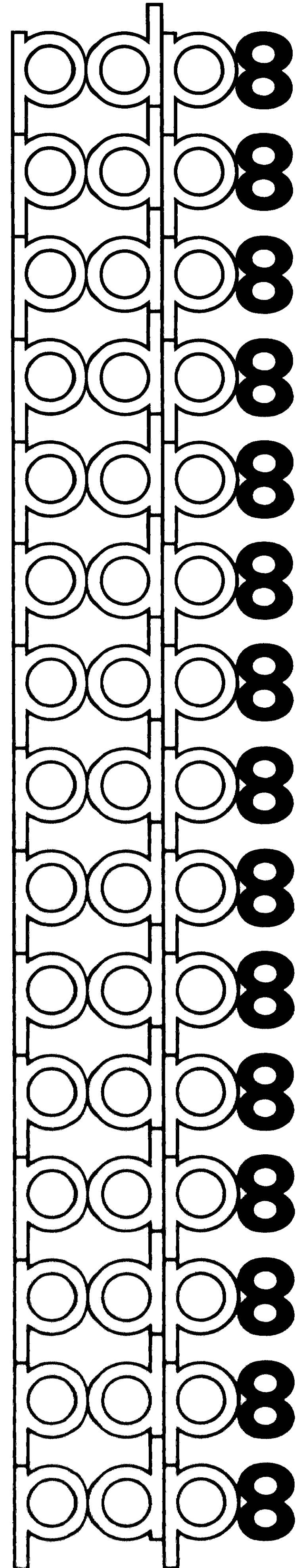
digital

8K basic



digital equipment corporation

DEC-08-LBSMA-A-D

8K BASIC

For additional copies, order No. DEC-08-LBSMA-A-D from Software Distribution Center, Digital Equipment Corporation, Maynard, Mass.

digital equipment corporation · maynard. massachusetts

First Printing, July 1973

Copyright © 1973 by Digital Equipment Corporation

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

CDP
COMPUTER LAB
COMTEX
COMSYST
DDT
DEC
DECCOMM
DECTAPE
DIBOL
DIGITAL
DNC
EDGRIN
EDUSYSTEM
FLIP CHIP
FOCAL
GLC-8
IDAC
IDACS
INDAC
KA10
LAB-8
LAB-8/e
LAB-K
OMNIBUS
OS/8
PDP
PHA
PS/8
QUICKPOINT
RAD-8
RSTS
RSX
RTM
SABR
TYPESET 8
UNIBUS

contents

Introduction 1

Numbers 2

Variables 3

Arithmetic Operations 3
- Priority of Arithmetic Operations 4
  - Parentheses 4

Relational Operators 5

Immediate Mode 6
- PRINT Command 6
- LET Command 6

BASIC Statements 7
- Example Program 7
- Statement Numbers 8
- Commenting the Program 9
  - REM 9
- Terminating the Program 9
  - END 9
  - STOP 10
- The Arithmetic Statement 10
  - LET 10
- Input/Output Statements 11
  - READ and DATA 11
  - RESTORE 12
  - INPUT 14
  - PTR 14
  - PRINT 15
  - LPT 19
  - PTP 20
  - TTY IN and TTY OUT 20

Loops 22
FOR, NEXT, and STEP 22
Subscripted Variables 25
DIM 27

Transfer of Control Statements 29
Unconditional Transfer—GOTO 29
Conditional Transfer—IF-THEN and IF-GOTO 30
Subroutines 30
GOSUB and RETURN 30
Functions 34
Sign Function—SGN(x) 35
Integer Function—INT(x) 35
Random Number Function—RND(x) 36
TAB Function 37
PUT and GET Functions 39
FNA Function 42

User-Defined Function—UUF 43
Coding Formats 43
Floating-Point Format 44
Addressing 45
Floating-Point Instruction Set 45
Writing the Program 47
Examples 48

Editing and Control Commands 53
Erasing Characters and Lines 53
SHIFT/O, RUBOUTS, and NO RUBOUTS 53
Listing and Punching a Program 55
LIST 55
PTP and LPT 55
Reading a Program 56
PTR 56
Running a Program 56
RUN 56
PTP and LPT 57
Stopping a Run 57

CTRL/C 57
CTRL/O 57
Erasing a Program in Core 58
SCR 58
Loading and Operating Procedures 58
BASIC Compiler 58
User-Defined Function 59
8K BASIC Error Messages 59
8K BASIC Symbol Table 61
Statement and Command Summaries 67
Edit and Control Commands 67
BASIC Statements 67
Appendix A A-1
Appendix B B-1

8K basic

INTRODUCTION

8K BASIC is an interactive programming language with a variety of applications. It is used in scientific and business environments to solve both simple and complex mathematical problems with a minimum of programming effort. It is used by educators and students as a problem-solving tool and as an aid to learning through programmed instruction and simulation.

In many respects the BASIC language is similar to other programming languages (such as FOCAL and FORTRAN), but BASIC is aimed at facilitating communication between the user and the computer. The BASIC user types in the computational procedure as a series of numbered statements, making use of common English words and familiar mathematical notations. Because of the small number of commands necessary and its easy application in solving problems, BASIC is one of the simplest computer languages to learn. With experience, the user can add the advanced techniques available in the language to perform more intricate manipulations or express a problem more efficiently and concisely.

8K BASIC is an extended version of DEC's 4K BASIC,[1] but has additional features and requires 8K of core. The user who has no familiarity with the BASIC language may wish to refer to the *EduSystem Handbook* for a background description of the language fundamentals, and for information pertaining to working with BASIC at the computer.

The minimum system configuration for 8K BASIC is a PDP-8

[1] 4K BASIC, or EduSystem 10, is the most fundamental BASIC in DEC's series of EduSystems. This series is directed primarily for use in an educational environment. Information concerning the EduSystems may be obtained from DEC's PDP-8 Educational Marketing Department.

series computer with 8K of core memory. Supported options include a high-speed reader and punch, and an LP08 line printer.

New features provided by 8K BASIC include one and two-dimensional subscripting, faster execution time, user-coded functions, use of the LP08 line printer and high-speed reader/punch, and specification of input and output devices from any part of a program.

Loading and operating instructions and a command summary are included at the end of the manual.

NUMBERS

BASIC treats all numbers (real and integer) as decimal numbers —that is, it accepts any number containing a decimal point, and assumes a decimal point after an integer. The advantage of treating all numbers as decimal numbers is that the programmer can use any number or symbol in any mathematical expression without regard to its type.

In addition to integer and real formats, a third format is recognized and accepted by 8K BASIC and is used to express numbers outside the range .01<=x<=1,000,000. This format is called exponential or E-type notation, and in this format, a number is expressed as a decimal number times some power of 10. The form is:

xxEn

where E represents "times 10 to the power of"; thus the number is read: "xx times 10 to the power of n." For example:

$$23.4E2 = 23.4*10^2 = 2340$$

Data may be input in any one or all three of these forms. Results of computations are output as decimals if they are within the range previously stated; otherwise, they are output in E format. BASIC handles seven significant digits in normal operation and input/output, as illustrated below:

Value Typed In	Value Output By BASIC
.01	.01
.0099	9.900000E−3
999999	999999
1000000	1.000000E+6

BASIC automatically suppresses the printing of leading and trailing zeros in integer and decimal numbers, and, as can be seen from the preceding examples, formats all exponential numbers in the form:

(sign) x.xxxxxx E (+ or −) n

where x represents the number carried to six decimal places, E stands for "times 10 to the power of," and n represents the exponential value. For example:

−3.470218E+8 is equal to −347,021,800
7.260000E−4 is equal to .000726

VARIABLES

A variable in BASIC is an algebraic symbol representing a number, and is formed by a single letter or a letter followed by a digit. For example:

Acceptable Variables	Unacceptable Variables
I	2C − a digit cannot begin a variable
B3	AB − two or more letters cannot form a variable
X	

The user may assign values to variables either by indicating the values in a LET statement, or by inputting the values as data; these operations are discussed further on in the manual.

ARITHMETIC OPERATIONS

BASIC performs addition, subtraction, multiplication, division and exponentiation, as well as more complicated operations explained in detail later in the manual. The five operators used in writing most formulas are:

Symbol Operator	Meaning	Example
+	Addition	A + B
−	Subtraction	A − B
*	Multiplication	A * B
/	Division	A / B
↑	Exponentiation (Raise A to the Bth power)	A ↑ B

Priority of Arithmetic Operations

In any given mathematical formula, BASIC performs the arithmetic operations in the following order of evaluation:

1. Parentheses receive top priority. Any expression within parentheses is evaluated before an unparenthesized expression.
2. In absence of parentheses, the order of priority is:
   a. Exponentiation
   b. Multiplication and Division (of equal priority)
   c. Addition and Subtraction (of equal priority)
3. If either 1 or 2 above does not clearly designate the order of priority, then the evaluation of expressions proceeds from left to right.

The expression A↑B↑C is evaluated from left to right as follows:

1.	A↑B	= step 1
2.	(result of step 1)↑C	= answer

The expression A/B*C is also evaluated from left to right since multiplication and division are of equal priority:

1.	A/B	= step 1
2.	(result of step 1)*C	= answer

PARENTHESES

Parentheses may be used by the programmer to change the order of priority (as listed in rule 2 above), as expressions within parentheses are always evaluated first. Thus, by enclosing expressions appropriately, the programmer can control the order of evaluation. Parentheses may be nested, or enclosed by a second set (or more) of parentheses. In this case, the expression within the innermost parentheses is evaluated first, and then the next innermost, and so on, until all have been evaluated.

Consider the following example:

```
A=7*((B↑2+4)/X)
```

The order of priority is:

1.	B↑2	= step 1
2.	(result of step 1)+4	= step 2
3.	(result of step 2)/X	= step 3
4.	(result of step 3)*7	= A

Parentheses also prevent any confusion or doubt as to how the expression is evaluated. For example:

```
A*B↑2/7+B/C+D↑2

((A*B↑2)/7)+((B/C)+D↑2)
```

Both of these formulas will be executed in the same way. However, the inexperienced programmer or student may find that the second is easier to understand.

Spaces may be used in a similar manner. Since the BASIC compiler ignores spaces, the two statements:

```
10 LET B = D↑2 + 1

10LETB=D↑2+1
```

are identical, but spaces in the first statement provide ease in reading.

RELATIONAL OPERATORS

A program may require that two values be compared at some point to discover their relation to one another. To accomplish this, BASIC makes use of the following relational operators:

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

Depending upon the result of the comparison, control of program execution may be directed to another part of the program, or the validity of the relationship may cause a value of 0 to 1 to be associated with a variable (that is, if a condition is true, a value of 1 is assigned; if a condition is not true, then the value of 0 is returned). Relational operators are used in conjunction with IF and LET statements, both of which are discussed in greater detail later in the manual.

The meaning of the equal (=) sign should be clarified. In algebraic notation, the formula X=X+1 is meaningless. However, in BASIC (and most computer languages), the equal sign designates replacement rather than equality. Thus, this formula is actually translated: "add one to the current value of X and store

the new result back in the same variable X." Whatever value has previously been assigned to X will be combined with the value 1. An expression such as A=B+C instructs the computer to add the values of B and C and store the result in a third variable A. The variable A is not being evaluated in terms of any previously assigned value, but only in terms of B and C. Therefore, if A has been assigned any value prior to its use in this statement, the old value is lost; it is instead replaced by the value of B+C.

IMMEDIATE MODE

There are two commands available which allow BASIC to act as a calculator—PRINT and LET. The user types in the algebraic expression which is to be calculated, and BASIC types back the result. This is called immediate mode since the user is not required to write a detailed program to calculate expressions and equations, but can use BASIC to produce results immediately.

PRINT Command

The PRINT command is of the form:

PRINT expression

and instructs BASIC to compute the value of the expression and print it on the Teletype. The expression may be made up of any decimal number, the arithmetic operators mentioned previously, and the functions which are discussed further on in the manual. (These may be used in conjunction with a string of text, as explained in the section concerning the PRINT statement.) For example:

```
PRINT 1/8↑8
 5.960464E-08
```

LET Command

Values may be assigned to variables by use of the LET command as follows:

LET variable = expression

The computer does not type anything in response to this command, but merely stores the information. This information may then be used in conjunction with a PRINT command to calculate results. For example:

```
LET P1=3.14159

PRINT P1*4↑2
 5Ø.26544
```

BASIC STATEMENTS

Example Program

The following example program is included at this point as an illustration of the format of a BASIC program, the ease in running it, and the type of output that may be produced. This program and its results are for the most part self-explanatory. Following sections cover the statements and commands used in BASIC programming.

```
1Ø REM - PROGRAM TO TAKE AVERAGE OF
15 REM - STUDENT GRADES AND CLASS GRADES
2Ø PRINT "HOW MANY STUDENTS, HOW MANY GRADES PER STUDENT";
3Ø INPUT A,B
4Ø LET I=Ø
5Ø FOR J=I TO A-1
55 LET V=Ø
6Ø PRINT "STUDENT NUMBER =";J
75 PRINT "ENTER GRADES"
76 LET D=J
8Ø FOR K=D TO D+(B-1)
81 INPUT G
82 LET V=V+G
85 NEXT K
9Ø LET V=V/B
95 PRINT "AVERAGE GRADE =";V
96 PRINT
99 LET Q=Q+V
1ØØ NEXT J
1Ø1 PRINT
1Ø2 PRINT
1Ø3 PRINT "CLASS AVERAGE =";Q/A
1Ø4 STOP
14Ø END
```

```
RUN
HOW MANY STUDENTS, HOW MANY GRADES PER STUDENT? 5,4
STUDENT NUMBER = 0
ENTER GRADES
?78
?86
?88
?74
AVERAGE GRADE = 81.5

STUDENT NUMBER = 1
ENTER GRADES
?59
?86
?70
?87
AVERAGE GRADE = 75.5

STUDENT NUMBER = 2
ENTER GRADES
?58
?64
?75
?80
AVERAGE GRADE = 69.25

STUDENT NUMBER = 3
ENTER GRADES
?88
?92
?85
?79
AVERAGE GRADE = 86

STUDENT NUMBER = 4
ENTER GRADES
?60
?78
?85
?80
AVERAGE GRADE = 75.75


CLASS AVERAGE = 77.6

READY.
```

Statement Numbers

An integer number is placed at the beginning of each line in a BASIC program. BASIC executes the statements in a program in numerically consecutive order, regardless of the order in which they have been typed. A common practice is to number lines by

fives or tens, so that additional lines may be inserted in a program without the necessity of renumbering lines already present.

Multiple statements may be placed on a single line by seperating each statement from the preceding statement with a backslash (SHIFT/L). For example:

```
10 A=5\B=.2\C=3\PRINT "ENTER DATA"
```

All of the statements in line 10 will be executed before BASIC continues to the next line. Only one statement number at the beginning of the entire line is necessary. However, it should be remembered that program control cannot be transferred to a statement within a line, but only to the first statement of the line in which it is contained (see the section entitled Transfer of Control Statements).

Commenting the Program

REM

The REM or REMARK statement allows the programmer to insert comments or remarks into a program without these comments affecting execution. The BASIC compiler ignores everything following REM. The form is:

(line number) REM (message)

In the Example Program, lines 10 and 15 are REMARK statements describing what the program does. It is often useful to put the name of the program and information relating to its use at the beginning where it is available for future reference. Remarks throughout the body of a long program will help later debugging by explaining the purpose of each section of code within the program.

Terminating the Program

END

The END statement (line 140 in the Example Program), if present, must be the last statement of the entire program. The form is:

(line number) END

This statement acts as a signal that the entire program has been executed. Use of the statement is optional. However, if the program contains an END statement, after execution, variables and

arrays are left in an undefined state, thereby losing any values they have been assigned during execution.

STOP

The STOP statement is used synonymously with the END statement to terminate execution, but while END occurs only once at the end of a program, STOP may occur any number of times. The format of the STOP statement is:

(line number) STOP

This statement signals that execution is to be terminated at that point in the program where it is encountered.

The Arithmetic Statement

LET

The Arithmetic (LET) statement is probably the most commonly used BASIC statement and is used whenever a value is to be assigned to a variable. It is of the form:

(line number) (LET) x = expression

where x represents a variable, and the expression is either a number, another variable, or an arithmetic expression. The word 'LET' is optional; thus the following statements are treated the same:

```
1ØØ LET A=A↑B+1Ø            11Ø LET C=F/G

1ØØ A=A↑B+1Ø                11Ø C=F/G
```

As mentioned earlier, relational operators may be used in a LET statement to assign a value of 0 (if false) or 1 (if true) to a variable depending upon the validity of a relationship. For example:

```
1ØØ A=1\B=2
11Ø C=A=B
12Ø D=A>B
13Ø E=A<>B
14Ø PRINT C,D,E
15Ø END
```

Translated, this actually means "let C=1 if A=B (0 otherwise); let D=1 if A>B (0 otherwise)" and so on. Thus, the values of C, D, and E are printed as follows:

```
RUN
 0                   0                   1

READY.
```

There is no limit to the number of relationships that may be tested in the statement.

