# NEVADA BASIC™

ELLIS COMPUTING™

SOFTWARE TECHNOLOGY

# NEVADA BASIC(tm)

## Users' Reference Manual



Contributing Authors for the CP/M(r) version:

Ian D. Kettleborough
John A. Starkweather, Ph.D.

**Ellis Computing, Inc.**
**3917 Noriega Street**
**San Francisco, CA 94122**

## COPYRIGHT



## TRADEMARKS



## DISCLAIMER



Printed in the U.S.A.

# NEVADA BASIC

## A BASIC INTERPRETER FOR CP/M

## TABLE OF CONTENTS

SECTION

# SECTION 1

## INTRODUCTION

NEVADA BASIC is a special adaptation of BASIC (Beginner's all-purpose Symbolic Instruction Code) for use with the CP/M Operating System. The BASIC interpreter was selected for adaptation because it is simple and easy to learn while providing the powerful capabilities of a high-level language. Thus, it is ideal for the user who is a novice at using programming languages as well as for the advanced user who wants to work with subroutines, functions, strings, and machine-level interfaces.

Some of the outstanding features available in NEVADA BASIC are:

1. Fully-formatted output to a variety of devices.
2. Many function subprograms, including mathematical, string, and video functions.
3. Program and data storage on floppy disk.
4. Full eight-digit precision and twelve-digit precision.
5. User-defined functions on one or more lines.
6. Calculator mode for immediate answers.
7. Full screen editing on video display.
8. Complete capability for string handling.
9. Functions and statements for communicating with any number of input/output channels.
10. Ability to view memory locations, change values, and branch to absolute addresses.
11. DATA files.
12. Matrix functions including INVert.

BASIC is a conversational language, which means that you can engage in a dialog with BASIC by typing messages at a terminal and receiving messages from a display device. For example:

```
BASIC:  Ready                          -BASIC indicates it is
                                        ready to receive
                                        instructions.
User:   10 PRINT "WHAT IS THE VALUE OF X" <CR>  -The user
        20 INPUT X <CR>                 enters the lines of a
        30 LET Y = X^3 <CR>             program each followed by a
        40 PRINT "X CUBED IS ";X^3 <CR>  carriage return.
        DEL 30 <CR>                     -User deletes line 30.
        LIST <CR>                       -User tells BASIC to
                                        list what has been typed.
BASIC:  10 PRINT "WHAT IS THE VALUE OF X" -BASIC lists all
        20 INPUT X                          but line 30,
        30 PRINT "X CUBED IS ";X^3          which was deleted.

User:   RUN <CR>                       -The user tells BASIC to
                                        execute the program.
BASIC:  WHAT IS THE VALUE OF X
User:   ?3 <CR>                        -The user types 3 in
BASIC:  X CUBED IS  27                  response to the ? prompt.
        Ready
```

**HOW TO USE THIS BOOK**

This book is intended as a description of this particular version of BASIC, namely NEVADA BASIC. Several useful beginning books are listed in Appendix 6 for those who need more background.

Read this book from cover to cover first, as a text. The material is presented in increasing difficulty from front to back. After you are familiar with NEVADA BASIC, you can use the book as a reference. In addition, statement and command summaries are given in Appendix 1. Appendix 2 is a function summary.

Section 2 gives background information needed for working with BASIC. It presents the fundamental definitions and modes of operation, and tells how to initialize and leave BASIC.

Section 3 describes the mechanics of writing BASIC programs, executing them, saving programs on diskette, and retrieving them at the appropriate time.

Section 4 describes an introductory set of statements, the instructions that make up a BASIC program. The statements described in section 4 are the simplest in the language, but

they can be used to solve many math and business applications.

Section 5 is referred to as "Advanced BASIC", but do not be taken aback by the term "Advanced". All of BASIC is, as the name implies, relatively simple to learn. Section 5 merely goes further into the language by teaching the use of subroutines and functions, how to work with strings of characters, saving data on diskette, and formatting output data.

Section 6 is for specialists. Those of you who have expanded your computer to send and receive data at a number of input/output ports will be interested in reading about the machine-level interfaces of BASIC.

