HEWLETT *hp* PACKARD

HP BASIC

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HP BASIC

HEWLETT **hp** PACKARD

11000 Wolfe Road
Cupertino, California 95014

Second Edition

PREFACE

The purpose of this text is to help the programmer learn how
to use BASIC rather than how to program. A "semi-programmed"
frame format is used for ease in reference and self-instruction.

The contents of this second edition are organized in sequence
from least to most difficult. Since the sections are modu-
lar, experienced programmers may follow any sequence of in-
struction. Related features are grouped by section and cross-
referenced where appropriate. Any examples presented in color
may be used as a practice exercise.

Operating instructions are presented in Section VIII.

NEW AND CHANGED INFORMATION

This text replaces the "BASIC LANGUAGE" Reference Manual
published August, 1968.

The external specifications of HP BASIC have been changed to
allow interface with the HP Magnetic Tape System. Material
pertinent to this change is contained in Section VII, "For
Advanced Programmers".

The format has been changed to facilitate self-instruction
and reference. More examples, including sample programs,
have been added.

A "Quick Reference" section is included in the Appendix
section; it is a summary of all BASIC features explained
in the text.

CONVENTIONS USED IN THIS TEXT

SAMPLE	EXPLANATION
RUN	Black, all capitals in examples indicates computer-output information.
And then...	Mixed upper and lower case black is used for regular text.
20 PRINT X,Y	Green, all capitals indicates a statement or command typed by the programmer.
statement _number_	Black lower case italics indicates a general form, derived from BASIC syntax requirements (Section VI). Green underlining indicates an essential part of a general form; each underlined item is a separate, essential element.
return _linefeed_ _esc_ _ctrl_ _alt-mode_	Represents the terminal keys: Return, Linefeed, Escape, Control, and Alt-Mode.
Note: _Both X and..._	Mixed upper and lower case italics is used for notes.
CALL	Oversize black is used for page headings.
O	The letter "O"
Ø	Zeroes are slashed.

Please examine the sample on the next page.

PAGE FORMAT

The reference page format is as uniform as possible. This
sample shows how positioning and typeface relate to content.
Black frames are used on reference pages.

EXAMPLES: ————————   Several sample
          ————————   statements or commands

GENERAL FORM: ———— ———— ————
          (Each essential element underlined in green.)

PURPOSE

A clear and concise explanation of the purpose or function.

COMMENTS

A series of several items containing:

        Pertinent information
        Additional explanation or examples
        Helpful hints.

Reference to other sections or subsections related to the
contents of this page.

Section No. ____   Page No. ____

CONTENTS

CONTENTS

CONTENTS

CONTENTS

HOW TO USE THIS BOOK

If your purpose is:	Read:
Quickly acquiring a minimum working knowledge of HP BASIC:	Sections I and II.
Acquiring a good working knowledge of HP BASIC:	Sections I, through VII, in sequence.
Learning the complete HP BASIC system:	The entire book, in sequence.
Learning to operate the computer:	Section VIII.
Reference only:	1. Contents 2. Appendix C 3. The index 4. Index tabs to locate the appropriate section.

SECTION I

COMMUNICATING WITH THE COMPUTER

There are many types of languages. English is a
natural language used to communicate with people.

To communicate with the computer, formal languages
are used. A formal language is a combination of
simple English and algebra; for example, BASIC is
a formal language used to communicate with the
computer.

Like natural languages BASIC has grammatical rules,
but they are much simpler. For example, this series
of BASIC statements (which calculates the average of
five numbers given by you, the user) shows the funda-
mental rules:

```
1Ø INPUT A,B,C,D,E
2Ø LET S = (A+B+C+D+E)/5
3Ø PRINT S
4Ø GO TO 1Ø
5Ø END
```

The following pages show how to interpret these rules.
Notice how the statements are written. What they do
is explained later.

1-1

STATEMENTS

This is a BASIC *statement*:

1Ø INPUT A,B,C,D,E

COMMENTS

A *statement* contains a maximum of 72 characters
(one teletypewriter line).

A *statement* may also be called a line.

STATEMENT NUMBERS

Each BASIC statement begins with a _statement number_ (in this example, 2∅):

    2∅ LET S=(A+B+C+D+E)/5

COMMENTS

The number is called a _statement number_ or a _line number_.

The statement number is chosen by you, the programmer. It may be any integer from 1 to 9999 inclusive.

Each statement has a unique _statement number_. The computer uses the numbers to keep the statements in order.

Statements may be entered in any order; they are usually numbered by fives or tens so that additional statements can be easily inserted. The computer keeps them in numerical order no matter how they are entered. For example, if statements are input in the sequence 3∅,1∅,2∅; the computer arranges them in the order: 1∅,2∅,3∅.

INSTRUCTIONS

The statement then gives an _instruction_ to the
computer (in this example, PRINT):

3Ø PRINT S

COMMENTS

Instructions are sometimes called _statement types_
because they identify a type of statement. For
example, the statement above is a "print" statement.

OPERANDS

If the instruction requires further details, _operands_
(numeric details) are supplied (In this example, 1Ø;
on the previous page, "S"):

40 GO TO 1Ø

COMMENTS

The _operands_ specify what the instruction acts upon; for
example, what is PRINTed, or where to GO.

The sequence of BASIC statements given on the previous pages is called a *program*.

The last statement in a program, as shown here, is an END statement.

```
1Ø INPUT A,B,C,D,E
2Ø LET S=(A+B+C+D+E)/5
3Ø PRINT S
4Ø GO TO 1Ø
5Ø END
```

COMMENTS

The last (highest numbered) statement in a program must be an END statement.

The END statement informs the computer that the program is finished.
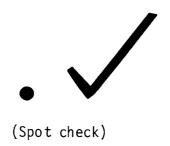
FREE-FORMAT LANGUAGE

BASIC is a "free format" language--the computer
ignores extra blank spaces in a statement. For
example, these three statements are equivalent:

```
30 PRINT S
30 PRINT  S
30PRINTS
```

COMMENTS

When possible, leave a space between words and
numbers in a statement. This makes a program
easier for people to read.

✔

•

(Spot check)

Be sure you are familiar with these terms before
continuing:

> statement
> instruction
> statement type
> statement number
> line number
> operand
> program

All of these terms are defined in the context of
this section.

BEFORE WORKING WITH THE COMPUTER

The following pages explain how to correct mistakes
and list programs.

Since you will probably have to make several correc-
tions in your first attempts to use the computer,
these features should be learned before beginning.

RETURN

The _return_ key must be pressed after each statement.

EXAMPLES: 1Ø INPUT A,B,C,D,E, _return_
                     2Ø LET S=(A+B+C+D+E)/5 _return_
                     3Ø PRINT S _return_
                     4Ø GO TO 1Ø _return_
                     5Ø END _return_

COMMENTS

Pressing _return_ informs the computer that the statement is complete. The computer then checks the statement for mistakes.

ENTERING A PROGRAM

This frame shows how to enter the sample program.
If you are not sure how the computer responds when
a statement is entered, use it as a practice
exercise.

```
10 INPUT A,B,C,D,E return

linefeed
20 LET S = (A+B+C+D+E)/5 return

linefeed
30 PRINT S return

linefeed
40 GO TO 10 return

linefeed
50 END return

linefeed
```

COMMENTS

The computer responds with a *linefeed* (terminal skips
a line) after each statement is entered, indicating
that the statement has been checked, accepted, and the
computer is ready for another statement.

MISTAKES AND CORRECTIONS

The reverse arrow (←) key acts as a backspace,
deleting the immediately preceding character.

Typing:	2Ø LR←ET S=1Ø _return_
is equivalent to typing:	2Ø LET S=1Ø _return_
And typing:	3Ø LET← ← ← PRINT S _return_
is equivalent to typing:	3Ø PRINT S _return_

COMMENTS

The ← character is a "shift" O on most terminals.

DELETING OR CHANGING A STATEMENT

To delete the statement being typed, press the _esc_ or _alt-mode_ key. This causes a \ to be printed, and deletes the entire line being typed.

To _delete_ a previously typed statement, type the statement number followed by a _return_.

To _change_ a previously typed statement, retype it with the desired changes. The new statement replaces the old one.

Pressing the _esc_ key deletes the statement being typed:

  2∅ LET S = _esc_

  NOTE: The computer responds with a \ when esc is typed, like this:

  2∅ LET S = \

To _delete_ statement 5 in the sequence:

  5 LET S = ∅
  1∅ INPUT A,B,C,D,E
  2∅ LET S = (A+B+C+D+E)/5

  NOTE: \ and / are different, and have very different functions.

type:

  5 _return_

Or, to _change_ statement 5 in the above sequence, type:

  5 LET S = 5 _return_

The old statement is replaced by the new one.

Typing an _esc_ (or _alt-mode_) before a _return_ prevents replacement of a previously typed statement.

For example, typing:

  5 LET _esc_

or:

  5 _esc_

has no effect on the original statement 5.

RUNNING A PROGRAM

This frame shows what happens when the sample program is run. The program does not begin execution (does not run) until the command RUN followed by a _return_ is typed.

NOTE: _The sample program (averaging 5 numbers) has been entered._

COMMENTS

The computer responds with a _linefeed_ indicating that the command is being executed.

The question mark indicates that input is expected. The five numbers being averaged should be typed in, SEPARATED BY COMMAS, and followed by a _return_.

The answer is printed.

? indicates that five more numbers are expected.

The answer is printed.

NOTE: _This program continues executing indefinitely, unless terminated by the user. To terminate, type an S return when more input is requested._

The program is finished.

RUN _return_
linefeed

? 95.6,87,3,5,9Ø,82.8 _return_
linefeed

84.24 _return_
linefeed

? -12.5,-5Ø.6,-32,45.6,6Ø _return_

2.1 _return_
linefeed

? S _return_

STOPPING A PROGRAM

When RUN or LIST is typed, BASIC "takes over" the
terminal until the program finishes executing or
the listing is complete.

To stop a program that is
running or being listed,
press, then release, any
key.   *esc* (or any key)

BASIC then responds with
the STOP message: STOP

COMMENTS

Remember that: S *return* is used to end input loops.

Line 1Ø tells the computer that five numbers will
be input, and that they should be given the labels
A,B,C,D,E in sequence. The first number input is
labeled "A" by the computer, the second "B", etc.
A,B,C,D, and E are called variables.

`1Ø INPUT A,B,C,D,E`

After line 1Ø is executed, the variables and
their assigned values, typed in by the user,
are stored. For example, using the values en-
tered by the user in the previous example, this
information is stored: A = -12.5; B = -50.6;
C = -32; D = 45.6; E = 60

Line 2Ø declares that a variable called S
exists, and is assigned the value of the sum
of the variables A,B,C,D,E divided by 5:

`2Ø LET S = (A+B+C+D+E)/5`

Line 3Ø instructs the computer to output the
value of S to user's terminal:

`3Ø PRINT S`

NOTE: *If the PRINT statement were not given,*
       the value of S would be calculated and
       stored, but not printed. The computer
       must be given explicit instruction for
       each operation to be performed.

Line 4Ø tells the computer to go to line 1Ø
and execute whatever instruction is there:

`4Ø GO TO 1Ø`

NOTE: *A "loop" is formed by lines 1Ø to 4Ø.*
       The sequence of statements in this
       loop execute until the user breaks
       the loop. This particular kind of
       loop is called an input loop (because
       the user must consistently input data).
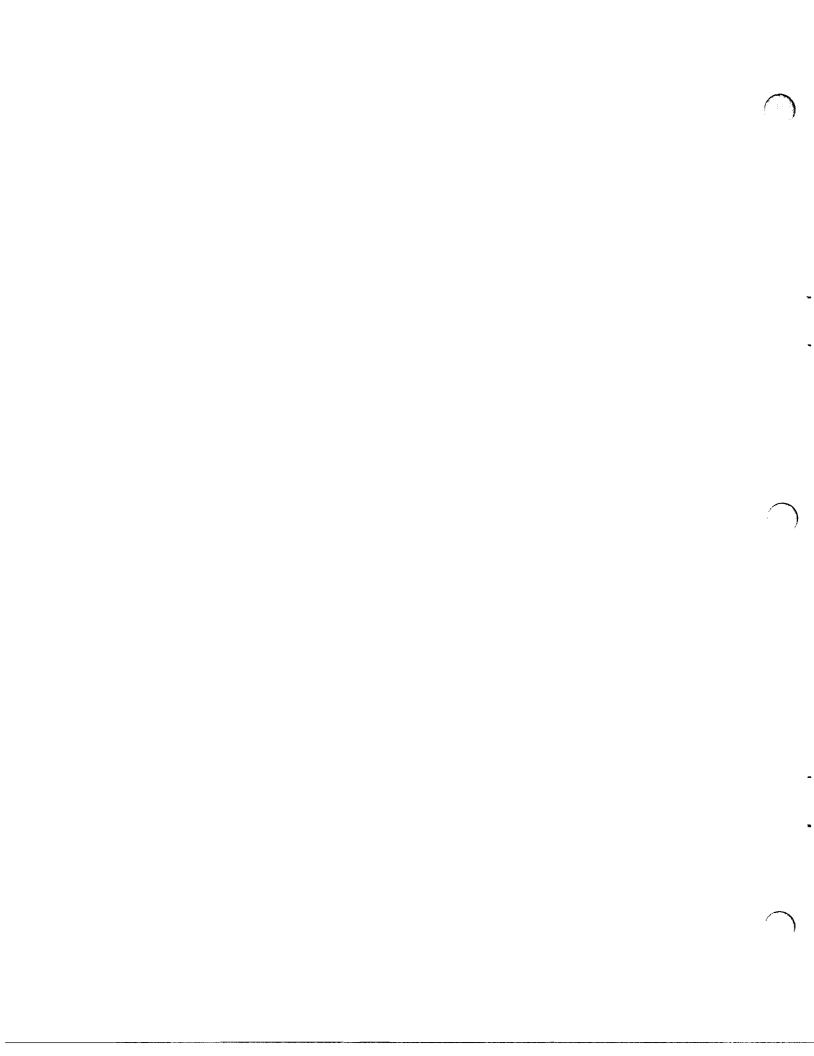
TYPING: S WHEN INPUT IS REQUESTED
BY A "?" IS THE ONLY WAY TO BREAK AN
INPUT LOOP. Other, more controlled
loops are explained later. Line 5Ø
is not executed until the loop is
broken by typing: S when input is
requested.

Line 5Ø informs the computer that the program   5Ø END
is finished:

SECTION II

THE ESSENTIALS OF BASIC

This section contains enough information to allow
you to use BASIC in simple applications.

Proceed at your own pace. The information in the
vocabulary and operators subsections is included
for completeness; experienced programmers may skip
these.

The "Operators" pages contain brief descriptions,
rather than explanations, of the logical operators.
The novice should not expect to gain a clear under-
standing of logical operators from this presenta-
tion. Section V presents more details and examples
of logical operations. Readers wishing to make best
use of logical capabilities should consult this sec-
tion. Those unfamiliar with logical operations should
also refer to an elementary logic text.

A simple program is included at the end of this section
for reference; it contains a running commentary on the
uses of many of the BASIC statements presented in the
section.

TERM: SIMPLE VARIABLE

DEFINED IN BASIC AS: A letter (from A to Z); or a
                        letter immediately followed
                        by a digit (from Ø to 9).

EXAMPLES: AØ   B
                        M5   C2
                        Z9   D

COMMENTS

Variables are used to represent numeric values.
For instance, in the statement:

                1Ø LET M5 = 96.7

M5 is a variable; 96.7 is the value of the variable
M5.

There is one other type of variable in BASIC, the
array variable; its use is explained in Section IV.

TERM: NUMBER

DEFINED IN BASIC AS: A decimal number (the sign is optional) between an approximate minimum of:

10^{-38} (or 2^{-129})

and an approximate maximum of:

10^{38} (or 2^{127})

Zero is included in this range.

TERM: E NOTATION

DEFINED IN BASIC AS: A means of expressing numbers having more than six decimal digits, in the form of a decimal number raised to some power of 10.

EXAMPLES: 1.00000E+06 is equal to 1,000,000 and is read: "1 times 10 to the sixth power" (1×10^6).

1.02000E+04 is equal to 10,200

1.02000E-04 is equal to .000102

COMMENTS

"E" notation is used to print numbers having more than six significant digits. It may also be used for input of any number.

When entering numbers in "E" notation, leading and trailing zeroes may be omitted from the number; the + sign and leading zeroes may be omitted from the exponent.

The precision of numbers is 6 to 7 decimal digits (23 binary digits).

TERM: EXPRESSION

DEFINED IN BASIC AS:	A combination of variables, constants and operators which evaluates to a numeric value.
EXAMPLES:	(P + 5)/27
	(where P has previously been assigned a numeric value.)
	Q - (N + 4)
	(where Q and N have previously been assigned numeric values.)

TERM: ARITHMETIC EVALUATION

DEFINED IN BASIC AS:	The process of calculating the value of an expression.

THE ASSIGNMENT OPERATOR

SYMBOL: =

EXAMPLES: 1∅ LET A = B2 = C = ∅
                         2∅ LET A9 = C5
                         3∅ LET Y = (N-(R+5))/T
                         4∅ LET N5 = A + B2
                         5∅ LET P5 = P6=P7=A=B=98.6

GENERAL FORM: LET *variable* = *expression*

PURPOSE

Assigns an arithmetic or logical value to a variable.

COMMENTS

When used as an assignment operator, = is read "takes
the value of," rather than "equals". It is, therefore,
possible to use assignment statements such as:

LET X = X+2

This is interpreted by BASIC as: "LET X take the value
of (the present value of) X, plus two."

Several assignments may be made in the same statement, as
in statements 1∅ and 5∅ above.

See Section V, "Logical Operations" for a description of
logical assignments.

RELATIONAL OPERATORS

```
SYMBOLS:          =  #  <>  >  <  >=  <=

EXAMPLES:         100 IF A=B THEN 900
                  110 IF A+B >C THEN 910
                  120 IF A+B < C+E  THEN 920
                  130 IF C>=D*E  THEN 930
                  140 IF C9<= G*H THEN 940
                  150 IF P2#C9 THEN 950
                  160 IF J <> K THEN 950
```

PURPOSE

Determines the logical relationship between two expressions, as

```
            equality:  =
          inequality:  #  or   <>
        greater than:  >
           less than:  <
greater than or equal to:  >=
   less than or equal to:  <=
```

COMMENTS

NOTE: It is not necessary for the novice to understand the nature of logical evaluation of relational operators, at this point. The comments below are for the experienced programmer.

Expressions using relational operators are logically evaluated, and assigned a value of "true" or "false" (the numeric value is 1 for "true," and 0 for false).

When the = symbol is used in such a way that it might have either an assignment or a relational function, BASIC assumes it is an assignment operator. For a description of the assignment statement using logical operators, see Section V, "Logical Operations."

THE AND OPERATOR

```
SYMBOL:     AND

EXAMPLES:   6Ø IF  A9<B1   AND   C#5   THEN 1ØØ
            7Ø IF  T7#T    AND   J=27  THEN 15Ø
            8Ø IF  P1 AND R>1   AND   N AND V2   THEN 1Ø
            9Ø PRINT X AND Y
```

PURPOSE

Forms a logical conjunction between two expressions. If both are "true," the conjunction is "true"; if one or both are "false," the conjunction is "false."

NOTE: It is not necessary for the novice to understand how this operator works. The comments below are for experienced programmers.

COMMENTS

The numeric value of "true" is 1, of "false" is Ø.

All non-zero values are "true." For example, statement 9Ø would print either a Ø or a 1 (the logical value of the expression X AND Y) rather than the actual numeric values of X and Y.

Control is transferred in an IF statement using AND, only when all parts of the AND conjunction are "true." For instance, example statement 8Ø requires four "true" conditions before control is transferred to statement 1Ø.

See Section V, "Logical Operations" for a more complete description of logical evaluation.

THE OR OPERATOR

SYMBOL: OR

EXAMPLES: 1ØØ IF A>1 OR B<5 THEN 5ØØ
            11Ø PRINT C OR D
            12Ø LET D = X OR Y
            13Ø IF (X AND Y) OR (P AND Q) THEN 6ØØ

PURPOSE

Forms the logical disjunction of two expressions. If
either or both of the expressions are true, the OR dis-
junction is "true"; if both expressions are "false," the
OR disjunction is "false."