Input/Output Statements

Input/Output statements allow the user to bring data into a program and output results or data at any time during execution. The Teletype keyboard, low or high-speed reader/punch, and LP08 line printer are all available as I/O devices in 8K BASIC. Statements which control their use are described next.

READ AND DATA

READ and DATA statements are used to input data into a program. One statement is never used without the other. The form of the READ statement is:

(line number) READ x1, x2, . . . xn

where x1 through xn represent variable names. For example:

```
10 READ A,B,C
```

A, B, and C are variables to which values will be assigned. Variables in a READ statement must be separated by commas. READ statements are generally placed at the beginning of a program, but must at least logically occur before that point in the program where the value is required for some computation.

Values which will be assigned to the variables in a READ statement are supplied in a DATA statement of the form:

(line number) DATA x1, x2, . . . xn

where x1 through xn represent values. The values must be separated by commas and occur in the same order as the variables which are listed in the corresponding READ statement. A DATA statement appropriate for the preceding READ statement is:

```
70 DATA 1,2,3
```

Thus, at execution time A=1, B=2, and C=3.

The DATA statement is usually placed at the end of a program (before the END statement) where it is easily accessible to the programmer should he wish to change the values.

A READ statement may have more or fewer variables than there are values in any one DATA statement. The READ statement causes BASIC to search all available DATA statements in the order of their line numbers until values are found for each variable in the READ. A second READ statement will begin reading values where the first stopped. If at some point in the program an attempt is made to read data which is not present or if the data is not separated by commas, BASIC will stop and print the following message at the console:

```
DATA ERROR AT LINE XXXX
```

where XXXX indicates the line which caused the error.

RESTORE

If it should become necessary to use the same data more than once in a program, the RESTORE statement will make it possible to recycle through the DATA statements beginning with the lowest numbered DATA statement. The RESTORE statement is of the form:

(line number) RESTORE

An example of its use follows:

```
15 READ B,C,D
•
•
•
55 RESTORE
60 READ E,F,G
•
•
•
80 DATA 6,3,4,7,9,2
•
•
100 END
```

The READ statements in lines 15 and 60 will both read the first three data values provided in line 80. (If the RESTORE statement

had not been inserted before line 60, then the second READ would pick up data in line 80 starting with the fourth value.)

The programmer may use the same variable names the second time through the data, or not, as he chooses, since the values are being read as though for the first time. In order to skip unwanted values, the programmer may insert replacement, or dummy, variables. Consider:

```
1 REM - PROGRAM TO ILLUSTRATE USE OF RESTORE
20 READ N
25 PRINT "VALUES OF X ARE:"
30 FOR I=1 TO N
40 READ X
50 PRINT X,
60 NEXT I
70 RESTORE
185 PRINT
190 PRINT "SECOND LIST OF X VALUES"
200 PRINT "FOLLOWING RESTORE STATEMENT:"
210 FOR I=1 TO N
220 READ X
230 PRINT X,
240 NEXT I
250 DATA 4,1,2
251 DATA 3,4
300 END
```

```
RUN
VALUES OF X ARE:
 1              2              3              4
SECOND LIST OF X VALUES
FOLLOWING RESTORE STATEMENT:
 4              1              2              3
READY.
```

The second time the data values are read, the first X picks up the value originally assigned to N in line 20, and as a result, BASIC prints:

```
4              1              2              3
```

To circumvent this, the programmer could insert a dummy variable which would pick up and store the first value, but would not be represented in the PRINT statement, in which case the output would be the same each time through the list.

INPUT

The INPUT statement is used when data is to be supplied by the user from the Teletype keyboard while a program is executing, and is of the form:

(line number) INPUT x1, x2, . . . xn

where x1 through xn represent variable names. For example:

```
25 INPUT A,B,C
```

This statement will cause the program to pause during execution, print a question mark on the Teletype console, and wait for the user to type in three numerical values. The user must separate the values by commas; they are entered into the computer by his pressing the RETURN key at the end of the list.

If the user does not insert enough values to satisfy the INPUT statement, BASIC prints another question mark and waits for more values to be input. When the correct number has been entered, execution continues. If two many values are input, BASIC ignores those in excess of the required number. The values are entered when the user types the RETURN key.

PTR

A PTR statement is used when data is to be input from the high-speed paper tape reader. The format of the data on the paper tape must be the same as it would be if it were input from the Teletype keyboard. If more than one value is to be input at a time, the values must be separated by commas. The tape must be positioned in the reader before it is called by the program; while it is reading, there is no echo (type out) on the Teletype. The form is:

(line number) PTR

The PTR statement is most useful for inputting large amounts of data in conjunction with the INPUT command. The following program accepts 20 data values from the high-speed reader, prints a heading, the value input, and its sine on the Teletype:

```
5Ø PTR
6Ø PRINT "SINE TABLE"
1ØØ FOR J=1 TO 2Ø
11Ø INPUT A
12Ø LET B=SIN(A)
13Ø PRINT A,B
14Ø NEXT J
15Ø END
```

```
RUN
SINE TABLE
-.97            -.8248857
-.911           -.79Ø1171
-.872           -.7656171
-.723           -.6616371
-.719           -.6586325
-.61            -.5728675
-.5Ø2           -.4811798
-.346           -.3391376
-.33            -.324Ø43
-.283           -.2792376
-.175           -.174Ø81
-.155           -.15438Ø1
-.Ø2            -.Ø1999867
 .Ø3             .Ø299955
 .Ø93            .Ø92866
 .127            .1266589
 .13             .1296341
 .42             .4Ø776Ø5
 .529            .5Ø467Ø3
 .632            .59Ø7596

READY.
```

PRINT

The PRINT statement is used to output results of computations, comments, values of variables, or plot points of a graph on the Teletype. The format is:

(line number) PRINT expression

When used without an expression, a blank line will be output on the Teletype. For more complicated formats, the type of expression and the type of format control characters following the word PRINT determines which formats will be created.

In order to have the computer print out the results of a computation, or the value of a variable at any point in the program, the

user types the line number, PRINT, and the variable name(s) separated by a format control character, in this case, commas:

```
5 A=16\B=5\C=4
10 PRINT A,C+B,SQR(A)
```

In BASIC, a Teletype line is formatted into five fixed columns (called print zones) of 14 spaces each. In the above example, the values of A, C+B, and the square root of A will be printed in the first three of these zones as follows:

```
RUN
 16            9             4

READY.
```

A statement such as:

```
5 A=2.3\B=21\C=156.75\D=1.134\E=23.4
10 PRINT A,B,C,D,E
```

will cause the values of the variables to be printed in the same format using all five columns:

```
RUN
 2.3           21            156.75        1.134         23.4

READY.
```

When more than five variables are listed in the PRINT statement, the sixth value begins a new line of output.

The PRINT statement may also be used to output a message or line of text. The desired message is simply placed in quotation marks in the PRINT statement as follows:

```
10 PRINT "THIS IS A TEST"
```

When line 10 is encountered during execution, the following will be printed:

```
THIS IS A TEST
```

A message may be combined with the result of a calculation or a variable as follows:

```
80 PRINT "AMOUNT PER PAYMENT ="R
```

Assuming R=344.9617, when line 80 is encountered during execution, this will be output as:

```
RUN
AMOUNT PER PAYMENT = 344.9617

READY.
```

It is not necessary to use the standard five zone format for output. The control character semicolon (;) causes the text or data to be output immediately after the last character printed (separated by one space.) If neither a comma nor a semicolon is used, BASIC assumes a semicolon. Thus both of the following:

```
80 PRINT "AMOUNT PER PAYMENT ="R
80 PRINT "AMOUNT PER PAYMENT =";R
```

will result in:

```
AMOUNT PER PAYMENT = 344.9617
```

The PRINT statement can also cause a constant to be printed on the console. (This is similar to the PRINT command used in Immediate Mode.) For example:

```
10 PRINT 1.234,SQR(10014)
```

will cause the following to be output at execution time:

```
1.234          100.07
```

Any algebraic expression in a PRINT statement will be evaluated using the current value of the variables. Numbers will be printed according to the format previously specified.

The following example program illustrates the use of the control characters[2] in PRINT statements:

[2] The user may wish to refer to the section entitled Functions for information pertaining to three functions available for additional character control—TAB, PUT, and GET.

```
10 READ A,B,C
20 PRINT A,B,C,A↑2,B↑2,C↑2
30 PRINT
40 PRINT A;B;C;A↑2;B↑2;C↑2
50 DATA 4,5,6
60 END
```

```
RUN
 4              5              6              16             25
 36

 4  5  6  16  25  36

READY.
```

As this example illustrates, if a number should be too long to be printed on the end of a single line, BASIC automatically moves the entire number to the beginning of the next line.

Another use of the PRINT statement is to combine it with an INPUT statement so as to identify the data expected to be entered. As an example, consider the following program:

```
10 REM - PROGRAM TO COMPUTE INTEREST PAYMENTS
20 PRINT "INTEREST IN PERCENT";
25 INPUT J
26 LET J=J/100
30 PRINT "AMOUNT OF LOAN";
35 INPUT A
40 PRINT "NUMBER OF YEARS";
45 INPUT N
50 PRINT "NUMBER OF PAYMENTS PER YEAR";
55 INPUT M
60 LET N=N*M
65 LET I=J/M
70 LET B=1+I
75 LET R=A*I/(1-1/B↑N)
78 PRINT
80 PRINT "AMOUNT PER PAYMENT =";R
85 PRINT "TOTAL INTEREST     =";R*N-A
88 PRINT
90 LET B=A
95 PRINT " INTEREST     APP TO PRIN     BALANCE"
100 LET L=B*I
110 LET P=R-L
120 LET B=B-P
130 PRINT L,P,B
140 IF B>=RGC TO 100
150 PRINT B*I,R-B*I
160 PRINT "LAST PAYMENT ="B*I+B
200 END
```

```
RUN
INTEREST IN PERCENT?9
AMOUNT OF LOAN?2500
NUMBER OF YEARS?2
NUMBER OF PAYMENTS PER YEAR?4

AMOUNT PER PAYMENT = 344.9617
TOTAL INTEREST     = 259.6932

 INTEREST       APP TO PRIN       BALANCE
 56.25           288.7117        2211.288
 49.75399        295.2077        1916.081
 43.11182        301.8498        1614.231
 36.32019        308.6415        1305.589
 29.37576        315.5859        990.0035
 22.27508        322.6866        667.317
 15.01463        329.947         337.3699
 7.590824        337.3708
LAST PAYMENT = 344.9608

READY.
```

As can be noticed in this example, the question mark is grammatically useful in a program in which several values are to be input by allowing the programmer to formulate a verbal question which the input value will answer.

LPT

The LPT statement is used to generate output on the LP08 line printer, and is of the form:

(line number) LPT

By inserting this statement anywhere in a program, all subsequent output, with the exception of error messages, will be printed on the line printer. The LPT statement is particularly advantageous for outputting large amounts of calculated data, as can be seen from this and following examples:

```
100 LPT
110 FOR F=30 TO 60 STEP 3
120 PRINT F,F↑2
130 NEXT F
140 END
```

```
30              900
33              1089
36              1296
39              1521
42              1764
45              2025
48              2304
51              2601
54              2916
57              3249
60              3600
```

When the END statement is encountered in the program, the output device is reset to the Teletype.

PTP

The high-speed paper tape punch is also available as an output device in 8K BASIC, permitting users to save data or output files quickly on paper tape. When the statement is encountered, all output is diverted from the Teletype to the high-speed punch. Control automatically returns to the Teletype when the END statement is encountered. The form is:

(line number) PTP

By substituting this statement in line 100 of the previous program, all output, with the exception of error messages, will be sent to the high-speed paper tape punch instead of the line printer.

TTY IN AND TTY OUT

The Teletype may be placed under program control so that, during execution of a program, I/O may be obtained or sent alternately between any available device. By issuing the statement:

(line number) TTY IN

control of input is returned to the Teletype if it has been previously set to another device. Similarly, the statement:

(line number) TTY OUT

returns output control to the Teletype.

The following program makes use of most all the available I/O devices. The output, with the exception of paper tape, is also included.

```
100 LPT
110 PRINT "FIRST DEGREE EQUATION CALCULATION"
120 TTY IN
130 TTY OUT
135 PRINT "TYPE X1 Y1 THEN X2 Y2"
140 INPUT X1,Y1,X2,Y2
150 X=X2-X1
160 Y=Y2-Y1
170 M=Y/X
180 B=Y2-M*X2
190 IF B>=0 THEN 300
200 PRINT "Y="M"X"B
210 LPT
220 PRINT "Y="M"X"B
230 GO TO 400
300 PRINT "Y="M"X+"B
310 LPT
320 PRINT "Y="M"X+"B
400 FOR Y=0 TO 10 STEP 2
410 FOR X=0 TO 10 STEP .5
420 LET T=M*X+B-Y
430 IF T<>0 THEN 480
440 PRINT X,Y
450 PTP
460 PRINT X,Y
470 LPT
480 NEXT X
490 NEXT Y
500 END

RUN
TYPE X1 Y1 THEN X2 Y2
?-3,-4,-1,0
Y= 2 X+ 2

READY.
```

The line printer output is the following:

```
FIRST DEGREE CALCULATION
Y= 2 X+ 2
 0            2
 1            4
 2            6
 3            8
 4            10
```

NOTE

The Teletype low-speed reader and punch may be used as I/O devices at any time. No special statement is required. To read in data from the low-speed reader, position the tape over the sprocket wheel and set the reader to START when input is required. The tape will begin reading in. To punch a tape, set the low-speed punch to ON and all ouput will be punched on the low-speed punch.

Using the low-speed I/O devices is, in effect, the same as using the Teletype keyboard. Characters will be typed on the Teletype keyboard as tapes are being read in or punched.

Loops

FOR, NEXT, AND STEP

FOR and NEXT statements define the beginning and end of a loop. A loop is a set of instructions which are repeated over and over again, each time being modified in some way until a terminal condition is reached. The FOR statement is of the form:

(line number) FOR v=x1 TO x2 STEP x3

where v represents a variable name, and x1, x2, and x3 all represent formulas (a formula in this case means a numerical value, variable name, or mathematical expression). v is termed the index, x1 the initial value, x2 the terminal value, and x3 the incremental value. For example:

```
15 FOR K=2 TO 20 STEP 2
```

This means that the loop will be repeated as long as K is less than or equal to 20. Each time through the loop, K is incremented by 2, so the loop will be repeated a total of 10 times.

A variable used as an index in a FOR statement must not be subscripted, although a common use of loops is to deal with subscripted variables, using the value of the index as the subscript of

a previously defined variable (this is illustrated in the section concerning Subscripted Variables).

The NEXT statement is of the form:

(line number) NEXT

and signals the end of the loop. When execution of the loop reaches the NEXT statement, the computer adds the STEP value to the index and checks to see if the index is less than or equal to the terminal value. If so, the loop is executed again. If the value of the index exceeds the terminal value, control falls through the loop to the following statement, with the value of the index equaling the value it was assigned the final time through the loop.[3]

If the STEP value is omitted, a value of +1 is assumed. Since +1 is the usual STEP value, that portion of the statement is frequently omitted. The STEP value may also be a negative number.

The following example illustrates the use of loops. This loop is executed 10 times: the value of I is 10 when control leaves the loop. +1 is the assumed STEP value.

```
10 FOR I=1 TO 10
20 NEXT I
30 PRINT I
40 END




RUN
 10

READY.
```

If line 10 had been:

```
10 FOR I=10 TO 1 STEP -1
```

the value printed by the computer would be 1.

As indicated earlier, the numbers used in the FOR statement

---

[3] The user should note that this method of handling loops varies among different versions of BASIC.

are formulas; these formulas are evaluated upon first encountering the loop. While the index, initial, terminal and STEP values may be changed within the loop, the value assigned to the initial formula remains as originally defined until the terminal condition is reached. To illustrate this point, consider the last example program. The value of I (in line 10) can be successfully changed as follows:

```
10 FOR I=1 TO 10
15 LET I=10
20 NEXT I
```

The loop will only be executed once since the value 10 has been reached by the variable I and the terminal condition is satisfied.

If the value of the counter variable is originally set equal to the terminal value, the loop will execute once, regardless of the STEP value. If the starting value is beyond the terminal value, the loop will also execute only once.

It is possible to exit from a FOR-NEXT loop without the index reaching the terminal value. (This is known as a conditional transfer and is explained in the section entitled Transfer of Control Statements.) Control may only transfer into a loop which has been left earlier without being completed, ensuring that the terminal and STEP values are assigned.

