during execution, the BASIC system types a question mark and waits for the user to enter the value for the argument. The user responds with a number terminated by a carriage-return. If more than one argument is specified in the INPUT statement the system will assign to the first argument the first value typed in. It will respond with another question mark. The user should then type the value to be assigned to the second argument, and if a third argument is specified, the system will respond with a third question mark, etc., until values for all the arguments have been entered, at which point the system will automatically continue execution of the program. Each value must, of course, be terminated by a carriage-return as it is typed. If the user makes an error in typing a number (he types a comma instead of a decimal point, for example) the computer will return a backwards slash "\" followed by another question mark, and he must enter the value again. Frequently, INPUTs are preceded by PRINT statement containing a message to inform the user what the question mark signifies. For example,

```
100 PRINT " THE VALUES OF A, B, AND C ARE";  
110 INPUT A, B, C
```

and during execution the computer types:

```
THE VALUES OF A, B, AND C ARE ?
```

Note that the semicolon at the end of the PRINT statement causes the computer to type the question mark at the end of the message instead of on the next line.

## PRINT and TAB

The PRINT statement may be used either as a program statement or as a keyboard command to cause printing operations on the Teletype. (The special character semi-colon (;) may be used interchangeably with the word PRINT. On listing, it will be converted to PRINT.) As a statement in a program it can be used a) to print results of computations, b) to print verbatim a message stored in the program, c) to print a combination of the two, and d) to skip a line. As a keyboard command it can be used e) to print the value of variables defined in the user's program or functions of these variables, and f) it can be used to print the results of computations on numbers in desk calculator fashion. Some examples of how the PRINT statement may be used in a program are:

1000	PRINT	" X VALUE", "SINE", "COSINE"	(b)
2000	PRINT	X, SIN (X), COS (X)	(a)
3000	PRINT	" THE SQUARE ROOT OF"; X; "IS"; SQR (X)	(c)
4000	PRINT		(d)

Formatting of output from a PRINT statement is automatically performed by the system unless the user specifies another format for his output. Whenever the arguments of the PRINT statement are separated by commas the BASIC system employs automatic formatting under which the teletypewriter line is divided into five zones starting at positions 0, 15, 30, 45 and 60. (If the comma has been initialized to represent 14 spaces, the five zones would start at positions 0, 14, 28, 42, 56.) Each comma in the PRINT statement causes the system to move to the next zone before typing the value of the next variable, or, if the fifth zone has been filled, to move to the first zone of the next line. The end of a PRINT statement signals a new line and causes a carriage-return/line-feed on the teletype unless this action is suppressed by placing a comma (or semicolon) at the end of the PRINT statement.

The user may override automatic formatting to specify compact formatting or particular line formats by means of the semicolon and the TAB function. Compact formatting, or elimination of spacing between the printing zones of the arguments being printed, is caused by the use of the semicolon to separate the variables in the PRINT statement. For example, the statement

```
1000 PRINT X, Y, Z
```

would cause the values of the variables X, Y and Z to be printed under automatic

formatting on a single teletypewriter line beginning at print positions 0, 15, and 30 respectively; whereas the statement

100 PRINT X:Y:Z

would cause the values of the three variables to be printed on a single line beginning at print position 0 with no spacing between each of the argument's print zones. The size and spacing within each of the variable's print zones depends on the value and type of the number being printed as described below. The caret symbol, ^, represents a space typed on the teletypewriter. If the absolute value of the variable is in the range  $10^{-1} \leq \text{Variable} < 10^6$  or if the absolute value can be represented exactly as a six-digit decimal fraction, then fixed format is used. Non-significant leading and trailing zeros are suppressed. The decimal point is omitted from integers.

examples:

-.001  
^ 425.321 ^  
^ .~~000000~~1 ^

Floating format is used in all other cases and has the form  $\bar{a}.d.dddddE\bar{+}ee$  where each d is a decimal digit and ee is a one or two digit exponent of ten. Numbers are preceded by a minus sign or a space and are followed by a space.