*NOTE: It is not necessary for the novice to understand how
this operator works. The comments below are for
experienced programmers.*

COMMENTS

The numeric values are: "true" = 1, "false" = Ø.

All non-zero values are true; all zero values are false.

Control is transferred in an IF statement using OR, when
either or both of the two expressions evaluate to "true."

See Section V, "Logical Operations" for a more complete
description of logical evaluation.

THE NOT OPERATOR

SYMBOL: NOT

EXAMPLES: 30 LET X = Y = 0
            35 IF NOT A THEN 300
            45 IF (NOT C) AND A THEN 400
            55 LET B5 = NOT P
            65 PRINT NOT (X AND Y)
            70 IF NOT (A=B) THEN 500

PURPOSE

Logically evaluates the complement of a given expression.

NOTE: It is not necessary for the novice to understand
*   how this operator works. The comments below are*
*   intended for experienced programmers.*

COMMENTS

If A = 0, then NOT A = 1; if A has a non-zero value,
NOT A = 0.

The numeric values are: "true" = 1, "false" = 0; for
example, statement 65 above would print "1", since the
expression NOT (X AND Y) is true.

Note that the logical specifications of an expression may
be changed by evaluating the complement. In statement 35
above, if A equals zero, the evaluation would be "true"
(1); since A has a numeric value of 0, it has a logical
value of "false," making NOT A "true."

See Section V, "Logical Operations" for a more complete
description of logical evaluation.

ORDER OF PRECEDENCE

The order of performing operations is:

↑   *highest precedence*

NOT   unary +   unary -

*   /

+   -

Relational Operators

AND

OR   *lowest precedence*

COMMENTS

If two operators are on the same level, the order of
execution is left to right, for example:

    5 + 6*7   is evaluated as: 5 + (6x7)

    7/14*2/5   is evaluated as: $\frac{(7/14)\times2}{5}$

Parentheses override the order of precedence in all cases,
for example:

    5 + (6x3) is evaluated as: 5 + 18
 and
    3 + (6+(2↑2)) is evaluated as: 3 + (6+4)

Unary + and - may be used; the parentheses are assumed by
BASIC. For example:

    A + + B is interpreted: A + (+B)
    C - + D -5 is interpreted: C - (+D)-5

Leading unary + signs are omitted from output by BASIC, but
remain in program listings.

STATEMENTS

Statements are instructions to the computer. They are contained
in numbered lines within a program, and execute in the order of
their line numbers. Statements cannot be executed without run-
ning a program. They tell the computer what to do while a pro-
gram is running.

Commands are also instructions. They are executed immediately,
do not have line numbers, and may not be used in a program. They
are used to manipulate programs, and for utility purposes.

Here are some examples mentioned in Section I:

Statements	Commands
LET	RUN
PRINT	TAPE
INPUT	LIST

Do not attempt to memorize every detail in the "Statements"
subsection; there is too much material to master in a single ses-
sion. By experimenting with the sample programs and attempting to
write your own programs, you will learn more quickly than by
memorizing.

THE LET STATEMENT

EXAMPLES: 1Ø LET A = 5.Ø2
               2Ø LET X = Y7 = Z = Ø
               3Ø LET B9 = 5* (X↑2)
               4Ø LET D = (3*C2↑N)/(A*(N/2))

GENERAL FORM:

statement number **LET** _variable = number or expression or variable..._

PURPOSE

Used to assign or specify the value of a variable. The value
may be an expression, a number, or a variable.

COMMENTS

The assignment statement must contain:

1. A statement number,
2. LET,
3. The variable to be assigned a value (for example, B9
    in statement 3Ø above),
4. The assignment operator, an = sign,
5. The number, expression or variable to be assigned to
    the variable (for example, 5*(X↑2) in statement 3Ø above).

Statement 2Ø in the example above shows the use of an assignment
to give the same value (Ø) to several variables. This is a useful
feature for initializing variables in the beginning of a program.

REM

EXAMPLES: 10 REM--THIS IS AN EXAMPLE
           20 REM: OF REM STATEMENTS
           30 REM-----/////*****!!!!!
           40 REM. STATEMENTS ARE NOT EXECUTED BY BASIC

GENERAL FORM: *statement number* REM *any remark or series of characters*

PURPOSE

Allows insertion of a line of remarks or comment
in the listing of a program.

COMMENTS

Must be preceded by a line number. Any series of
characters may follow REM.

REM lines are part of a BASIC program and are printed
when the program is listed or punched; however, they
are ignored when the program is executing.

Remarks are easier to read if REM is followed by a
punctuation mark, as in the example statements.

INPUT

This program shows several variations of the INPUT statement and their effects.

Sample Program Using INPUT

```
 5 FOR M=1 TO 2
1Ø INPUT A
2Ø INPUT A1,B2,C3,ZØ,Z9,E5
3Ø PRINT "WHAT VALUE SHOULD BE ASSIGNED TO R";
4Ø INPUT R
5Ø PRINT A;A1;B2;C3;ZØ;Z9;E5;"R=";R
6Ø NEXT M
7Ø END
```

-------------------------- RESULTS ----------------------------

RUN *return*

?1 *return*

?2,3,4,5,6,7 *return*

WHAT VALUE SHOULD BE ASSIGNED TO R?27 *return*

```
   1     2     3     4     5     6     7      R=27
```

?1.5 *return*

?2.5,3.5,4.5,6.,7.2 *return*

?8.1 *return*   *? indicates that more input is expected*

WHAT VALUE SHOULD BE ASSIGNED TO R?-99 *return*

```
   1.5     2.5     3.5     4.5     6      7.2
   8.1      R=-99
```

General Form:

      statement number INPUT *variable , variable ,...*

PURPOSE

Assigns a value input from the teleprinter to a variable.

COMMENTS

The program comes to a halt, and a question mark is printed when the INPUT statement is used. The program does not continue execution until the input requirements are satisfied.

Only one question mark is printed for each INPUT statement. The statements:

```
10 INPUT A, B2, C5, D, E, F, G
20 INPUT X
```

each cause a single "?" to be printed. The "?" generated by statement 10 requires seven input items, separated by commas, while the "?" generated by statement 20 requires only a single input item.

The only way to terminate or exit a program when input is required is entering: S _return_. Note that the S ends the program; it must be restarted with the RUN command.

Relevant Diagnostics:

? indicates that input is required.

See PRINT in this section for output variations.

PRINT

This sample program gives a variety of examples of the PRINT statement.
The results are shown below.

```
1Ø LET A=B=C=1Ø
2Ø LET D1=E9=2Ø
3Ø PRINT A,B,C,D1,E9
4Ø PRINT A/B,B/C/D1+E9
5Ø PRINT "NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE"
6Ø PRINT "VALUE IN THE SAME STATEMENT."
7Ø PRINT
8Ø PRINT
9Ø REM* "PRINT" WITH NO OPERAND CAUSES THE TELEPRINTER TO SKIP A LINE.
1ØØ PRINT "'A' DIVIDED BY 'E9' =";A/E9
11Ø PRINT "11111", "22222", "33333", "44444", "55555", "66666"
12Ø PRINT "11111"; "22222"; "33333"; "44444"; "55555"; "66666"
13Ø END
```

--------------------------- RESULTS ----------------------------

RUN *return*

```
 1Ø             1Ø             1Ø             2Ø             2Ø
  1             2Ø.Ø5
NOTE THE POWER TO EVALUATE AN EXPRESSION AND PRINT THE
VALUE IN THE SAME STATEMENT.


'A' DIVIDED BY 'E9' = .5
11111              22222          33333          44444          55555
66666
111112222233333444445555566666
```

*NOTE: The "," and ";" used in statements 11Ø and 12Ø have
very different effects on the format.*

GENERAL FORM:

statement number PRINT *expression , expression , ...*

or

statement number PRINT *"any text" ; expression ; ...*

or

statement number PRINT *"text" ; expression ; "text" , "text" , ...*

or

statement number PRINT *any combination of text and/or expressions*

or

statement number PRINT

PRINT

PURPOSE

Causes the *expressions* or *"text"* to be output to
the terminal.

Causes the teleprinter to skip a line when used
without an operand.

COMMENTS

Note the effects of , and ; on the output of the
sample program. If a comma is used to separate
PRINT operands, five fields are printed per tele-
printer line. If semicolon is used, up to twelve
"packed" numeric fields are output per teleprinter
line (72 characters).

Text in quotes is printed literally.

Remember that variable values must be defined in
an assignment, INPUT, READ or FOR statement before
being used in a PRINT statement.

Although the format of the PRINT statement is "automatic" to help beginning programmers, the experienced programmer may use several features to control his output format.

Each line output to the terminal is divided into five print fields when commas are used as separators (as in statement 3∅ in the sample program). The fields begin at print spaces ∅, 15, 3∅, 45, and 6∅. The first four fields contain fifteen spaces, and the last field contains twelve. The comma signals the computer to move to the next print field, or if in the last field, to move to the next line.

More information may be printed on a line by using semicolons as separators. Twelve numbers may be printed per line by using semicolons. (See the output from statements 11∅ and 12∅ in the sample program for an example of the differences in the two separators.)

Spacing within a print field depends on the value and type of the number being printed. A number is always printed in a field larger than itself and is left-justified. The space required for a number is determined by these formulas:

Value of Number	Type of Number	Output Field Size
$-999 \leq n \leq +999$	Integer	$\Lambda xxx \Lambda \Lambda$*
$-32768 < n < -1000$ $+1000 \leq n \leq +32767$	Integer	$\Lambda xxxxx \Lambda \Lambda$
$.1 \leq n \leq 999999.5$	Large Integer or Real	$\Lambda xxxxxxx \Lambda \Lambda \Lambda \Lambda$ (Decimal point printed as one of the x's; trailing zeros suppressed.)
$n < .1$ $999999.5 < n$	Large Integer or Real	$\Lambda x.xxxxxE\pm ee \Lambda \Lambda \Lambda$

*The Λ symbol indicates a space.

PRINT, CONTINUED

Ending a PRINT statement with a semicolon causes the output
to be printed on the same line, rather than generating a
return *linefeed* after the statement is executed. For example,
the sequence:

```
2Ø LET X = 1
3Ø PRINT X;
4Ø LET X=X+1
5Ø GO TO 2Ø
        ⋮
```

produces output in this format:

```
1    2    3    4    5    6    7    8    9    1Ø   11   12
13   14   15   16   17   18   19   20   21   22   23   24
```

Similarly, ending a PRINT statement with a comma causes output to
fill all five fields on a line before moving to the next line.
The trailing comma in statement 3Ø in the sequence:

```
2Ø LET X = 1
3Ø PRINT X,
4Ø LET X=X+1
5Ø GO TO 2Ø
        ⋮
```

produces output in this format:

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

A PRINT statement without an operand (statements 7Ø and 8Ø in
the sample program) generates a *return* *linefeed*.

Three general rules for planning output formats are:

1. If a number is an integer with a value between
    -32768 and +32767, inclusive, the decimal point
    is not printed.

2-19

2. If the number is an integer out of the above range or if the number is real and has an absolute value between .1 and 999999.5, the number is rounded to six digits and printed with a decimal point. Zeros trailing the decimal point are suppressed.

3. If a number is either greater than 999999.5 or less than .1, it is rounded to six places; the teletypewriter then prints a space (if positive) or minus sign (if negative), the first digit, the decimal point, the next five digits, the letter E (indicating exponent), the sign of the exponent, and the exponent.

See the description of the TAB function in Section III for more information on controlling output format.

GO TO

```
EXAMPLES:      1Ø LET X = 2Ø

                  .
                  .
                  .

               5Ø GOTO 1ØØ
               8Ø GOTO 1Ø

GENERAL FORM:

statement number GO TO statement number
```

PURPOSE

Transfers control to the specified statement.

COMMENTS

GO TO may be written: GOTO or GO TO.

This statement must be followed by the statement
number to which control is transferred.

GO TO overrides the normal execution sequence of
statements in a program.

Useful for repeating a task infinitely, or "jumping"
(GOing TO) another part of a program if certain con-
ditions are present.

GO TO should not be used to enter FOR-NEXT loops;
doing so may produce unpredictable results or fatal
errors. (See "FOR...NEXT" in this section for de-
tails on loops.)
To get out of a GO TO loop, type: STOP return.

2-21

IF...THEN

```
SAMPLE PROGRAM:        1Ø LET N = 1Ø
                       2Ø READ X
                       3Ø IF X <=N THEN 6Ø
                       4Ø PRINT "X IS OVER"; N
                       5Ø GO TO 1ØØ
                       6Ø PRINT "X IS LESS THAN OR EQUAL TO"; N
                       7Ø GO TO 2Ø
                       8Ø STOP
                          .
                          .
       GENERAL FORM:      .
```

statement number IF expression relational op expression THEN statement number

PURPOSE

Transfers control to a specified statement if a specified condition is true.

COMMENTS

Sometimes described as a conditional transfer; "GO TO" is implied by IF...THEN, if the condition is true. In the example above, if X<=1Ø, the message in statement 6Ø is printed (statement 6Ø is executed).

Since numbers are not always represented exactly in the computer, the = operator should be used carefully in IF...THEN statements. Limits, such as <=,>=, etc. should be used in an IF expression, rather than =, whenever possible.

If the specified condition for transfer is not true, the program will continue executing in sequence. In the example above, if X>1Ø, the message in statement 4Ø prints.

The relational operator is optional in logical evaluations.

See Section V, "Logical Operations," for a more complete description of logical evaluation.

FOR...NEXT

```
EXAMPLES:          1ØØ FOR P1 = 1 TO 5
                   11Ø FOR Q1 = N TO X
                   12Ø FOR R2 = N TO X STEP 2.5
                   13Ø FOR S  = 1 TO X STEP Y
                   14Ø NEXT S
                   15Ø NEXT R2
                   16Ø NEXT Q1
                   17Ø NEXT P1
```

- -

Sample Program - Variable Number Of Loops

```
4Ø PRINT "HOW MANY TIMES DO YOU WANT TO LOOP";
5Ø INPUT A
6Ø FOR J = 1 TO A
7Ø PRINT "THIS IS LOOP"; J
8Ø READ N1, N2, N3
9Ø PRINT "THESE DATA ITEMS WERE READ:" N1; N2; N3
1ØØ PRINT "SUM ="; (N1+N2+N3)
11Ø NEXT J
12Ø DATA 5, 6, 7, 8, 9, 1Ø, 11, 12
13Ø DATA 13, 14, 15, 16, 17, 18, 19, 2Ø, 21
14Ø DATA 22, 23, 24, 25, 26, 27, 28, 29, 3Ø
15Ø DATA 31, 32, 33, 34
16Ø END
```

GENERAL FORM:

statement number FOR *simple variable* = *initial value* TO *final value*

or

statement no. FOR *simple var.* = *initial value* TO *final value* STEP *step value*

:

statement number NEXT *simple variable*

NOTE: *The same simple variable must be used in both the FOR and NEXT state-*
        ments of a loop.

PURPOSE

Allows controlled repetition of a group of statements within a program.

COMMENTS

Initial value, final value and *step value* may be any expression.

STEP and *step value* are optional; if no step value is specified, the computer will automatically increment by one each time it executes the loop.

How the loop works:

The simple variable is assigned the value of the *initial value;* the value of the simple variable is increased by 1 (or by the *step value*) each time the loop executes. When the value of the *simple variable* passes the *final value,* control is transferred to the statement following the "NEXT" statement.

The *initial, final,* and *step values* are all evaluated upon entry to the loop and remain unchanged after entry. For example,
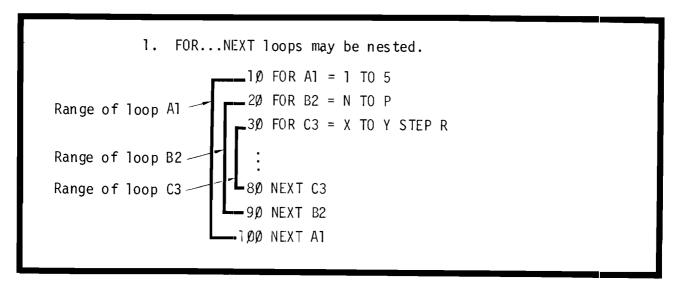
        FOR I = 1 TO I + 5

goes from 1 to 6; that is, the final value does not "move" as I increases with each pass through the loop.

For further details on the STEP feature, see "FOR...NEXT with STEP" in Section III.

Try running the sample program if you are not sure what happens when FOR... NEXT loops are used in a program.

NESTING FOR...NEXT LOOPS

Several FOR...NEXT loops may be used in the same program;
they may also be nested (placed inside one another). There
are two important features of FOR...NEXT loops:

---

1. FOR...NEXT loops may be nested.

```
            ┌──10 FOR A1 = 1 TO 5
            │┌─20 FOR B2 = N TO P
Range of loop A1 ─┤│┌─30 FOR C3 = X TO Y STEP R
            ││
Range of loop B2 ─┤│  ⋮
Range of loop C3 ─┤└─80 NEXT C3
            └──90 NEXT B2
            └─100 NEXT A1
```

---

2. The range of FOR...NEXT loops may not
    overlap. The loops in the example above
    are nested correctly. This example shows
    improper nesting.

```
            ┌──10 FOR I = 1 TO 5
            │
            │  ⋮
            │
            │┌─30 FOR J = 1 TO N
            ││
The range of loops  ││  ⋮
I and J overlap.  └─50 NEXT I
            │
            │  ⋮
            │
            └─90 NEXT J
```

2-25

READ, DATA AND RESTORE

```
                  Sample Program using READ and DATA


      15 FOR I=1 TO 5
      2Ø READ A
      4Ø LET X=A↑2
      45 PRINT A;" SQUARED =";X
      5Ø NEXT I
      55 DATA 5.24,6.75,30.8,72.65,89.72
      6Ø END
```

Each data item may be read only once in this
program. BASIC keeps track of data with a
"pointer." When the first READ statement is
encountered, the "pointer" indicates that the
first item in the first DATA statement (the one
with the lowest statement number) is to be read;
the pointer is then moved to the second item of
data, and so on.

In this example, after the loop has executed five
times, the pointer remains at the end of the data
list. To reread the data, it is necessary to re-
set the pointer. A RESTORE statement moves the
pointer back to the first data item.

Sample Program Using RESTORE with READ and DATA.

```
           20 FOR I = 1 TO 5
           30 READ A
           40 LET X=A↑2
           50 PRINT A; "SQUARED =";X
           60 NEXT I
           80 RESTORE
          100 FOR J=1 TO 5
          110 READ B
          120 LET Y=B↑4
          130 PRINT B; "TO THE FOURTH POWER =";Y
          140 NEXT J
          150 DATA 5.24,6.75,30.8,72.65,89.72
          160 END
GENERAL FORM:
    statement number READ variable , variable ,...
    statement number DATA number , number , ...
    statement number RESTORE
```

READ, DATA & RESTORE

PURPOSE

The READ statement instructs BASIC to read an item from a DATA statement.

The DATA statement is used for specifying data in a program. The data is read in sequence from first to last DATA statements, and from left to right within the DATA statement.

The RESTORE statement resets the pointer to the first data item, allowing data to be reread.

WAIT

EXAMPLE: 900 WAIT (1000)

               990 WAIT (3000)

GENERAL FORM: *statement number* WAIT (*expression* *max. value of 32767*)

PURPOSE

Introduces delays into a program. WAIT causes the program to wait the specified number of milliseconds (maximum 32767 milliseconds) before continuing execution.

COMMENTS

The time delay produced by WAIT is not precisely the number of milliseconds specified because there is no provision to account for time elapsed during calculation or terminal-computer communication.

One millisecond = 1/1000 second.

END AND STOP

EXAMPLES:

```
200 IF A # 27.5 THEN 350
       .
       .
       .
300 STOP
       .
       .
       .
500 IF B # A THEN 9999
       .
       .
       .
550 PRINT "B = A"
600 END
9999 END
```

GENERAL FORM:

any statement number STOP

any statement number END

highest statement number in program END

PURPOSE

Terminates execution of the program.

