May 1971

Introduction

The BASIC language is described in detail in a primer by Donald D. Spencer, A Guide to BASIC Programming: A Time-Sharing Language., published by Addision-Wesley, 1970. The purpose of the document which follows is to indicate where the TSS implementation of BASIC differs from the language described by Spencer as well as to provide a brief description of the language for those who are familiar with other programming languages. It also provides details of how the interaction between the user and the system works.

May 1971

CHAPTER 1 - BASIC STATEMENTS

1.1 STATEMENT TYPES

BASIC accepts three types of statements. The first type, called a direct statement, is executed immediately and most standard statements can be of this type. With a direct statement a value can be printed out or a variable changed. Some statements only make sense when executed directly, e.g., the statement which starts execution.

The second type of statement, called an indirect statement, is saved in a program to be executed later. Indirect statements are saved in a text file by using the third type of statement, standard Editor requests. (See 2.2 Editor under PART THREE - The CAL Time-Sharing System.) All of the editing requests of the text editor can be used to create and edit a file of indirect requests to be executed when the program is run.

Every indirect statement must start with a line number. When the program is run, the lines are executed in order of line number regardless of their order in the text file. Therefore it is advisable to number lines by 10's in order to leave room for lines which might have to be added while debugging. Although the lines may be in any order, it is less confusing if they are kept in order. It is also convenient to keep debugging statements at the bottom of the file.

If a program is not in order and/or the line numbers have uneven gaps between them, the program can be cleaned up by calling the subsystem RENUM. (See Section 3.3 below.)

1.2 BASIC Statements

The following is a brief description of the statements which the BASIC on the CAL TSS accepts. In the following descriptions square brackets indicate items which are optional. If the first bracket is followed by three dots, then any number of the items in brackets is allowed. CAL TSS BASIC accepts some statements and capabilities which are not in the BASIC described in A Guide to BASIC Programming: A Time-Sharing Language by Donald D. Spencer. In the descriptions which follow, these features are indicated by a footnote. A few features described by Spencer are not in this BASIC and are listed in Section 3.1.3 below. Unless otherwise noted, a statement has no effect if there is an error during its execution.

1.2.1 Declaration Statements

REM any character string

Remark. This is a comment and has no action when executed.

Example: REM THIS IS A COMMENT

DATA value [...,value]

When executed this statement has no effect. All the data statements in a program form a data bank of values for the READ statements to use. The values must be numerical constants or string constants. String constants do not need to have quotes around them if they start with a letter and contain only letters and digits.

Examples: DATA 10,20,6.3E+52 - 3 numerical values.

DATA "MINUS:+10",-10,TEN,+10 - A string, a number, a string, and a number.

DIM array name (dimension list) [...,array name (dimension list)]

The DIM statement reserves space for an array. If there was an array with the same name, it is deleted and a new array is reserved without any data in it. The dimension list is one or more expressions separated by commas.[1] If the value of the expression is not an integer, it is truncated to the next smallest integer. Any number of dimensions are allowed. If there is only one dimension, it is called a list and the index can go from zero to the value of the expression. If there is more than one dimension, the indices go from one to the value of the expression. String arrays (those with a $ in the name) can only have one dimension. If the total amount of space to be reserved is greater than 5000, an error occurs.

A one or two dimensional array can be referenced without a DIM statement explicitly dimensioning it. If this is done, the array is dimensioned to be of size 10 or 10 by 10.

Examples: DIM A(60), B$(11), C(5,N,3*N)
A is dimensioned to be a 61 element list with indices from 0 to 60. B$ is a 12 element list of strings. C

---

[1] Here Spencer restricts you to constants and one or two dimensions.

becomes a 3 dimensional array whose size depends on the value of N when the DIM statement is executed.

```
DIM B(I,J,I*J,2)
```

B becomes a four dimensional array whose indices can range from 1 to the value of the expressions given.

SIG *expression* [2]

The number of significant digits to be printed for a number is changed to the value of the *expression*. The value must be between 1 and 13. The default value is 7. When numbers are printed they are rounded to this many digits.