TAB clauses or functions, each with an argument of its own, may be inserted in PRINT statements to cause the Teletype to move to specific positions on the line before printing the variables. The argument of the TAB function may be either an integer or an expression and is evaluated modulo 70 or 75 depending upon whether the space has been initialized to represent 14 or 15 spaces respectively. If it is the latter, the value of the expression will be computed and its integer part taken. If the comma has been initialized to represent 15 spaces, <sup>and</sup> then the value of the argument is greater than 71 or less than 75 the teletype will print at the end of the current line. There are only 72 print positions on a teletypewriter, numbered 0 through 71. Commas following TAB clauses have no effect on positioning, but commas following arguments will cause automatic formatting unless the next argument is preceded by a TAB function.

If the user, having interrupted a program's execution by striking the ESC key, wishes to know the current value of the program variable X, he may employ the PRINT statement as a keyboard command. If he types

PRINT X

(e)

without a preceding line number, he would cause the current value of this variable to be printed out just following the keyboard command. A user who at any time wishes to perform a quick calculation may do so using the PRINT statement or a combination of the

LET and PRINT statements in keyboard mode.

For example,

```
PRINT SQR (5*5+12*12)
```

or

```
LET Q=(5*5+12*12)
```

```
PRINT SQR (Q)
```

(f)

will cause the system to return the square root of  $(5^2 + 12^2)$  on the same line as the PRINT commands.

### GO TO

The GOTO statement is used to direct the flow of execution in a program and may be employed as a keyboard command. When a GOTO statement is encountered in a program it causes the system to continue execution from the line number specified. It is possible to transfer control to a non-executable statement (a DATA statement, for example) in which case control passes to the next sequential executable statement. For example,

```
5Ø DATA 2.Ø, 3.1416, 7.8, 4.1
6Ø LET X1=SQR(B*B-4*A*C)
:
:
2ØØ GO TO 5Ø
```

### IF-THEN (IF-GOTO)

The IF-THEN statement is used to cause branching to another point in a program when a condition is fulfilled. For this reason it is also known as a conditional GOTO statement and is typed by the system as IF-GOTO when the keyboard command LIST is given. Both IF-THEN and IF-GOTO are perfectly acceptable variations of the statement for input to the system. Most commonly, the IF-THEN statement will have the form:

```
IF [ expression ] [ relational operator ] [ expression ] THEN
[ line number ]
```

Thus, a statement to cause branching when the variable A was not equal to 2 would be

```
1ØØ IF A <> 2 THEN 5Ø
```

where the concatenated symbols "<>" mean "is not equal to".

In a mathematically more general sense, the IF-THEN statement would have the form:

```
IF [ expression ] THEN [ line number ]
```

where the expression is evaluated and the result treated as true when non-zero, or

false when equal to zero. Using this form the above example would become

1000 IF A = 2 GOTO 500

This would cause the system to take its next instruction from line number 500 so long as A does not equal 2. If A = 2 control would pass to the next sequential instruction following line 1000.

The first form is more readily readable and is thus more often used. The following six standard relational operators are permitted:

<u>SYMBOL</u>	<u>EXAMPLE</u>	<u>MEANING</u>
=	X = Y	Is equal to .(X is equal to Y)
<	X < Y	Is less than .(X is less than Y)
<=	X <= Y	Is less than or equal to .(X is less than or equal to Y)
>	X > Y	Is greater than .(X is greater than Y)
>=	X >= Y	Is greater than or equal to .(X is greater than or equal to Y)
<>	X <> Y	Is not equal to .(X is not equal to Y)

The IF-THEN statement may not be used as a keyboard command.

### FOR, NEXT, and STEP

The FOR and NEXT statements and the STEP clause are used in setting up and operating a loop in the user's program. They must always appear together: the FOR statement, with or without a STEP specified, at the entrance to the loop; and the NEXT statement at the exit, pointing back to the entrance.

Every FOR statement is of the form

FOR [ *variable* ] = [ *expression* ] TO [ *expression* ] STEP [ *expression* ]  
where the variable may be any nonsubscripted variable. Most often, the expressions will be integers with the STEP clause omitted in which case the system assumes a step size of +1. Note that when specified, the step-size may be an expression which resolves to either a positive or negative number. Note too that the FOR and NEXT statements are not acceptable as keyboard commands.