COMMENTS

The highest numbered statement in the program must be an END statement.

END and STOP statements may be used in any portion of the program to terminate execution.

END and STOP have identical effects; the only difference is that the highest numbered statement in a program must be an END statement.

SAMPLE PROGRAM

If you understand the effects of the statement
types presented up to this point, skip to the
"COMMANDS" section.

The sample program on the next two pages uses
several BASIC statement types.

Running the program gives a good idea of the
various effects of the PRINT statement on tele-
printer output. If you choose to run the program,
you may save time by omitting the REM statements.

After running the program, compare your output
with that shown under "RUNNING THE SAMPLE PROGRAM."
If there is a difference, LIST your version and
compare it with the one presented on the next
two pages. Check the commas and semi-colons; they
must be used carefully.

```
10   REMARK:  "REMARK" OR "REM" IS USED TO INDICATE REMARKS OR COMMENTS
20   REMARK:  THE USER WANTS TO INCLUDE IN THE TEXT OF HIS PROGRAM.
30   REM:  THE COMPUTER LISTS AND PUNCHES THE "REM" LINE, BUT DOES NOT
40   REM:  EXECUTE IT.
50   REM:  "PRINT" USED ALONE GENERATES A "RETURN" "LINEFEED"
60   PRINT
70   PRINT "THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY."
80   PRINT
90   PRINT "IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS."
100  PRINT
110  PRINT "PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY."
120  PRINT
130  PRINT
140  REM:  FIRST, ALL VARIABLES USED IN THE PROGRAM ARE INITIALIZED
150  REM:  TO ZERO (THEIR VALUE IS SET AT ZERO).
160  LET A=N=R1=S=0
180  REM:  NOW THE USER WILL BE GIVEN A CHANCE TO SPECIFY HOW MANY
190  REM:  NUMBERS HE WANTS TO AVERAGE.
200  PRINT "HOW MANY NUMBERS DO YOU WANT TO AVERAGE";
210  INPUT N
220  PRINT
230  PRINT "O.K., TYPE IN ONE OF THE ";N;"NUMBERS AFTER EACH QUES. MARK."
240  PRINT "DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH NUMBER."
250  PRINT
260  PRINT "NOW, LET'S BEGIN"
270  PRINT
280  PRINT
300  REM:  "N" IS NOW USED TO SET UP A "FOR-NEXT" LOOP WHICH WILL READ
310  REM:  1 TO "N" NUMBERS AND KEEP A RUNNING TOTAL.
320  FOR I=1 TO N
330  INPUT A
340  LET S=S+A
350  NEXT I
```

```
360 REM:  "I" IS A VARIABLE USED AS A COUNTER FOR THE NUMBER OF TIMES
370 REM:  THE TASK SPECIFIED IN THE "FOR-NEXT" LOOP IS PERFORMED.
380 REM:  "I" INCREASES BY 1 EACH TIME THE LOOP IS EXECUTED.
390 REM:  "A" IS THE VARIABLE USED TO REPRESENT THE NUMBER TO BE
400 REM:  AVERAGED.  THE VALUE OF "A" CHANGES EACH TIME THE
410 REM:  USER INPUTS A NUMBER.
420 REM:  "S" WAS CHOSEN AS THE VARIABLE TO REPRESENT THE SUM
430 REM:  OF ALL NUMBERS TO BE AVERAGED.
440 REM:  AFTER THE LOOP IS EXECUTED "N" TIMES, THE PROGRAM CONTINUES.
460 REM:  A SUMMARY IS PRINTED FOR THE USER.
470 PRINT
480 PRINT
490 PRINT N; "NUMBERS WERE INPUT."
500 PRINT
510 PRINT "THEIR SUM IS:";S
520 PRINT
530 PRINT "THEIR AVERAGE IS:";S/N
540 PRINT
550 PRINT
570 REM:  NOW THE USER WILL BE GIVEN THE OPTION OF QUITTING OR
580 REM:  RESTARTING THE PROGRAM.
590 PRINT "DO YOU WANT TO AVERAGE ANOTHER GROUP OF NUMBERS?"
600 PRINT
610 PRINT "TYPE 1 IF YES, 0 IF NO"
620 PRINT "BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER."
630 PRINT
640 PRINT "YOUR REPLY";
650 INPUT R1
660 IF R1=1 THEN 120
670 REM:  THE FOLLOWING LINES ANTICIPATE A MISTAKE IN THE REPLY.
680 IF R1#0 THEN 700
690 GO TO 720

700 PRINT "TO REITERATE, YOU SHOULD TYPE 1 IF YES, 0 IF NO."
710 GO TO 640
720 END
```

RUNNING THE SAMPLE PROGRAM

RUN *return*

THIS PROGRAM WILL AVERAGE ANY GROUP OF NUMBERS YOU SPECIFY.

IT WILL ASK ALL NECESSARY QUESTIONS AND GIVE INSTRUCTIONS.

PRESS THE RETURN KEY AFTER YOU TYPE YOUR REPLY.

HOW MANY NUMBERS DO YOU WANT TO AVERAGE?

O.K.,TYPE IN ONE OF THE 5 NUMBERS AFTER EACH QUES. MARK.

DON'T FORGET TO PRESS THE RETURN KEY AFTER EACH NUMBER.

NOW, LET'S BEGIN

? 99 *return*

? 87.6 *return*

? 92.7 *return*

? 79.5 *return*

? 84 *return*

5 NUMBERS WERE INPUT.

THEIR SUM IS: 442.8

THEIR AVERAGE IS: 88.56

DO YOU WISH TO AVERAGE ANOTHER GROUP OF NUMBERS?

TYPE 1 IF YES, Ø IF NO

BE SURE TO PRESS THE RETURN KEY AFTER YOUR ANSWER.

YOUR REPLY? 2 *return*

TO REITERATE, YOU SHOULD TYPE 1 IF YES, Ø IF NO.

YOUR REPLY? 1 *return*

HOW MANY NUMBERS DO YOU WISH TO AVERAGE?

COMMANDS

Remember the difference between commands and
statements. (See "Statements" in this section.)

Commands are direct instructions to the computer,
and are executed immediately. They are used for
utility purposes and for program manipulation.

Do not try to memorize all of the details in the
COMMANDS subsection. The various commands and
their functions will become clear to you as you
begin to write your own programs.

RUN

EXAMPLE:	RUN *return*
GENERAL FORM:	RUN

PURPOSE

Starts execution of a program at the lowest numbered statement.

COMMENTS

A running program may be terminated by pressing any key. To terminate a running program at some point when input is required, type:

S *return*

LIST

EXAMPLE:	LIST *return*
	or
	LIST 1ØØ *return*
GENERAL FORM:	LIST
	LIST *statement number*

PURPOSE

Produces a listing of all statements in a program (in statement number sequence) when no statement number is specified.

When a statement number is specified, the listing begins at that statement.

COMMENTS

A listing may be stopped by pressing any key.

SCRATCH

```
EXAMPLE:          SCRATCH return

GENERAL FORM:     SCRATCH
                      or
                  SCR
```

PURPOSE

Deletes (from memory) the program currently being accessed from the teleprinter.

COMMENTS

SCRATCH erases *everything* in the user's area of computer memory.

SCRATCHed programs are not recoverable. For information about saving programs on paper tape, see the PLIST command in this section.

TAPE

EXAMPLES:	TAPE _return_
GENERAL FORM:	TAPE
	or
	TAP

PURPOSE

Informs the computer that following input is from
paper tape being read from the terminal tape reader.

COMMENTS

BASIC responds to the TAPE command with a _linefeed_.

TAPE suppresses _linefeeds_ following statements.

Error messages are printed as the tape is input; the
tape reader is held inactive while they are being
printed.

PTAPE

EXAMPLES: PTAPE *return*

GENERAL FORM: PTAPE
                     or
                   PTA

PURPOSE

Causes the computer to read in a program from the
punched tape photoreader.

COMMENTS

If the computer does not have a photoreader, the
message:

                   STOP
                   READY

is printed on the terminal, and BASIC waits for
further input.

BASIC responds to the PTAPE command with a *linefeed*.

PLIST

```
EXAMPLE:              PLIST return

GENERAL FORM:         PLIST
                        or
                      PLIST statement number
```

PURPOSE

Causes the program in memory to be punched onto paper tape,
with leading and trailing guide holes; also produces a list-
ing of the program on the HP modified ASR-33 terminal; one
listing is produced on the HP modified ASR-35 in 'KT' mode.

COMMENTS

Be sure to press the "ON" button on the terminal paper tape
punch before pressing return after PLIST.

If there is no paper tape punch on the terminal, a listing is
printed.

BASIC uses the high-speed punch if available, otherwise the
terminal punch is used.

SECTION III

ADVANCED BASIC

This section describes further capabilities
of BASIC.

The experienced programmer has the option of
skipping the "Vocabulary" subsection, and
briefly reviewing the commands and functions
presented here. Matrices are explained in
the next section.

The inexperienced programmer need not spend
a great deal of time on programmer-defined
and standard functions. They are shortcuts,
and some programming experience is necessary
before their applications become apparent.

TERM: ROUTINE

DEFINED IN BASIC AS: A sequence of program statements
which produces a certain result.

PURPOSE

Routines are used for frequently performed
operations, saving the programmer the work
of defining an operation each time he uses
it, and saving computer memory space.

COMMENTS

A routine may also be called a program,
subroutine, or sub-program.

The task performed by a routine is defined
by the programmer.

Examples of routines and subroutines are
given in this section.

3-2

TERM: ARRAY OR MATRIX

DEFINED IN BASIC AS: An ordered collection of numeric data (numbers).

COMMENTS

Arrays are divided into columns (vertical) and rows (horizontal):

```
C | ROWS
0 |
L |
U |
M |
N |
S |
```

Arrays may have one or two dimensions. For example,

```
1.0
2.1
3.2
4.3
```

is a one-dimensional array, while

```
6 , 5 , 4

3 , 2 , 1

0 , 9 , 8
```

is a two-dimensional array.

Array elements are referenced by their row and column position
For instance, if the two examples above were arrays A and Z
respectively, 2.1 would be A(2); similarly, 0 would be Z(3,1).
The references to array elements are called subscripts, and
set apart with parentheses. For example, P(1,5) references the
fifth element of the first row of array P; 1 and 5 are the sub-
scripts. In X(M,N) M and N are the subscripts.

TERM: STRING

```
DEFINED IN BASIC AS:    Ø to 65 teleprinter characters
                        enclosed by quotation marks
                        (one line on a teleprinter terminal).
```

COMMENTS

Sample strings: "ANY CHARACTERS!?*/---"
                 "TEXT 1234567..."

Quotation marks may not be used within a string.
Strings are used only in PRINT statements.

The statement number PRINT, and quotation marks are not included
in the 65 character count. Each statement may contain up to 72
characters. Maximum string length is 72 characters minus 6
characters for "PRINT", two for the quotation marks, and the num-
ber of characters in the statement number.

TERM: FUNCTION

```
DEFINED IN BASIC AS:    The mathematical relationship between
                        two variables (X and Y, for example)
                        such that for each value of X there is
                        one and only one value of Y.
```

COMMENTS

The independent variable in a function is called an argument;
the dependent variable is the function value. For instance,
if X is the argument, the function value is the square root
of X, and Y takes the value of the function.

TERM: WORD

DEFINED IN BASIC AS: The amount of computer memory
                        space occupied by two teleprinter
                        characters.

COMMENTS

Numbers require two words of memory space when stored as
numbers. When used within a string, numbers require 1/2
word of space per character in the number.

SUBROUTINES AND FUNCTIONS

The following pages explain BASIC features useful for re-
petitive operations -- subroutines, programmer-defined
functions and standard functions.

The programmer-defined features, such as GOSUB,
FOR...NEXT with STEP, and DEF FN become more useful as
the user gains experience and learns to use them as
shortcuts.

Standard mathematical and trigonometric functions are
convenient timesavers for programmers at any level.
They are treated as numeric expressions by BASIC.

GOSUB...RETURN

```
EXAMPLE:              5Ø READ A2

                      6Ø IF A2<1ØØ THEN 8Ø

                      7Ø GOSUB 4ØØ
                         .
                         .
                    38Ø STOP  (STOP,END, or GO TO frequently precede
                                  the first statement of a subroutine
                                  to prevent accidental entry.)

                    39Ø REM--THIS SUBROUTINE ASKS FOR A 1 OR Ø REPLY.

                    4ØØ PRINT "A2 IS>1ØØ"

                    41Ø PRINT "DO YOU WANT TO CONTINUE";

                    42Ø INPUT N

                    43Ø IF N #Ø THEN 45Ø

                    44Ø LET A2 = Ø

                    45Ø RETURN
                         .
                         .
                    6ØØ END
```

GENERAL FORM: <u>statement number</u> <u>GOSUB</u> <u>statement number starting subroutine</u>
                         .
                         .
               <u>statement number</u> <u>RETURN</u>

PURPOSE

GOSUB transfers control to the specified statement number.

RETURN transfers control to the statement following the GOSUB statement which transferred control.

GOSUB...RETURN eliminates the need to repeat frequently used groups of statements in a program.

COMMENTS

The portion of the program to which control is transferred must logically end with a RETURN statement.

RETURN statements may be used at any desired exit point in a subroutine. GOSUB...RETURN'S may be "nested" to a level of nine during execution. There is no limit on physical nesting in the listing.

FOR...NEXT WITH STEP

```
EXAMPLES:        20 FOR I5 = 1 TO 20 STEP 2
                 40 FOR N2 = 0 TO -10 STEP  2
                 80 FOR P = 1 TO N STEP X5
                 90 FOR X = N TO W STEP (N-7-V)
                    .
                    .
                    .
GENERAL FORM:
statement no. FOR simple var. = expression TO expression STEP expression
```

PURPOSE

Allows the user to specify the size of the
increment of the FOR variable.

COMMENTS

The step size need not be an integer. For instance,
        100 FOR N = 1 TO 2 STEP .01
is a valid statement which produces approximately 100
loop executions, incrementing N by .01 each time.
Since no binary computer represents all decimal numbers
exactly, round-off errors may increase or decrease the
number of steps when a non-integer step size is used.

A step size of 1 is assumed if STEP is omitted from a
FOR statement.

A negative step size may be used, as shown in statement
40 above.

GENERAL MATHEMATICAL FUNCTIONS

EXAMPLES: 642 PRINT EXP(N); ABS(N)

                       652 IF RND (Ø)>=.5 THEN 9ØØ

                       662 IF INT (R) # 5 THEN 91Ø

                       672 PRINT SQR (X); LOG (X)

GENERAL FORM: The general mathematical functions may be used as
                       expressions, or as parts of an expression.

PURPOSE

Facilitates the use of common mathematical functions by pre-defining them as:

ABS (*expression*) the absolute value of the expression;

EXP (*expression*) the constant e raised to the power of the expression
                    value (in statement 642 above, $e{\uparrow}N$)

INT (*expression*) the largest integer $\leq$ the expression;

LOG (*expression*) the logarithm of the positively valued expression to the
                    base e;

RND (*expression*) a random number between 1 and Ø; the expression is a
                    dummy argument;

SQR (*expression*) the square root of the positively valued expression.

COMMENTS

The RND function is restartable; the sequence of random numbers using RND
is identical each time a program is RUN.

TRIGONOMETRIC FUNCTIONS

EXAMPLES:

```
500 PRINT SIN(X): COS(Y)
510 PRINT 3*SIN(B); TAN (C2)
520 PRINT ATN (22.3)
530 IF SIN (A2) <1 THEN 800
540 IF SIN (B3) = 1 AND SIN(X) <1 THEN 90
```

PURPOSE

Facilitates the use of common trigonometric functions by pre-defining them, as:

SIN (_expression_)   the sine of the expression (in radians);

COS (_expression_)   the cosine of the expression (in radians);

TAN (_expression_)   the tangent of the expression (in radians);

ATAN (_expression_)   the arctangent of the expression.

COMMENTS

The function is of the value of the expression (the value in parentheses, also called the argument).

The trigonometric functions may be used as expressions or parts of an expression.

ATN returns the angle in radians.

DEF FN

EXAMPLE: 60 DEF FNA (B2) = A$\uparrow$2 + (B2/C)

                   70 DEF FNB (B3) = 7*B3$\uparrow$2

                   80 DEF FNZ (X) = X/5

GENERAL FORM:

statement no. <u>DEF</u> <u>FN</u> *single letter A to Z* <u>(</u> *simple var.* <u>)</u> <u>=</u> *expression*

PURPOSE

Allows the programmer to define functions.

COMMENTS

A maximum of 26 programmer-defined functions are possible in a program (FNA to FNZ).

Any operand in the program may be used in the defining expression; however such circular definitions as:

              10 DEF FNA (Y) = FNB (X)

              20 DEF FNB (X) = FNA (Y)

cause infinite looping.

See the vocabulary at the beginning of this section for a definition of "function" and an explanation of "arguments".

COM

```
EXAMPLES:        1 COM A(10), B(3,3) first program
                 1 COM C(5), D(5), F(3,3) subsequent program

GENERAL FORM:

lowest statement no.COM subscripted array var., separated by commas
```

PURPOSE

Allows a BASIC program to store data in memory for retrieval
by a subsequent BASIC program.

COMMENTS

The data designated by a COM statement is accessible only as
an array; since COM designates a common array of data, the
same array variable can not appear in both DIM and COM state-
ments within a program.

COM must be the first statement entered and the lowest numbered
statement in a program.

COM, CONTINUED

The common area is a block of contiguous data in memory (two computer words per number.) The storage space is allotted in the order that the arrays appear in the COM statement; the elements within an array are stored row by row.

It is the user's responsibility to see that the portions of the common area are accessed properly by subsequent programs. For example, if the first program starts with the statement "1 COM A(10), B(3,3)" and a subsequent program with "1 COM C(5), D(5), F(3,3)", the common storage area elements are assigned as follows:

Element Position	First Program Reference	Second Program Reference
1	A(1)	C(1)
2	A(2)	C(2)
3	A(3)	C(3)
4	A(4)	C(4)
5	A(5)	C(5)
6	A(6)	D(1)
7	A(7)	D(2)
8	A(8)	D(3)
9	A(9)	D(4)
10	A(10)	D(5)
11	B(1,1)	F(1,1)
12	B(1,2)	F(1,2)
13	B(1,3)	F(1,3)
14	B(2,1)	F(2,1)
15	B(2,2)	F(2,2)
16	B(2,3)	F(2,3)
17	B(3,1)	F(3,1)
18	B(3,2)	F(3,2)
19	B(3,3)	F(3,3)

A reference in the first program to B(1,1) accesses the same element as a reference to F(1,1) in the second program. If A contained only 9 elements, however, the B(1,1) and F(1,1) references would access different elements.

The length of the common area may vary between programs, but for any two programs, information may be transferred only via the portion which is common to both.

If the first program declares "1 COM A(10), B(5,5)" and a succeeding program contains "1 COM D(10), E(5,5), F(10)", the values of F would be unpredictable. If the second program contained "1 COM D(10)" only, the contents of B would be destroyed.

THE TAB AND SGN FUNCTIONS

EXAMPLES:
```
500 IF SGN (X) # 0 THEN 800
510 LET Y = SGN(X)
520 PRINT TAB (5); A2; TAB (20)"TEXT"
530 PRINT TAB (N),X,Y,Z2
540 PRINT TAB (X+2) "HEADING"; R5
```

GENERAL FORM: *The TAB and SGN may be used as expressions, or parts of an expression. The function forms are:*

TAB (*expression indicating number of spaces to be moved*)

SGN (*expression*)

PURPOSE

TAB (*expression*) is used only in a PRINT statement, and causes the terminal typeface to move to the space number specified by the expression (0 to 71). The *expression* value after TAB is rounded to the nearest integer. Expression values greater that 71 cause a *return linefeed* to be generated.

SGN (*expression*) returns a 1 if the expression is greater than 0, returns a 0 if the expression equals 0, returns a -1 if the expression is less than 0.

SECTION IV

MATRICES

This section explains matrix manipulation. It is intended
to show the matrix capabilities of BASIC and assumes that
the programmer has some knowledge of matrix theory.

TERM: MATRIX (ARRAY)

DEFINED IN BASIC AS: An ordered collection of numeric data
(numbers).