Nesting Loops

It is often useful to have one or more loops within a loop. This technique is called nesting, and is allowed as long as the field of one loop (the numbered lines from the FOR statement to the corresponding NEXT statement, inclusive) does not cross the field of another loop. A diagram is the best way to illustrate acceptable nesting procedures:

ACCEPTABLE NESTING TECHNIQUES	UNACCEPTABLE NESTING TECHNIQUES

Two Level Nesting

```
Acceptable:                 Unacceptable:
┌─FOR                       ┌FOR
│┌FOR                      ┌┼FOR
│└NEXT                     │└NEXT
│┌FOR                      └─NEXT
│└NEXT
└─NEXT
```

Three Level Nesting

```
Acceptable:                 Unacceptable:
┌──FOR                      ┌──FOR
│┌─FOR                      │┌─FOR
││┌FOR                      ││┌FOR
││└NEXT                     ││└NEXT
││┌FOR                      ││┌FOR
││└NEXT                     │└┼NEXT
│└─NEXT                     │ └NEXT
└──NEXT                     └──NEXT
```

A maximum of eight(8) levels of nesting is permitted. Exceeding that limit will result in the error message:

```
FOR ERROR AT LINE XXXX
```

where XXXX is the number of the line in which the error occurred.

Subscripted Variables

In addition to single variable names, BASIC accepts another class of variables called subscripted variables. Subscripted variables provide the programmer with additional computing capabilities for handling lists, tables, matrices, or any set of related variables. Variables are allowed one or two subscripts. A single letter forms the name of the variable; this is followed by one or two integers in parentheses and separated by commas, indicating the place of that variable in the list. Up to 26 arrays are possible in any program (corresponding to the letters of the alphabet), subject only to the amount of core space available for data storage. For example, a list might be described as A(I) where I goes from 1 to 5, as follows:

```
A(1),A(2),A(3),A(4),A(5)
```

This allows the programmer to reference each of the five elements in the list A. A two dimensional matrix A(I, J) can be defined in a similar manner, but the subscripted variable A can only be used once (i.e., A(I) and A(I,J) cannot be used in the same program). It is possible however, to use the same variable name as both a subscripted and an unsubscripted variable. Both A and A(I) are valid variable names and can be used in the same program.

Subscripted variables allow data to be input quickly and easily, as illustrated in the following program (the index of the FOR statement in lines 20, 42, and 44 is used as the subscript):

```
10 REM - PROGRAM DEMONSTRATING READING
11 REM - OF SUBSCRIPTED VARIABLES
15 DIM A(5),B(2,3)
18 PRINT "A(I) WHERE A=1 TO 5;"
20 FOR I=1 TO 5
25 READ A(I)
30 PRINT A(I);
35 NEXT I
38 PRINT
39 PRINT
40 PRINT "B(I,J) WHERE I=1 TO 2:"
41 PRINT "         AND  J=1 TO 3:"
42 FOR I=1 TO 2
43 PRINT
44 FOR J=1 TO 3
48 READ B(I,J)
50 PRINT B(I,J);
55 NEXT J
56 NEXT I
60 DATA 1,2,3,4,5,6,7,8
61 DATA 8,7,6,5,4,3,2,1
65 END


RUN
A(I) WHERE A=1 TO 5;
 1  2  3  4  5

B(I,J) WHERE I=1 TO 2:
         AND  J=1 TO 3:

 6  7  8
 8  7  6
READY.
```

DIM

From the preceding example, it can be seen that the use of subscripts requires a dimension (DIM) statement to define the maximum number of elements in the array. The DIM statement is of the form:

$$(\text{line number})\ \text{DIM}\ v_1\ (n_1),\ v_2\ (n_2,\ m_2)$$

where v indicates an array variable name and n and m are integer numbers indicating the largest subscript value required during the program. For example:

```
15 DIM A(6,10)
```

The first element of every array is automatically assumed to have a subscript of zero. Dimensioning A(6, 10) sets up room for an array with 7 rows and 11 columns. This matrix can be thought of as existing in the following form:

$$\begin{matrix} A_{0,0} & A_{0,1} & \cdots & A_{0,10} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,10} \\ A_{2,0} & A_{2,1} & \cdots & A_{2,10} \\ \vdots & \vdots & & \vdots \\ A_{6,0} & A_{6,1} & \cdots & A_{6,10} \end{matrix}$$

and is illustrated in the following program:

```
10 REM - MATRIX CHECK PROGRAM
15 DIM A(6,10)
20 FOR I=0 TO 6
22 LET A(I,0)=I
25 FOR J=0 TO 10
28 LET A(0,J)=J
30 PRINT A(I,J);
35 NEXT J
40 PRINT
45 NEXT I
50 END
```

```
RUN
 0  1  2  3  4  5  6  7  8  9  10
 1  0  0  0  0  0  0  0  0  0  0
 2  0  0  0  0  0  0  0  0  0  0
 3  0  0  0  0  0  0  0  0  0  0
 4  0  0  0  0  0  0  0  0  0  0
 5  0  0  0  0  0  0  0  0  0  0
 6  0  0  0  0  0  0  0  0  0  0

READY.
```

Notice that a variable assumes a value of zero until another value has been assigned. If the user wishes to conserve core space by not making use of the extra variables set up within the array, he should set his DIM statement to one less than necessary, DIM A(5,9). This results in a 6 by 10 array which may then be referenced beginning with the A(0, 0) element.

More than one array can be defined in a single DIM statement:

```
10 DIM A(20), B(4,7)
```

This dimensions both the list A and the matrix B.

A number must be used to define the maximum size of the array. A variable inside the parentheses is not acceptable and will result in an error message by BASIC at run time. The amount of user core not filled by the program will determine the amount of data the computer can accept as input to the program at any one time. In some programs a TOO-BIG ERROR may occur, indicating that core will not hold an array of the size requested. In that event,

the user should change his program to process part of the data in one run and the rest later.

Transfer of Control Statements

Certain control statements cause the execution of a program to jump to a different line either unconditionally or depending upon some condition within the program. Looping is one method of jumping to a designated point until a condition is met. The following statements give the programmer added capabilities in this area.

UNCONDITIONAL TRANSFER—GOTO

The GOTO (or GO TO) statement is an unconditional statement used to direct program control either forward or back in a program. The form of the GOTO statement is:

(line number) GOTO n

where n represents a statement number. When the logic of the program reaches the GOTO statement, the statement(s) immediately following will not be executed; instead execution is transferred to the statement beginning with the line number indicated.

The following program never ends; it does a READ, prints something, and jumps back to the READ via a GOTO statement. It attempts to do this over and over until it runs out of data, which is sometimes an acceptable, though not advisable, way to end a program.

```
10 REM - PROGRAM ENDING WITH ERROR
11 REM - MESSAGE WHEN OUT OF DATA
20 READ X
25 PRINT "X="X,"X↑2="X↑2
30 GO TO 20
35 DATA 1,5,10,15,20,25
40 END
```

```
RUN
X= 1            X↑2= 1
X= 5            X↑2= 25
X= 10           X↑2= 100
X= 15           X↑2= 225
X= 20           X↑2= 400
X= 25           X↑2= 625

DATA ERROR AT LINE 20
```

CONDITIONAL TRANSFER—IF-THEN AND IF-GOTO

If a program requires that two values be compared at some point, control of program execution may be directed to different procedures depending upon the result of the comparison. In computing, values are logically tested to see whether they are equal, greater than, less than another value, or possibly a combination of the three. This is accomplished by use of the relational operators discussed earlier.

IF-THEN and IF-GOTO statements allow the programmer to test the relationship between two formulas (variables, numbers, or expressions). Providing the relationship described in the IF statement is true at the point it is tested, control will transfer to the line number specified, or perform the indicated operation. The statements are of the form:

(line number) IF v1 <relation> v2 {GOTO | THEN} x or expression

where v1 and v2 represent variable names or expressions, x represents a line number, and expression represents an operation to be performed. The use of either THEN or GOTO is acceptable.

The following two examples are equivalent (the value of the variable A is changed or remains the same depending upon A's relation to B):

```
    •
    •
100 IF A>B THEN 120
110 A=A↑B-1
120 C=A/D
    •
    •
```

```
    •
    •
100 IF A<=B THEN A=A↑B-1
110 C=A/D
    •
    •
```

Subroutines

GOSUB AND RETURN

A subroutine is a section of code performing some operation that is required at more than one point in the program. Often a

complicated I/O operation for a volume of data, a mathematical evaluation which is too complex for a user-defined function, or any number of other processes may best be performed in a subroutine.

Subroutines are generally placed physically at the end of a program, usually before DATA statements, if any, and always before the END statement. Two statements are used exclusively in BASIC to handle subroutines; these are the GOSUB and RETURN statements.

A program begins execution and continues until it encounters a GOSUB statement of the form:

(line number) GOSUB x

where x represents the first line number of the subroutine. Control then transfers to that line. For example:

```
50 GOSUB 200
```

When program execution reaches line 50, control transfers to line 200; the subroutine is processed until execution encounters a RETURN statement of the form:

(line number) RETURN

which causes control to return to the statement following the GOSUB statement. Before transferring to the subroutine, BASIC internally records the next statement to be processed after the GOSUB statement; thus the RETURN statement is a signal to transfer control to this statement. In this way, no matter how many different subroutines are called, or how many times they are used, BASIC always knows where to go next.

The following program demonstrates a simple subroutine:

```
1 REM - THIS PROGRAM ILLUSTRATES GOSUB AND RETURN
1Ø DEF FNA(X)=ABS(INT(X))
2Ø INPUT A,B,C
3Ø GOSUB 1ØØ
4Ø LET A=FNA(A)
5Ø LET B=FNA(B)
6Ø LET C=FNA(C)
7Ø PRINT
8Ø GOSUB 1ØØ
9Ø STOP
1ØØ REM - THIS SUBROUTINE  PRINTS OUT THE SOLUTIONS
11Ø REM - OF THE EQUATION: A(X↑2) + B(X) + C = Ø
12Ø PRINT "THE EQUATION IS   "A"*X↑2  + "B"*X  + "C
13Ø LET D=B*B-4*A*C
14Ø IF D<>Ø THEN 17Ø
15Ø PRINT "ONLY ONE SOLUTION... X ="-B/(2*A)
16Ø RETURN
17Ø IF D<Ø THEN 2ØØ
18Ø PRINT "TWO SOLUTIONS... X =";
185 PRINT (-B+SQR(D))/(2*A)"AND X ="(-B-SQR(D))/(2*A)
19Ø RETURN
2ØØ PRINT "IMAGINARY SOLUTIONS... X = (";
2Ø5 PRINT -B/(2*A)","SQR(-D)/(2*A)") AND  (";
2Ø7 PRINT -B/(2*A)","-SQR(-D)/(2*A)")"
21Ø RETURN
9ØØ END
```

```
RUN
?1,.5,-.5
THE EQUATION IS     1 *X↑2   +   .5 *X  +  -.5
TWO SOLUTIONS... X = .5 AND X =-1

THE EQUATION IS     1 *X↑2   +   Ø *X   +   1
IMAGINARY SOLUTIONS... X = ( Ø , 1 )  AND ( Ø ,-1 )

READY.
```

Line 100 begins the subroutine. There are several places in which control may return to the main program, depending upon a certain condition being satisfied. The subroutine is executed from line 30 and again from line 80. When control returns to line 90, the program encounters the STOP statement and execution is terminated.

It is important to remember that subroutines should generally be kept distinct from the main program. The last statement in the main program should be a STOP or GOTO statement, and subroutines are normally placed following this statement.

More than one subroutine may be used in a single program, in which case these can be placed one after another at the end of the program (in line number sequence). A useful practice is to assign distinctive line numbers to subroutines. For example, if the main program is numbered with line numbers up to 199, 200 and 300 could be used as the first numbers of two subroutines.

Nesting Subroutines

Nesting of subroutines occurs when one subroutine calls another subroutine. If a RETURN statement is encountered during execution of a subroutine, control returns to the statement following the GOSUB which called it. From this point, it is possible to transfer to the beginning or any part of a subroutine, even back to the calling subroutine. Multiple entry points and RETURN statements make subroutines more versatile.

The maximum level of GOSUB nesting is about thirty-three (33) levels, which should prove more than adequate for all normal uses. Exceeding this limit will result in the message:

```
GOSUB ERROR AT LINE XXXX
```

where XXXX represents the line number where the error occurred. An example of GOSUB nesting follows (execution has been stopped by typing a CTRL/C, as the program would otherwise continue in an infinite loop; see Stopping a Run.)

```
10 REM FACTORIAL PROGRAM USING GOSUB TO
15 REM RECURSIVELY COMPUTE THE FACTORS
40 INPUT N
50 IF N>20 THEN 120
60 X=1
70 K=1
80 GOSUB 200
90 PRINT "FACTORIAL"N" ="X
110 GO TO 40
120 PRINT "MUST BE 20 OR LESS"
130 GO TO 40
200 X=X*K
210 K=K+1
220 IF K<=N THEN GOSUB 200
230 RETURN
240 END
```

```
RUN
?2
FACTORIAL 2  = 2
?4
FACTORIAL 4  = 24
?5
FACTORIAL 5  = 120
?
STOP.
READY.
```

Functions

BASIC performs several mathematical calculations for the programmer, eliminating the need for tables of trig functions, square roots, and logarithms. These functions have a three letter call name, followed by an argument, x, which can be a number, variable, expression, or another function. Table 1 lists the functions available in 8K BASIC. Most are self-explanatory; those that are not and are provided in greater detail are marked with asterisks.

Table 1 8K BASIC Functions

Function	Meaning
SIN(x)	Sine of x (x is expressed in radians)
COS(x)	Cosine of x (x is expressed in radians)

Table 1 8K BASIC Functions (Cont.)

Function	Meaning
TAN(x)	Tangent of x (x is expressed in radians)
ATN(x)	Arctangent of x (result is expressed in radians)
EXP(x)	e^x (e=2.718282)
LOG(x)	Natural log of x ($\log_e x$)
*SGN(x)	Sign of x—assign a value of +1 if x is positive, 0 if x is zero, or −1 if x is negative
*INT(x)	Integer value of x
ABS(x)	Absolute value of x ($\|x\|$)
SQR(x)	Square root of x ($\sqrt{x}$)
*RND(x)	Random number
*TAB(x)	Print next character at space x
*GET(x)	Get a character from input device
*PUT(x)	Put a character on output device
*FNA(x)	User-defined function
*UUF(x)	User-coded function (machine language code)

SIGN FUNCTION—SGN(X)

The sign function returns the value +1 if x is a positive value, 0 if x is zero, and −1 if x is negative. For example, SGN(3.42)=1, SGN(−42)=−1, and SGN(23−23)=0. The following example in which X is assigned the sign of y illustrates the use of this function:

```
25 X=SQR(A↑2+2*BC)*SGN(A)
```

INTEGER FUNCTION—INT(X)

The integer function returns the value of the nearest integer not greater than x. For example, INT(34.67)=34. By specifying

INT(x+.5) the INT function can be used to round numbers to the nearest integer; thus, INT(34.67+.5)=35. INT can also be used to round numbers to any given decimal place by specifying:

INT (x*10↑D+.5)/10↑D

where D is the number of decimal places desired. The following program illustrates this function; execution has been stopped by typing a CTRL/C:

```
10 REM - INT FUNCTION EXAMPLE
20 PRINT "NUMBER TO BE ROUNDED";
30 INPUT A
40 PRINT "NO. OF DECIMAL PLACES";
50 INPUT D
60 LET B=INT(A*10↑D+.5)/10↑D
70 PRINT "A ROUNDED = "B
80 GO TO 20
90 END
```

```
RUN
NUMBER TO BE ROUNDED?55.65342
NO. OF DECIMAL PLACES?2
A ROUNDED =  55.65
NUMBER TO BE ROUNDED?78.375
NO. OF DECIMAL PLACES?-2
A ROUNDED =  100
NUMBER TO BE ROUNDED?67.89
NO. OF DECIMAL PLACES?-1
A ROUNDED =  70
NUMBER TO BE ROUNDED?
STOP.
READY.
```

If the argument is a negative number, the value returned is the largest negative integer (rounded to the higher value) contained in the number. For example, INT(−23)=−23 but INT(−14.39)=−15.