Example: SIG N

PAUSE [*string*] [2]

Execution pauses for debugging. When this statement is executed, *string* is printed if it exists, otherwise a message indicating the next line to be executed appears. During the pause, BASIC can accept direct statements and/or editing requests, after which execution can be resumed by typing CONTINUE.

Examples:
```
PAUSE
PAUSE "WE SHOULDN'T BE HERE"
```

END

The last line in every program must be an END statement. When executed, it stops execution.

STOP

This statement stops execution of the program; it acts like a jump to the END statement.

DEF FN *letter* (*parameter*) = *expression*

When executed this statement defines a one line function with one parameter. The dummy parameter used in defining the function is not affected when the function is called.

---

[2] Not mentioned in Spencer.

Example: DEF FNG(X3) = X3/10 - A0/X3

After this statement is executed FNG(expression) can be used as a sub-expression in any expression. For example one could write:

LET A(FNG(N+2)+1) = FNG(2*A3)

This would execute exactly as if one had written

LET A((N+2)/10 -A0/(N+2) + 1) = (2*A3)/10 - A0/(2*A3)

Note that X3 is only a dummy parameter and is not affected, and that the value used for A0 is equal to the value A0 has when the function is called, not when it is defined.

1.2.2 Input/Output Statements

READ variable [...,variable]

Values from the data bank formed by DATA statements are stored into the variables. If there is an error when this statement is executed, the variables before the one with the error in it are altered, but the data bank pointer is not moved so that the line can be re-executed.

Example: READ F$,C,G2
Read a string and 2 numbers from the data bank.

INPUT variable [...,variable]

INPUT reads values typed by the user at the teletype. When executed it types a question mark and expects the person at the teletype to type values which are appropriate. If the person makes a mistake, such as typing too many values or typing a string instead of a number, he is told about his mistake and that line is ignored. If he does not type enough values, the first ones are accepted and a question mark is typed asking for more values. It is best for the program to have a PRINT statement to tell the user what value to input.

Example: INPUT A,B,A$(N)
The user should type two numbers and a string (separated by commas) when this is executed. The line to be typed can be anything that would be legal after a DATA statement, i.e., numbers and strings separated by commas.

May 1971

PRINT [...item]

PRINT outputs characters on the teletype. Each item specifies what is to be printed. In addition to the items listed below, a carriage return will be done at the end of a line unless the last item was a comma, semi-colon, or colon. A carriage return is also done if a line exceeds 72 characters. The following table describes the allowable items in a print statement.

item	action
numerical expression	The expression is evaluated and then rounded to the appropriate number of digits (see SIG request above). If the number is negative, a minus sign is printed. If it is positive, a blank is typed. Then the numerical value of the number is printed in a format that seems best.
string variable	The contents of the string variable (either simple or subscripted) are printed.
"characters"	The characters enclosed in quotes will be printed. There may be any number of characters. The only character not allowed in the string is a double quote.
TAB(expression)	The expression is evaluated and truncated to an integer. Then blanks are typed until the teletype is positioned to type the next character in the column specified by the integer. There is an error if the result is not between one and seventy-two. If the teletype head is beyond the column specified by the expression, a carriage return is performed, then blanks are typed until the head is in the proper column.
,	Blanks are printed until the beginning of the next 14 character zone is reached. This is the most common and easiest way of getting numbers to line up correctly. From one to 14 blanks will be printed.

;	Two blanks are printed.
:	A colon[3] does not cause anything to be printed, but it is useful as a separator to print such things as the concatenation of two string variables, or to end a line without any blanks and without a carriage return.

Examples:

PRINT does a carriage return.

PRINT A$,B1,TAN(B1*B1) prints the string in A$; moves to the next print zone; prints the value in B1; moves to the next print zone; prints the tangent of the value in B1 squared; and ends with a carriage return.

PRINT "RESULTS PRINTED IN A TIGHT FORMAT":A1;2*A2;3*A3 prints the message immediately followed by the three values separated by 2 blanks, and terminated by a carriage return.

PRINT TAB(N):N prints the value of N starting in the N-th column - providing N is between 1 and 72. Note that since N is positive, column N will be left blank to indicate that the sign of N is positive.