Section 7 involves special statements, preceded by MAT, which involve the manipulation of matrices (two-dimensional arrays).

**Symbols and Conventions**

The symbols below are used in examples throughout this document:

<CR> The user depresses the carriage return key.
<LF> The user depresses the line feed key.

Command and statement forms use uppercase and lowercase characters to differentiate between characters to be typed literally and data to be supplied by the programmer. For example, the following command form indicates that the word LIST should be typed followed by a number selected by the user:

```
LIST n
```

Punctuation in command and statement forms should be interpreted literally. For example, the statement form below indicates that the word INPUT should be followed by one or more variable names separated by commas:

```
INPUT var1, var2, ...
```

The ellipsis (...), three consecutive periods, indicate that preceeding arguments can be repeated.

Optional parts of command and statement forms are enclosed in square brackets. For example, the form SCR[ATCH] indicates that both SCR and SCRATCH are valid forms of the command. The form EXecute indicates that only the first two characters need be typed.

# SECTION 2

# OPERATING PROCEDURES

## HARDWARE REQUIREMENTS

1. 8080/Z80/8085 microprocessor.
   (Z80 is a trademark of Zilog.)
2. A minimum of 32K RAM.
3. Any disk drive.
4. CRT or Video display and keyboard.

## SOFTWARE REQUIREMENTS

The CP/M(r) operating system version 1.4, 2.2 or 3.0. CP/M is a registered trademark of Digital Research, Inc.

## FILES ON THE DISTRIBUTION DISKETTE

NVBASIC.COM is the BASIC interpreter 8-digit version.
NVBAS12.COM is the BASIC interpreter 12-digit version.
BASKEY.COM is used to change Editing Control keys.
NVBASIC.PRN is a source code listing of the user changeable CRT driver.
SAMPLE.BAS is a BASIC source code sample program.

## FILE TYPE CONVENTIONS

| | |
|---|---|
| BASIC source code files | .BAS |
| Assembly source code files | .ASM |
| COBOL source code files | .CBL |
| FORTRAN source code files | .FOR |
| Object code run time files | .OBJ |
| Printer listing files | .PRN |
| Symbol table listing files | .SYM |
| Error files | .ERR |
| Work files | .WRK |

## GETTING STARTED

If the master disk is not write protected, do it now!

1. NEVADA BASIC is distributed on a DATA DISK without the CP/M operating system. There is no information on the

system tracks, so don't try to "boot it up", it won't work!

2. On computer systems with the ability to read several disk formats, such as the KayPro computer, the master diskette must be used in disk drive B.

3. Do not try to copy the master diskette with a COPY program! On most systems it won't work. You must use the CP/M PIP command to copy the files.

4. First, prepare a CP/M system's diskette for use as your NEVADA BASIC operations diskette. On 5 1/4 inch diskettes you may have to remove (use the CP/M ERA command) most of the files in order to make room for the BASIC files. None of the CP/M files are needed for NEVADA BASIC, but PIP.COM and STAT.COM are useful if you have the space. You may want to do a CP/M STAT command on the distribution disk so you will know how much space you need on your operational diskette. For more information read the CP/M manuals about the STAT command.

5. Then insert the newly created CP/M diskette in disk drive A, and insert the NEVADA BASIC diskette in drive B and type (ctl-c) to initialize CP/M. Now copy all the files from the BASIC diskette onto the CP/M diskette:

```
PIP A:=B:*.*[VO]
```

If you get a BDOS WRITE ERROR message from CP/M during the PIP operation it usually means the disk is full and you should erase more files from the operational diskette.

At this point, put the NEVADA BASIC diskette in a safe place. You will not need it unless something happens to your operations diskette. By the way, back up your operations diskette with a copy each week! If your system malfunctions you can then pat yourself on the back for having a safe back up copy of your work.

Now, boot up the newly created NEVADA BASIC operations diskette. Notice that CP/M displays the amount of memory for which this version of CP/M has been specialized. The amount of memory available determines the size of the programs that can be run. The more memory available the larger the program that can be run.