RANDOM NUMBER FUNCTION—RND(X)

The random number function produces a random number between 0 and 1. The numbers are not reproducible, a fact the programmer should keep in mind when debugging or checking his

program. The argument x in the RND(x) function call can be any number, as that value is ignored. The following program illustrates the use of this function to generate a table of random numbers:

```
1Ø REM - RANDOM NUMBER EXAMPLE
25 PRINT "RANDOM NUMBERS"
3Ø FOR I=1 TO 3Ø
4Ø PRINT RND(Ø),
5Ø NEXT I
6Ø END
```

```
RUN
RANDOM NUMBERS
 .95476Ø9       .289Ø875       .1416765       .2482717       .2145417
 .Ø528Ø478      .3859534       .84Ø4774       .5692836       .8514Ø56
 .98488Ø8       .2466345       .61588         .4755698       .31Ø4984
 .5828625       .7Ø26891       .97Ø3719       .498Ø298       .2548316
 .Ø4672124      .9868434       .5ØØ5693       .1218251       .2258269
 .2585353       .51877Ø1       .7858Ø24       .Ø4588368      .2Ø3Ø8Ø7
READY.
```

It is possible to generate random numbers over any range by using the following formula:

(B−A)*RND(0)+A

This produces a random number (n) in the range A<n<B.

In order to obtain random integer digits in the range 0< = n<9, line 40 in the previous example is changed to read:

```
4Ø PRINT INT(9*RND(Ø)),
```

When the program is run again, the results will look as follows:

```
RANDOM NUMBERS
 8              8              3              Ø              Ø
 3              Ø              1              Ø              4
 8              3              1              4              6
 2              2              Ø              6              5
 7              6              7              7              6
 2              Ø              2              8              6
READY.
```

Notice that the range has changed to 0< = n<9. This is because the INT function returns the value of the nearest integer not greater than n.

TAB FUNCTION

The TAB function allows the user to position the printing of characters anywhere on the Teletype (or line printer) line. Print

positions can be thought of as being numbered from 1 to 72 across the Teletype from left to right. (For printing devices with long lines, the number of positions may be as large as 255, but it is unlikely that more than 160 spaces will be required for most printers.) The form of this function is:

TAB (n)

where the argument n represents the position (from 1 to the total number of spaces available) in which the next character will be typed.

Each time the TAB function is used, positions are counted from the beginning of the line, not from the current position of the printing head. For example, TAB(3) causes the character to be printed at position 3; the following statement:

```
10 PRINT "X =";TAB(3);"/";3.14159
```

will print the slash on top of the equal sign, as shown below:

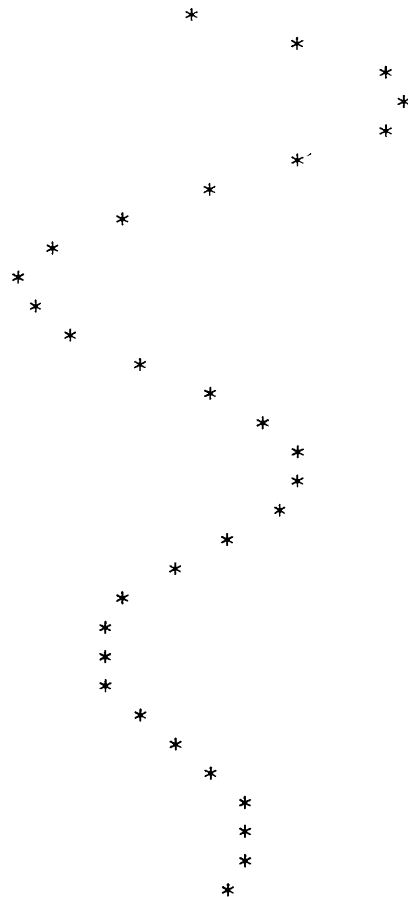
```
X ≠ 3.14159

READY.
```

The following is an example of the sort of graph that can be drawn with BASIC using the TAB function:

```
30 FOR X=0 TO 15 STEP .5
40 PRINT TAB(30+15*SIN(X)*EXP(-.1*X));"*"
50 NEXT X
60 END


RUN
```



```
READY.
```

PUT AND GET FUNCTIONS

8K BASIC provides two additional functions, PUT and GET, to increase input/output flexibility. Using these statements, the programmer can "PUT" an ASCII character on the current output device, or "GET" a character from the current input device. GET is of the form:

GET (x)

where the argument x is a dummy variable which may be any value. GET (x) will be assigned the decimal value of the ASCII code of the next character input on the current input device.

For example, if the following statement appears in a program:

```
10 LET L=GET(X)
```

and the next character input is an M, the variable L will be assigned the value $77_{(10)}$.

PUT is of the form:

PUT (x)

where the argument x represents the decimal value of the ASCII code of the character to be output. For example, the statement:

```
15 L=PUT(GET(V))
```

will wait for a character to be read from the current input device and then print it on the current output device. A statement such as:

```
30 PRINT PUT(Q)
```

will print the character typed as well as the decimal value of the ASCII code for that character. To get 10 characters from a paper tape and print them on the line printer, a suitable program is:

```
100 LPT
110 PTR
120 FOR A=1 TO 10
130 LET B=PUT(GET(0))
140 NEXT A
150 END
```

The GET(0) will contain the most recently obtained character which is then "PUT" to the line printer. The user should be careful to position the tape on the first character to be input. Otherwise

blank tape may be entered, resulting in spaces being printed as output.

The PUT statement can also be used to format output. For example, to print a trig table on the line printer with a heading and 50 data lines per page, the line feed character ($12_{(10)}$) can be "PUT" to the printer as follows:

```
100 LPT
110 GOSUB 1000
120 GOSUB 500
125 REM - SET UP TRIG TABLE
130 FOR J=0 TO 360 STEP .5
140 LET L=L+1
150 LET B=J/180*3.14
160 PRINT J,SIN(B),COS(B),TAN(B),ATN(B)
165 REM - PRINT 50 ENTRIES IN TABLE
170 IF L=50 THEN GOSUB 500
180 NEXT J
190 GOSUB 1000
200 GOSUB 1000
210 STOP
500 REM PRINT HEADER
505 GOSUB 1000
510 PRINT
520 PRINT
530 PRINT "ANGLE","SINE","COSINE","TANGENT","ARCTANGENT"
540 PRINT
550 RETURN
1000 REM PRINT FORM FEEDS TO ADVANCE PAPER
1005 X=PUT(12)
1010 L=0
1020 RETURN
1030 END
```

The beginning of the line printer output from this program follows. The first page of the table continues through an angle of 24.5 degrees: then the header and the next 50 entries are printed on the next page, and so on until the values have been output for all angles through 360 degrees (in steps of .5).

ANGLE	SINE	COSINE	TANGENT	ARCTANGENT
Ø	Ø	1	Ø	Ø
.5	8.722112E-Ø3	.999962	8.722444E-Ø3	8.722ØØ1E-Ø3
1	.Ø1744356	.9998479	.Ø1744621	.Ø1744268
1.5	.Ø2616368	.9996577	.Ø2617264	.Ø261607
2	.Ø3488181	.9993915	.Ø349Ø3Ø5	.Ø3487474
2.5	.Ø4359729	.999Ø492	.Ø4363878	.Ø435835
3	.Ø523Ø945	.99863Ø9	.Ø5238116	.Ø5228564
3.5	.Ø61Ø1763	.9981367	.Ø6113154	.Ø6Ø97986
4	.Ø6972117	.9975665	.Ø6989125	.Ø6966486
4.5	.Ø784194	.99692Ø5	.Ø7866164	.Ø7833935
5	.Ø8711167	.9961986	.Ø8744408	.Ø87ØØ2Ø4
5.5	.Ø9579731	.99540Ø9	.Ø9623993	.Ø9565166
6	.1Ø44757	.9945274	.1Ø5Ø5Ø6	.1Ø42869
6.5	.1131461	.9935784	.1138774	.1129Ø67
7	.1218Ø79	.9925537	.1227217	.1215Ø95
7.5	.13Ø46Ø4	.9914535	.131585	.13ØØ944
.				
.				
.				
24	.4Ø65426	.9136318	.4449743	.396494
24.5	.414496	.91ØØ512	.4554645	.4Ø38923

The GET statement cannot be used to get binary characters.

FNA FUNCTION

In some programs it may be necessary to execute the same mathematical formula in several different places. 8K BASIC allows the programmer to define his own function in the BASIC language and then call this function in the same manner as the square root or a trig function is called. Only one such user-defined function may be included per program. The function is defined once at the beginning of the program before its first use, and consists of a DEF statement in combination with a three-letter function name, the first two letters of which must be FN. The format of the defining statement is as follows:

(line number) DEF FNA(x)=formula(x)

A may be any letter. The argument (x) has no significance; it is strictly a dummy variable, but must be the same on each side of the equal sign. The function itself can be defined in terms of numbers, several variables, other functions, or mathematical expressions. For example:

```
1Ø DEF FNA(X)=X↑2+3*X+4
```

or

```
2Ø DEF FNC(X)=SQR(X+4)+1
```

The function:

```
10 DEF FNA(S)=S↑2
```

will cause the later statement:

```
20 LET R=FNA(4)+1
```

to be evaluated as R=17.

The user-defined function can be a function of only one variable.

USER-DEFINED FUNCTION-UUF

A special user-coded function is available for the programmer who wishes to define an additional 8K BASIC function permanently or one which cannot be defined with one BASIC expression, as an FNA function must be. The UUF function routine is coded in PDP-8 assembly language, assembled with one of the available assemblers, and loaded as an overlay to 8K BASIC. While 8K BASIC is running, the special function can be used in a fashion analogous to the regular 8K BASIC functions. The user-coded function, if present, is referenced in the BASIC program as:

UUF(n)

where n can be any BASIC expression.

The programmer who defines the UUF function should be familiar with the information on assembly language programming which is in *Introduction to Programming 1972* chapters 1-5, and the material on the Floating Point Package, chapter 8. He should also be familiar with the information on the assembler he intends to use by reading the appropriate manual.

Coding Formats

8K BASIC uses a floating point package which has been modified to allow 27-bit, sign-magnitude mantissa floating point. In sign-magnitude convention the sign bit, rather than the mantissa, expresses the sign of the entire number. This format is described more fully below. All coding must be compatible with this format. The floating point instructions are discussed later in this manual.

Upon entrance to the UUF subroutine the value of the argument is in the FAC (floating accumulator). The value which is calculated for the function must be in the FAC in normalized form on exit.

When floating point statements are to be included in the program, the start of a series of floating point instructions must be indicated by the instruction:

FENTER

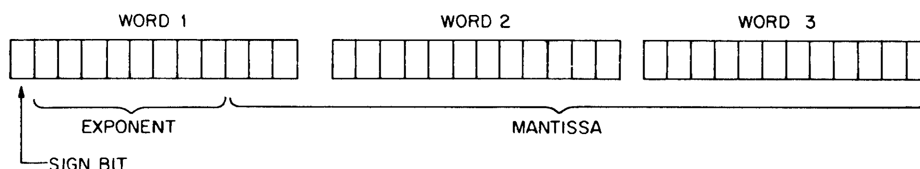
immediately before the first floating point instructions. Each series of floating point instructions is terminated by the instruction:

FEXIT

immediately after the last floating point statement. There can be as many sections of floating point code as necessary in the program, but each must be delimited in this manner.

Floating-Point Format

The floating-point format used by 8K BASIC allocates three storage words to each number as follows:



The FAC occupies five locations on page 0:

Location Name	Location Number	Contents
ACS	0024	Sign
ACE	0025	Exponent (200_8 biased)
AC1	0020	High-order word
AC2	0017	Mid-order word
AC3	0016	Low-order word

The constant 200_8 is added to the exponent to make its range 0 to 377.

All of BASIC's mathematical operations are in floating point format. Therefore, if any temporary storage locations are to be used, they will require three words, for example:

```
UTEMP,0;0;0
```

Addressing

The floating point package uses only relative addressing. Therefore all statements that require an address specification must include one of the operators FWD or BKWD plus a reference to the current location. Such a reference is generally of the form:

instruction+FWD+LTEMP–.

or

instruction+BKWD+.–LTEMP

where LTEMP is the first of the three locations containing the number to be used. The operator FWD is used when the address of the location to be referenced is numerically greater than the address of the instruction; BKWD is used when the address of the location to be referenced is numerically less than the address of the instruction. The floating point interpreter uses the number of locations between the instruction and the data to locate the data. The location referenced must be within 200_8 locations of the instruction.

The following two examples cause the contents of LTEMP to be added to the contents of the FAC, and the result left in the FAC:

```
00200 4210              FAD+FWD+LTEMP-.
  .
  .
00210 0000 LTEMP        0
00211 0000              0
00212 0000              0
```

or

```
00200 0000 LTEMP        0
00201 0000              0
00202 0000              0
  .
  .
00210 4610              FAD+BKWD+.-LTEMP
```

Floating-Point Instruction Set

The legal instructions in the modified Floating-Point Package used by 8K BASIC are explained in Table 2:

Table 2 Floating-Point Instructions

Instruction	Value	Meaning
FST	2000	Store the contents of the floating accumulator (FAC). The contents of the FAC are not changed.
FLD	3000	Load FAC with contents of relative address.
FAD	4000	Add contents of relative address to FAC.
FSB	5000	Subtract contents of relative address from FAC.
FMP	6000	Multiply the contents of the FAC by the contents of the relative address.
FDV	7000	Divide FAC by contents of relative address.
FJMP	1000	Floating-point jump to relative address.
FENTER	4435	Start floating-point code.
FEXIT	0000	Exit floating-point code. Return to PDP-8 code.
FWD	0200	Access a relative location in the forward direction.
BKWD	0600	Access a relative location in the backward direction.
FSNE	0040	Skip if FAC $\neq 0$
FSEQ	0050	Skip if FAC $= 0$
FSGE	0100	Skip if FAC $\geqslant 0$
FSLT	0110	Skip if FAC < 0
FSGT	0140	Skip if FAC > 0
FSLE	0150	Skip if FAC $\leqslant 0$

The following list contains floating-point instructions for indirect relative addressing. The indirect addressing is similar to the I construction used in regular PDP-8 assembly language coding.

Floating-Point Instructions (Indirect Relative Addressing)

Instruction	Value	Operation
FSTI	2400	Store
FLDI	3400	Load
FADI	4400	Add
FSBI	5400	Subtract
FMPI	6400	Multiply
FDVI	7400	Divide
FJMPI	1400	Jump

Writing the Program

UUF must be made a defined function for 8K BASIC. This is done by inserting the starting address of the UUF subroutine in BASIC's table of subroutine addresses. The subroutine address must be placed in location 1156 of field 0. If UUF is the first location of the subroutine, the following code is sufficient:

```
*1156
        UUF
```

The UUF subroutine may be placed in the area of core normally occupied by the RIM and BIN loaders, location 7600-7777 of field 0. To do this, the loaders are placed in field 1. The loading instructions for UUF are contained in the section called Loading and Operating Procedures.

If mass storage devices are in use, they may destroy the data break locations on the last page of field 0. If TC08 DECtape is used, locations 7752 and 7753 must be reserved. If an RF08 or DF32 disk is used, locations 7750 and 7751 must be reserved.

There are three subroutines in 8K BASIC which are available to maintain a floating-point format acceptable to the modified floating-point package in 8K BASIC. These subroutines are described below. The listing of 8K BASIC is available from the Software Distribution Center for the programmer who wishes to call other subroutines in the compiler.

BEGFIX — If a value is to be returned to the FAC as a result of the UUF function, that value must be in normalized floating point format in the FAC on exit from the subroutine. If floating point arithmetic is used throughout the user function, then the value in the FAC is in normalized floating point format and need not be converted. If fixed point arithmetic (single word) is used anywhere in the function, then the subroutine BEGFIX must be called to initialize the FAC before the fixed point number is placed in the FAC and subsequently converted to floating point (see ANORM below). After BEGFIX is called, the 12-bit number is stored by a simple DCA AC3 instruction and then ANORM is called. BEGFIX is located at 3762 and is called with a JMS instruction; on return from BEGFIX the AC is clear.

ANORM — If a fixed point value is placed in the FAC, ANORM may be called to normalize the FAC. After the fixed point value has been placed in AC3, ANORM may be called to supply the acceptable values for ACE, ACS, AC1, and AC2. ANORM is located at 4600; on return, the AC is clear.

FIX — When the value in the FAC must be made into an integer, FIX may be called to perfom that job. The 12-bit value of the FAC is left in AC3 and that value plus 1 is left in the AC. FIX is located at 4744.

Examples

The following examples illustrate the method of writing and calling a UUF routine.