PRINT "INPUT COUNT": prints the message INPUT COUNT and then leaves the teletype where it stops. If the next statement were an INPUT command, it would type a question mark right after COUNT.

PRINT A "=" B prints the contents of A followed by = immediately followed by the value in B.

PRINT A(I,J); prints elements I,J in matrix A followed by two blanks but no carriage return.

1.2.3 Control Statements

RESTORE

When executed this statement restores the pointer into the data bank to the top so that the bank is in the same state as it was when execution started.

---

[3] Not mentioned in Spencer.

May 1971

GO TO line number

This statement transfers control to the designated line.

ON expression GOTO line number [...,line number]

The value of expression designates which line is to be executed next. If the value is 1, execution jumps to the line with the first line number. If the value is 2, the second line number is used, etc. The expression is evaluated and truncated to an integer. The result must be between one and the number of line numbers given. This statement can be used to start or restart execution when used as a direct command.

Example: ON A/10 GOTO 10,25,50,65 divides A by ten, then truncates to an integer, which must be between 1 and 4. If the result is 1, execution will resume at line 10. If it is 2, execution will continue at line 25, and so forth for 3 and 4.

IF logical expression THEN line number
IF logical expression GOTO line number

These requests are interchangeable. The logical expression is evaluated to be either true or false. If it is true, then execution jumps to the line with the given line number; otherwise, it falls through to the next line. When used as a direct statement, execution resumes at the indicated line if the expression is true; otherwise nothing happens.

Examples: IF A < B & A$(N)><"ONE" GO TO 65 jumps to line 65 if the value in A is less than the value in B and the Nth string in list A$ is not equal to the string "ONE".

IF NOT (A < B ! C >= D) THEN 70
If it is not* the case that either A is less than B or C is greater than or equal to D, then jump to line 70.

FOR simple variable = expression TO expression [STEP expression]
NEXT simple variable

The FOR and NEXT statements go together to generate loops. When the FOR statement is executed the three expressions are evaluated

---

* The operators NOT, &, and ! are not allowed in the BASIC described by Spencer.

first. If the STEP expression is missing, its value is assumed to be one. Then the simple variable is given the value of the first expression and the other two are saved for incrementing and terminating the loop. If the value of the simple variable is greater than the second expression (less than if the step value is negative), the loop is not executed at all. Otherwise the loop is executed down to the NEXT statement. The NEXT statement adds the step value to the simple variable (which must match the simple variable in the corresponding FOR statement), and re-executes the loop unless the variable is greater (less if step negative) than the TO expression. When the loop terminates, the simple variable retains the value it had during the last iteration of the loop. Note that the value of the simple variable can be changed in the loop but not the step or terminating value. FOR loops can be nested but the simple variables on paired NEXT and FOR statements must match.

Examples: FOR I = -2 TO 2 STEP N/10
I is set to -2 and the value of N/10 is saved for incrementing.

FOR J = I TO N-3
J is set to the value of I and incremented by 1 until it has a value greater than N-3.

NEXT J
When executed, it adds 1 to J and then tests for the value in J being greater than N-3 was when the loop was entered.

NEXT I
When executed, it adds the value which N/10 had to I and loops if I is less than or equal to 2.

FOR K = -1 TO +1 STEP -1
NEXT K
This loop will not execute even once since K is already less than +1, but K will be given the value -1.

GO SUB line number

GO SUB acts like a GOTO except that it saves the address of the next line so that a RETURN can go to that line. It can be executed directly in which case the return statement stops execution.

RETURN

Execution goes to the line after the last GO SUB for which a RETURN has not been executed. If the GO SUB was a direct statement, a message is typed and the program counter is restored to what it was when the direct GO SUB was executed. Then execution halts.

1.2.4 Assignment Statement

LET variable = [...variable =] expression

Each variable takes on the value of the expression.

Examples: LET A = SIN(A) stores the sine of A back into A.

LET C2 = B3 = A(5*A) = 10/.76
Variables C2, B3 and the 5*A element of array A, are all set to 10/.76.