Matrix elements are referenced by subscripts following the
matrix variable, indicating the row and column of the ele-
ment. For example, if matrix A is:

$$1 \quad 2 \quad 3$$
$$4 \quad 5 \quad 6$$
$$7 \quad 8 \quad 9$$

the element 5 is referenced by A(2,2); likewise, 8 is A(3,2).

See Section III,"Vocabulary" for a more complete description
of matrices

DIM

EXAMPLES:

```
11Ø DIM A (5Ø), B(2Ø,2Ø)
12Ø DIM Z (5,2Ø)
13Ø DIM S (5,25)
14Ø DIM R (4,4)
```

GENERAL FORM:

statement number DIM *matrix variable* (*integer*) ...

or

statement number DIM *matrix variable* (*integer* , *integer*) ...

PURPOSE

Reserves working space in memory for a matrix.

The maximum *integer* value (matrix bound) is 255.

COMMENTS

The integers refer to the number of matrix elements if only one dimension
is supplied, or to the number of rows and columns respectively, if two
dimensions are given.

A matrix (array) variable is any single letter from A to Z.

Arrays not mentioned in a DIM statement are assumed to have 10 elements
if one-dimensional, or 10 rows and columns if two-dimensional.

The working size of a matrix may be smaller than its physical size. For
example, an array declared 9 x 9 in a DIM statement may be used to store
fewer than 81 elements; the DIM statement supplies only an upper bound
on the number of elements.

The absolute maximum matrix size depends on the memory size of the computer.

MAT...ZER

EXAMPLES:

    3Ø5 MAT A = ZER
    31Ø MAT Z = ZER (N)
    315 MAT X = ZER (3Ø, 1Ø)
    32Ø MAT R = ZER (N, P)

GENERAL FORM:

statement number MAT *matrix variable* = ZER

          or

statement number MAT *matrix variable* = ZER (*expression*)

          or

statement number MAT *matrix variable* = ZER (*expression* , *expression*)

PURPOSE

Sets all elements of the specified matrix
equal to Ø; a new working size may be
established.

COMMENTS

The new working size in a MAT...ZER is an
implicit DIM statement, and may not exceed
the limit set by the DIM statement on the
total number of elements in an array.

Since Ø has a logical value of "false,"
MAT...ZER is useful in logical initialization.

MAT...CON

EXAMPLES:

```
205 MAT C = CON
210 MAT A = CON (N,N)
220 MAT Z = CON (5,20)
230 MAT Y = CON (50)
```

GENERAL FORM:

statement number MAT *matrix variable* = CON

      or

statement number MAT *matrix variable* = CON (*expression*)

      or

statement number MAT *matrix variable* = CON (*expression* , *expression*)

PURPOSE

Sets up a matrix with all elements equal to 1;
a new working size may be specified, within the
limits of the original DIM statement on the total
number of elements.

COMMENTS

The new working size (an implicit DIM statement)
may be omitted as in example statement 205.

Note that since 1 has a logical value of "true,"
the MAT...CON statement is useful for logical
initialization.

The expressions in new size specifications should
evaluate to integers. Non-integers are rounded
to the nearest integer value.

INPUTTING SINGLE MATRIX ELEMENTS

EXAMPLES:

```
6ØØ INPUT A(5)
61Ø INPUT B(5,8)
62Ø INPUT R(X), N, A(3,3),S,T
63Ø INPUT Z(X,Y), P3, W$
64Ø INPUT Z(X,Y), Z(X+1, Y+1), Z(X+R3, Y+S2)
```

GENERAL FORM:

statement number <u>INPUT</u> *matrix variable* <u>(</u> *expression* <u>)</u> ...

or

statement number <u>INPUT</u> *matrix variable* <u>(</u> *expression* <u>,</u> *expression* <u>)</u> ...

PURPOSE

Allows input of a specified matrix element
from the teleprinter.

COMMENTS

The subscripts (in *expressions*) used after the matrix
variable designate the row and column of the matrix
element. Do not confuse these expressions with work-
ing size specifications, such as those following a
MAT READ statement.

Expression used as subscripts should evaluate to
integers. Non-integers are rounded to the
nearest integer value.

Inputting, printing, and reading individual array
elements are logically equivalent to simple vari-
ables and may be intermixed in INPUT, PRINT, and
READ statements.

PRINTING SINGLE MATRIX ELEMENTS

```
EXAMPLES:            8ØØ PRINT A(3)
                     81Ø PRINT A(3,3);
                     82Ø PRINT F(X);E; C5;R(N)
                     83Ø PRINT G(X,Y)
                     84Ø PRINT Z(X,Y), Z(1,5), Z(X+N), Z(Y+M)
```

GENERAL FORM:

statement number PRINT *matrix variable* (*expression*) ...

                     or

statement number PRINT *matrix variable* (*expression* , *expression*) ...

PURPOSE

Causes the specified matrix element(s) to be printed.

COMMENTS

Expressions used as subscripts should evaluate to
integers. Non-integers are rounded to the
nearest integer value.

A trailing semicolon packs output into twelve
elements per teleprinter line, if possible (statement
81Ø above). A trailing comma or *return* prints five
elements per line.

Expressions (or subscripts) following the matrix
variable designate the row and column of the matrix
element. Do not confuse these with new working
size specifications, such as those following a MAT
IDN statement.

4-6

MAT PRINT

EXAMPLES:

        5ØØ MAT PRINT A

        5Ø5 MAT PRINT A;

        515 MAT PRINT A,B,C

        52Ø MAT PRINT A,B,C;

GENERAL FORM:

statement number MAT PRINT *matrix variable*

                      or

statement number MAT PRINT *matrix variable* , *matrix variable* ...

PURPOSE

Causes an entire matrix to be
printed, row by row, with double
spacing between rows.

COMMENTS

Matrices may be printed in "packed"
rows up to 12 elements wide by us-
ing the ";" separator, as in example
statement 5Ø5. Separation with commas
or a *return* prints 5 elements per row.

READING MATRIX ELEMENTS

EXAMPLES: 900 READ A(6)

910 READ A(9,9)

920 READ C(X); P; R7

930 READ C(X,Y)

940 READ Z(X,Y), P(R2, S5), X(4)

GENERAL FORM:

statement number READ *matrix variable* (*expression*)

or

statement number READ *matrix variable* (*expression* , *expression*) ...

PURPOSE

Causes the specified matrix element to be
read from the current DATA statement.

COMMENTS

Expressions (used as subscripts) should
evaluate to integers. Non-integers are
rounded to the nearest integer.

Expressions following the matrix variable
designate the row and column of the matrix
element. Do not confuse these with working
size specifications, such as those following
MAT READ statement.

The MAT READ statement is used to read an
entire matrix from DATA statements. See
details in this section.

MAT READ

EXAMPLES: 35Ø MAT READ A

                   37Ø MAT READ B(5),C,D

                   38Ø MAT READ Z (5,8)

                   39Ø MAT READ N (P3,Q7)

GENERAL FORM:

statement number MAT READ _matrix variable_

                   _or_

statement number MAT READ _matrix variable_ (_expression_) ...

                   _or_

statement number MAT READ _matrix variable_ (_expression_ , _expression_) ...

PURPOSE

Reads an entire matrix from DATA statements.
A new working size may be specified, within
the limits of the original DIM statement.

COMMENTS

MAT READ causes the entire matrix to be filled
from the current DATA statement in the row,
column order: 1,1; 1,2; 1,3; etc. In this
case, the DIM statement controls the number of
elements read.

MATRIX ADDITION

EXAMPLES: 31Ø MAT C = B + A

                   32Ø MAT X = X + Y

                   33Ø MAT P = N + M

GENERAL FORM:

statement number MAT *matrix variable* = *matrix variable* + *matrix variable*

PURPOSE

Establishes a matrix equal to the
sum of two matrices of identical
dimensions; addition is performed
element-by-element.

COMMENTS

The resulting matrix must be previously
mentioned in a DIM statement if it has
more than 10 elements, or 10 x 10 ele-
ments if two-dimensional. Dimensions
must be the same as the operand matrices.

The same matrix may appear on both sides
of the = sign, as in example statement 32Ø.

4-10

MATRIX SUBTRACTION

EXAMPLES: 550 MAT C = A - B

                     560 MAT B = B - Z

                     570 MAT X = X - A

GENERAL FORM:

statement number MAT *matrix variable* = *matrix variable* - *matrix variable*

PURPOSE

Establishes a matrix equal to the difference
of two matrices of identical dimensions; sub-
traction is performed element-by-element.

COMMENTS

The resulting matrix must be previously
mentioned in a DIM statement if it has more
than 10 elements, or 10 x 10 elements if
two-dimensional. Its dimension must be the
same as the operand matrices.

The same matrix may appear on both sides of
the = sign, as in example statement 560.

MATRIX MULTIPLICATION

EXAMPLES: 93Ø MAT Z = B * C

                   94Ø MAT X = A *A

                   95Ø MAT C = Z * B

GENERAL FORM:

statement number MAT *matrix variable* = *matrix variable* * *matrix variable*

PURPOSE

Establishes a matrix equal to the product
of the two specified matrices.

COMMENTS

Following the rules of matrix multiplication,
if the dimensions of matrix B = (P,N) and ma-
trix C = (N,Q), multiplying matrix B by matrix
C results in a matrix of dimensions (P,Q).

Note that the product matrix must have an
appropriate working size.

The same matrix variable may not appear on
both sides of the = sign.

SCALAR MULTIPLICATION

EXAMPLES:

110 MAT A = (5) * B
115 MAT C = (10) * C
120 MAT C = (N/3) * X
130 MAT P = (Q7*N5) * R

GENERAL FORM:

statement number MAT _matrix variable_ = (_expression_) * _matrix variable_

PURPOSE

Establishes a matrix equal to the pro-
duct of a matrix multiplied by a speci-
fied expression (number); that is, each
element of the original matrix is multi-
plied by the number.

COMMENTS

The resulting matrix must be previously
mentioned in a DIM statement if it con-
tains more than 10 elements (10 x 10
if two-dimensional).

The same matrix variable may appear on
both sides of the = sign.

Both matrices must have the same working
size.

COPYING A MATRIX

EXAMPLES: 4Ø5 MAT B = A
                   41Ø MAT X = Y
                   42Ø MAT Z = B

GENERAL FORM:

statement number MAT *matrix variable* = *matrix variable*

PURPOSE

Copies a specified matrix into a
matrix of the same dimensions;
copying is performed element-by-
element.

COMMENTS

The resulting matrix must be previously
mentioned in a DIM statement if it has
more than 10 elements, or 10 x 10 if two-
dimensional. It must have the same di-
mensions as the copied matrix.

IDENTITY MATRIX

EXAMPLES:

2Ø5 MAT A = IDN

21Ø MAT B = IDN (3,3)

215 MAT Z = IDN (Q5, Q5)

22Ø MAT S = IDN (6, 6)

GENERAL FORM:

statement number MAT *array variable* = IDN

or

statement number MAT *array variable* = IDN (*expression* , *expression*)

PURPOSE

Establishes an identity matrix (all Ø's, with
a diagonal from left to right of all 1's); a
new working size may be specified.

COMMENTS

The IDN matrix must be two-dimensional and square.

Specifying a new working size has the effect of a
DIM statement.

Sample identity matrix: 1 Ø Ø
                         Ø 1 Ø
                         Ø Ø 1

MATRIX TRANSPOSITION

```
EXAMPLES:              959 MAT Z = TRN (A)
                       969 MAT X = TRN (B)
                       979 MAT Z = TRN (C)

GENERAL FORM:

statement number MAT matrix variable = TRN ( matrix variable )
```

PURPOSE

Establishes a matrix as the transposition of
a specified matrix (transposes rows and columns).

COMMENTS

Sample transposition:

Original	Transposed
1 2 3	1 4 7
4 5 6	2 5 8
7 8 9	3 6 9

Note that the dimensions of the resulting matrix
must be the reverse of the original matrix. For
instance, if A has dimensions of 6,5 and MAT C =
TRN (A), C must have dimensions of 5,6.

Matrices cannot be transposed or inverted into
themselves.

MATRIX INVERSION

EXAMPLES: 38Ø MAT A = INV(B)

                       39Ø MAT C = INV(A)

                       4ØØ MAT Z = INV(Z)

GENERAL FORM:

statement number MAT matrix variable = INV (matrix variable)

PURPOSE

Establishes a square matrix as the inverse of
the specified square matrix of the same dimensions.

COMMENTS

The inverse is the matrix by which you multiply
the original matrix to obtain an identity matrix.

For example,

Original   Inverse   Indentity

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad X \quad \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \quad = \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Number representation in BASIC is accurate to 6-7
decimal digits; matrix elements are rounded
accordingly.

SECTION V

LOGICAL OPERATIONS

LOGICAL VALUES AND NUMERIC VALUES

A distinction should be made between logical values and
the numeric values produced by logical evaluation, when
using the logical capability of BASIC.

The <u>logical value</u> of an expression is determined by defi-
nitions established in the user's program.

The <u>numeric values produced by logical evaluation</u> are as-
signed by BASIC. The user may not assign these values.

Logical value is the value of an expression or statement,
using the criteria:

    any nonzero expression value = "true"
    any expression value of zero = "false"

When an expression or statement is logically evaluated,
it is assigned one of two numeric values, either:

    1, meaning the expression or statement is "true",
                        or
    0, meaning the expression or statement is "false".

RELATIONAL OPERATORS

There are two ways to use the relational operators in logical evaluations:

1. As a simple check on the numeric value of an expression.

EXAMPLES:

```
150 IF B=7 THEN 600
200 IF A9#27.65 THEN 700
300 IF (Z/10)>0 THEN 800
```

When a statement is evaluated, if the "IF" condition is currently true (for example, B = 7 in statement 150), then control is transferred to the specified statement; if it is not true, control passes to the next statement in the program.

Note that the numeric value produced by the logical evaluation is unimportant when the relational operators are used in this way. The user is concerned only with the presence or absence of the conditon indicated in the IF statement.

2. As a check on the numeric value produced by logically
    evaluating an expression, that is: "true" = 1, "false" = Ø.

EXAMPLES: 61Ø LET X=27

                           615 PRINT X=27

                           62Ø PRINT X#27

                           63Ø PRINT X>=27

The example PRINT statements give the numeric values produced by logi-
cal evaluation. For instance, statement 615 is interpreted by BASIC
as "Print 1 if X equals 27, Ø if X does not equal 27." There are only
two logical alternatives; 1 is used to represent "true," and Ø "false."

The numeric value of the logical evaluation is dependent on, but dis-
tinct from, the value of the expression. In the example above, X equals
27, but the numeric value of the logical expression X=27 is 1 since
it describes a "true" condition.

RELATIONAL OPERATORS

BOOLEAN OPERATORS

There are two ways to use the Boolean Operators.

1. As logical checks on the value of an expression or expressions.

EXAMPLES:

51Ø IF Al OR B THEN 67Ø

52Ø IF B3 AND C9 THEN 68Ø

53Ø IF NOT C9 THEN 69Ø

54Ø IF X THEN 7ØØ

Statement 51Ø is interpreted: "If either Al is true (has a non-zero value) or B is true (has a non-zero value), then transfer control to statement 67Ø

Similarly, statement 54Ø is interpreted: "If X is true (has a non-zero value), then transfer control to statement 7ØØ."

The Boolean operators evaluate expressions for their logical values only: these are "true" = any non-zero value, "false" = zero. For example, if B3 = 9 and C9 = -5, statement 52Ø would evaluate to "true," since both B3 and C9 have a non-zero value.

2. As a check on the numeric value produced by logically evaluating an expression, that is: "true" = 1, "false" = Ø.

EXAMPLES:

49Ø LET B = C = 7

5ØØ PRINT B AND C

51Ø PRINT C OR B

52Ø PRINT NOT B

Statements 5ØØ - 52Ø return a numeric value of either 1, indicating that the statement has a logical value of "true", or Ø, indicating a logical value of "false".

Note that the criteria for determining the logical values are:

> true = any non-zero expression value
> false = an expression value of Ø.

The numeric value 1 or Ø is assigned accordingly.

SOME EXAMPLES

These examples show some of the possibilities for combining logical operators in a statement.

It is advisable to use parentheses wherever possible when combining logical operators.

```
EXAMPLES:    31Ø IF (A9 AND B7)=Ø OR (A9 + B7)>1ØØ THEN 9ØØ
             31Ø PRINT (A>B) AND (X<Y)
             32Ø LET C = NOT D
             33Ø IF (C7 OR D4) AND (X2 OR Y3) THEN 93Ø
             34Ø IF (A1 AND B2) AND (X2 AND Y3) THEN 94Ø
```

The numerical value of "true" or "false" may be used in algebraic operations. For example, this sequence counts the number of zero values in data statements.

```
                    :
 9Ø LET X = Ø
1ØØ FOR I = 1 TO N
11Ø READ A
12Ø LET X = X+(A=Ø)
13Ø NEXT I
14Ø PRINT N; "VALUES WERE READ."
15Ø PRINT X; "WERE ZEROES."
16Ø PRINT (N-X); "WERE NONZERO."
                    :
```
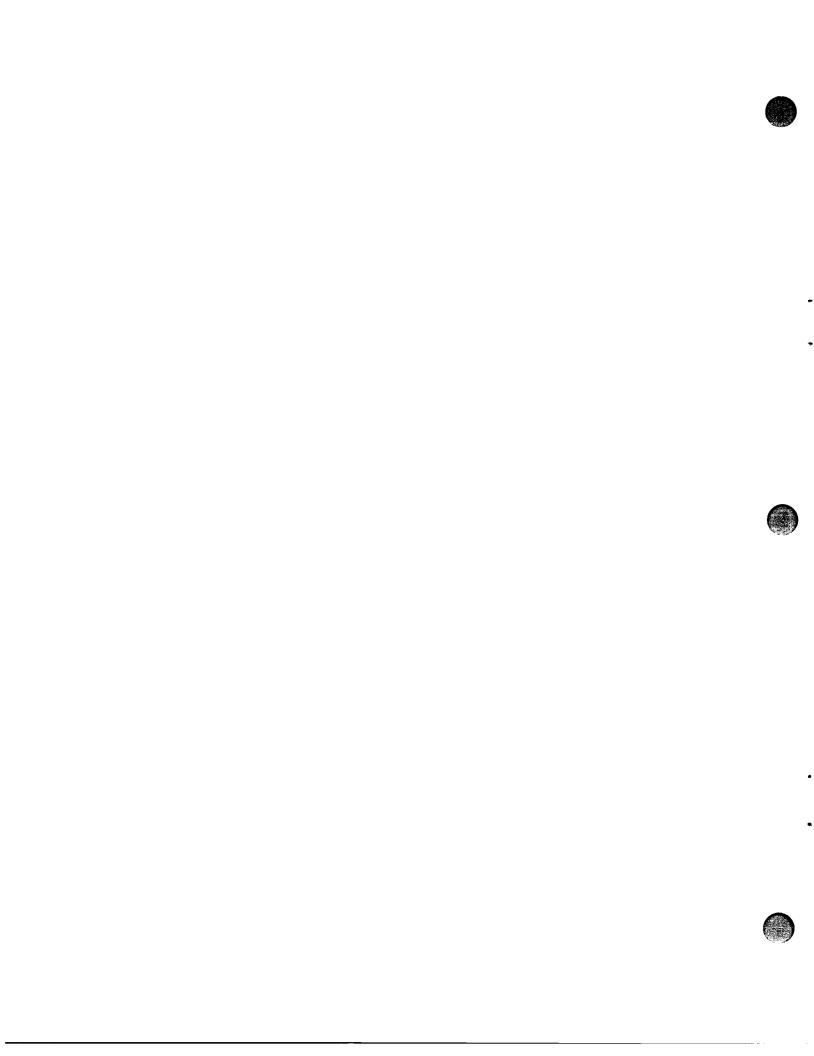
Note that X is increased by 1 or Ø each time A is read; when A = Ø, the expression A = Ø is true, and X is increased by 1.

SECTION VI

SYNTAX REQUIREMENTS OF BASIC

LEGEND

::= "is defined as..."

| "or"

< > enclose an element of BASIC

LANGUAGE RULES

1. The <com statement>, if any exists, must be the first statement
    presented and have the lowest sequence number; the last state-
    ment must be an <END statement>.