Example 1:

This UUF routine is an example of a fixed point calculation. The value of UUF(X) is 3X+2.

```
                /UUF(X)=3X+2
                /ENTER WITH X IN FAC
                /EXIT WITH UUF(X) IN FAC
                /USE FIXED POINT ARITHMETIC

        0016    AC3=16
        4744    FIX=4744
        3762    BEGFIX=3762
        4600    ANORM=4600

        0000    FIELD 0

        1156    *1156
01156   7600            UUF

        7600    *7600                                 which is in FAC
07600   0000    UUF,    0
07601   4623            JMS I IFIX          /MAKE X A 12-BIT INTEGER.
07602   7200            CLA                 / Set AC=0
07603   3222            DCA ANSWER          / Set ANSWER=0
07604   4215            JMS LOOP            /MULTIPLY X BY 3
07605   4215            JMS LOOP
07606   4215            JMS LOOP
07607   4624            JMS I IBEG
07610   7326            CLA CLL CML RTL     /SET AC=2
07611   1222            TAD ANSWER          /FETCH 3X
07612   3016            DCA AC3             /RETURN 3X+2 TO FAC
07613   4625            JMS I INORM         /NORMALIZE
07614   5600            JMP I UUF           /--RETURN--

07615   0000    LOOP,   0
07616   1222            TAD ANSWER          /ADD AC3 TO ACCUMULATED SUM
07617   1016            TAD AC3
07620   3222            DCA ANSWER
07621   5615            JMP I LOOP          /--RETURN--

07622   0000    ANSWER, 0
07623   4744    IFIX,   FIX
07624   3762    IBEG,   BEGFIX
07625   4600    INORM,  ANORM

                $
```

The following BASIC program calls UUF(X) to print X and 3X+2 for a number of values of X:

```
READY.


100 FOR X=-3 TO 3 STEP .5
110 PRINT X, UUF(X)
120 NEXT X
130 END
```

```
RUN
-3              11
-2.5            11
-2              8
-1.5            8
-1              5
-.5             5
 0              2
 .5             2
 1              5
 1.5            5
 2              8
 2.5            8
 3              11

READY.
```

Example 2:

This UUF routine is an example of a floating point calculation. Like example 1, this routine returns a value of UUF(X)=3X+2.

```
/UUF(X)=3X+2                                PAL8-V7  5/25/72   PAGE 1

                /UUF(X)=3X+2
                /ENTER WITH X IN FAC
                /EXIT WITH UUF(X) IN FAC
                /USE FLOATING POINT ARITHMETIC

        4435    FENTER=4435
        2000    FST=2000
        0200    FWD=200
        0000    FEXIT=0000
        6000    FMP=6000
        4000    FAD=4000
        0016    AC3=16
        3762    BEGFIX=3762
        4600    ANORM=4600

        0000    FIELD 0

        1156    *1156
01156   7600            UUF

        7600    *7600
07600   0000    UUF,    0
07601   4435            FENTER
07602   2216            FST+FWD+X-.  /STORE X FROM FAC INTO LOC. X
07603   0000            FEXIT
07604   7325            CLA CLL CML IAC RAL     /SET AC=3
07605   4224            JMS FLOAT /GET A FLOATING POINT 3 IN THE FAC
07606   4435            FENTER
07607   6211            FMP+FWD+X-.      /MULTIPLY X BY 3
07610   2210            FST+FWD+X-.      /SAVE IT FOR LATER
07611   0000            FEXIT
07612   7326            CLA CLL CML RTL /GET A 2 IN THE FAC.
07613   4224            JMS FLOAT
```

```
07614  4435           FENTER
07615  4203           FAD+FWD+X-,    /ADD 2 TO 3X
07616  0000           FEXIT          /LEAVE RESULT IN FAC
07617  5600           JMP I UUF      /--RETURN--


07620  0000  X,       0;0;0
07621  0000
07622  0000

07623  0000  TEMP,    0

07624  0000  FLOAT,   0
07625  3223           DCA TEMP       /STORE CONSTANT TEMPORARILY
07626  4633           JMS I IBEG     /PREPARE FAC TO RECEIVE VALUE
07627  1223           TAD TEMP
07630  3016           DCA AC3        /PUT CONSTANT IN FAC
07631  4634           JMS I INORM    /NORMALIZE IT
07632  5624           JMP I FLOAT    /--RETURN--

07633  3762  IBEG,    BEGFIX
07634  4600  INORM,   ANORM

/UUF(X)=3X+2                         PAL8-V7   5/25/72   PAGE 1-1

              $
```

The following BASIC program calls UUF(X) to print X and 3X+2 for a number of values of X. The results differ from those in example 1 because of the capability of floating point arithmetic to handle fractions.

```
READY.

100 FOR X=-3 TO 3 STEP .5
110 PRINT X, UUF(X)
120 NEXT X
130 END

RUN
-3            -7
-2.5          -5.5
-2            -4
-1.5          -2.5
-1            -1
-.5            .5
 0             2
 .5            3.5
 1             5
 1.5           6.5
 2             8
 2.5           9.5
 3             11

READY.
```

Example 3:

This UUF routine computes the square of the argument in floating point format.

```
/UUF(X)=X^2                              PAL8-V7  5/25/72   PAGE 1

               /UUF(X)=X^2
               /ENTER WITH X IN FAC
               /EXIT WITH UUF(X) IN FAC
               /USE FLOATING POINT ARITHMETIC

       4435    FENTER=4435
       2000    FST=2000
       0200    FWD=200
       6000    FMP=6000
       0000    FEXIT=0000

       0000    FIELD 0

       1156    *1156
01156  7700            UUF

       7700    *7700
07700  0000    UUF,    0
07701  4435            FENTER
07702  2204            FST+FWD+X-.    /STORE ARGUMENT IN X
07703  6203            FMP+FWD+X-.    /MULTIPLY FAC BY LOC. X (X*X)
07704  0000            FEXIT          /RESULT IS IN FAC
07705  5700            JMP I UUF      /--RETURN--

07706  0000    X,      0;0;0
07707  0000
07710  0000
```

The following BASIC program uses the above UUF to produce a table of squares and square roots:

```
READY.

100 FOR A=1 TO 10 STEP 1
110 PRINT A, UUF(A), SQR(A)
120 NEXT A
130 END

RUN
 1             1             1
 2             4             1.414214
 3             9             1.732051
 4             16            2
 5             25            2.236068
 6             36            2.44949
 7             49            2.645751
 8             64            2.828427
 9             81            3
 10            100           3.162278

READY.
```

EDITING AND CONTROL COMMANDS

Errors made while typing at the console keyboard are easily corrected. BASIC provides special commands to facilitate the editing procedure.

Erasing Characters and Lines

SHIFT/O, RUBOUTS, NO RUBOUTS

There are two methods available for erasing a character or series of characters one at a time. Typing a SHIFT/O causes the deletion of the last character typed, and echoes as a back arrow (←) on the Teletype. One character is deleted each time the key is typed.

The RUBOUT key may also be used for deletion of characters one at a time providing the command:

```
RUBOUTS
```

has been typed on the keyboard before the editing is done. This command enables the RUBOUT key to be used. If the user has neglected to type this command, he may not use the RUBOUT key. A later command of:

```
NO RUBOUTS
```

disables the key for use. (This is desirable when programs created on other systems which use rubouts as null characters are to be read into core. See the section entitled PTP AND LPT under Listing and Punching a Program.) For example:

```
10 LEB←T A=10*B
```

The user types a B instead of T and immediately notices the mistake. He may type SHIFT/O (or RUBOUT key, if enabled) once to delete the B, and as many times more as characters, including spaces, are to be deleted. After the correction is made, he may continue typing the line. The typed line enters the computer only when the RETURN key is pressed. Before that time any number of corrections can be made to the line.

```
20 DEN F←←←F FNA(X,Y)=X↑2+3*Y
```

When the RETURN key is typed, the line is input as:

```
20 DEF FNA(X,Y)=X↑2+3*Y
```

Notice that spaces, as well as printing characters, may be erased.

The user may erase an entire line (provided the RETURN key has not been typed) by typing the ALTMODE key (ESCAPE key on some keyboards). BASIC echos back:

```
DELETED
```

at the end of the line to indicate that the line has been removed. The user continues as though it were a new line. If the RETURN key has already been typed, the user may still correct the line by simply typing the line number and retyping the line correctly. He may delete the line by typing the RETURN key immediately after the line number, thus removing both the line number and line from his program.

If the line number of a line not needing correction is accidentally typed, the SHIFT/O or RUBOUT key may be used to delete the number(s); the user may then type in the correct numbers. Assume the line:

```
10 IF A>5 GO TO 230
```

is correct. The programmer intends to insert a line 15, but instead types:

```
10 LET
```

He notices the mistake and makes the correction as follows:

```
10 LET←←←←←5 LET X=X-3
```

Line 10 remains unchanged, and line 15 is entered.

Following an attempt to run a program, error messages may be

output on the Teletype indicating illegal characters or formats, or other user errors in the program. Most errors can be corrected by typing the line number(s) and the correction(s) and then re-running the program. As many changes or corrections as desired may be made before runs.

Listing and Punching a Program

LIST

An indirect program or data can be listed on the active output device by typing the command:

```
LIST
```

followed by the RETURN key. The entire program (or data) will be listed.

A part of a program may be listed by typing LIST followed by a line number. This causes that line and all following lines in the program to be listed. For example:

```
LIST 1ØØ
```

will list line 100 and all remaining lines in the program.

PTP AND LPT

The LIST command may be issued in conjunction with the LPT or PTP commands as follows:

```
PTP             LPT
LIST            LIST
```

This will list the current program on the high-speed paper tape punch or line printer respectively. Control is reset to the Teletype after the listing is completed.

Occasionally, when 8K BASIC is reading in a program from the low-speed reader, it may drop a character since the Teletype buffer cannot accept input at a prolonged fast rate. To eliminate this possibility, use LIST as follows when punching out paper tapes:

```
PTP
LIST*
```

This inserts null characters after carriage returns and is recommended when punching any tapes that will later be read in from the low-speed paper tape reader. (8K BASIC does not use rubouts as null characters.)

Reading a Program

PTR

The PTR command can be issued to read in a paper tape from the high-speed reader. This mode is particularly useful for reading in a user-coded "load and go" BASIC program. The tape should be positioned in the reader before the command is issued; if not, or if the reader runs out of tape, BASIC prints:

```
TTY
```

on the Teletype to indicate that there is no more input from the high-speed reader, and that it is waiting for input from the Teletype.

The user may cause tapes to be read in from the low-speed reader by simply placing the tape over the sprocket wheel and setting the reader to START.

Running a Program

RUN

After a BASIC program has been typed and is in core, it is ready to be run. This is accomplished by simply typing the command:

```
RUN
```

followed by the RETURN key. The program will begin execution. If errors are encountered, appropriate error messages will be typed on the keyboard; otherwise, the program will run to completion, printing whatever output was requested. When the END statement is reached, BASIC stops execution and prints:

```
READY.
```

PTP AND LPT

Either the high-speed paper tape punch or LP08 line printer, if available, can be used in conjunction with the RUN command. After the command is issued, all output during program execution is diverted from the Teletype to the specified device. The command sequence is:

```
PTP          LPT
RUN          RUN
```

This procedure eliminates the need to insert the PTP (or LPT) statement within the program. Output returns to the Teletype after execution.

Stopping a Run

CTRL/C

To stop a program during execution or to return to BASIC at any time, type a CTRL/C (accomplished by typing the CTRL key and the C simultaneously). This causes the current operation to be aborted immediately, and the message:

```
STOP.
READY.
```

to be printed indicating that an 8K BASIC command can now be issued.

CTRL/O

The command CTRL/O (caused by typing the CTRL and O keys down simultaneously) is used to stop output temporarily. The program will continue to execute but output will not be printed on any output device unless an error occurs or unless BASIC is waiting for a command or for data from an input statement. In the latter case, the Teletype is the expected input device. This feature is particularly useful for programs that print lengthy introductions and then request a user-specified parameter. Typing CTRL/O after the program is started will cause BASIC to bypass printing the introduction and wait until the parameter is specified, thereby saving the time required to print the message. A second CTRL/O will resume output.

NOTE

For most programs that do not wait for input from the Teletype, processing of the program after an initial CTRL/O will be completed before a second CTRL/O can be typed. Thus, it is very possible for no output to be printed rather than the anticipated partial output.

Erasing a Program in Core
SCR

The command:

```
SCRATCH
```

or

```
SCR
```

is provided to allow the programmer to clear his storage area, deleting any commands, or a program which may have been previously entered, and leaving a clean area in which to work. If the storage area is not cleared before entering a new program, lines from previous programs may be executed along with the new program, causing errors or misinformation. The SCRATCH command eliminates all old statements and numbers and should be used before any tapes are read into core, or new programs created.

LOADING AND OPERATING PROCEDURES

BASIC Compiler

The following procedure may be used to load in the 8K BASIC binary tape.

1. Toggle the RIM Loader into field 0 and, using the appropriate reader, read the Binary Loader into field 0. (Refer to Appendix A for details.) 8K BASIC will not use locations 7600 to 7777, thereby preserving the Binary Loader if it is present.
2. Place the 8K BASIC binary tape in the appropriate reader; set switches 6-8 = 0, and 9-11 = 0; press EXTD ADDRess LOAD.
3. Set the Switch Register = 7777 and press ADDRess LOAD.

4. If using high-speed reader, set the Switch Register = 3777 and press CLEAR and CONTinue; otherwise, simply press CLEAR and CONTinue.
5. After the tape has read in, set the Switch Register = 1000.
6. Press ADDRess LOAD, and CLEAR and CONTinue. BASIC responds by typing READY.
7. BASIC programs on paper tapes may be read in using the PTR command explained earlier, or created on-line.

User-Defined Function

The following procedure may be used to load in a user-defined function.

1. Load the Binary Loader into field 1.
2. Load BASIC into field 0.
3. Load the user-function (binary paper tape overlay) into field 0 with BIN loader.
4. Set Switch Register = 1000; press ADDRess LOAD and START.

Note that the Binary Loader is destroyed. To reload BASIC, steps 1 through 6 must be repeated.

8K BASIC ERROR MESSAGES

The computer checks all commands before executing them. If for some reason it cannot execute the command, it indicates this by typing one of the error messages. The number of the line in which the error was found is also typed out. The form is:

ERROR MESSAGE AT LINE XXXX

Table 3 lists the errors 8K BASIC checks for and reports before execution:

Table 3 8K BASIC Error Messages

Message	Meaning
ARGUMENT ERROR	A function has been given an illegal argument; for example: SQR(-1)
DATA ERROR	There are no more items in the data list.
FOR ERROR	FOR loops are nested too deeply.

FUNCTION ERROR	The user has attempted to call a function which has not been defined.
GOSUB ERROR	Subroutines are nested too deeply.
LINE NO ERROR	A GOTO, GOSUB, or IF references a non-existent line.
NEXT ERROR	FOR and NEXT statements are not properly paired.
RETURN ERROR	RETURN statement issued when not under control of a GOSUB.
SUBSCRIPT ERROR	A subscript has been used which is outside the bounds defined in the DIM statement.
SYNTAX ERROR	The command does not correspond to the language syntax. Common examples of syntax errors are misspelled commands, unmatched parentheses, and other typographical errors. Reference to an undefined UUF will also produce this diagnostic.
TOO-BIG ERROR	The combination of program size and number of variables exceeds the capacity of the computer. Reducing one or the other may help. If the program has undergone extensive revision, punching it out, typing SCRATCH and reloading should be tried.

The following programming errors are not reported by 8K BASIC, but instead are used in the computation as specified. They are included here for the programmer's reference.

1. Attempting to use a number in a computation which is too large for BASIC to handle will produce a result which is meaningless.
2. Attempting to use a number in a computation which is too small for BASIC to handle will result in the value zero being used instead.
3. Attempting to divide by zero will produce a result which is meaningless.

BASIC SYMBOL TABLE

Table 4 lists 8K BASIC's symbols and their values. This information is useful when writing user-coded (machine language) functions.