1.2.5 Matrix Statements

There are 11 matrix statements. Some can deal with an array of any dimension, some work on an array of 3 or less dimensions, while others must have a two dimensional matrix. In all cases the array must be dimensioned before the statement is executed. In the description of the statements which follow, the letters a, b and c represent the name of numerical arrays. Matrix statements do not work with string lists.

MAT READ c

Read values from the data bank into array c. The array c must have three dimensions or less. If the data bank runs out of data, there is an error. (The data goes into the matrix but the pointer into the data bank does not move.) The data are read in sequentially for lists (i.e., c(0), c(1),...,c(n)). For matrices, the data is read in by rows (i.e., c(1,1), c(1,2), ..., c(1,n), c(2,1),...,c(m,n). For three dimensional arrays, values are read in by rows then planes (i.e, c(1,1,1), c(1,2,1), ..., c(1,n,1),c(2,1,1),..., c(m,n,1),c(1,1,2), ..., c(m,n,p).)

Example: MAT READ Z

MAT PRINT c [separator]

Print out the array c. The separator is the same as the allowed separators for PRINT statements (i.e., comma, semi-colon, colon). If the separator is missing, it is assumed to be a comma. The values are printed out in the same order as they are read in for a

May 1971

MAT READ statement. The spacings betweeen values are determined by the separator as in a PRINT statement. For one dimensional lists, the separator has no meaning since a carriage return is done after every value. For two dimensional matrices, a carriage return is done after every row. In three dimensional arrays, a carriage return is done after every row and four blank lines are printed between planes. More than 3 dimensions are not allowed. If an element has not been stored in, then the message UNINT is printed rather than a value.

MAT c = TRN(a)

Matrix c becomes the transpose of matrix a. If an element of a is uninitialized, there is no error message, and the corresponding element in c becomes uninitialized. a and c must have 2 dimensions.

MAT c = ZER

Every element in array c is set to zero. Any number of dimensions are allowed on c.

MAT c = IDN

The square matrix c is set to the identity matrix. All elements are zero except the diagonal, which is set to one.

MAT c = CON

The array c is set to all ones. The array c can have any number of dimensions.

MAT c = a + b

Array c takes on the sum of array a and b. Arrays a, b, and c must have the same shape. Any number of dimensions is allowed.

MAT c = a - b

Array c takes on the difference between array a and array b. Arrays a, b, and c must have the same shape. Any number of dimensions is allowed.

May 1971

MAT c = (expression) * b

Array c takes on the scalar product of the expression and array b. Any number of dimensions is allowed.

MAT c = INV(a)

Matrix c takes on the inverse of matrix a. If there are any errors and c is different from a, c may be changed into an intermediate value.

1.2.6 Direct Statements

Most statements can be executed immediately by simply typing them and are therefore referred to as direct statements. The exceptions are the following: DATA, END, STOP, PAUSE, FOR, NEXT, RETURN. When statements such as GOTO, ON, and IF cause a jump in the program, execution is resumed at the line specified with no change in the state of variables or the GOSUB return stack. If a direct GOSUB is executed, a special return is saved so that when the RETURN is executed a message specifying which GOSUB is returning is printed. Control then returns to the teletype such that a CONTINUE (see below) will act as it would have before the subroutine was called.

There are some statements which must be direct:

LIMIT integer

The integer specifies a limit to the number of statements that may be executed without control being returned to the console. The count of statements executed is reset to zero every time the execution of the program is resumed. To remove the limit type LIMIT without an integer or with a zero.

RUN

Execution of this direct statement clears all variables, arrays, function definitions, and GOSUB calls. It resets values such as the number of significant digits, the clock and the random number generator seed. The data bank is restored. It sets up the program to be just as it was before it was ever executed, and initiates execution at the line with the lowest line number.

CONTINUE

Execution continues where it last stopped. If execution was stopped by an error, it resumes at the line that caused the error. Some statements may be partially executed when an error occurs, but they can all be re-executed with no problem. If the line which should be executed has been deleted, execution resumes at the next line.