## HOW TO INITIALIZE AND LEAVE BASIC

NEVADA BASIC is stored on diskette under the name NVBASIC for the 8-digit version or under the name NVBAS12 for the 12-digit version.

1. To create your specialized working version, type:

        NVBASIC <CR>
    or  NVBAS12 <CR>

2. Next the screen is filled with terminal choices:

```
NVBASIC VERSION 2.1 (0) CONFIGURING
COPYRIGHT (C) 1983 ELLIS COMPUTING, INC.
@ ANSI MODE TERMINAL
A ADVANTAGE
B APPLE COMPUTER, 40 COLUMN DISPLAY
C APPLE COMPUTER + VIDEX 80 COLUMN BOARD
D BEEHIVE 150 OR CROMENCO 3100
E COMMODORE 64
F FREEDOM 100
G HAZELTINE 1400 SERIES
H HAZELTINE 1500 SERIES
I HEATH H19/H89 OR ZENITH Z19/Z89
J HEWLETT-PACKARD 2621

TYPE A SINGLE LETTER TO SELECT TERMINAL.
<CARRIAGE RETURN> FOR MORE TERMINALS

K IBM PERSONAL COMPUTER+ BABY BLUE CARD
L INFOTON I-100
M LEAR-SEIGLER ADM-3A
N LEAR-SEIGLER ADM-31
O MICROTERM ACT-IV
P OSBORNE I
Q PERKIN-ELMER 550 (Bantom)
R PROCESSOR TECHNOLOGY SOL OR VDM
S SOROC IQ-120/140
T SUPERBRAIN
U TELEVIDEO 950
V TRS-80, MOD II (P. & T. CP/M)
W NONE OF THE ABOVE
TYPE A SINGLE LETTER TO SELECT TERMINAL.
<CARRIAGE RETURN> FOR MORE TERMINALS R
```

BASIC.COM is now saved on the default drive.

Since BASIC is stored as a (.COM) file, you can now enter BASIC by simply typing BASIC <CR>.

After you are finished working in BASIC, you can exit to CP/M by simply typing BYE <CR>.

If you have a BASIC program stored as a CP/M (.BAS), and you want to load BASIC, and load and run the program at the same time, use the one command:

        BASIC file-name

using the program's file name. The program will begin execution automatically.

**DEMONSTRATION PROGRAMS**

The BASIC disk contains demonstration programs which illustrate the power of this version of BASIC and may be studied as examples of advanced programming techniques, by LISTING them.

## DEFINITIONS OF COMMANDS AND STATEMENTS

Whenever you type a line of text ending with a carriage return in the BASIC environment, BASIC interprets it as a command or as a statement. A command is an instruction that is to be executed immediately, while a statement is an instruction that is to be executed at a later time, probably in a sequence with other statements.

BASIC differentiates between commands and statements by the presence or absence of line numbers. A statement is preceded by a line number. A Command is not. Examples of command lines are:

```
LIST 10, 90, <CR>
DEL 70 <CR>
BYE <CR>
```

Examples of statement lines are:

```
 10 LET A = 100 <CR>
 70 PRINT A1,Z7 <CR>
100 INPUT X,Y,C <CR>
```

You can enter more than one statement on a line by using the colon as a separator. For example:

```
10 LET X = 0 : GO TO 150
```

is the same as

```
10 LET X = 0
20 GO TO 150
```

When entering multiple statements on a line, precede only the first statement with a line number. For example:

```
100 INPUT A,B,C:LET X = A - B*C
```

A command or statement has a keyword that tells what is to be done with the rest of the line. In the examples above, the keywords are LIST, DEL, BYE, LET, PRINT, and INPUT. Keywords can be abbreviated by eliminating characters on the right and following the abbreviation with a period. For example, the following statements are equivalent:

```
10 PRINT X,Y
10 PRIN. X,Y
10 PRI. X,Y
10 PR. X,Y
10 P. X,Y
```

The minimum number of characters allowed in the abbreviation is determined by the number of characters required to uniquely identify the keyword and by a hierarchy of keywords in statements or commands. Appendices 1 and 2 indicate the minimum abbreviations allowed for all command and statement keywords.