Table 4 8K BASIC Symbol Table

Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value
ABCDEF	1756	BARROW	2666	CT3	0014	EPTR	0056
ABDGET	0006	BCDEFG	1757	CVTLOO	5024	ERROR	4142
ABOP	0325	BCKWDS	4502	DATAER	1667	EVAL	1004
ABS	6425	BEGFIX	3762	DBAD	7513	EVALGO	1007
AC1	0020	BIDLE	6713	DBBAD	7532	EXECUT	0213
AC2	0017	BKWD	0600	DBGOT	7420	EXIT	2402
AC3	0016	BREAK	6522	DBISRT	7547	EXP	6000
ACCEPT	7473	BSKIP	2730	DBLIT	7526	EXPGOO	5242
ACE	0025	BUSY	6737	DBPUT	7556	EXPLON	5764
ACN	4417	CARRET	2700	DDLAST	7512	EXPOK	5265
ACOUNT	0022	CCINTK	7465	DECEXP	0043	FAD	4000
ACS	0024	CCXRA	7342	DECFRA	3366	FADEXT	1314
ADA1	0025	CDEVCO	0000	DEEPER	0526	FADI	4400
ADA2	0026	CDINP	7445	DEF	1576	FATNAX	6273
ADA3	0027	CHECKW	2346	DELAY	7463	FATNC	6337
ADACPT	0007	CHKFIT	6400	DELETE	6501	FATNC1	6304
ADB	7477	CLAB	6133	DELOUT	0142	FATNC2	6307
ADC	7477	CLBA	6136	DEVCOM	7175	FATNC3	6312
ADCCOR	0064	CLC	7477	DEVCON	7176	FATNC4	6315
ADCL	6530	CLCA	6137	DICD	6051	FATNC5	6320
ADCUNT	0010	CLEAR	7432	DIGIN	3224	FATNC6	6323
ADCX	0012	CLEARV	2462	DIGIT	3204	FATNC7	6326
ADDRES	0067	CLEN	6134	DIGLUP	6557	FATNC8	6331
ADLE	6536	CLKSTS	0003	DILC	6050	FATNC9	6334
ADLM	6531	CLOCKI	0175	DILE	6056	FATNCH	6342
ADRB	6533	CLOE	6132	DILX	6053	FATNCJ	6345
ADRS	6537	CLRCNT	0360	DILY	6054	FATNSX	6272
ADSE	6535	CLS	7477	DIM	6472	FATNT	6276
ADSK	6534	CLSA	6135	DIMFLA	0034	FATNTT	6301
ADST	6532	CLSK	6131	DINP	7511	FCNTLC	6665
AGET	0301	CLTEMP	0011	DIRE	6057	FCNTLO	6703
AL1	4654	CLZE	6130	DISAUT	0015	FDIGIT	3360
ALGNLP	4466	CNCLR	0143	DISB	0136	FDV	7000
ALL3	3146	CNTLCF	6700	DISD	6052	FDVI	7400
ALLOC	1461	CNTLCR	6670	DIVLP	4705	FENTER	4435
ALTMOD	2663	CNTLO	0133	DIVXTE	3364	FEXIT	0000
AMATCH	6506	CODELO	0004	DIXY	6055	FEXPC1	6072
ANORM	4600	COLUMN	0126	DLAST	7511	FEXPC2	6075
APUT	0263	COMMON	3400	DOAD	0362	FEXPC3	6100
APUT1	0023	COMPAR	2136	DOADLP	0366	FEXPC4	6103
APUT2	0024	CONST	1367	DOITNO	1247	FEXPC5	6106
AR1	4402	COS	5616	DOTZER	7371	FEXPC6	6111
ARGERR	7363	COWT	7326	DPFLAG	3365	FEXPF	6067
ARRLOC	0003	COWTFP	7343	DQINTX	3170	FEXPI	6061
ATEMP	0323	COWTLP	7331	DSCREW	0375	FEXPU	6064
ATEMP2	0324	COWTO	7340	DVLOOP	5245	FINDIT	0557
ATLINE	6451	COWTW	7344	EDIT	2405	FINDLU	0565
ATN	6200	CRINTX	3076	END	2567	FIX	4744
ATNBIG	6265	CRLF	6531	ENDLIN	7643	FIXEXI	4773
ATNLOW	6220	CRLFPR	3740	ENDNUM	3331	FIXITU	5200
ATNNOT	6237	CT1	0016	ENDPDL	7704	FIXLIN	2113
AUTEMP	0063	CT2	0015	EOFAD	4526	FIXLUP	4750

FIXUP	5146	FSB	5000	GPTR	0060	INTOU	6765
FJMP	1000	FSBI	5400	GRB	7224	INWDTM	4064
FJMPI	1400	FSEQ	0050	GRDELA	7222	IPNOPE	4024
FJUMP	1130	FSGE	0100	GSBEND	7755	IPOINT	7034
FLD	3000	FSGT	0140	GSBPTR	0165	ISDEF2	3512
FLDI	3400	FSHIFT	7443	GSS1	1562	ISDIG	6532
FLOGC1	6175	FSIN10	5641	GSS2	1563	ISDIM	1473
FLOGC2	6156	FSINC1	5713	GTBKLP	1710	ISET	7407
FLOGC3	6161	FSINC3	5716	GTEMP	7254	ISIT	4566
FLOGC4	6164	FSINC4	5721	GWHERE	7270	ISITDF	0550
FMP	6000	FSINC5	5724	HFOUND	7321	ISITFU	1110
FMPI	6400	FSINC6	5727	HIGHWD	4333	ISITLI	4104
FMT1	5125	FSINC7	5732	HLOOP	2722	ISLIT	4133
FMT2	5053	FSINM4	5735	HPTR	0061	ISSOME	1644
FMT3	5130	FSINOK	5657	HRCHAR	7256	ISUMIN	1013
FMTENF	5123	FSINZ	5705	HRLOP	7302	ITSDEF	3514
FN	5453	FSINZZ	5710	HRMES	7323	ITSDP	3256
FNERR	0352	FSLE	0150	IAMLES	2156	ITSE	3263
FNEXIT	1200	FSLT	0110	IDLEAC	6732	ITSOP	1220
FOR	0413	FSNE	0040	IDLECD	6725	ITSP	3300
FORCT	0063	FSQRX	5407	IDLECI	6726	JBPENT	3707
FORDON	0663	FST	2000	IDLELK	6731	JDIGIT	3124
FORERR	0501	FSTI	2400	IDLEPC	6733	JISDIG	3367
FORLIM	0721	FTANT1	5677	IF	0375	JMATCH	2766
FORLIS	7705	FTANT2	5702	IGNORE	2115	JPUTCH	0777
FORSTE	0724	FUNTAB	1131	IIXR	7414	JTXXIT	3123
FORVAR	0452	FUPRC1	5762	IMMED	2454	JUST0	3150
FOUND	0575	FWD	0200	IN	3431	JUST0F	3160
FOURLF	3557	FXXPFX	6023	INCHAR	7255	JUST0P	3163
FPADD	4456	GALT	7247	INDEV	0127	JUST1	3145
FPADDR	4304	GDIM2	1564	INDEX1	0013	JUST2	3147
FPDIV	4667	GET	0001	INDEX2	0014	KEYWD	0231
FPDOIT	4237	GETADD	1400	INLCTM	4065	L4LUP	3664
FPFLAG	0156	GETARY	7462	INLOOP	0572	LBEGIN	7563
FPGOTO	4273	GETBLK	1674	INLUPF	0432	LCF	6662
FPJMP	4317	GETCH	7201	INODUN	6710	LET	0312
FPJUMP	4274	GETJ	1770	INOPPP	6641	LETDO	0205
FPLAC	4351	GETLIN	2603	INOTTT	6645	LETTER	3446
FPLOOP	4202	GETLRE	2600	INPLUP	4034	LFXLUP	2333
FPMUL	4530	GETOPR	1015	INPPTR	4063	LHALF	3070
FPNOAD	4270	GETVAR	0311	INPUT	4007	LIMIT	0003
FPOPER	4305	GETWD	0177	INSERT	2032	LINBUF	7512
FPPGZ	4227	GLOOP	2711	INSRT5	2030	LINENO	0052
FPSKIP	4314	GOBOTH	0532	INT	6434	LINFIX	2330
FPSTO	4322	GOLIST	7725	INTAC	6734	LIST	3600
FPSUB	4453	GOSUB	0505	INTCDF	6770	LIST2	3640
FPT	4200	GOTEMP	0053	INTCIF	6771	LIST3	3655
FPTEMP	4576	GOTO	0517	INTECD	6762	LIST4	3661
FPTR	0057	GOTOPR	1202	INTEMP	6736	LIST5	3676
FPZDIV	4736	GOTSS	1074	INTER	6600	LISTAL	3616
FRNDX	5404	GOTSTE	0634	INTEXT	6744	LISTLU	3620
FRSTNE	2155	GOUT	7251	INTL	6735	LISTSO	3617

LITRAL	3131	NOBUMP	4633	O212	0010	O7706C	3473
LLLJMP	7457	NOCOMM	0335	O215	0007	O7715	2775
LLLJMS	7446	NOINT	0134	O23	1366	O7725A	3100
LLLUUU	7140	NONBLN	3110	O233	3772	O7725B	3371
LLS	6666	NONZER	5016	O240	0051	O7737	3122
LNOEND	3630	NOPARE	1035	O253	5150	O7740	0054
LOADED	4127	NOPCR	2216	O255	5151	O7741	3745
LOCCTR	0045	NORLFT	6423	O256A	5156	O7743	4743
LOCTEM	0671	NORMED	5220	O256B	6575	O7745	0162
LOCTMP	1673	NORMIT	5207	O260	0011	O7746	7264
LOG	6114	NORUBO	5574	O27	3370	O7753	3104
LOGACE	6170	NOSS1	1460	O305	5152	O7762	2327
LOGFWD	6167	NOSS2	1453	O32	3121	O7763A	0774
LOGOKW	6172	NOT	3427	O36	2767	O7763B	3101
LOWLOC	2171	NOTBAD	2127	O3737B	2773	O7764A	1274
LPTOUT	7163	NOTBIG	4620	O3754	1162	O7764B	3102
LSF	6661	NOTCR	3023	O3755	1273	O7764C	3374
LSTLOC	2160	NOTFRS	2061	O377	0071	O7764D	6777
LUP	3405	NOTHER	0435	O4	0160	O7766	5154
LUPF	0426	NOTKWD	0313	O40	2770	O7770	0055
MACHIN	0000	NOTNOW	2000	O4001	7525	O7771	5155
MAYZER	4612	NOTSGN	3301	O4014	1163	O7772	5157
MENDLI	0041	NOTTXT	2236	O42	3106	O7773	5346
MENDPD	2363	NOTVAR	1105	O4200	3105	O7774	1566
MEVAL	7415	NOTX10	5236	O4213	1164	O7775	6776
MEVALG	7431	NPSPER	7555	O5400	5347	O7776	3372
MGOLIS	0720	NSYMTA	0006	O6201A	6615	O7777	7114
MGSBEN	0525	NULCMD	7454	O6201B	7005	OADD	4435
MINUS	1316	NULJOB	7415	O6202	6775	OBHIGH	1177
MLBEGI	0173	NULLOP	7430	O7	0072	OBLOW	1165
MLEND	0174	NUMBUF	5335	O70	6774	OBOP	7101
MLINBU	0040	O1000	3550	O700	1272	OCC	5205
MNSONE	0736	O11	0504	O7000A	2560	OCMLIM	7115
MOREDI	6470	O110	2361	O7000B	3474	OCOR	7113
MOREIN	4000	O12	0065	O7000C	7456	OCOUNT	7042
MORERD	1621	O122	2771	O7077	1275	ODEV	0132
MOVE	2012	O13	1567	O7520	5153	OFLAG	7100
MOVLUP	2072	O132	7265	O7545	4777	OFLOW	7115
MPY	5321	O137	7261	O7570	6456	OJUMP	1276
MPYLUP	4552	O14	2360	O7577	4577	OLDOP	0066
MTXXIT	3201	O140A	0775	O7577B	7554	ONE	0151
MULCLR	4571	O140B	2772	O7600	4345	ONEDIM	1064
MULEXP	3346	O143	7266	O7601	7260	ONESS	1076
MULXTE	3363	O17	5147	O7603	2774	ONLY1	3322
MUSTBE	4570	O1742	3375	O7610	5345	OO7600	5452
NDELAY	7420	O175	7267	O7612	7262	OO7736	2326
NEWCHA	2615	O177	0027	O7640A	0765	OP1	0023
NEWLIN	2610	O1774	3376	O7640B	2763	OP2	0022
NEXT	0600	O2	0062	O7640C	7263	OP3	0021
NEXTER	0673	O200	0155	O7673	3316	OPDONE	1203
NEXTVA	0637	O2040A	3373	O77	0070	OPE	0030
NINTEC	7457	O2062	5344	O7706A	6544	OPERAN	0073

OPNUL	7Ø67	PGOTOP	Ø1Ø7	PRENT	2316	PZERDO	6545
OPOINT	7Ø65	PHRCHA	674Ø	PRESET	Ø137	QBIDLE	7257
OPOTAB	7Ø73	PIGNOR	Ø471	PRINBL	2277	QERROR	417Ø
OPRST	7155	PINCHA	6741	PRINCO	23Ø3	QHRCHA	7156
OPS	ØØ26	PINT	5676	PRINHA	226Ø	RANDAE	Ø451
OPUTC	7Ø41	PISITL	Ø175	PRINQU	2225	RBSWCH	Ø135
OTEMP	1271	PJSET	2461	PRINRE	2242	READ	1623
OTHER	3ØØØ	PLBEGI	Ø172	PRINSE	2312	READLO	ØØ46
OTST1	7112	PLETDO	Ø2Ø4	PRINT	2173	READY	6525
OUTD2	Ø13Ø	PLETTE	31Ø3	PRINTC	22Ø7	REALTI	7473
OUTDEL	7146	PLIMIT	2561	PRINTG	22Ø6	REJECT	7473
OUTDEV	Ø131	PLINBU	ØØ37	PRINTH	2222	RELATE	1342
OUTIT	7Ø43	PLINFI	Ø161	PRINTX	37Ø2	REMPAC	3Ø43
OUTNUM	5ØØØ	PLIST	2563	PRINUM	3747	RESET1	7116
OV	ØØ12	PLITRA	3377	PRINVA	3653	RESET2	7141
PACN	4742	PLOG	5775	PRLOOP	3711	RESTOR	3773
PAL1	Ø15Ø	PLOT	74ØØ	PRSUBR	3724	RETNER	Ø713
PANORM	Ø146	PLOTB	7514	PRTEMP	ØØ42	RETURN	Ø677
PAR1	Ø147	PLUS	1312	PRTXRE	3722	RHALF	3Ø74
PARGER	ØØ47	PMEVAL	7441	PSGN	5675	RMLEFT	6413
PASSCR	Ø472	PMPY	516Ø	PSKIPI	1617	RND	5353
PASSUM	744Ø	PNBF6	5161	PSLOOP	Ø116	RNDJMP	535Ø
PBEGFI	1776	PNOCR	Ø757	PSPACE	1565	RTERR	7373
PBIDLE	7161	PNONBL	Ø122	PSTICK	Ø121	RUBO	5573
PBOMB	Ø367	PNOTNO	2566	PSTOVA	Ø112	RUN	2452
PBUSY	7157	PNUMBU	ØØ44	PSXERR	Ø1ØØ	RUN2IN	2543
PCCUNT	Ø744	POADD	Ø157	PSYMTA	ØØØ5	RUN2LU	2514
PCHKFI	Ø163	POFLAG	6742	PTABDE	557Ø	RUN2NO	2537
PCOMMO	3357	POP	3551	PTABFL	5571	RUNIN	25Ø3
PCOWT	Ø141	POP3	4434	PTABLE	2777	RUNLUP	2465
PDEVCO	7442	POPERA	3127	PTEN	Ø145	RUNNOT	2477
PDL	ØØ36	POTHER	2776	PTEXT	ØØ76	SCHMOR	1657
PDLIST	7644	POUTIT	6743	PTPOUT	7164	SCRATC	244Ø
PEDIT	Ø12Ø	POUTNU	Ø117	PTRIN	7172	SEARCH	166Ø
PERMSY	7Ø22	PPAC1	ØØ42	PTUBIG	3Ø21	SETCLO	7473
PERROR	ØØ77	PPAC2	ØØ43	PUSERF	5545	SETRAT	7473
PEVAL	Ø1Ø1	PPAC3	ØØ44	PUSH	2364	SETSGN	4512
PEVALG	224Ø	PPACE	ØØ45	PUTCDF	7Ø37	SETUP	74ØØ
PEXECU	Ø1Ø3	PPACS	ØØ46	PUTCH	Ø741	SGN	Ø726
PEXP	5776	PPASSC	Ø11Ø	PUTCIN	7Ø23	SIMPLV	3466
PFINDI	Ø672	PPDLIS	Ø125	PUTER	7ØØØ	SIN	5624
PFIX	Ø1Ø6	PPERMS	2565	PUTJ	1761	SINCHA	7162
PFNERR	5546	PPFLOO	4741	PUTLOC	2172	SJUMP	Ø24Ø
PFPLOO	4575	PPFORL	176Ø	PUTLP	7ØØ7	SKIPIT	Ø467
PGETAD	Ø1Ø2	PPINT	6Ø6Ø	PUTXRA	Ø776	SLASH	1332
PGETBL	Ø115	PPOP	Ø1Ø5	PXFORL	Ø556	SLOOP	27Ø7
PGETCH	ØØ32	PPRINR	2241	PXLINB	3746	SLSHTM	1337
PGETLI	Ø124	PPRINT	Ø114	PXXCRL	3125	SNUMFL	ØØ64
PGETLR	3Ø22	PPRINU	Ø123	PXXEOF	2564	SPACER	Ø37Ø
PGETOP	Ø111	PPUSH	Ø1Ø4	PXXEXI	3126	SPECIN	Ø14Ø
PGETVA	Ø113	PPUTCH	ØØ33	PXXLIT	313Ø	SPLEFT	Ø144
PGOLIS	Ø164	PPXRA	716Ø	PXXTHE	2562	SQEXIT	545Ø