CHAPTER 2 - BASIC EXPRESSIONS

2.1 EXPRESSION TYPES

There are three types of expressions in BASIC: numerical, string, and logical. All expressions can use any level of parenthesis nesting to group sub-expressions.

Most expressions are numerical; when evaluated, they produce a number. The variables allowed in a numerical expression are simple variables, consisting of a letter optionally followed by a digit, and subscripted variables, consisting of a letter followed by a list of expressions separated by commas and enclosed in parentheses. User defined and/or library functions (see 2.3 below) may be used in numerical expressions; numerical constants are also allowed. Numerical expressions can use the arithmetic operators +, -, *, /, and ↑ (see 2.2 below). Two operators must be separated by a parenthesis or one of the above elements, e.g., the expression A/-B is illegal, and must be written A/(-B). The error message which would be typed for this error is ERROR EXPRESSION MISSING.

String expressions consist of a string variable (subscripted or simple) or a string constant. There are no string operators or functions in BASIC. There are 26 simple string variables; A$ -Z$. There are also 26 string array names, A$ -Z$, which can be only one-dimensional lists. Strings must always be 15 characters or less. Strings can be assigned, printed, or tested for lexicographical order.[5] There are no other operations involving strings.

Logical expressions produce a value of true or false, and can only occur in IF commands. The relational operators, <, >, =, <=, =>, <> operate on numerical (or string for = and <>) expressions to produce a value of true or false. The logical operators &, !, and NOT operate on expressions having the value true or false. Note that the NOT operator is always a unary operator and must not occur directly following another operator.

2.2 OPERATORS

The following list gives all the operators in BASIC. Those with the highest binding strength occur first. Groups of operators not

---

[5] Lexicographical order corresponds to that shown in Table 1 of Appendix A of the CAL TSS section.

separated by blank lines have the same binding strength.

Arithmetic Operators

↑	Exponentiation
*	Multiplication
/	Division
+	Addition (may be unary)
-	Subtraction (may be unary)

Relational Operators

=	Equal
<>,><,#	Not equal
<	Less than
<=, =<	Less than or equal
>	Greater than
=>, >=	Greater than or equal

Logical Operators

!	Logical OR
&	Logical AND
NOT	Logical NOT

In addition to the regular numerical constants, the variable PI can be used as a constant. Its value is the best approximation to pi that is possible on this machine.

2.3 PREDEFINED FUNCTIONS

The following list gives the functions that are predefined as part of CAL TSS BASIC. They are all called with one argument which may be any numerical expression. In the descriptions below the argument of the function is referred to as X, but it may be any valid numerical expression.

Name	Use
ABS(X)	Returns the absolute value of X.
ACS(X)	Returns arc-cosine of X.
ASN(X)	Returns arc-sine of X.
ATN(X)	Returns the arc-tangent of X.
COS(X)	Returns the cosine of X. X is in radians.
EXP(X)	Returns the value e^X.
INT(X)	Returns the integer part of X. It truncates X to the integer with the next lowest absolute value. If X is an integer it has no effect.

May 1971

LOG(X)	Returns the natural logarithm of X.
LGT(X)	Returns $\log_{10}$ of X.
RND(X)	If X is zero or RND has been called with a non-zero argument it returns the next pseudo-random number. If X is positive and RND has not been called with a non-zero argument, the sequence of random numbers is affected by X in a repeatable manner. If X is negative and RND has never been called with a non-zero argument, the real time clock is used to affect the sequence of random numbers.
SGN(X)	Returns the sign of X. If X is negative, it returns -1. If X is positive, it returns +1. If X is 0, it returns 0.
SIN(X)	Returns the sine of X. X is in radians.
SQR(X)	Returns the square root of X.
TAN(X)	Returns the tangent of X. X is in radians.
TIM(X)	Returns the amount of computer time (in seconds) used. The RUN command resets the clock to zero.

CHAPTER 3 - FEATURES OF USING CAL TSS BASIC

3.1 CAL TSS BASIC VS. GE 265 BASIC

This section describes the differences between the CAL TSS implementation and the implementation on the GE 265 as described in A Guide to BASIC Programming: A Time-Sharing Language by Spencer.