## DESCRIPTION OF BASIC STATEMENTS

A statement is preceded by a line number which must be an integer between 1 and 65534. This line number determines the statement's place in a sequence of statements. The first word following the statement number tells BASIC what operation is to be performed and how to treat the rest of the statement. For example:

```
200 PRINT "THIS IS AN EXAMPLE"

                Indicates what is to be printed.
          Tells BASIC that a printing operation is
          to take place.
     Indicates that this statement will be executed
     before statements with line numbers greater than
     200 and after statements with line numbers less
     than 200.
```

Blanks do not affect the meaning of a statement in BASIC. That is the following are equivalent statements:

```
20 GO TO 200
20GOTO200
```

BASIC automatically removes blanks from statements as you enter them. Blanks in strings (discussed later) are not altered.

BASIC statements specify operations on constants, variables, and expressions. These terms are discussed in the units below.

### Constants

A constant is a quantity that has a fixed value. In NEVADA BASIC constants are either numerical or string. A numerical constant is a number, and a string constant is a sequence of characters.

A numerical constant can be expressed in any of the following forms:

| | |
|---|---|
| Integer | 1, 400, 32543, -17 |
| Floating point | 1.73, -1123.01, .00004 |
| Exponential | 3.1001E-5, 10E4, 230E-12 |

A string constant is indicated by enclosing a string of characters in quotation marks. For example:

```
"Nevada"
"The answer is"
```

Strings are discussed in more detail in section 5.

**Variables**

A variable is an entity that can be assigned a value. In NEVADA BASIC a variable that can be assigned a numerical value has a name consisting of a single letter or a single letter followed by a digit. The following are examples of numerical values being assigned to numerical varibles:

```
A = 17
B9 = 147.2
```

A variable that can be assigned a string value has a name consisting of a single letter followed by a ($) dollar sign or a single letter followed by a digit followed by a ($) dollar sign.

Examples of string values being assigned to string variables are:

```
A$ = "J. PAUL JONES"
X$ = "711 N. Murry"
R9$ = "Payables, Dex. 9"
```

**Expressions**

An expression is any combination of constants, variables, functions, and operators that has a numerical or string value. Examples are:

```
X^2 + Y - A*B
22 + A
"NON" + A$
NOT N
```

A numerical expression is an expression with a numerical value. It may include any of the following arithmetic operators:

```
^  expoentiate
*  multiply
/  divide
+  add
-  subtract
```

However, a negative number cannot be raised to a power

((-X)^Y) since the result could be a complex number. In an expression arithmetic operators are evaluated in the order shown below:

```
highest          -  (unary negate)
next highest     ^
next highest     * and /
lowest           + and -
```

Expressions in parentheses are evaluated before any other part of an expression. For example:

```
A / 2 * B - (4/C) ^ 2
third       first
              second
fourth
        fifth
```

Numerical expressions can also include logical and relational operators. These are introduced in section 4.

Operations in string expressions are described in section 5.

**DEFINITION OF A PROGRAM**

A program is a stored sequence of instructions to the computer. The instructions are specified in statements arranged to solve a particular problem or perform a task. The statement numbers determine the sequence in which the instructions are carried out. For example, the following program averages numbers:

```
10 PRINT "HOW MANY NUMBERS DO YOU WANT TO AVERAGE";
20 INPUT N
30 PRINT "TYPE ", N; "NUMBERS"
40 FOR I = 1 TO N
50 INPUT X
60 S = S + X
70 NEXT I
80 PRINT "THE AVERAGE IS ", S/N
```

**THE CALCULATOR MODE OF BASIC**

Earlier, a statement was described as a user-typed line preceded by a statement number and a command was described as a user-typed line without a statement number. In NEVADA BASIC you can also type a statement without a statement number and it will be treated as a command. That is, BASIC executes the statement as soon as you type the carriage return at the end of the line. For example:

```
User:   PRINT "5.78 SQUARED IS ", 5.78^2 <CR>
```

```
BASIC:  5.78 SQUARED IS 33.3084
```

Thus, you can use BASIC as a calculator to perform immediate computations.