SQLOOP	5435	U776Ø	ØØ52	UUAC1	Ø654	XXEND	7232
SQR	5412	U7767	ØØ35	UUAC2	Ø661	XXEOF	75Ø2
SSERR	157Ø	U7775	ØØ21	UUAC3	Ø666	XXEQ	7Ø54
SSFIX	4775	U7777	ØØ5Ø	UUDATA	7567	XXEXIT	75Ø5
SSONE	Ø344	UABAD	Ø62Ø	UUDEVC	Ø7Ø5	XXEXP	7126
SSTWO	Ø345	UAC1	ØØ37	UUFUDG	Ø735	XXFINI	75Ø4
STAR	1327	UAC2	ØØ4Ø	UUJMP	Ø722	XXFN	71Ø7
START	1ØØØ	UAC3	ØØ41	UUJMS	Ø7Ø7	XXFOR	7235
STICKI	643Ø	UACCPT	Ø453	UUMEVA	Ø673	XXGE	7Ø4Ø
STOP	257Ø	UADCB	Ø6ØØ	UUNOAD	746Ø	XXGET	7173
STOVAR	Ø341	UADCIN	Ø613	UUPFIX	Ø677	XXGOSU	7241
SUBRA	2161	UADCMY	Ø615	UUSETF	ØØ3Ø	XXGOTO	7246
SXERR	6441	UADCN	Ø622	UUUJMP	Ø73Ø	XXGT	7Ø6Ø
TAB	5547	UCLC	Ø345	UUUJMS	Ø721	XXIF	726Ø
TABDES	6367	UCLOOP	Ø217	UUULLL	Ø352	XXINPU	7263
TABDO	635Ø	UCLS	Ø34Ø	UVP	75Ø2	XXINT	7142
TABFLG	2345	UDEVCO	ØØ56	UWAIT	Ø437	XXLBRA	71Ø5
TABOK	636Ø	UDOAD	ØØ51	UWAITC	Ø44Ø	XXLE	7Ø35
TABTHR	2362	UDOPER	1363	VAR	Ø343	XXLET	727Ø
TAN	56ØØ	UFFUD	Ø756	VARTEM	Ø553	XXLIS	7162
TBEGFI	5572	UFJMP	Ø757	VSCHIN	3524	XXLIST	7156
TEN	ØØØØ	UGETWD	ØØ57	VSCHLU	3475	XXLITØ	75Ø6
THESKI	1353	UGH1	3562	VSCHNO	352Ø	XXLOG	7123
THISTX	31Ø7	UIEXT	Ø234	WAIT	7473	XXLPT	7361
TIM	7477	UIEXT2	Ø243	WAITC	7473	XXLT	7Ø56
TIM1	ØØØ4	UIEXT3	Ø245	WDTEMP	1Ø75	XXMINU	7Ø25
TIM2	ØØØ5	UIEXT4	Ø26Ø	WORD	ØØ5Ø	XXNE	7Ø43
TMP	ØØ31	UINAC	Ø631	XEXECU	Ø412	XXNEXT	7274
TOOLON	5162	UJMP	ØØ54	XGISIT	4Ø66	XXNRUB	7377
TPRINT	637Ø	UJMS	ØØ55	XGMUST	7345	XXOPEN	71Ø3
TRALUP	21Ø5	UMEVAL	ØØ31	XISIT	4Ø73	XXPLOT	7422
TRANSF	21Ø3	UMOPER	1321	XMUST	7352	XXPLUS	7Ø23
TRYAGI	5133	UNDERF	4645	XRESTA	1ØØ3	XXPRIN	73ØØ
TRYSTE	Ø626	UPAGET	Ø653	XXABS	7134	XXPTP	7364
TST	7421	UPARR2	4365	XXACPT	7466	XXPTR	7367
TSTFX	7417	UPARRO	6457	XXADB	7223	XXPUT	7153
TSTP	7431	UPARRX	574Ø	XXADC	7176	XXRBRA	7Ø7Ø
TTYIN	7173	UPCOMD	Ø731	XXATN	712Ø	XXREAD	7337
TTYOUT	7165	UPFIX	ØØ32	XXBSLS	723Ø	XXREAL	7426
TUBIG	2657	UPFUN	Ø747	XXCLC	722Ø	XXREJT	7473
TWIDTH	2357	UPJMP	Ø736	XXCLEA	74Ø6	XXREM	7327
TWOSS	1Ø77	UREAL	Ø46Ø	XXCLOS	7Ø66	XXRETR	73Ø5
TXTPAK	3Ø46	UREJT	Ø456	XXCLS	7215	XXRND	7145
U1Ø	ØØ36	USE	7446	XXCOMM	7Ø62	XXRSTO	7322
U1ØØ	ØØ13	USERFN	162Ø	XXCOS	7112	XXRUB	7372
U17	ØØ47	USETC	Ø411	XXCRLF	7226	XXRUN	7165
U177	ØØ17	USETF	Ø55Ø	XXDATA	7343	XXSCR	717Ø
U1P	75Ø3	USETM	Ø416	XXDEF	7333	XXSEMI	7Ø64
U2P	75Ø4	USETR	Ø4ØØ	XXDELA	7412	XXSETC	7443
U47Ø7	ØØ2Ø	USKIPI	7461	XXDIM	7316	XXSETR	7435
U5Ø1Ø	ØØ34	UTEMP	ØØ53	XXEG	7Ø46	XXSGN	7137
U7	ØØ33	UTIM	Ø541	XXEL	7Ø51	XXSIN	715Ø

```
XXSLAS  7Ø31
XXSQR   7131
XXSTAR  7Ø27
XXSTEP  7Ø76
XXSTOP  7312
XXTAB   72Ø7
XXTAN   7115
XXTEXT  75Ø1
XXTHEN  7253
XXTIME  72Ø4
XXTO    7Ø72
XXTTY   72Ø1
XXTTYI  7347
XXTTYO  7354
XXUCOM  7462
XXUNAR  75Ø3
XXUPAR  7Ø33
XXUSE   7416
XXUUF   7212
XXWAIT  7452
YYWAIT  7456
ZERDON  5144
ZERO    Ø152
ZFIXEX  4767
ZZADB   Ø775
ZZADC   Ø77Ø
```

STATEMENT AND COMMAND SUMMARIES

Summaries of the editing and program control commands available in 8K BASIC are presented below.

Edit and Control Commands

Command	Abbreviation	Action
CTRL/C		Stops a running program. and returns to the editing phase of BASIC.
CTRL/O		Stops output of a running BASIC program. Remains in this state until BASIC requests INPUT, an error occurs, or until another CTRL/O is typed.
LIST	LIS	Lists the entire program in core.
	LIS n	Lists line n through end of program.
NO RUBOUTS		Disables the RUBOUT key.
RUBOUTS		Enables the RUBOUT key.
RUN	RUN	Compiles and runs the program currently in core.
SCRATCH	SCR	Erases the current program from core.

BASIC Statements

Statement	Example of Form	Explanation
DATA	DATA n1, n2, . . . nn	Numbers n1 through nn are to be associated with corresponding variables in a READ statement.
DEF	DEF FNB (x) = f(x) DEF FNB (x, y) =f(x, y)	The user may define his own function to be called within his program by putting a DEF statement at the beginning of a program. The function name begins with FN and must have three letters. The function is then equated to a formula f(x) which must be only one line long.

Statement	Example of Form	Explanation
DIM	DIM v(s)	Enables the user to create a table or array with the specified number of elements where v is the variable name and s is the maximum subscript value. Any number of arrays can be dimensioned in a single DIM statement.
END	END	Last statement in the program. Signals completion of the program.
FOR-TO-STEP	FOR v=f1 TO f2 STEP f3	Used to implement loops; the variable v is set equal to the formula f1. From this point the loop cycle is completed following which v is incremented after each cycle by f3 until its value is greater than f2. If STEP f3 is omitted, f3 is assumed to be +1. f3 may also be negative.
GOSUB	GOSUB n	Allows the user to enter a subroutine at several points in the program. Control transfers to line n.
GOTO	GOTO n	Transfers control to line n and continues execution from there.
IF-GOTO	IF f1 r f2 GOTO n	Same as IF-THEN.
IF-THEN	IF f1 r f2 THEN n	If the relationship r between the formulas f1 and f2 is true, transfers control to line n (n may also represent an operation); if not, continues in regular sequence.
INPUT	INPUT v1, v2, . . . vn	Causes typeout of a ? to the user and waits for the user to supply the values of the variables v1 through vn.
LET	LET v=f	Assigns the value of the formula f to the variable v.
LPT	LPT	Assigns line printer as output device.
NEXT	NEXT v	Used to tell the computer to return to the FOR statement and execute the loop again until v is greater than or equal to f2.

Statement	Example of Form	Explanation
PRINT	PRINT a1, a2, . . . an	Prints the values of the specified arguments, which may be variables, text or format control characters (, or ;).
PTP	PTP	Assigns high-speed paper tape punch as output device.
PTR	PTR	Assigns high-speed paper tape reader as input device.
READ	READ v1, v2, . . . vn	Variables v1 through vn are assigned the value of the corresponding numbers in the DATA string.
REM	REM	When typed as the first three letters of a line, allows typing of remarks within the program.
RESTORE	RESTORE	Sets pointer back to the beginning of the string of DATA values.
RETURN	RETURN	Must be at the end of each subroutine to enable control to be transferred to the statement following the last GOSUB.
STOP	STOP	Terminates execution at that point at which the statement is reached in the program.
TTY IN	TTY IN	Assigns a console terminal as input device.
TTY OUT	TTY OUT	Assigns a console treminal as output device.

During input to the editor or when executing an INPUT command, the following messages may be printed in response to the input:

Message	Explanation
LINE TOO LONG	The line just typed exceeded the available core buffer and must be retyped.
DELETED	The line has been deleted in response to an ALTMODE character and must be retyped.
←	Back arrow is printed any time a RUBOUT or SHIFT/O is used. The previous character is deleted.
TTY	BASIC prints TTY to indicate that there is no more input from the high-speed reader and that it is waiting for input from the Teletype.

appendix a
loading procedures

Initializing the system

Before using the computer system, it is good practice to initialize all units. To initialize the system, ensure that all switches and controls are as specified below.

1. Main power cord is properly plugged in.
2. Teletype is turned OFF.
3. Low-speed punch is OFF.
4. Low-speed reader is set to FREE.
5. Computer POWER key is ON.
6. PANEL LOCK is unlocked.
7. Console switches are set to 0.
8. SING STEP is not set.
9. High-speed punch is OFF.
10. DECtape REMOTE lamps OFF.

The system is now initialized and ready for your use.

Loaders

READ-IN MODE (RIM) LOADER

When a computer in the PDP-8 series is first received, it is nothing more than a piece of hardware; its core memory is completely demagnetized. The computer "knows" absolutely nothing, not even how to receive input. However, the programmer can manually load data directly into core using the console switches.

The RIM Loader is the very first program loaded into the computer, and it is loaded by the programmer using the console

switches. The RIM Loader instructs the computer to receive and store, in core, data punched on paper tape in RIM coded format (RIM Loader is used to load the BIN Loader described below.)

There are two RIM loader programs: one is used when the input is to be from the low-speed paper tape reader, and the other is used when input is to be from the high-speed paper tape reader. The locations and corresponding instructions for both loaders are listed in Table A-1.

The procedure for loading (toggling) the RIM Loader into core is illustrated in Figure A-1.

Table A-1. RIM Loader Programs

	Instruction	
Location	Low-Speed Reader	High-Speed Reader
7756	6032	6014
7757	6031	6011
7760	5357	5357
7761	6036	6016
7762	7106	7106
7763	7006	7006
7764	7510	7510
7765	5357	5374
7766	7006	7006
7767	6031	6011
7770	5367	5367
7771	6034	6016
7772	7420	7420
7773	3776	3776
7774	3376	3376
7775	5356	5357
7776	0000	0000

After RIM has been loaded, it is good programming practice to verify that all instructions were stored properly. This can be done by performing the steps illustrated in Figure A-2, which also shows how to correct an incorrectly stored instruction.

When loaded, the RIM Loader occupies absolute locations 7756 through 7776.

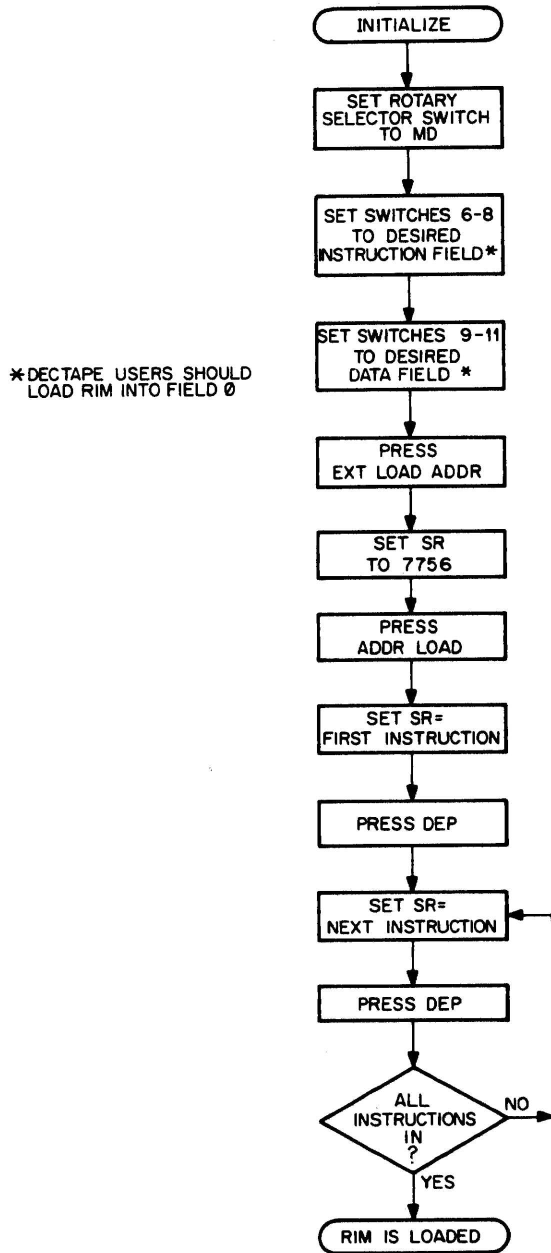


Figure A-1. Loading the RIM Loader

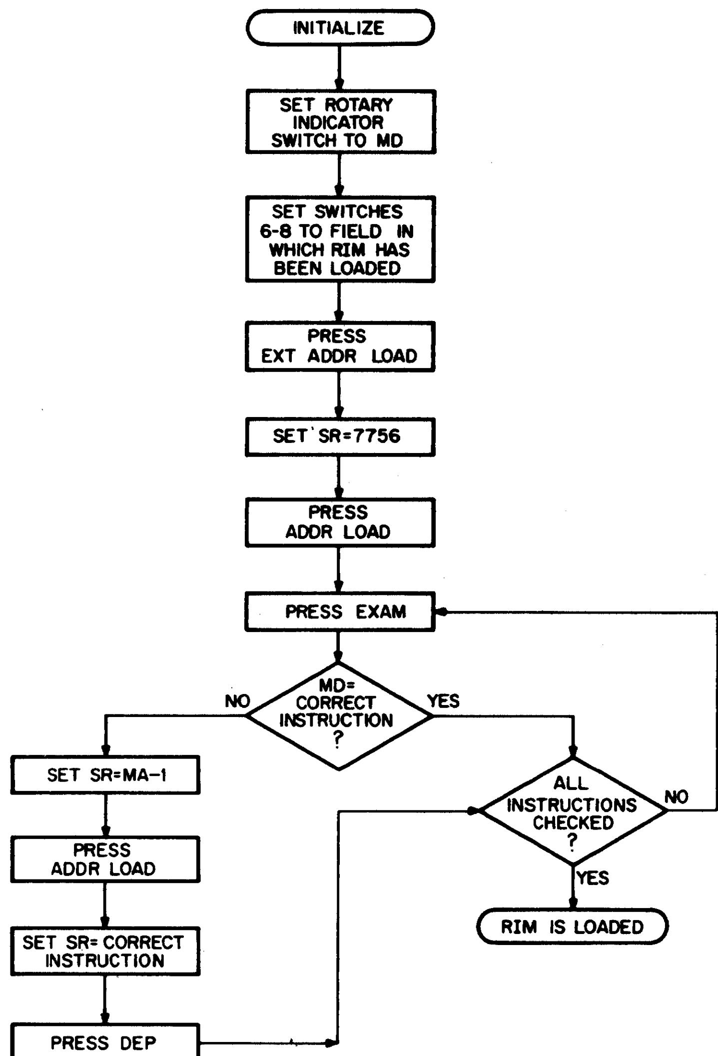


Figure A-2. Checking the RIM Loader

BINARY (BIN) LOADER—

The BIN Loader is a short utility program which, when in core, instructs the computer to read binary-coded data punched on paper tape and store it in core memory. BIN is used primarily to load the programs furnished in the software package (excluding the loaders and certain subroutines) and the programmer's binary tapes.