### GOSUB and RETURN

The GOSUB and RETURN statements are used to transfer control to, and return from, subroutines in the user's program. The subroutine is entered by means of a GOSUB statement where the argument is the line number of the first statement in the subroutine. At the logical end of the subroutine there should be a RETURN statement



(which has no argument) directing the computer to return control to the statement following the GOSUB that called the subroutine. GOSUBs may be "nested," that is, one subroutine may call another, but it is necessary that each subroutine be exited by a RETURN statement when it completes its work. This is because actual return addresses are stored in a push-down list as each GOSUB is executed and must be picked off by RETURN statements in the reverse sequence. If the user exits a subroutine by means of a GOTO instead of a RETURN it is extremely likely that he will foul the correct passage of control between levels of subroutines. This does not mean that it is impossible to use several RETURN statements in a given subroutine if it is to be exited at several different points, but that the user should keep in mind how the system works. GOSUBS may not be used as keyboard commands.

### DIM

The DIM statement is used to inform the system to allocate a specific amount of storage for a one or two-dimensional array with a given name. Array names must consist of a single letter. One-dimensional arrays may have a maximum of 256 entries ( , DIM A (255) since there will be a zeroth entry) and two-dimensional arrays may have a maximum of 1024 entries ( , DIM B (31, 31) or DIM C (15, 63) since there will be both a row 0 and a column 0. For example, the statements

10 DIM X (17)

20 DIM Y (14, 9)

would cause the BASIC system to allocate storage for a one-dimensional array, X, with 18 entries, and for a two-dimensional array, Y, with 150 entries.

### Functions and DEF

#### A) Standard Functions

Data General Corporation's single-user BASIC allows the use in expressions of the five arithmetic operations (addition, subtraction, multiplication, division, exponentiation, +, -, \*, /, ^ ) and the eleven functions normally defined for BASIC. The arguments of these functions may themselves be expressions. The eleven functions are:

<u>Function</u>	<u>Meaning</u>	
SIN (X)	Find the sine of X	} where X is interpreted as an angle measured in radians
COS (X)	Find the cosine of X	
TAN (X)	Find the Tangent of X	
ATN (X)	Find the Arctangent of X (result expressed in radians)	
EXP (X)	Find $e^X$	
LOG (X)	Find the natural logarithm of X (lnX)	
SQR (X)	Find the square root of X ( $\sqrt{x}$ )	
ABS (X)	Find the absolute value of X (  X  )	
SGN (X)	Find the sign of X	
INT (X)	Find the largest integer $\leq$ X	
RND (X)	Generate a random number between $\emptyset$ and 1	

Most of these are well known functions and need no explanation, but the following few comments should be noted.

The arguments of the functions SIN, COS, TAN, ATN and ABS are confined to the range of acceptable real numbers ( $2 \times 10^{-39} \leq |x| \leq 1.7 \times 10^{38}$ ).

The LOG and SQR functions require their arguments to be positive. A negative or zero argument in the LOG function will cause the system to respond with the largest possible negative number (-1.70141 E+38) plus an error message (error 16). A negative argument in the SQR function will cause the system to respond with the square root of the same positive number plus an error message (error 16).

The argument of the EXP function is confined to those values which will generate the largest and smallest acceptable real numbers, eg,  $e^{88} \approx 1.7 \times 10^{38}$  and hence arguments greater than 88 will cause the system to return 1.70141E+38 as the answer along with error message 16.

The SGN function generates as its result a +1 if its argument is a positive number, a  $\emptyset$  if its argument is a zero, and a -1 if its argument is a negative number. Thus,  $SGN (.452) = 1$ ,  $SGN (\emptyset.\emptyset\emptyset) = \emptyset$ , and  $SGN (-24.8) = -1$ .