2. A sequence number may not exceed 9999 and must be non-zero.

3. Exponent integers may not have more than two digits.

4. A formal bound may not exceed 255 and must be non-zero.

5. A subroutine number must lie between 1 and 63, inclusive.

6. Strings may not contain the quote character (").

7. A <bound part> for an IDN must be doubly subscripted.

8. An array may not be inverted or transposed into itself.

9. An array may not be replaced by itself multiplied by another array.

SECTION VI

SYNTAX REQUIREMENTS

<basic program>   ::= <program statement>|<basic program><program statement> (1)

<program statement>   ::= <sequence number><basic statement>carriage return

<sequence number>   ::= <integer> (2)

<basic statement>   ::= <let statement>|<dim statement>|<com statement>|
                                   <def statement>|<rem statement>|<go to statement>|
                                   <if statement>|<for statement>|<next statement>|
                                   <gosub statement>|<return statement>|<end statement>|
                                   <stop statement>|<wait statement>|<call statement>|
                                   <data statement>|<read statement>|<restore statement>|
                                   <input statement>|<print statement>|<mat statement>

<let statement>   ::= <let head><formula>

<let head>   ::= LET<variable>=|<let head><variable>=

<formula>   ::= <conjunction>|<formula>OR<conjunction>

<conjunction>   ::= <boolean primary>|<conjunction>AND<boolean primary>

<boolean primary>   ::= <arithmetic expression>|<boolean primary>
                                   <relational operator><arithmetic expression>

<arithmetic expression>   ::= <term>|<arithmetic expression> + <term>|
                                   <arithmetic expression> - <term>

<term>   ::= <factor>|<term>*<factor>|<term>/<factor>

<factor>   ::= <primary>|<sign><primary>|NOT<primary>

<primary>   ::= <operand>|<primary>↑<operand>

<relational operator>   ::= >|<|>=|<=|=|#|<>

<operand>   ::= <variable>|<unsigned number>|<system function>|
                                   <function>|<formula operand>

<variable>   ::= <simple variable>|<subscripted variable>

<simple variable>   ::= <letter>|<letter><digit>

<subscripted variable>   ::= <array identifier><subscript head><subscript>
                                   <right bracket>

<array identifier>   ::= <letter>

<subscript head>   ::= <left bracket>|<left bracket><subscript>

<subscript>   ::= <formula>

<letter>   ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<digit>   ::= 0|1|2|3|4|5|6|7|8|9

<left bracket>   ::= (|[

<right bracket>   ::=)|]

<sign>   ::= +|-

<unsigned number>   ::= <decimal part>|<decimal part><exponent>

\<decimal part\>	::=	\<integer\>|\<integer\>.\<integer\>|.\<integer\>
\<integer\>	::=	\<digit\>|\<integer\>\<digit\>
\<exponent\<	::=	E\<integer\>|E\<sign\>\<integer\> [3]
\<system function\>	::=	\<system function name\>\<parameterpart\>
\<system function name\>	::=	SIN|COS|TAN|ATN|EXP|LOG|ABS|SQR|INT|RND|SGN
\<parameter part\>	::=	\<left bracket\>\<actual parameter\>\<right bracket\>
\<actual parameter\>	::=	\<formula\>
\<function\>	::=	FN\<letter\>\<parameter part\>
\<formula operand\>	::=	\<left bracket\>\<formula\>\<right bracket\>
\<dim statement\>	::=	DIM\<formal array list\>
\<formal array list\>	::=	\<formal array\>|\<formal array list\>,\<formal array\>
\<formal array\>	::=	\<array identifier\>\<formal bound head\>\<formal bound\>\<right bracket\>
\<formal bound head\>	::=	\<left bracket\>|\<left bracket\>\<formal bound\>,
\<formal bound\>	::=	\<integer\> (4)
\<com statement\>	::=	COM\<formal array list\>
\<def statement\>	::=	DEF FN\<letter\>\<left bracket\>\<formal parameter\>\<right bracket\>=\<formula\>
\<formal parameter\>	::=	\<simple variable\>
\<rem statement\>	::=	REM\<character string\>
\<character string\>	::=	any teletype character except carriage return, alt mode, escape, rubout, or line feed, or null, control B, control C, left arrow
\<goto statement\>	::=	GO TO\<sequence number\>
\<if statement\>	::=	IF\<formula\>THEN\<sequence number\>
\<for statement\>	::=	\<for head\>|\<for head\>STEP\<step size\>
\<for head\>	::=	FOR\<for variable\>=\<initial value\>TO\<limit value\>
\<for variable\>	::=	\<simple variable\>
\<initial value\>	::=	\<formula\>
\<limit value\>	::=	\<formula\>
\<step size\>	::=	\<formula\>
\<next statement\>	::=	NEXT\<for variable\>
\<gosub statement	::=	GOSUB\<sequence number\>
\<return statement\>	::=	RETURN
\<end statement\>	::=	END
\<stop statement\>	::=	STOP

SYNTAX

`<wait statement>`	`::= WAIT<parameter part>`
`<call statement>`	`::= CALL<call head><right bracket>`
`<call head>`	`::= <left bracket><subroutine number>|<call head>,`
	`<actual parameter>`
`<subroutine number>`	`::= <integer>` [(5)]
`<data statement>`	`::= DATA<constant>|<data statement>,<constant>`
`<constant>`	`::= <unsigned number>|<sign><unsigned number>`
`<read statement>`	`::= READ<variable list>`
`<variable list>`	`::= <variable>|<variable list>, <variable>`
`<restore statement>`	`::= RESTORE`
`<input statement>`	`::= INPUT<variable list>`
`<print statement>`	`::= <print head>|<print head><print formula>`
`<print head>`	`::= PRINT|<print head><print part>`
`<print part>`	`::= <string>|<string><delimiter>|<print formula>`
	`<delimiter>|<print formula><string>|`
	`<print formula><string><delimiter>`
`<string>`	`::= "<character string>"` [(6)]
`<delimiter>`	`::= ,|;`
`<print formula>`	`::= <formula>|TAB<parameter part>`
`<mat statement>`	`::= MAT<mat body>`
`<mat body>`	`::= <mat read>|<mat print>|<mat replacement>`
`<mat read>`	`::= READ<actual array>|<mat read>, <actual array>`
`<actual array>`	`::= <array identifier>|<array identifier><bound part>`
`<bound part>`	`::= <actual bound head><actual bound><right bracket>`
`<actual bound head>`	`::= <left bracket>|<left bracket><actual bound>,`
`<actual bound>`	`::= <formula>`
`<mat print>`	`::= PRINT<mat print part>|PRINT<mat print part><delimiter>`
`<mat print part>`	`::= <array identifier>|<mat print part><delimiter>`
	`<array identifier>`
`<mat replacement>`	`::= <array identifier>=<mat formula>`
`<mat formula>`	`::= <array identifier>|<mat function>|<array identifier>`
	`<mat operator><array identifier>|<formula operand>*`
	`<array identifier>`

<mat function>	::=	<mat initialization>|<mat initialization><bound part>| INV<array parameter>|TRN<array parameter>
<mat initialization>	::=	ZER|CON|IDN[7]
<array parameter>	::=	<left bracket><array identifier><right bracket>[8]
<mat operator>	::=	+|-|*[9]

SYNTAX

SECTION VII

FOR ADVANCED PROGRAMMERS

MODIFYING HP BASIC

As indicated in the configuration instructions, an HP BASIC
system configured with PBS may include user-written assembly
language subroutines. These subroutines are accessed with a
CALL statement while a BASIC program is running. HP BASIC
may also be run under the HP Magnetic Tape System (MTS), pro-
vided that the amount of core memory in the configured tape of
HP BASIC is the same as the MTS under which it is run.

The information in this section is intended to help the pro-
grammer in modifying HP BASIC. Users are reminded that HP
cannot be responsible for non-standard or user-modified
software.

CALL

```
EXAMPLE:        2Ø CALL (5, A(10),1, 1188, 10)

GENERAL FORM:   statement number CALL ( statement number , parameter list )
```

PURPOSE

Allows addition of absolute assembly language routines (such as input-output drivers) to BASIC, for specialized configurations. CALL transfers control to the specified assembly language subroutine.

COMMENTS
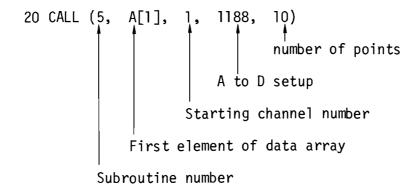
Subroutines executed by CALL are not constrained by BASIC and have absolute control of the computer. The assembly language subroutine may, therefore, alter any portion of the system, including BASIC. For this reason, it is recommended that only programmers proficient in assembly language attempt to add CALL subroutines to BASIC programs.

CALL subroutines are "loaded into the computer" through the photoreader or terminal tape reader either at configuration time or as a load-time overlay.

The CALL _subroutine number_ is a positive integer between 1 and 63 specifying the desired subroutine. If no such subroutine number exists, the statement is rejected.

CALL, CONTINUED

The other parameters, separated by commas, may be any formula and their number
is dependent upon the subroutine called. For example, a subroutine designated
by 5 is appended to the system to take readings from an A to D subsystem and
store them in an array. The parameters specify the array into which the val-
ues are inserted, the channel number of the first point to be measured, the
setup for the A to D converter and the number of points to be measured. A
representative call for this subsystem is:

```
20 CALL (5,  A[1],  1,  1188,  10)
                                    number of points

                                A to D setup

                          Starting channel number

              First element of data array

        Subroutine number
```

When using the CALL statement, it is important that correct parameters be
specified. Accidentally reversing the first and second parameters could de-
stroy the core-resident BASIC system, unless precautions have been taken by
the writer of the called subroutine to protect the BASIC system.

The parameters of a CALL statement provide the dynamic link between BASIC and
the called subroutine. Prior to transferring control to the subroutine, BASIC
evaluates the parameters and stacks the addresses of the results. Upon enter-
ing the subroutine, the A-register contains the address of this stack (i.e.,
the address of the addresses of the parameter values.) The A-register initial-
ly points to the address of the first parameter; successively decrementing the
A-register causes it to point to successive parameter addresses. For example,
if the A-register = 17302 when a subroutine is entered, the first parameter
address is 17302, the second 17301, the third 17300, etc.

CALL

The parameter addresses passed by BASIC give the subroutine access to values
in the BASIC program. The only way a called subroutine can transmit results
to a BASIC program is to store them by means of a parameter address.

Transmittal of quantities of data between a BASIC program and a called sub-
routine is most conveniently handled through arrays. Since only addresses are
passed to a subroutine, an array parameter must be an element of the array (in
general this would be the first element of the array). It is important to re-
member that arrays are stored by rows, and that each element is a floating
point number occupying two (16-bit) words. Hence, if an array A has M columns
per row, the address of A[I,J] is (address A[1,1]+ 2(M(I-1) + (J-1))).

To output from a subroutine to the terminal:

1. Load a buffer address into the B-register.
2. Load a character count into the A-register.
3. Execute a JSB 102B, I.

The referenced block of core is then interpreted as an ASCII string and output,
followed by a *return linefeed* if the count was negative.

Whenever data is transferred from a called subroutine through the address of a
parameter, there is a danger that the BASIC system or a program might be de-
stroyed. This situation can arise when parameters are specified incorrectly
or insufficient space is allocated in a data array. For example, constants
such as 2 or -1.1 in a BASIC program are stored in the program as they appear;
therefore, storing through the address of a constant parameter changes the
actual constant in the CALL statement. A subsequent execution of that state-
ment may lead to unpredictable results. A parameter that is an expression
(for example, A AND B or NOT A AND B) is evaluated and the result placed in a
temporary location. Since the parameter address references this temporary

location, storing into it will not harm the BASIC system or program. However, the value stored there is lost to the BASIC program. If a called subroutine stores more values in an array than the array can hold, the resulting overflow of data may destroy the BASIC system or program.

Users of CALL statements should be cautioned against using unsuitable parameters in CALL statements (especially against using a simple variable or a constant where an array element is expected). Also, when using arrays as parameters it is good practice to include the dimensions of the array as additional parameters to allow a means of checking within the subroutine.

An effective protection requires additional programming effort. BASIC contains sets of pointers delimiting the areas of memory within which different types of parameters exist. By checking parameter addresses against these bounds, the subroutine can verify that they are of the expected type. If X represents the parameter address, the following applies:

    a. Constant parameter $(112_8) < X < (113_8)$

    b. Simple variable parameter $(116_8) < X < (117_8)$

    c. Array parameter 1) In common storage $(110_8) \leq X < (112_8)$

                         2) Not in common storage $(113_8) \leq X < (115_8)$

    d. Expression parameter $(115_8) < X < (120_8)$

    where (112_8) means the contents of location number octal 112.

CALL

BYE

```
EXAMPLES:              BYE

GENERAL FORM:          BYE
```

PURPOSE

Produces a HLT 77_8 when used under the HP BASIC system;
or causes transfer of control from the HP BASIC system
to the Magnetic Tape System (MTS) executive when used in
an MTS based HP BASIC system.

COMMENTS

HP BASIC may be configured as part of an HP Magnetic Tape
System.

If it is intended to run under the Magnetic Tape System, PBS
may be configured separately or together with the HP BASIC
interpreter.

User-written assembly language subroutines may be added to an
MTS based HP BASIC system; they may be configured along with
the drivers and interpreter using PBS or added while preparing
the MTS.

Note that configuration of an HP BASIC system cannot be done
under the control of an MTS, rather a configured system may be
one of the subsystems supplied when creating an MTS.

Remember that an HP BASIC system running under MTS must specify
the same core memory size as the MTS.

FIRST AND LAST WORDS OF AVAILABLE MEMORY

> The first word of available memory (FWAM) is
> contained in location 110_8 in the HP BASIC sys-
> tem.
>
> The last word of available memory (LWAM) is con-
> tained in location 111_8 in the HP BASIC system.

COMMENTS

When HP BASIC is run under MTS, FWAM is contained
in location 110_8; LWAM is dynamically determined
and placed in location 106_8 after the system is
loaded.

FIRST WORD AVAILABLE IN BASE PAGE

The address of the first word available in base
page is contained in location 114_8. All locations
from the location referenced in 114_8 through 1777_8
are not used by BASIC, and are therefore available
for CALL subroutines or other modifications.

MEMORY

LINK POINTS

For ease in user modification, locations 201_8
through 322_8 contain links to various sub-
portions and subroutines of BASIC in creating
customized systems. The identity and locations
of these links is fixed (will not change with
subsequent versions), but the contents of these
locations are subject to change if the routines
they point to move as a result of future re-
visions. The assembly language listings of the
HP BASIC interpreter captions each link briefly.
Since these links are an integral part of BASIC,
the user is responsible for interpreting and
using this information.

LINKS

LINKAGES TO SUBROUTINES

BASIC accesses called subroutines through a table containing
linkage information. Entries in the table, one per subroutine,
are two words in length. Bits 5-0 of the first word contain
the number identifying the subroutine (chosen freely from 1 to
77_8 inclusive) and bits 15-8 contain the number of parameters
passed to the subroutine. (CALL statements with an incorrect
number of parameters are rejected by the syntax analyzer.)
The second word contains the absolute address of the entry point
of the subroutine. (Control is transferred via a JSB.) Although
subroutine numbers need not be assigned in any particular order,
all entries in the table must be contiguous. An acceptable aux-
iliary tape contains the following:

1. An ORG statement to origin the program at an address greater
    than that of the last word of the BASIC system. The address
    of this last word + 1 is contained in location 110_8 of the
    standard BASIC system. Hence, a suitable lower limit for the
    origin address can be determined by loading BASIC and examin-
    ing location 110_8.

2. The subroutine linkage table described above.

3. The assembly language subroutines.

4. Code to set the following linkage addresses:

    a. In location 110_8 put the address of the last word + 1 used
        in the auxiliary tape.

    b. In location 121_8 put the address of the first word of the
        subroutine linkage table.

    c. In location 122_8 put the address of the last word + 1 of the
        subroutine linkage table.

LINKS

Assuming, for example, that location 110_8 of the standard
BASIC system contains 13142_8; an acceptable auxiliary tape
could be assembled from the following code:

```
          ORG 13142B
SBTBL     OCT 2406     Subroutine 6 has 5 parameters
          DEF SB6
          OCT 1421     Subroutine 17 has 3 parameters
          DEF SB17
ENDTB     EQU *
SB6       NOP

            :          Subroutine #6 body

          JMP SB6,I
SB17      NOP

            :          Subroutine #17 body

          JMP SB17, I
LSTWD     EQU *
          ORG 110B
          DEF LSTWD
          ORG 121B
          DEF SBTBL
          DEF ENDTB
          END
```

Acceptable calls to subroutines SB6 and SB17 might be

        CALL (6, A, B, 1, N*3, SIN(X+Y)

        CALL (17, A[1], 5, N)

NOTE: *Location 111_8 of the standard BASIC system contains*
       the address of the last word of available memory. It
       is not possible to create a system which requires more
       space than that existing between the addresses in loca-
       tions 110_8 and 111_8. Systems using all or most of this
       space leave very little space for the user of the system.

HOW TO MAKE MORE PROGRAM SPACE
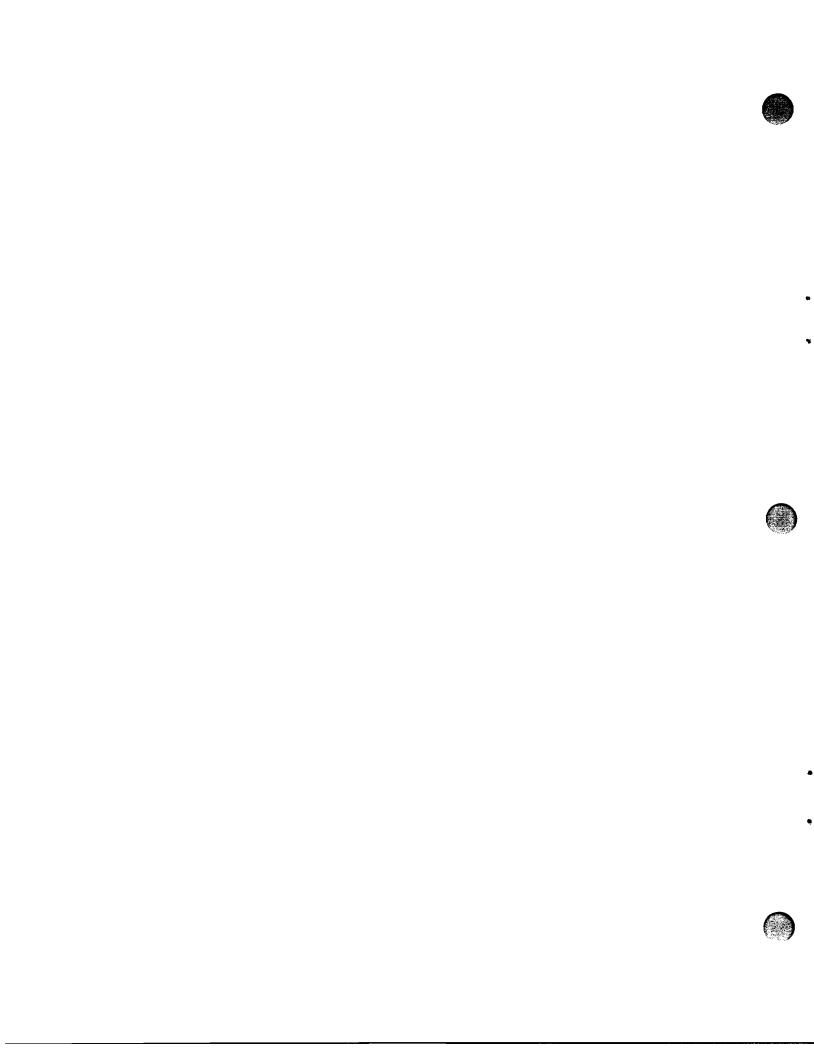
(DELETING THE MATRIX SUBROUTINES)

This assembly language pseudo-program shows a method of
deleting the MAT execution package to gain more user space,
or for replacing it with CALL routines or other customized code.

$$ORG \quad <contents \ of \ 210_8>$$
$$OCT \quad 0,0$$
$$ORG \quad 110B$$
$$DEF \quad <contents \ of \ 211_8>$$

This sequence has the effect of preventing the syntax processor
from recognizing "MAT" and of resetting the first word of avail-
able memory pointer to the first word of the matrix execution
package.