BIN is furnished to the programmer on punched paper tape in RIM-coded format. Therefore, RIM must be in core before BIN can be loaded. Figure A-3 illustrates the steps necessary to properly load BIN. And when loading, the input device (low- or high-speed reader) must be that which was selected when loading RIM.

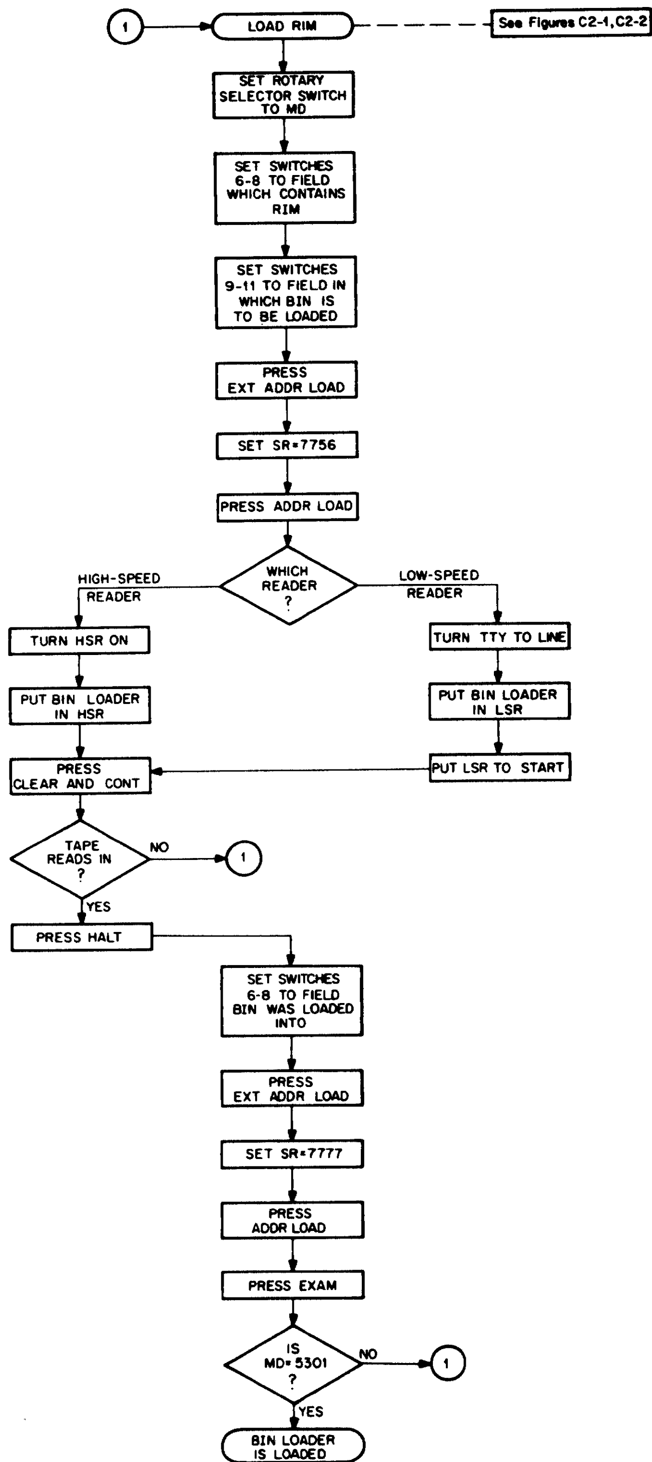


Figure A-3 Loading the BIN Loader

When stored in core, BIN resides on the last page of core, occupying absolute locations 7625 through 7752 and 7777.

BIN was purposely placed on the last page of core so that it would always be available for use—the programs in DEC's software package do not use the last page of core (excluding the Disk Monitor). The programmer must be aware that if he writes a program which uses the last page of core, BIN will be wiped out when that program runs on the computer. When this happens, the programmer must load RIM and then BIN before he can load another binary tape.

Binary tapes to be loaded should be started on the leader-trailer code (Code 200), otherwise zeros may be loaded into core, destroying previous instructions.

Figure A-4 lilustrates the procedure for loading binary tapes into core.

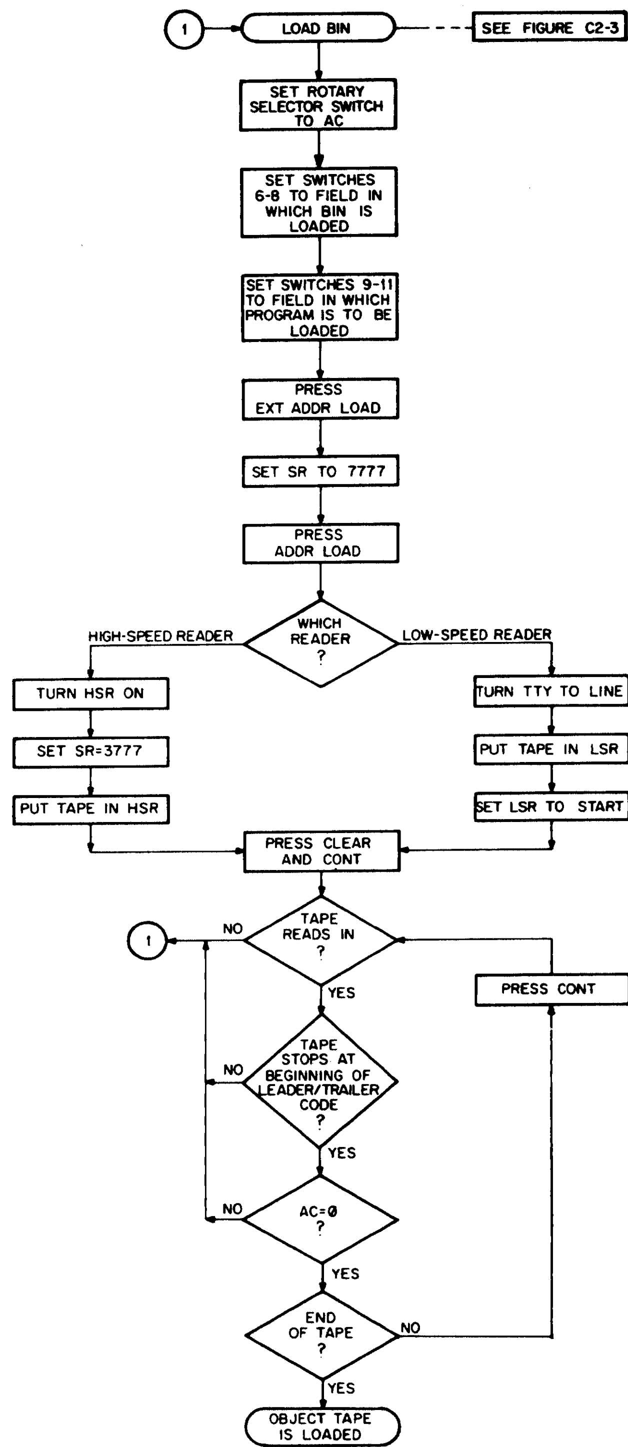


Figure A-4. Loading A Binary Tape Using BIN

appendix b
character codes

ASCII-1[1] Character Set

Character	8-Bit Octal	6-Bit Octal	Decimal Equivalent (A1 Format)	Character	8-Bit Octal	6-Bit Octal	Decimal Equivalent (A1 Format)
A	301	01	96	!	241	41	−1952
B	302	02	160	"	242	42	−1888
C	303	03	224	#	243	43	−1824
D	304	04	288	$	244	44	−1760
E	305	05	352	%	245	45	−1696
F	306	06	416	&	246	46	−1632
G	307	07	480	'	247	47	−1568
H	310	10	544	(	250	50	−1504
I	311	11	608	)	251	51	−1440
J	312	12	672	*	252	52	−1376
K	313	13	736	+	253	53	−1312
L	314	14	800	,	254	54	−1248
M	315	15	864	-	255	55	−1184
N	316	16	928	.	256	56	−1120
O	317	17	992	/	257	57	−1056
P	320	20	1056	:	272	72	−352
Q	321	21	1120	;	273	73	−288
R	322	22	1184	<	274	74	−224
S	323	23	1248	=	275	75	−160
T	324	24	1312	>	276	76	−96
U	325	25	1376	?	277	77	−32
V	326	26	1440	@	300		32
W	327	27	1504	[	333	33	1760
X	330	30	1568	\	334	34	1824
Y	331	31	1632	]	335	35	1888
Z	332	32	1696	↑(∧)[2]	336	36	1952
0	260	60	−992	←(—)[2]	337	37	2016
1	261	61	−928	Leader/Trailer	200		
2	262	62	−864	LINE FEED	212		
3	263	63	−800	Carriage RETURN	215		
4	264	64	−736	SPACE	240	40	−2016
5	265	65	−672	RUBOUT	377		
6	266	66	−608	Blank	000		
7	267	67	−544	BELL	207		
8	270	70	−480	TAB	211		
9	271	71	−416	FORM	214		

[1] **An abbreviation for American Standard Code for Information Interchange.**

[2] The character in parentheses is printed on some Teletypes.

INDEX

AC1, 44
AC2, 44
AC3, 44
Acceptable nesting
techniques, 25
ACE, 44
ACS, 44
Addressing, 45
Indirect relative, 47
Relative, 45
ALTMODE, 54
ANORM, 48
Argument, 36
Arithmetic operations, 3
Priority of, 4
Arithmetic statement, 10
Array, 25
Maximum size of an, 28
ASCII character set, B-1

Backslash, 9
BASIC compiler, 58
BEGFIX, 48
BIN loader, A-4
loading the, A-5
loading a binary tape,
A-7
BKWD, 45

Character codes, B-1
Character set, B-1
Characters,
Format control, 15
Checking the RIM loader,
A-4
Coding formats (UUF), 43
Command,
LET, 6
LIST, 55
LPT, 55, 57
PRINT, 6
PTP, 55, 57
PTR, 56
RUN, 56
SCR, 58
Command summary, 67
Commands,
Editing and Control, 53,
67
Commenting the program, 9
Conditional transfer, 24,
30
Control characters,
Format, 15
Control commands,
Editing and, 53, 67
CTRL/C, 36, 57
CTRL/O, 57

DATA statement, 11
DEF statement, 42
Devices,
I/O, 22
DIM statement, 27
Directing program control,
29
Dummy variable, 13, 39, 42

E-type notation, 2
Editing and Control
commands, 53, 67
END statement, 9
Equal sign,
Meaning of the, 5
Erasing a program in core,
58
Erasing characters and
lines, 53
Error messages, 59
Evaluation,
Order of, 4
Example Program, 7
Examples (UUF), 48
Exponential notation, 2

FAC, 44
FENTER, 44
FEXIT, 44
Field, 24
FIX, 48
Floating accumulator, 44
Floating-point format, 44
Floating-point instruction
set, 45
Floating-point interpreter,
45
Floating-point package, 43
FNA function, 42
FOR statement, 22
Format control characters,
15
comma, 16
semicolon, 17
Formats, coding (UUF), 43
Formatting output, 41
Formula, 22, 24
Function, 34
FNA, 42
GET, 39
INT, 35
PUT, 39
RND, 36
SGN, 35
TAB, 36
User-defined, 59
UUF, 43
Functions, 34
FWD, 45

Generating random numbers
over any range, 37
GET function, 39
GOSUB nesting,
maximum level of, 33
GOSUB statement, 30, 31
GOTO statement, 29

I/O devices, 22
IF GOTO statement, 30
IF THEN statement, 30
Immediate mode, 6
Incremental value, 22
Index, 22
Indirect relative
addressing, 47
Initial value, 22
Initializing the system,
A-1
INPUT statement, 14
Input/Output statements, 11
Instruction set,
Floating-point, 45
INT function, 35
Integer function, 35
Introduction, 1

LET command, 6
LET statement, 10
Level of GOSUB nesting, 33
List, 25
LIST command, 55
Listing and punching
program, 55
Loaders, A-1
Loading a binary tape
(using BIN), A-7
Loading and operating
procedures, 58
Loading procedures, A-1
Loading the BIN loader, A-5
Loading the RIM loader, A-3
Loops, 22
Nesting, 24
LPT command, 55, 57
LPT statement, 19

Mass storage devices, 47
Maximum level of GOSUB
nesting, 33
Maximum size of an array,
28
Meaning of the equal sign,
5
Minimum system
configuration, 1
Mode,
Immediate, 6

Nesting loops, 24
Nesting subroutines, 33
Nesting techniques,
Acceptable, 25
Unacceptable, 25
Nesting, maximum level of
GOSUB, 33
NEXT statement, 22, 23
NO RUBOUTS, 53
Normalized form, 44
Numbers, 2

Operating procedures,
Loading and, 58
Operators, 3
Relational, 5
Order of evaluation, 4
Output,
Formatting, 41

Parentheses, 4
PRINT command, 6
Print positions, 37
PRINT statement, 15
Print zones, 16
Priority of arithmetic
operations, 4
Program control,
Directing, 29
Programming errors, 60
PTP command, 55, 57
PTP statement, 20
PTR command, 56
PTR statement, 14
PUT function, 39

Random number function, 36
READ statement, 11
Reading a program, 56
Relational operators, 5
Relative addressing, 45
Indirect, 47
REM statement, 9
RESTORE statement, 12
RETURN, 54
RETURN statement, 31
RIM, A-1
RIM loader,
Checking the, A-4
Loading the, A-3
RIM loader programs, A-2
RND function, 36
Rounding numbers, 36
RUBOUT, 53
RUN command, 56
Running a program, 56

SCR command, 58
SGN function, 35
SHIFT/L, 9

SHIFT/O, 53
Sign function, 35
Sign-magnitude convention, 43
Statement,
  Arithmetic, 10
  DATA, 11
  DEF, 42
  DIM, 27
  END, 9
  FOR, 22
  GOSUB, 30, 31
  GOTO, 29
  IF GOTO, 30
  IF THEN, 30
  INPUT, 14
  LET, 10
  LPT, 19
  NEXT, 22, 23
  PRINT, 15
  PTP, 20
  PTR, 14
  READ, 11
  REM, 9
  RESTORE, 12
  RETURN, 31
  STEP, 23
  STOP, 10
  TTY IN, 20
  TTY OUT, 20
Statement numbers, 8
Statement summary, 67
Statements, 7, 67
  Input/Output, 11
  Transfer of control, 29
STEP statement, 23
STOP statement, 10
Stopping a run, 57
Subroutines, 30
  Nesting, 33
Subscript, 27
Subscripted variables, 22, 25
Summary,
  Command, 67
  Statement, 67
Supported options, 2
Symbol Table, 62-66
System configuration, minimum, 1

TAB function, 36
Table,
  Symbol, 62
Terminal value, 22
Terminating the program, 9
Transfer,
  Conditional, 24, 30
  Unconditional, 29
Transfer of control statements, 29
TTY IN statement, 20
TTY OUT statement, 20
Two-dimensional matrix, 26

Unacceptable nesting techniques, 25
Unconditional transfer, 29
User-defined function, 59, 43
UUF function, 43

Variable, 3
  Dummy, 13, 39, 42
  Subscripted, 22, 25

Writing the program, 47

8K BASIC
DEC-08-LBSMA-A-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback--your critical evaluation of this document.

Did you find errors in this document? If so, please specify by page.

____

____

____

____

____

How can this document be improved?

____

____

____

____

____

How does this document compare with other technical documents you have read?

____

____

____

____

____

Job Title________________________ Date:____________

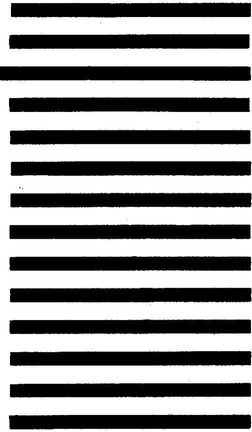
Name:____________________ Organization:____________

Street:__________________ Department:______________

City:______________ State:________ Zip or Country________

Fold Here

Do Not Tear - Fold Here and Staple



BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Digital Equipment Corporation
Software Information Service
Software Engineering and Services
Maynard, Massachusetts 01754