The INT function yields as its result the largest integer less than or equal to (not greater than) its argument, where the argument is limited to a six digit number with absolute value less than  $2^{31}$  or 2,147,483,648. Arguments of larger absolute value will be reduced to  $\pm 2.14748E+9$ . The INT function is frequently used in rounding operations:  $INT (X+.5)$  will round X to its nearest integer, whereas  $INT (10*+.5)/10$  will round X to one decimal place, or, more generally,  $INT (1ED*X+.5)/1ED$  will round X to D decimal places.

The RND function is used to generate a random number between  $\emptyset$  and 1. Although the form of the function requires an argument, it has no significance, and any number, or previously defined variable, may be substituted as the argument.

#### B) DEF

The DEF Statement is used to define as functions those expressions which a programmer uses several times in the course of his program, and which he wishes to abbreviate after having once written them. The name of each defined function must consist of three letters, the first two of which are "FN". Hence, up to 26 functions may be defined, *eg*, FNA, FNB, . . . , FNZ. For example, a function equivalent to  $X^2 + e^{3x+2}$  could be defined by the line

```
50 DEF FNT (X) = X*X + EXP (3X + 2)
```

Thereafter in his program the user would call for various values of the function simply by using FNT (2.4), FNT ( $-\emptyset.1$ ), FNT (A), etc. A DEF statement may occur anywhere in a program, and the expression to the right of the equal sign may be any formula which can be fitted on a single line. It may include any combination of other functions, including ones defined by other DEF statements, and it can involve other variables besides the one denoting the argument of the function. Functions are not actually defined until the DEF Statement has been executed.

#### STOP

The STOP statement is used to cause a halt in the execution of the user's program at any point. When a STOP is encountered, the BASIC system will cease execution of the user's program, type the message "STOP @ XXXX", where XXXX is the line number of the STOP statement, and wait for the user to type in additional program statements or alterations to statements, or, to type in a keyboard command.

#### REM

The REM statement is used to insert explanatory remarks in a program. Everything following the REM is stored exactly as typed and is reproduced when a listing is requested by a LIST statement. The BASIC system completely ignores the whole statement during execution. For example,

```
100 REM THIS IS A REMARK STATEMENT !
```

## END

The END statement is used to signal the end of execution of a program. Every program must have an END statement and it is often assigned the highest line number in the program. Its form is:

```
9999 END
```

After an END statement has been executed the BASIC system will inform the user that it has completed processing of the program by typing the message "\*\*READY\*\*".

## APPENDIX A

### Punching a Tape of a User's Program

To punch a BASIC program tape with blank leader and trailer, follow this procedure:

1. Turn the power switch on the front of the teletypewriter to the LOCAL position. Press the ON button on top of the teletype punch unit. Then press the HERE IS\* key to punch blank tape. When enough blank tape has been produced press the OFF button on top of the punch unit and turn the power switch back to the LINE position.
  2. Type the Keyboard command LIST, but do not press the RETURN key yet. Press the ON button on top of the teletype punch unit and then press the RETURN key. After waiting until the punching and listing is complete press the OFF button on the teletype punch unit.
  3. Repeat step 1 to punch a length of blank trailer on the tape.
  4. Remove the tape by pulling straight up. This produces arrows on the ends of the tape to indicate the direction in which the tape should be read through the reader. It is usually wise to write the name of the program on the tape for easy identification at a later time.
- \* On some teletypewriters the HERE IS key may not punch blank tape. In this case the user should hold down CTRL, SHIFT, REPT, and P keys simultaneously to punch blank tape. The REPT and P keys should be released first.

Loading a Tape of a User's Program

To load a BASIC program tape, follow this procedure:

1. Set the switch on the teletypewriter's paper tape reader to the STOP position. Release the plastic tape guide on the tape reader and mount the beginning end of the tape in the reader with the arrow at the end of the tape pointing forward and the punched portion hanging down behind. Be sure the sprocket holes in the tape leader are fitted on the sprocket wheel.
2. Close the plastic tape guide and move the switch on the reader to the START position. Wait until the entire tape has been read and listed, and then move the reader switch to the FREE position, and pull out the remaining tape.