PROGRAM
SPACE

SECTION VIII

OPERATING INSTRUCTIONS

The minimum hardware configuration for HP BASIC is a computer
(HP 2114, 2115, 2116 series) with 8K of core memory and an HP
modified ASR-33 or ASR-35 teletype terminal. A photoreader
and high-speed paper tape punch are optional.

Minimum software modules for an HP BASIC system are the HP BASIC
interpreter binary tape and the Prepare BASIC System (PBS) binary
tape. Additional user-written assembly language subroutines may
be included in an HP BASIC system.

The HP BASIC tape contains the BASIC interpreter only. The PBS
tape contains drivers for the terminal, photoreader, high-speed
punch, and the routines necessary to configure these elements
into an HP BASIC system. User-written assembly language sub-
routines may be included in a configuration produced by PBS.

Since HP BASIC is designed to allow the user to include "custom"
software such as CALL subroutines in his HP BASIC system, user-
written subroutines or modifications may be included on an HP
BASIC system tape produced by PBS, or they may be loaded into
an HP BASIC system separately through the photoreader or term-
inal tape reader.

The following pages explain how to configure an HP BASIC system
using PBS, and how to load the configured HP BASIC tape into the
computer. Steps are included in the PBS preparation procedures
for including user-written subroutines on the composite system
tape produced by PBS.

The most convenient method of loading a configured HP BASIC
system is to have included all software elements of the HP
BASIC system on the composite tape produced by PBS. Users
with constantly changing CALL subroutines or modifications
to the standard software have the option of configuring a
system with PBS, (without including the other software ele-
ments) and then loading the various other elements (such as
the BASIC interpreter, CALL subroutines, and modifications)
into the computer separately. If the user chooses to load
non-standard software tapes separately from the PBS configured
system tape and the BASIC interpreter, they must be loaded af-
ter the configured system tape and the interpreter (loaded
separately if not on the configured tape).

INSTRUCTIONS

HP 2114 COMPUTER

HOW TO LOAD A CONFIGURED HP BASIC SYSTEM TAPE

1. Make sure the computer and photoreader (tape reader) are turned on.

2. Make sure the terminal control switch is in the LINE position.

3. Put the tape in the photoreader (tape reader) with

    a. the arrows on the tape facing up, and to the right;

    b. the tape threaded through the guides, under the lamp and between the rollers. (With a tape reader, you may have to press the LOAD button first.)

4. Raise the tape guide (exposing the word RUN); or with the tape reader, press the READ button.

5. Touch the PRESET and LOAD buttons simultaneously. (The tape is read by the computer.)

6. When the tape stops, check the MEMORY DATA register for 102077_8 (lights 15,10,5,4,3,2,1,0 on).

7. Remove and rewind the tape.

8. Touch the CLEAR REGISTER button.

9. Set the SWITCH REGISTER to 100_8 (Touch button 6.)

10. Touch the LOAD ADDRESS button.

11. Touch the RUN button.

    BASIC responds: READY on the terminal.

HP 2114

HOW TO LOAD A CONFIGURED HP BASIC SYSTEM TAPE

1. Make sure the computer, power supply, and photoreader (tape reader) are turned on.

2. Make sure the terminal control switch is in the LINE position.

3. Put the tape in the photoreader (tape reader) with:

   a. the arrows on the tape facing up, and to the right.

   b. the tape threaded through the guides, under the lamp and between the rollers. (With a tape reader, you may have to press the LOAD button first.)

4. Raise the tape guide (exposing the word RUN); or on the tape reader, press the READ button.

5. Set the SWITCH REGISTER to 077700_8 by putting switches 14,13,12,11,9,8,7,6 in the up position; all others should be down.

6. Press the LOAD ADDRESS button.

7. Set the SWITCH REGISTER TO 000000 (all switches down).

8. Move the LOADER switch to the ENABLED position.

9. Press the PRESET button.

10. Press the RUN button. (Tape is read by the computer.)

11. When the tape stops, check the T-REGISTER for 102077_8 (lights 15,10,5,4,3,2,1,0 on).

12. Move the LOADER switch to the PROTECTED position.

13. Remove and rewind the tape.

14. Set the SWITCH REGISTER to 100_8 (switch 6 in the up position; all others down).

15. Press the LOAD ADDRESS button.

16. Press the PRESET button.

17. Press the RUN button.

    BASIC responds: READY on the terminal.

CONFIGURING A BASIC SYSTEM USING PBS TAPE

NOTE: This set of instructions is not restartable before Step 21.

1. Make sure that the computer tape reader (photoreader) and high-speed punch are turned on.

2. Make sure the terminal control switch is in the LINE position.

3. Touch the HALT button.

4. Place the PBS binary tape in the photoreader or tape reader:

   a. Arrows on the tape face up and point to the right.

   b. Thread the tape through the guide, under the lamp, and between the rollers. (With the tape reader, press the LOAD button.)

5. Raise the tape guide, exposing the word RUN. (On the tape reader, press the READ button.)

6. Touch the CLEAR REGISTER button.

7. Touch the PRESET and LOAD buttons simultaneously. (Tape is read by the computer.)

8. When the tape stops, check the MEMORY DATA register for 102077_8 (lights 15,10,5,4,3,2,1,0 on).

9. Remove and rewind the tape.

10. Repeat steps 3 through 9, using the BASIC binary tape; then go to step 11.

11. Touch the CLEAR REGISTER button.

12. Set the SWITCH REGISTER to 000002_8 (touch button 1).

13. Touch the LOAD ADDRESS button.

14. Touch the CLEAR REGISTER button.

15. Set the SWITCH REGISTER to the select code of the terminal (octal number, right-justified) by touching the appropriate SWITCH REGISTER buttons.*

16. Touch the RUN button.

*Touch button 15 if a serial teletype driver is desired; a buffered teletype terminal driver is provided if button 15 is not set.

HP 2114

CONFIGURING A BASIC SYSTEM USING PBS TAPE

17. PBS queries: PHOTOREADER I/O ADDRESS?

    Enter the select code for the photoreader from the terminal keyboard
    (octal number), followed by a _return_. If there is no photoreader, type
    a _return_ only.

18. PBS queries: PUNCH I/O ADDRESS?

    Enter the select code for the high-speed tape punch (octal number) from
    the terminal keyboard, followed by a _return_. If there is no punch, type
    a _return_.

19. PBS queries: SYSTEM DUMP I/O ADDRESS?

    Enter the select code of the high-speed tape punch (octal number) from
    the terminal keyboard, followed by a _return_. If there is no punch, type
    a _return_.

20. PBS queries: CORE SIZE?

    Enter the core memory size of the computer on which the HP BASIC system
    is to be run (8,16,24, or 32), followed by a _return_. Typing only a _return_
    indicates an 8K core memory size.

21. PBS then punches a configured BASIC tape on the high-speed punch. If
    there is no punch, the message: TURN ON TTY PUNCH, PRESS RUN is printed.
    Respond by turning the terminal punch on and pressing the RUN button.

22. Touch the CLEAR REGISTER button.

23. Repeat steps 3 to 9 above, using the configured tape, then go to step 24.

24. Set the SWITCH REGISTER to 100_8 (touch button 6).

25. Touch the LOAD ADDRESS button.

26. Touch the RUN button. BASIC responds: READY.

HP 2115, HP 2116 COMPUTERS

CONFIGURING A BASIC SYSTEM USING PBS TAPE

NOTE: *The PBS loading sequence is not restartable. If you make a mistake or the tapes do not load properly, begin at step 1.*

1. Make sure the computer, power supply, photoreader, and high speed punch are turned on.

2. Turn the terminal control switch to the LINE position.

3. Press the HALT button (on the computer).

4. Move the LOADER switch to the PROTECTED position.

5. Set the SWITCH REGISTER to 077700_8 by putting switches 14,13,12,11,10,9, 8,7,6 in the up position; all others should be down.

6. Press the LOAD ADDRESS button.

7. Check the P-Register and M-Register for 077700_8 (lights 14,13,12,11,10,9, 8,7,6 on).

8. Put the tape in the photoreader (on the tape reader, press the LOAD, button first) with:

    a. Arrows on facing up and pointing to the right.
    b. Tape threaded through the guide, under the lamp, and between the rollers.

9. Raise tape guide, exposing the word RUN. (The rollers start moving.) With the tape reader, press the READ button.

10. Set SWITCH REGISTER to 000000 (switches 0 to 15 down).

11. Make sure the LOADER switch is in the ENABLED position.

12. Press the PRESET button.

13. Press the RUN button. (Tape is read.)

14. When the tape stops moving, check the T-REGISTER for 102077_8 (lights 15,10,5,4,3,2,1,0 on).

15. Remove and rewind the tape.

16. Repeat steps 8 to 15, using the BASIC binary tape instead of PBS; then go to step 17.

17. Repeat steps 8 to 15, using user-written assembly language routine tapes (if any); then go to step 18.

18. Move the LOADER switch to the PROTECTED position.

19. Set the SWITCH REGISTER to 000002_8 by putting switch 1 in the up position; all others are down.

20. Press the LOAD ADDRESS button.

21. Set SWITCH REGISTER to the select code for the terminal (octal number, right justified) using switches 0 to 5.*

22. Press the RUN button.

23. PBS queries: PHOTOREADER I/O ADDRESS?

    Enter the select code for the photoreader from the terminal keyboard (octal number), followed by a _return_. If there is no photoreader, type a _return_ only.

24. PBS queries: PUNCH I/O ADDRESS?

    Enter the select code for the high speed tape punch (octal number) from the terminal keyboard, followed by a _return_. If there is no punch, type a _return_.

25. PBS queries: SYSTEM DUMP I/O ADDRESS?

    Enter the select code of the high speed tape punch (octal number) from the terminal keyboard, followed by a _return_. If there is no punch, type a _return_.

26. PBS queries: CORE SIZE?

    Enter the core memory size of the system or which HP BASIC is to be run (8,16,24, or 32) followed by a _return_. Typing only a _return_ indicates an 8K core memory.

27. Tear the punched tape off, and rewind it.

28. See "How to Load a Configured HP BASIC System Tape" for further instructions.

---

*Set switch 15 in the up position if using a serial teletype terminal; leaving switch 15 down configures the teletype terminal driver for a buffered teletype.

APPENDIX A

HOW TO PREPARE PAPER TAPE OFF-LINE
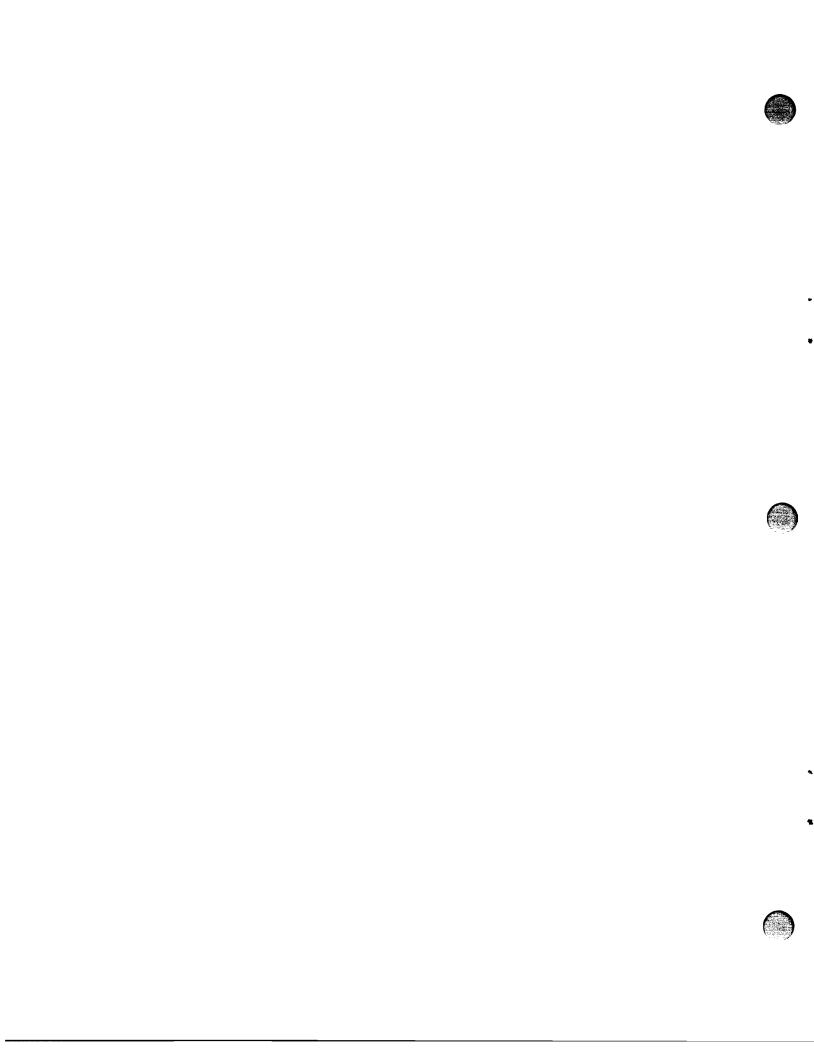
To prepare a paper tape for input:

1. Turn teleprinter control knob to "LOCAL".

2. Press the "ON" button (on tape punch).

3. Press the "HERE IS" key; or press @^{C} (control shift "p") several times to put leading holes on the tape.

4. Type program as usual, following each line with _return_ _linefeed_.

5. Press "HERE IS"; or press @^{C} several times to put trailing holes on the tape.

6. Press the "OFF" button on the tape punch.

COMMENTS

The standard on-line editing features, such as _esc,_ ←, and repeating the same line number may be punched on tape; _esc_ must be followed by _return_ _linefeed_.

Pressing the "B.SP." (backspace) button on the tape punch, then the "RUBOUT" key physically deletes the previous character from a paper tape.

APPENDIX B

SAMPLE PROGRAMS

BATNUM

DESCRIPTION

This program simulates a game called "The Battle of Numbers."

INSTRUCTIONS

The game is played with a pile of objects, some of which are
removed by you and the machine. You must specify whether
winning is defined as taking or not taking the last object,
the original number of objects in the pile, who goes first,
and the minimum and maximum number of objects that can be re-
moved at one time. Typing Ø for your move causes a forfeit,
and typing Ø for the pile size terminates the game.

```
10   PRINT "THIS PROGRAM PLAYS 'THE BATTLE OF NUMBERS.'"
20   PRINT
30   PRINT "IF YOU NEED INSTRUCTIONS TYPE A 1, OTHERWISE TYPE A 2: ";
40   INPUT I
50   IF I=1 THEN 70
60   GOTO 160
70   PRINT "THE GAME IS PLAYED WITH A PILE OF OBJECTS, SOME OF"
80   PRINT "WHICH ARE REMOVED ALTERNATELY BY YOU AND THE MACHINE."
90   PRINT "YOU MUST SPECIFY WHETHER WINNING IS DEFINED AS TAKING"
100   PRINT "OR NOT TAKING THE LAST OBJECT, THE ORIGINAL NUMBER OF"
110   PRINT "OBJECTS IN THE PILE, WHO GOES FIRST, AND THE MINIMUM"
120   PRINT "AND MAXIMUM NUMBER OF OBJECTS WHICH CAN BE REMOVED AT"
130   PRINT "ONE TIME.  TYPING 'Ø' FOR YOUR MOVE WILL CAUSE A"
140   PRINT "FORFEIT, AND TYPING 'Ø' FOR THE PILE SIZE WILL CAUSE"
150   PRINT "THE TERMINATION OF THE GAME."
160   PRINT "ENTER PILE SIZE: ";
170   INPUT N
180   IF N=Ø THEN 980
190   IF N=INT(N) THEN 210
200   GOTO 160
210   IF N<1 THEN 160
```

```
220     PRINT "ENTER WIN OPTION - 1 TO TAKE LAST, 2 TO AVOID LAST: ";
230     INPUT M
240     IF M=1 THEN 270
250     IF M=2 THEN 270
260     GOTO 220
270     PRINT "ENTER MIN AND MAX: ";
280     INPUT A,B
290     IF A>B THEN 270
300     IF A<1 THEN 270
310     IF A=INT(A) THEN 330
320     GOTO 270
330     IF B=INT(B) THEN 350
340     GOTO 270
350     PRINT "ENTER START OPTION - 1 MACHINE FIRST, 2 YOU FIRST: ";
360     INPUT S
370     IF S=1 THEN 400
380     IF S=2 THEN 400
390     GOTO 350
400     LET C=A+B
410     IF S=2 THEN 440
420     GOSUB 470
430     IF W=1 THEN 160
440     GOSUB 680
450     IF W=1 THEN 160
460     GOTO 420
470     LET Q=N
480     IF M=1 THEN 500
490     LET Q=Q-1
500     IF M=1 THEN 550
510     IF N>A THEN 590
520     LET W=1
530     PRINT "MACHINE TAKES";N;"AND LOSES"
540     RETURN
550     IF N>B THEN 590
560     LET W=1
570     PRINT "MACHINE TAKES";N;"AND WINS"
580     RETURN
590     LET P=Q-C*INT(Q/C)
600     IF P >= A THEN 620
610     LET P=A
620     IF P <= B THEN 640
630     LET P=B
640     LET N=N-P
650     PRINT "MACHINE TAKES";P;"AND LEAVES";N
660     LET W=0
670     RETURN
680     PRINT "YOUR MOVE: ";
690     INPUT P
700     IF P=0 THEN 720
710     GOTO 750
720     PRINT "MACHINE WINS BY FORFEIT"
```

```
730    LET W=1
740    RETURN
750    IF P=INT(P) THEN 770
760    GOTO 810
770    IF P >= A THEN 800
780    IF P=N THEN 860
790    GOTO 810
800    IF P <= B THEN 830
810    PRINT "ILLEGAL MOVE, REENTER IT: ";
820    GOTO 690
830    LET N=N-P
840    IF N=0 THEN 860
850    GOTO 930
860    IF M=1 THEN 900
870    PRINT "YOU LOSE"
880    LET W=1
890    RETURN
900    PRINT "YOU WIN"
910    LET W=1
920    RETURN
930    IF N >= 0 THEN 960
940    LET N=N+P
950    GOTO 810
960    LET W=0
970    RETURN
980    STOP
990    STOP
1000   END
```

FACTOR

DESCRIPTION

This program finds the prime factors of a number.

INSTRUCTIONS

The program requests the number to be factored and prints
out all prime factors and their multiplicity. Input a
zero or a negative number to terminate execution.

RESTRICTION

The number to be factored must be a positive integer less
than 32768.

```
9002    REM   FINDS PRIME FACTORS
9003    PRINT "PROGRAM TO FIND PRIME FACTORS OF A POSITIVE INTEGER."
9004    PRINT "TO TERMINATE EXECUTION INPUT A '0'."
9005    PRINT
9006    PRINT "WHAT NUMBER IS TO BE FACTORED";
9007    INPUT A
9008    IF A <= 32767 THEN 9012
9009    PRINT "SORRY!THIS PROGRAM IS ONLY DESIGNED TO FACTOR NUMBERS"
9010    PRINT "OF 5 DIGITS UP TO 32767 OR LESS!PLEASE TRY AGAIN"
9011    GOTO 9005
9012    LET D=A
9013    PRINT
9014    IF A=2 THEN 9044
9015    LET Q=0
9016    IF A>0 THEN 9018
9017    STOP
9018    LET C=2
9019    GOSUB 9023
9020    FOR C=3 TO SQR(A) STEP 2
9021    GOSUB 9023
9022    GOTO 9039
9023    LET B=0
9024    IF A=C*INT(A/C) THEN 9026
9025    GOTO 9029
9026    LET A=A/C
9027    LET B=B+1
9028    GOTO 9024
```

```
9029    IF B<1 THEN 9038
9030    IF Q=1 THEN 9037
9031    LET Q=1
9032    PRINT "THE PRIME FACTORS OF";D;"ARE:"
9033    PRINT
9034    PRINT "PRIME","MULTIPLICITY"
9035    PRINT "-----","------------"
9036    PRINT
9037    PRINT C,B
9038    RETURN
9039    NEXT C
9040    IF A=1 THEN 9005
9041    IF Q=0 THEN 9044
9042    PRINT A,1
9043    GOTO 9005
9044    PRINT "THE NUMBER";A;"IS PRIME"
9045    GOTO 9005
9046    STOP
9999    END
```

PLOT

DESCRIPTION

This program plots a given function on the terminal. It checks for minimum and maximum Y values over the domain (excluding the undefined points), calculates the Y-axis spacing, and plots the function.