3.1.1 Differences

The criterion for determining the format of a number when printing differs slightly.

A semi-colon always causes two spaces to be typed, rather than a number dependent on the size of the last value typed.

The teletype line is divided into 14 character zones rather than 15 character zones when printing values separated by commas.

3.1.2 Additional Features.

The SIG statement was added to allow more significant digits to be printed out. This can mess up the appearance of the output as a number can be more than the zone size used for commas.

The separator : was added in PRINT statements to allow values to be printed with no spacing.

The PAUSE statement was added to aid in debugging.

GOTO can be used as well as THEN in an IF command.

There is no limit to the number of dimensions allowed on an array.

The range of a subscript given in a DIM command can be any expression rather than a constant.

Some of the relational operators have alternative forms. See Section 2.2 above.

The logical operators, NOT, &, and ! were added to require fewer lines to make a decision.

The variable PI was added to provide a good value of pi.

May 1971

3.1.3 Missing Features

The PRINT USING command is not implemented.

The function CLK is not implemented.

3.2 RUNNING A BASIC PROGRAM UNDER TSS

An example of how to enter and debug a BASIC program should help clarify matters. BASIC is called by typing BASIC to the command processor. When BASIC is ready to accept statements, it will type BASIC HERE, followed on the next line by its prompt character, a colon (:). To create a program, insert mode is entered by typing I followed by a carriage return. Now the lines of the program can be typed into the file. Note that, as in the Editor, it is necessary to be in insert mode to add lines to the program. If a line to be added has an error in it, BASIC will type an error message and that line will not go into the file. Instead, the line remains as the old line in the Line Collector so that it can be corrected before being put into the program. If a bad line is in a file that is read in with the R command of the Editor, the line does not go into the program, and it is typed after the error message. A line already in the program can be changed or edited, but if the new "corrected" line has an error in it, it is typed after the error message and the change will not be implemented. To aid in debugging, some PAUSE statements may be added. PAUSE stops execution with a message giving the next line to be executed, thereby allowing the programmer to check values with direct PRINT statements, or to enter any other direct statements. The program may be modified or changed in any way during a PAUSE. Unless the last line is an END statement, the program cannot be executed.

Execution of a BASIC program is started by typing RUN. All variables, functions, and arrays are made to be undefined and execution of the program starts at the beginning. If there are some variables which should not be destroyed because they have been set with direct statements or from execution of a previous program, execution can be started with a direct GO TO. When the program encounters the END or a STOP statement, it stops with the message EXECUTION COMPLETE.

The program may stop in the middle of execution for several reasons. If there is an error in running, such as division by zero or a jump to a non-existent line, an error message is typed out and execution is halted. Direct statements and/or editing requests can then be entered to discover and fix the problem. If, for example, the program jumped to a non-existent line, it could be fixed by adding a line which was forgotten or correcting the line number in the GO TO statement. After the problem has been fixed, the program could be restarted with a RUN statement or CONTINUE, which would restart execution with the line where the error occurred. Execution can be halted by executing more

lines than specified by a direct LIMIT statement. By using LIMIT, a program can be prevented from getting caught in a loop. If a program gets in a loop, it is possible to get out by hitting the panic button - CTRL-SHIFT-P. A message is printed as if an error had occurred. Execution can be resumed by typing CONTINUE.

There are two ways to leave BASIC. These are the same statements as are used to leave the Editor. Q means to quit and to destroy the program that was created. F,fname saves the text of the program on the file specified so that it can be printed or loaded again later.

3.3 RENUMBERING A BASIC PROGRAM

To rearrange a sloopy BASIC program under CAL TSS, type RENUM fname where fname is the text file containing the program as a parameter (e.g., RENUM PROG). Provided the program is a legal BASIC program, the lines will be arranged in order of line number. Then RENUM asks the user if he wants to renumber the program. If the response is a line that begins with NO, RENUM returns to the Command Processor and the program lines are sorted but otherwise unchanged. Any other response will cause the lines specified to have the new line numbers specified by the user. All references to these lines will also be changed to the new line numbers. The user indicates which lines are to be changed and how, by responding to the four questions listed below. It is not possible to renumber lines so that the execution order of the program is changed. If that is attempted, there is an error. An appropriate error message is typed and some or all of the questions are reasked. The four questions take numbers as responses. Characters after the first non-numeric, non-blank character are ignored. If the number typed is zero or missing, the default value is used. The following table gives the questions and their meaning in the order in which they are asked.