Note: Before loading a tape of a BASIC program it is usual to give the keyboard command "NEW" which clears the user's space of all statements. If a NEW command is not given, the program will still be loaded, but any statements previously stored which bear the same line numbers as statements in the program being loaded will be replaced by the statements from the tape. Statements with line numbers not used in the program being loaded will not be affected. This feature is useful in loading subroutines or portions of programs from several pieces of tape.

APPENDIX B  
Programming Hints

The following suggestions are offered in the interest of making the most efficient use of user's storage. These hints become important if a Storage Overflow error occurs while entering or running a program.

1. Enter data which changes for each run by means of INPUT statements rather than by READ and DATA statements.
2. Keep remarks (REM statements) to a minimum.
3. Re-use variables and arrays used earlier in the program if the old value(s) will no longer be needed.
4. Dimension all arrays.
5. Dimension arrays as closely as possible to the largest subscripts that will be used.
6. Utilize the zeroth column and row of arrays.
7. Wherever possible utilize one lengthy statement (such as READ, DATA, INPUT, or PRINT) instead of two or more statements to accomplish the same results.
8. Define functions (DEF statement) whenever the same expression will be used at least twice in a program.

## APPENDIX C

### Short Summary of BASIC Statements

#### 1. Elementary BASIC

	<u>Purpose</u>	<u>Example</u>
READ	Reads data from data block	5Ø READ X, Y1, M(J, 3)
DATA	Stores data in data block	5Ø DATA -1, 2.Ø7, 31416E-4, 127829
PRINT	Types numbers and labels	5Ø PRINT "ANSWER="; X
;	Same as PRINT	
LET	Computes and assigns value	5Ø LET X2= X + Y 2
GO TO	Transfers control	5Ø GO TO 175
IF	Conditional transfer	5Ø IF T(I, J)=25 THEN 175
FOR	Sets up and operates a loop	5Ø FOR N=1Ø TO 1 STEP -1
NEXT	Closes loop	5Ø NEXT N
END	Final statement in program	5Ø END

#### 2. Advances BASIC

INPUT	Reads data from the Teletype	5Ø INPUT X, Y4, Z
DEF	Defines a function	5Ø DEF FNG (X) = 2* SIN (X)
GOSUB	Transfers to a subroutine	5Ø GO SUB 8ØØ
RETURN	Returns control from subroutine to statement following GOSUB	5Ø RETURN
RESTORE	Restores data block pointer to beginning of data block	5Ø RESTORE
REM	Permits comments	5Ø REM BEGINNING OF SUBROUTINE
DIM	Declares dimensions of arrays	5Ø DIM B (3, 5)
STOP	Stops program	5Ø STOP
TAB	Clause used in PRINT statement	5Ø PRINT TAB (2*A+5); ""



3. Notes

Variables	Single letter or single letter plus single digit
Array Names	Single letter
Arithmetic Operators	+, -, *, / ,
Relational Operators	<, <=, =, >, >=, < >
Functions	SQR, SIN, COS, TAN, ATN, LOG, EXP, ABS, SGN, INT, RND

## APPENDIX D

### Single-User BASIC Error Messages

00	Format Error
01	Illegal Character
02	Syntax Error
04	System Error
05	Illegal Statement Number
06	Too Many Variable Names
07	Spelling Error
08	Spelling Error
09	No Such Word
10	Incorrect Subscript Closure
11	Incorrect Parenthesis Closure
12	Not A Keyboard Command
13	No Such Line Number
14	Storage Overflow (While Inputting Program)
15	Read Statement Is Out Of Data
16	Arithmetic Overflow (Number Too Large)
18	Too Many Nested GOSUBs
19	Too Many RETURNS
20	Too Many Nested FORs
21	FOR Without NEXT
22	NEXT Without FOR
23	Out Of Storage (While Assigning Variable Storage)
24	Array Too Large
25	Attempt To Dimension Simple Variable
26	Variable Name Is Not Dimensionable
28	Redimensioned Array Is Larger Than Previously Defined
29	Expression Is Too Complex
30	Illegal Format In Defined Function
31	Subscript Exceeds Dimension
32	Undefined User Function
33	Too Many Nested Functions
34	Negative Subscript
35	Function Not Yet Implemented