INSTRUCTIONS

Define the function in line 1∅ by:

$$\text{DEF FNF}(X) = \dots$$

Example: 1∅ DEF FNF(X) = 25*COS(X)*SIN(X 2/2)/(X 2+1)

Type RUN. The program requests the following information:

    1. left X end point

    2. right X end point

    3. desired X increment

    4. points on the X-axis for which the function is undefined (for example, the denominator becomes zero).

The output gives:

    1. minimum Y

    2. maximum Y

    3. Y-axis spacing

    4. the plot with the Y-axis horizontal and the X-axis vertical on the paper.

RESTRICTIONS

The program will not handle functions where Y is a constant over the entire range.

PLOT, CONTINUED

```
LIST
10    DEF  FNF(X)=SQR(16-(X↑2))
15    DIM  Z[255]
20    REM  *****  PLOT  *****  MATHEMATICS PROGRAM  *****
30    REM  PLOTS A FUNCTION ON THE TTY.
40    LET  R1=0
50    LET  L1=0
60    LET  Q1=0
70    PRINT "PLEASE INPUT THE FOLLOWING PARAMETERS:"
80    PRINT "LEFT X-ENDPOINT";
90    INPUT A
100   PRINT "RIGHT X-ENDPOINT";
110   INPUT B
120   PRINT "X-SPACING";
130   INPUT D
140   PRINT "THE NUMBER OF UNDEFINED POINTS (IF NONE, ENTER 0)";
150   INPUT N9
160   IF N9=0 THEN 210
170   PRINT "ENTER THE UNDEFINED POINTS, FOLLOWING EACH WITH A RETURN"
180   FOR K7=1 TO N9
190   INPUT Z[K7]
200   NEXT K7
210   DEF FNG(X)=INT((Y7-L1)/D1+.5)+15
220   LET L2=R2=FNF(A)
230   FOR X=A TO B STEP D
240   FOR I=1 TO N9
250   IF X=Z[I] THEN 310
260   NEXT I
270   IF FNF(X)>L2 THEN 290
280   LET L2=FNF(X)
290   IF FNF(X)<R2 THEN 310
300   LET R2=FNF(X)
310   NEXT X
320   IF L2<0 THEN 350
330   LET R1=R2
340   GOTO 390
350   IF R2>0 THEN 370
360   GOTO 380
370   LET R1=R2
380   LET L1=L2
390   LET D1=(R1-L1)/50
400   IF L1<R1 THEN 430
410   PRINT "THIS IS THE FUNCTION Y=CONSTANT."
420   STOP
430   PRINT "THE MINIMUM VALUE OF THE FUNCTION IS";L2
440   PRINT "THE MAXIMUM VALUE OF THE FUNCTION IS";R2
450   PRINT "THE SPACING ON THE Y-AXIS IS";D1
460   PRINT ""
470   LET F=INT(-L1/D1+.5)+15
480   IF A <= 0 THEN 590
490   IF A/D>6 THEN 590
```

```
500   LET Q1=1
510   IF L1=0 THEN 530
520   PRINT TAB(F);"+"
530   PRINT
540   GOTO 780
550   FOR I=1 TO INT(A/D-.5)
560   PRINT TAB(F);"+"
570   NEXT I
580   LET Q1=0
590   FOR X=A TO B STEP D
600   IF D<1.00000E-04 THEN 630
610   IF ABS(X)>1.00000E-05 THEN 630
620   LET X=0
630   PRINT X,
640   FOR P=1 TO N9
650   IF X#Z[P] THEN 750
660   IF X#0 THEN 730
670   FOR I2=1 TO 50
680   PRINT "+";
690   NEXT I2
700   LET Q=1
710   PRINT "Y"
720   GOTO 1060
730   PRINT TAB(F);"+"
740   GOTO 1060
750   NEXT P
760   IF X*(X+D)>0 THEN 960
770   IF X<-D/2 THEN 960
780   FOR I=0 TO 50
790   IF Q1>0 THEN 820
800   LET Y7=FNF(X)
810   IF FNG(X)=I+15 THEN 850
820   IF I+15=F THEN 870
830   PRINT "+";
840   GOTO 880
850   PRINT "*";
860   GOTO 880
870   PRINT "O";
880   NEXT I
890   IF I+15#F THEN 910
900   PRINT "+";
910   PRINT "Y"
920   LET Q=1
930   IF (Q1+1)=1 THEN 1060
940   IF (Q1+1)=2 THEN 550
950   IF (Q1+1)=3 THEN 1150
960   IF X*(X-D)>0 THEN 980
970   IF X <= D/2 THEN 780
980   LET Y7=FNF(X)
990   IF FNG(X)>F THEN 1050
1000   IF FNG(X)=F THEN 1030
```

PLOT, CONTINUED

```
1010    PRINT TAB(FNG(X));"*";TAB(F);"+"
1020    GOTO 1060
1030    PRINT TAB(F);"*"
1040    GOTO 1060
1050    PRINT TAB(F);"+";TAB(FNG(X));"*"
1060    NEXT X
1070    IF X >= 0 THEN 1160
1080    IF -X/D>6 THEN 1160
1090    FOR I=1 TO INT(-X/D-.5)
1100    PRINT TAB(F);"+"
1110    NEXT I
1120    LET Q1=2
1130    PRINT
1140    GOTO 780
1150    PRINT TAB(F);"+"
1160    PRINT TAB(F);"X"
1170    IF Q=0 THEN 1190
1180    STOP
1190    PRINT
1200    PRINT
1210    PRINT
1220    FOR I=0 TO 50
1230    PRINT "+";
1240    NEXT I
1250    PRINT "Y"
1260    PRINT
1270    PRINT
1280    PRINT "                       SINCE THE REAL Y-AXIS IS OFF THE GRAPH."
1290    STOP
1300    END
```

CURFIT

This program performs a least-squares curve fit to the
following functions:

$$1. \quad Y = A + B(X)$$
$$2. \quad Y = A \exp (B * X)$$
$$3. \quad Y = A (X^B)$$
$$4. \quad Y = A + B/X$$
$$5. \quad Y = 1/(A + B * X)$$
$$6. \quad Y = X/(A + B * X)$$

INSTRUCTIONS

Before running the program, enter the following data beginning
in line 9900:

$$9900 \text{ DATA N}$$
$$9901 \text{ DATA } X_1, Y_1, X_2, Y_2 \ldots$$
$$\vdots$$
$$99\text{-- DATA } \ldots X_n, Y_n$$

Where: N = Number of data pairs

X_n, Y_n = the n-th data pair; X_n is the independent
variable and Y_n is the dependent variable.

The program prints summary data for the curve fits for the six
functions and requests the user to indicate which function he
wishes detailed information about (input 0, 1, 2, 3, 4, 5, or 6).
A zero terminates the program.

RESTRICTIONS

If there are more than 14 data pairs,change the dimension of
variables X, Y, U, V in statement 9003 to this number; the
HP BASIC system must have more than 8K of core memory to accom-
modate more than 14 data pairs in this program.

```
9000   REM   *****   CURFIT   *****   MATHEMATICS PROGRAM   *****
9001   REM   *****   VERSION 1   *****   7/31/69   *****
9002   REM   LEAST SQUARES CURVE FIT #1
9003   DIM X[14],Y[14],U[14],V[14],A[6],B[6],S[6],F[6]
9004   MAT F=CON
9005   READ N
9006   PRINT
9007   FOR I=1 TO N
9008   READ X[I],Y[I]
9009   NEXT I
9010   PRINT
9011   PRINT
9012   PRINT " ","LEAST SQUARES CURVES FIT"
9013   PRINT
9014   PRINT "CURVE TYPE"," INDEX  OF","   A","   B"
9015   PRINT " ","DETERMINATION"
9016   PRINT
9017   FOR I=1 TO 6
9018   MAT S=ZER
9019   GOSUB 9120
9020   IF (I-5)*(I-6)=0 THEN 9035
9021   IF (I-2)*(I-3)=0 THEN 9028
9022   FOR J=1 TO N
9023   LET V[J]=Y[J]
9024   GOSUB 9098
9025   NEXT J
9026   IF I=1 THEN 9045
9027   GOTO 9056
9028   FOR J=1 TO N
9029   IF Y[J] <= 0 THEN 9042
9030   LET V[J]=LOG(Y[J])
9031   GOSUB 9098
9032   NEXT J
9033   IF I=3 THEN 9050
9034   GOTO 9045
9035   FOR J=1 TO N
9036   IF Y[J]=0 THEN 9042
9037   LET V[J]=1/Y[J]
9038   GOSUB 9098
9039   NEXT J
9040   IF I=6 THEN 9056
9041   GOTO 9045
9042   PRINT "CAN'T FIT"
9043   LET F[I]=0
9044   GOTO 9063
9045   FOR J=1 TO N
9046   LET U[J]=X[J]
9047   GOSUB 9101
9048   NEXT J
9049   GOTO 9061
9050   FOR J=1 TO N
```

```
9051   IF X[J] <= 0  THEN 9042
9052   LET U[J]=LOG(U[J])
9053   GOSUB 9101
9054   NEXT J
9055   GOTO 9061
9056   FOR J=1  TO  N
9057   IF X[J]=0  THEN 9042
9058   LET U[J]=1/X[J]
9059   GOSUB 9101
9060   NEXT J
9061   GOSUB 9161
9062   PRINT C[I],A[I],B[I]
9063   NEXT I
9064   GOSUB 9105
9065   PRINT
9066   PRINT
9067   PRINT
9068   PRINT "DETAILS FOR CURVE TYPE (1 TO 6, 0 TO END PROGRAM.)";
9069   INPUT I
9070   IF I=0  THEN 9186
9071   LET K=I
9072   IF F[I]=1  THEN 9076
9073   GOSUB 9120
9074   PRINT " COULD NOT BE FIT."
9075   GOTO 9065
9076   GOSUB 9138
9077   IF (I-1)*(I-5)*(I-6)#0  THEN 9087
9078   FOR J=1  TO  N
9079   LET Y=A[I]+B[I]*X[J]
9080   IF I=1  THEN 9084
9081   LET Y=1/Y
9082   IF I=5  THEN 9084
9083   LET Y=X[J]*Y
9084   GOSUB 9176
9085   NEXT J
9086   GOTO 9065
9087   FOR J=1  TO  N
9088   IF I=2  THEN 9094
9089   IF I=3  THEN 9092
9090   LET Y=A[4]+B[4]/X[J]
9091   GOTO 9095
9092   LET Y=A[3]*(X[J]↑B[3])
9093   GOTO 9095
9094   LET Y=A[2]*EXP(B[2]*X[J])
9095   GOSUB 9176
9096   NEXT J
9097   GOTO 9065
9098   LET S[5]=S[5]+V[J]↑2
9099   LET S[3]=S[3]+V[J]
9100   RETURN
9101   LET S[1]=S[1]+U[J]
9102   LET S[2]=S[2]+U[J]↑2
```

```
9103   LET S[4]=S[4]+U[J]*V[J]
9104   RETURN
9105   FOR I=1 TO N-1
9106   LET M=I
9107   FOR J=I+1 TO N
9108   IF X[M] <= X[J] THEN 9110
9109   LET M=J
9110   NEXT J
9111   IF M=I THEN 9118
9112   LET P=X[M]
9113   LET Q=Y[M]
9114   LET X[M]=X[I]
9115   LET Y[M]=Y[I]
9116   LET X[I]=P
9117   LET Y[I]=Q
9118   NEXT I
9119   RETURN
9120   LET K=I
9121   IF K=1 THEN 9136
9122   IF K=2 THEN 9134
9123   IF K=3 THEN 9132
9124   IF K=4 THEN 9130
9125   IF K=5 THEN 9128
9126   PRINT "6. Y=X/(A+B*X) ";
9127   RETURN
9128   PRINT "5. Y=1/(A+B*X) ";
9129   RETURN
9130   PRINT "4. Y=A+(B/X)",
9131   RETURN
9132   PRINT "3. Y=A*(X+B)",
9133   RETURN
9134   PRINT "2. Y=A*EXP(B*X)";
9135   RETURN
9136   PRINT "1. Y=A+(B*X)",
9137   RETURN
9138   PRINT "    ";
9139   GOSUB 9121
9140   PRINT " IS A";
9141   IF K=1 THEN 9146
9142   IF K=2 THEN 9148
9143   IF K=3 THEN 9150
9144   PRINT " HYPERBOLIC";
9145   GOTO 9151
9146   PRINT " LINEAR";
9147   GOTO 9151
9148   PRINT "N EXPONENTIAL";
9149   GOTO 9151
9150   PRINT " POWER";
9151   PRINT " FUNCTION.   THE RESULTS"
9152   IF K=1 THEN 9154
9153   PRINT "         OF A LEAST-SQUARES FIT OF ITS LINEAR TRANSFORM"
9154   PRINT "           (SORTED IN ORDER OF ASCENDING VALUES OF X)"
```

```
9155    PRINT "        ARE AS FOLLOWS:"
9156    PRINT
9157    PRINT "X-ACTUAL","Y-ACTUAL"," Y-CALC"," PCT DIFFER"
9158    PRINT
9159    RETURN
9160    PRINT
9161    LET B=(N*S[4]-S[1]*S[3])/(N*S[2]-(S[1]↑2))
9162    LET A=(S[3]-B*S[1])/N
9163    LET S1=S[5]-(S[3]↑2)/N
9164    LET S2=(B↑2)*(S[2]-(S[1]↑2)/N)
9165    LET C[I]=S2/S1
9166    IF (I-1)*(I-4)*(I-5)=0 THEN 9173
9167    IF (I-2)*(I-3)=0 THEN 9171
9168    LET A[6]=B
9169    LET B[6]=A
9170    RETURN
9171    LET A[I]=EXP(A)
9172    GOTO 9174
9173    LET A[I]=A
9174    LET B[I]=B
9175    RETURN
9176    PRINT X[J],Y[J],Y,
9177    LET D=Y[J]-Y
9178    LET D=.1*SGN(D)*INT(1000*ABS(D/Y))
9179    IF D<0 THEN 9184
9180    IF D>0 THEN 9183
9181    PRINT "        0"
9182    RETURN
9183    PRINT "           ";
9184    PRINT D
9185    RETURN
9186    STOP
9900    DATA 6
9901    DATA 1,2,3,4,5,6,7,8,9,10,11,12
9999    END
```

LOW PASS FILTER

DESCRIPTION

This program uses constant K prototype T section, and
M derived (M = 0.6) termination L section to design low
pass filters. The program gives high attenuation at speci-
fied frequencies in the stop band by adding up to nine
additional M derived T sections.

INSTRUCTIONS

Enter the following information when requested by the program:

    1. Characteristic impedance

    2. Cut-off frequency in H_z

    3. Number of stop band attenuators

    4. Frequency (in H_z) for attenuators

The program then diagrams the filter and indicates maximum
attenuation.
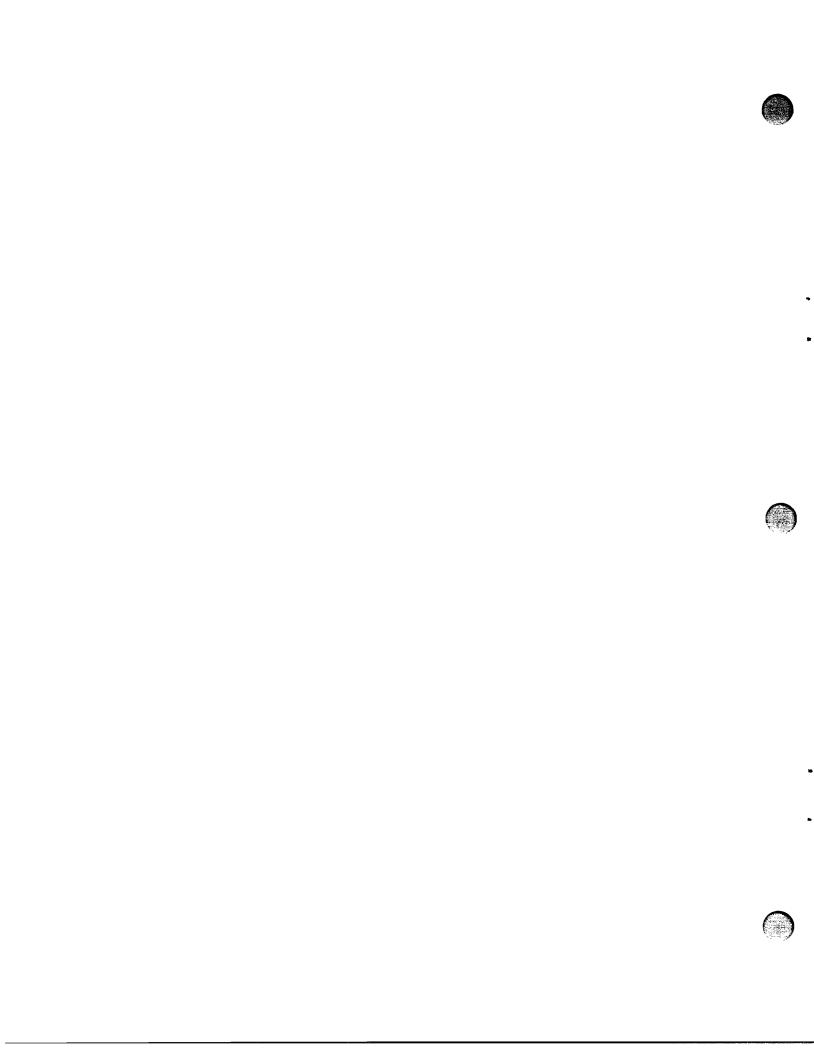
LOW PASS FILTER, CONTINUED

```
9002    REM   DESIGNS LOW PASS FILTER
9003    PRINT "PROGRAM FOR THE DESIGN OF A LOW PASS FILTER"
9004    PRINT
9005    PRINT "WHAT IS THE DESIRED CHARACTERISTIC IMPEDANCE IN OHMS ";
9006    INPUT R
9007    MAT F=ZER
9008    PRINT
9009    PRINT "WHAT IS THE DESIRED CUTOFF FREQUENCY IN HZ ";
9010    INPUT F[1]
9011    PRINT
9012    PRINT "HOW MANY ATTENUATORS ARE DESIRED IN THE STOP BAND ";
9013    INPUT A
9014    GOTO 9026
9015    FOR I=2 TO A+1
9016    PRINT "WHAT IS THE FREQUENCY FOR ATTENUATOR NUMBER "I-1;
9017    INPUT F[I]
9018    PRINT
9019    NEXT I
9020    LET L=1000*R/(3.14159*F[1])
9021    LET C=1.E+06/(3.14159*F[1]*R)
9022    LET L[1]=.6*L/2
9023    LET S[1]=.64*L/1.2
9024    LET C[1]=.6*C/2
9025    GOTO 9029
9026    PRINT
9027    IF A=0 THEN 9020
9028    GOTO 9015
9029    FOR I=2 TO 10
9030    IF F[I]=0 THEN 9036
9031    LET M[I]=SQR(1-F[1]/F[I]*F[1]/F[I])
9032    LET L[I]=M[I]*L/2
9033    LET S[I]=L*((1-M[I]*M[I])/(4*M[I]))
9034    LET C[I]=C*M[I]
9035    NEXT I
9036    PRINT
9037    PRINT
9038    GOSUB 9077
9039    GOSUB 9071
9040    GOSUB 9075
9041    GOSUB 9071
9042    PRINT ">",L/2+L[1],"MH","                    I"
9043    GOSUB 9071
9044    PRINT "+-------------",C,"   MFD  ------------------+"
9045    GOSUB 9071
9046    LET I=2
9047    IF F[I]=0 THEN 9065
9048    PRINT ">",L/2+L[I],"MH","              I"
9049    GOSUB 9071
9050    PRINT "+-------";S[2]"MH +",C[2],"MFD ---------+"
9051    GOSUB 9071
9052    FOR I=3 TO 10
9053    IF F[I]=0 THEN 9059
```

APPENDIX B

```
9054    PRINT ">",L[I]+L[I-1],"MH","                            I"
9055    GOSUB 9071
9056    PRINT "+--------";S[I]"MH +",C[I],"MFD ----------+"
9057    GOSUB 9071
9058    NEXT I
9059    PRINT ">",L[I-1]+L[1],"MH","                            I"
9060    GOSUB 9071
9061    GOSUB 9075
9062    GOSUB 9071
9063    GOSUB 9077
9064    GOTO 9079
9065    PRINT ">",L/2+L[1],"MH","                        I"
9066    GOSUB 9071
9067    GOSUB 9075
9068    GOSUB 9071
9069    GOSUB 9077
9070    GOTO 9079
9071    FOR N=1 TO 2
9072    PRINT "I"," "," ","                        I"
9073    NEXT N
9074    RETURN
9075    PRINT "+--------";S[1]"MH +",C[1],"MFD ----------+"
9076    RETURN
9077    PRINT "0<--------------",R,"OHM LINE","-------------->0"
9078    RETURN
9079    PRINT
9080    PRINT
9081    PRINT "TERMINATING SECT'S GIVE MAX. ATTEN. AT"1.25*F[1]"HZ"
9082    IF F[2]=0 THEN 9088
9083    PRINT "IN ADDITION TO THOSE SPECIFIED AT:"
9084    FOR I=2 TO 10
9085    IF F[I]=0 THEN 9088
9086    PRINT F[I]"HZ"
9087    NEXT I
9088    STOP
9999    END
```

APPENDIX C

QUICK REFERENCE TO HP BASIC

SPECIAL CHARACTERS

KEY	FUNCTION
alt-mode	Deletes a line being typed. (Same as *esc*).
S	Terminates an input loop and causes a jump to the END statement.
esc	Deletes a line being typed (same as *alt-mode*).
linefeed	Causes the teleprinter to advance one line.
return	1. Must follow every command or statement.
	2. Causes the teleprinter typeface to return to the first print position.
	3. BASIC responds with a *linefeed*.
←	Backspace. Deletes as many preceding characters as ←'s are typed in.