FROM?	Give the line number of the first line to renumber. Default is the first line in the program.
TO?	Give the line number of the last line to renumber. Default is the last line of the program.
START?	Give the number which the first renumbered line is to have. Default is 100.
INCREMENT?	Give the increments by which the succeeding line numbers are to differ. Default is 10.

CHAPTER 4 - FEATURES OF USING BATCH BASIC

4.1 BASIC STATEMENTS

The BASIC language is also available under the batch system, CALIDOSCOPE, in a slightly modified form. The first and most obvious difference is the absence of direct statements, since there is no programmer-machine interaction under a batch system.

Batch BASIC has the same set of indirect statements as CAL TSS BASIC, with the exclusion of the PAUSE statement which is not appropriate for batch processing. However, two of the I/O statements, INPUT and PRINT, behave differently from their CAL TSS counterparts since there is no teletype.

4.1.1 The INPUT Statement

The INPUT statement inputs values from cards immediately following the EOR card which ends the program (see Section 4.2 Deck Setup below). The data on the cards must be in the same format as they were in the teletype lines under CAL TSS BASIC. Data cards are read until all the variable addresses are satisfied. If only some of the values on a card are used by the INPUT statement, the rest are saved for the next INPUT statement. If a card contains a syntax error such as an illegal value, e.g., one lacking a closing quote or with too large an exponent, the card image, with blanks removed, is printed on the output fileset followed by an error message. Then the card image is discarded. However, an error in type, such as specifying a number where a string belongs, is treated as an execution error.

It is possible for INPUT to read card images from a file other than the fileset INPUT. The alternate fileset name is specified as the first parameter of the control card used to load and execute BASIC (see Section 4.3 Execution).

4.1.2 The PRINT Statement

The PRINT statement interacts with the line printer in the same way as it does with the teletype under CAL TSS BASIC. Lines are still restricted to a maximum of 72 characters. No carriage control characters are needed since a blank for single spacing is inserted at the beginning of each line. The output generated during execution follows immediately after the listing of the program generated by the BASIC interpreter.

May 1971

4.2 DECK SETUP

The program to be interpreted by BASIC must be punched onto cards, one line per card. Every line must begin with a line number and if a card is out of order (i.e., line numbers are not in sequential order ignoring gaps), a warning message is printed below the card image on the output fileset, and it is executed in the order specified by its line number. If a syntax error is detected on a card, an error message is printed below the card image and that card image is ignored. The deck is set up as follows:

```
Job Card
COMMON,BASIC.
LGO,BASIC[,input fname].
EOR
 [ Program deck
EOR
 [Data cards, if any
EOJ
```

The cards for loading and executing the BASIC interpreter (those between the Job Card and first EOR card) are subject to change. When an EOR (7-8-9) card or an EOJ (6-7-8-9) card is read, the program begins execution.

4.3 EXECUTION

The program is executed even if compilation errors were detected; the offending lines are simply ignored. If a fatal error occurs during execution, all variables which have been assigned values and all arrays which have been dimensioned are dumped to the output fileset. Values are printed with seven significant digits. Arrays of 3 dimensions or less are dumped as if a MAT PRINT statement were being executed. Arrays of more dimensions are dumped as they are stored in memory, i.e., according to the formula:

$$1+(i-1)+I*(j-1)+I*J*(k-1)+I*J*K*(l-1)+....$$

where I,J,K,L... are the dimensions and i,j,k,l... are the subscript values for a given element. For example, if any array has 4 dimensions (3,3,3,3) and the desired element is (3,3,1,1), it will appear as the 9th value in the dump:

$$1+(3-1)+3*(3-1)+3*3*(1-1)+3*3*3*(1-1)=9.$$

Successive values are printed across the page in rows.