OPERATORS

SYMBOL	SAMPLE STATEMENT	PURPOSE/MEANING/TYPE
=	11Ø LET A = Ø	Assignment operator; assigns a value to a variable
↑	12Ø PRINT X↑2	Exponentiate (as in X^2).
*	13Ø LET C5 = (A*B)*N2	Multiply
/	14Ø PRINT T5/4	Divide
+	15Ø LET P = R1 + 1Ø	Add
-	16Ø X3 = R3 - P	Subtract

NOTE: *The numeric values used in logical evaluation are:* *"true" = any non-zero number; "false" = Ø.*

=	17Ø IF D=E THEN 6ØØ	*expression* "equals" *expression*
#	18Ø IF (D+E)#(2*D)THEN 71Ø	*expression* does not equal" *expression*
<>	18Ø IF(D+E)<>(2*D)THEN 7ØØ	*expression* "does not equal" *expression*
>	19Ø IF X>1Ø THEN 62Ø	*expression* "is greater than" *expression*
<	2ØØ IF R8<P7 THEN 64Ø	*expression* "is less than" *expression*
>=	21Ø IF R8>=P7 THEN 71Ø	*expression* "is greater than or equal to" *expression*
<=	22Ø IF X2<=1Ø THEN 65Ø	*expression* "is less than or equal to" *expression*
AND	23Ø IF G2 AND H5 THEN 9ØØ	*expression 1* AND *expression 2* must both be "true" for statement to be "true"
OR	24Ø IF G2 OR H5 THEN 91Ø	If either *expression 1* OR *expression 2* is "true", statement is "true."
NOT	25Ø IF NOT G5 THEN 95Ø	Statement is "true" when *expression* (NOT G5) is "false."

STATEMENTS

NAME	EXAMPLE	PURPOSE
DATA	360 DATA 99,106.7, 16.2	Specifies data; read from left to right
DIM	310 DIM A(72)	Specifies maximum matrix size.
END	400 END	Terminates the program; the last statement in a program must be an END statement.
FOR...NEXT	350 FOR J=1 TO N STEP 3	Executes statements between FOR and NEXT the specified number of times (a loop), and in increments of the size indicated after STEP; STEP and *step size* may be omitted.
GO TO	330 GO TO 900	Transfers control (jumps) to specified statement number.
GOSUB	420 GOSUB 800	Begins executing the subroutine at specified statement (see RETURN).
IF...THEN	340 IF A#10 THEN 350	Logical test of specified condition; transfers control if "true".
INPUT	390 INPUT Y2,B4	Allows data to be entered from teleprinter while a program is running.
LET	300 LET A=B=C=0	Assigns variable a value; LET is optional.
NEXT	355 NEXT J	Sets the boundary of the FOR loop.
READ	360 READ A,B,C	Reads information from DATA statement.
REM	320 REM--ANY TEXT**!!	Inserts non-executable remarks in a program.
PRINT	356 PRINT A,B,"HELLO"	Prints the specified values; 5 fields per line when commas are used as separators.
	357 PRINT X;Y;P;Q;R(5)	Prints the specified values; 12 fields per line when semicolons are used as separators.
	358 PRINT	Causes the teleprinter to advance one line.
RESTORE	380 RESTORE	Permits re-reading data without re-running the program.

STATEMENTS, CONTINUED

NAME	EXAMPLE	PURPOSE
RETURN	850 RETURN	Transfers control to statement following its GOSUB.
STOP	410 STOP	Terminates the program; may be used anywhere in program.

COMMANDS

NOTE: Commands are executed immediately; they do not require statement numbers.

FULL NAME	EXAMPLE	PURPOSE
BYE	BYE	Returns user from BASIC to MTS executive.
LIST	LIST	Produces a listing of current program
	LIST 150	Produces a listing, starting at specified statement.
PTAPE	PTAPE	Allows input of a program on paper tape through the photoreader.
PLIST	PLIST	Punches the program in memory onto paper tape.
RUN	RUN	Starts program execution.
SCRATCH	SCRATCH	Erases current program.
TAPE	TAPE	Informs computer that following input is from paper tape.

FUNCTIONS

NOTE: PRINT is used for examples only; other statement types may be used.

FULL NAME	EXAMPLE	PURPOSE
DEF FN	3ØØ DEF FNA (X)=(M*X)+B	Allows the programmer to define functions; the function label (A) must be a letter from A to Z; the argument (X) must be mentioned in the function definition.
ABS (X)	31Ø PRINT ABS (X)	Gives the absolute value of the expression (X).
EXP (X)	32Ø PRINT EXP (X)	Gives the constant *e* raised to the power of the expression value (X); in this example, *e*↑X.
INT (X)	33Ø PRINT INT (X)	Gives the largest integer $\leq$ the expression (X).
LOG (X)	34Ø PRINT LOG (X)	Gives the natural logarithm of an expression; expression must have a positive value.
RND (X)	35Ø PRINT RND (X)	Generates a random number between Ø and 1; the expression (X) is a dummy argument.
SQR (X)	36Ø PRINT SQR (X)	Gives the square root of the expression (X); expression must have a positive value.
SIN (X)	37Ø PRINT SIN (X)	Gives the sine of the expression (X); X is real and in radians.
COS (X)	38Ø PRINT COS (X)	Gives the cosine of the expression (X); X is real and in radians.
TAN (X)	39Ø PRINT TAN (X)	Gives the tangent of the expression (X); X is real and in radians.
ATN (X)	4ØØ PRINT ATN (X)	Gives the arctangent of the expression (X); is real and in radians.
TAB (X)	42Ø PRINT TAB (X);A	Tabs to the specified position (X), then prints the specified value (A).
SGN (X)	44Ø PRINT SGN (X)	Gives: 1 if X>Ø, Ø if X=Ø, -1 if X<Ø

MATRICES

NOTES: *1. Maximum matrix size is 255 elements.*

2. Matrix variables must be a single letter from A to Z.

NAME	SAMPLE STATEMENT	PURPOSE
DIM	1Ø DIM A (1Ø, 2Ø)	Allocates space for a matrix of the specified dimensions.
MAT IDN	15 MAT X = IDN (m,n)	Establishes an identity matrix (with all ones down the diagonal). A new working size (m,n) may be specified;
MAT ZER	2Ø MAT B = ZER	Sets all elements of the specified matrix equal to Ø.
	25 MAT D = ZER (m,n)	A new working size (m,n) may be specified after ZER.
MAT CON	3Ø MAT C = CON	Sets all elements of the specified matrix equal to 1.
	35 MAT E = CON (m,n)	A new working size (m,n) may be specified after CON.
INPUT	4Ø INPUT A(5,5)	Allows input from the teleprinter of a specified matrix element.
MAT PRINT	5Ø MAT PRINT A	Prints the specified matrix on the teleprinter.
	55 PRINT A(X,Y)	Prints the specified element of a matrix on the teleprinter; element specifications X and Y may be any expression.
MAT READ	7Ø MAT READ A	Reads matrix from DATA statements.
	75 MAT READ A(5,5)	Reads matrix of specified size from DATA statements.
	8Ø READ A(X,Y)	Reads the specified matrix element from a DATA statement.

NAME	SAMPLE STATEMENT	PURPOSE
MAT +	1ØØ MAT C = A + B	Matrix addition; A and B must be the same size.
MAT -	11Ø MAT C = A - B	Matrix subtraction; A,B, and C must be the same size.
MAT*	12Ø MAT C = A * B	Matrix multiplication; no. columns in A must equal no. rows in B.
MAT =	13Ø MAT A = B	Establishes equality of two matrices; assigns values of B to A.
MAT TRN	14Ø MAT B = TRN (A)	Transposes an m by n matrix to an n by m matrix.
MAT INV	15Ø MAT C = INV (B)	Inverts a square matrix into a square matrix of the same size; matrix may be inverted into itself.

APPENDIX D

ERROR CODES, MEANING, AND PROBABLE CAUSE

ERROR CODE	MEANING	PROBABLE CAUSE
1	STATEMENT ENDS UNEXPECTEDLY.	Statement end (_return_) found by the syntax analyzer. Additional characters are needed to form a consistent statement.
2	INPUT EXCEEDS 71 CHARACTERS.	Too many characters in the line just typed.
3	SYSTEMS COMMAND NOT RECOGNIZED.	The line just typed begins with a letter, but the initial character string does not form a recognizable statement type. May be a missing statement number.
4	MISSING OR INCORRECT STATEMENT TYPE.	The characters immediately following the statement number do not form any recognizable statement type.
5	BAD EXPONENT.	A number appears followed by an E but not followed by a legitimate exponent integer.
6	SYMBOL FOLLOWING MAT NOT RECOGNIZED.	MAT not followed by PRINT, READ, or matrix variable.
7	LET STATEMENT HAS NO STORE.	No assignment operator appears in the formula following LET.
8	MULTIPLE OR MISPLACED COM STATEMENT.	A COM statement is not the first statement in the program, does not have the lowest sequence number, or is the second COM statement in the program.
9	MISSING OR INCORRECT FUNCTION IDENTIFIER IN DEF.	DEF is not followed by FN<letter>, or FN is not followed by a letter in a formula (for example, A+FN(3)).
10	MISSING PARAMETER IN DEF STATEMENT.	No simple variable is found following a 'DEF FN<letter>'.

MEANING PROBABLE CAUSE

11	MISSING ASSIGNMENT OPERATOR.	No assignment operator found in either a DEF statement, a FOR statement, or a MAT statement (e.g., FOR A STEP or MAT A *return*).
12	MISSING THEN.	An IF statement has no THEN following the decision formula (may be an incorrect formula as in IF A+BC THEN ...). Note that a missing * between B and C forces end-of-formula here.
13	MISSING OR INCORRECT FOR-VARIABLE.	A simple variable is not found following a FOR or a NEXT; for example, FOR A[1] = ... is not legal.
14	MISSING TO.	No TO found following the initial part of a FOR statement. May also be an incorrect formula as in FOR I = BC TO ...
15	INCORRECT STEP IN FOR STATEMENT.	Characters appear following the limit formula but do not form a correct STEP formula. May also be an incorrect formula as in FOR I = 1 to A+BC STEP Y (operator missing between B and C).
16	CALLED ROUTINE DOES NOT EXIST.	The first parameter of a CALL statement does not match any of the defined CALL routines.
17	WRONG NUMBER OF PARAMETERS IN CALL STATEMENT.	
18	MISSING OR INCORRECT CONSTANT IN DATA STATEMENT.	Caused by such things as: DATA 1,2, *return* (trailing comma) or DATA ++3 May also be caused by an incorrect reply to an INPUT statement, as +-4.
19	MISSING OR INCORRECT VARIABLE IN READ STATEMENT.	No variable appears following a READ, or there is a trailing comma in a READ statement; for example, READ A,B, *return*.
20	NO CLOSING QUOTE FOR PRINT STRING.	Unmatched " in a PRINT statement.

ERROR CODE	MEANING	PROBABLE CAUSE
21	MISSING PRINT DELIMITER OR BAD PRINT QUANTITY.	Caused by such things as missing operators or commas. Generally means that two formulas appear to be reversed without separating punctuation.
22	ILLEGAL WORD FOLLOWS MAT.	MAT is followed by two or more letters, but they do not form either PRINT or READ.
23	MISSING DELIMITER.	Sample causes: MAT READ A,B C (missing comma) MAT PRINT A;B1 (illegal variable for matrix.)
24	IMPROPER MATRIX FUNCTION.	MAT <letter> = is followed by two or more letters which do not form IDN, CON, ZER, TRN, or INV.
25	NO SUBSCRIPT WHERE EXPECTED.	IDN, CON, ZER followed by characters but they do not form a legitimate subscript.
26	MAY NOT INVERT OR TRANSPOSE MATRIX INTO SELF.	
27	MISSING MULTIPLICATION OPERATOR.	For example, MAT <letter> = (formula) (missing* between formula and A.)
28	IMPROPER MATRIX OPERATOR.	MAT letter = letter not followed by +, -, or *.
29	MATRIX MAY NOT BE BOTH OPERAND AND RESULT OF MATRIX MATRIX MULTIPLICATION.	
30	MISSING LEFT PARENTHESIS.	Cause should be obvious from inspection of the line.
31	MISSING RIGHT PARENTHESIS.	Cause should be obvious from inspection of the line typed.
32	OPERAND NOT RECOGNIZED.	No recognizable operand found where one is expected, for example, following a binary operator or left parenthesis. Usually a typing error.

ERROR CODE	MEANING	PROBABLE CAUSE
33	DEFINED ARRAY MISSING SUBSCRIPT PART.	An array appearing in a DIM or COM statement does not have a proper subscript-bound part; for example, DIM A[3],B,C[4,5]. (B has no subscript.)
34	MISSING ARRAY IDENTIFIER.	No array identifier found where one is expected in a DIM, COM, or MAT statement; for example, MAT A = B + _return_ (missing array identifier before _return_.)
35	MISSING OR BAD INTEGER.	A required integer is 0, or too large, or does not appear at all. Required integers appear as sequence numbers, formal bounds in DIM and COM statements, and as the first parameter of a CALL statement.
36	NON-BLANK CHARACTERS FOLLOWING STATEMENT'S LOGICAL END.	Something is missing from a statement at a place where the statement could logically end.
37	OUT OF STORAGE DURING SYNTAX PHASE.	Program is too large.
38	PUNCHED TAPE READER NOT READY.	Photoreader is off or the RUN-LOAD switch is not set to RUN when a PTAPE is given.
39	DOUBLY DEFINED FUNCTION.	The same function is defined in two DEF statements.
40	FOR STATEMENT HAS NO MATCHING NEXT STATEMENT.	
41	NEXT STATEMENT HAS NO MATCHING FOR STATEMENT.	The program contains an extra NEXT statement, or an improper nesting of FOR -- NEXT loops.
42	OUT OF STORAGE FOR SYMBOL TABLE.	Program is toolarge.
43	ARRAY APPEARS WITH INCONSISTENT DIMENSIONS.	An array is referenced as being singly-subscripted in one place and as doubly-subscripted in another.
44	LAST STATEMENT IS NOT END.	
45	ARRAY DOUBLY DIMENSIONED.	The same array appears twice in DIM statements, or in both a DIM statement and the COM statement.

ERROR CODES, CONTINUED

ERROR CODE	MEANING	PROBABLE CAUSE
46	NUMBER OF DIMENSIONS NOT OBVIOUS.	Several conditions may produce this message. Cured by specifying the bounds of all arrays which appear in MAT statements.
47	ARRAY TOO LARGE.	Number of array elements exceeds 32767.
48	OUT OF STORAGE DURING ARRAY ALLOCATION.	Program and arrays together are too large.
49	SUBSCRIPT EXCEEDS BOUND.	An actual subscript exceeds the declared (or dynamically redeclared value) bound; or an attempt is made to redimension an array with a bound greater than 255 (as in MAT READ A[300]).
50	ACCESSED OPERAND HAS UNDEFINED VALUE.	Attempted use of a variable or array element which has never been assigned a value.
51	NON-INTEGER POWER OF NEGATIVE NUMBER.	
52	ZERO TO ZERO POWER.	
53	MISSING STATEMENT.	Attempted GOTO, GOSUB, or IF... THEN to a non-existent statement.
54	GOSUBS NESTED 10 DEEP.	Attempted execution of ten GOSUBs in a row, with no intervening RETURN's. (Refers to the logical execution of a program, not the physical layout.)
55	RETURN FINDS NO ADDRESS.	RETURN is encoutered during execution when no GOSUBs are active.
56	OUT OF DATA.	More data has been requested in READ or MAT READ statements than exists in the DATA statements.
57	OUT OF STORAGE DURING EXECUTION.	Insufficient working space to execute the program.

ERROR CODE	MEANING	PROBABLE CAUSE
58	DYNAMIC ARRAY EXCEEDS ALLOCATED STORAGE.	An array redimensioning request requires more elements in the new working size than exist in the original array definition (for example, A[5,5] cannot be redimensioned to A[4,6], although the converse is true.).
59	DIMENSIONS NOT COMPATIBLE.	A MAT statement cannot be executed because the matrix arguments have incompatible dimensions for the operation attempted.
60	MATRIX OPERAND CONTAINS UNDEFINED ELEMENT.	Same as Error 50.
61	SINGULAR OR NEARLY SINGULAR MATRIX.	An array cannot be inverted, because all significance is lost in the calculations.
62	TRIGONOMETRIC FUNCTION ARGUMENT IS TOO LARGE.	Applies to SIN, COS, TAN.
63	ATTEMPTED SQUARE ROOT OF NEGATIVE ARGUMENT.	
64	ATTEMPTED LOG OF NEGATIVE ARGUMENT.	

WARNING-ONLY ERRORS
(Program continues executing.)

ERROR CODE	MEANING	PROBABLE CAUSE
65	NUMERICAL OVERFLOW, RESULT TAKEN TO BE + OR - INFINITY.	A calculated result or number input exceeds the capacity of numerical representation. The value is replaced by the largest representable number of appropriate sign.
66	NUMERICAL UNDERFLOW, RESULT TAKEN TO BE ZERO.	Number is too close to zero to be represented as other than zero.
67	LOG OF ZERO TAKEN TO BE -INFINITY.	

ERROR CODE	MEANING	PROBABLE CAUSE
68	EXP OVERFLOWS, RESULT TAKEN TO BE +INFINITY.	
69	DIVISION BY ZERO, RESULT TAKEN TO BE + OR -INFINITY.	
70	ZERO RAISED TO NEGATIVE POWER, RESULT TAKEN TO BE + INFINITY.	

ERROR CODES

INDEX

INDEX, CONTINUED

INDEX, CONTINUED

INDEX, CONTINUED

02116 — 9077