If you perform a sequence of operations in calculator mode, BASIC will remember the results of each statement just as it does in a program. For example:

```
User:   LET A = 20.78 <CR>
        INPUT X
BASIC:  ? 2 <CR>                The user types 2 in response
                                to the ?
User:   LET B = A*X <CR>
        IF B > X THEN PRINT B
BASIC:  41.56
```

In the documentation of individual statements in sections 4 and 5, statements that can be used in calculator mode are marked CALCULATOR.

Without exception, all numbers in BASIC are decimal. This includes not only data values in constants, variables, and expressions, but the operands of BASIC statements and commands when they call for numeric values.

## SECTION 3

## HOW TO CREATE, EDIT, EXECUTE, AND SAVE A PROGRAM

A BASIC program is a stored sequence of instructions to the computer. This section tells how to enter a program into the computer, view the text of the program and alter it, execute the program, save it for future use, and retrieve it from storage.

### CREATING A PROGRAM

To create a program, simply type statements of the program in BASIC. Precede each statement with a statement number and follow it with a carriage return. For example:

```
User:   10 INPUT X,Y,Z <CR>
        20 PRINT X+Y+Z <CR>
```

A program now exists in BASIC. When executed the program will accept three numbers from the terminal and then print their sum.

When entering statements be careful not to create lines that will be too long when formatted by BASIC. BASIC will expand abbreviated statements; for example P. will become PRINT in a listing or edit. BASIC will insert blanks to improve readability, if the program was typed without them. These two factors can expand a line beyond the limit set bye SET LL = length command or statement. For more information about line length errors, see "LL" in Appendix 3.

It is not necessary to enter statements in numerical order. BASIC will automatically arrange them in ascending order. To replace a statement, precede the new statement with the statement number of the line to be replaced. For example:

```
User:  20 INPUT X,Y <CR>                    The user enters the
       10 PRINT "TYPE X AND Y" <CR>         statements out of
       30 PRINT X*Y <CR>                    sequence.
       30 PRINT "THE PRODUCT IS ",X*Y <CR>  Duplicate
       LIST <CR>                             statement number.
        10 PRINT "TYPE X AND Y"              BASIC orders the
        20 INPUT X,Y                         statements and keeps
        30 PRINT "THE PRODUCT IS " X*Y       only the last line
                                             entered for a given
                                             statement number.
```

While entering statements or commands in BASIC, you can use any of the following keys on the terminal to correct the line being typed:

| | |
|---|---|
| Control-H | Deletes the current character and shifts the remainder of the line to the left. |
| Control-C | Aborts a running program, infinite loop, listing, and getting or saving operations. Deletes a line being typed. If used to stop a running program, all open files will be closed. |
| Control-M<br>RETURN | Terminates the line. The line remains as it appeared when the RETURN key was type. |
| Control-X | Cancels the line being typed, and positions the cursor on a new line. The cancelled line remains on the screen. May also be used while the user is typing a responce to an INPUT statement in a running program. |

If a control character (like CONTROL-X above) is typed at the beginning of a line on the video display or terminal, it will be displayed on the screen or terminal, and will be ignored by BASIC.

**COMMANDS TO AID IN CREATING A PROGRAM**

The commands described in this section are likely to be used while creating a program. The LIST command display the program. DELETE and SCRATCH are used to erase statements. REN lets you automatically renumber statements. The EDIT command makes the screen editor available.

**LIST** **LIST**

FUNCTION: To display the indicated program statements.

FORMAT-1:

LIST [n]

FORMAT-2:

LIST [n1,n2]

RULES:

1. The command LIST will display the entire program.

2. The command LIST n will display the statement number n.

3. The command LIST n1, will display the statement number n1 through the end of the program.

4. The command LIST ,n2 will display all the statements from the first through the statement number n2.

5. The command LIST n1,n2 will display statements numbered n1 through n2.

6. The LIST command displays the indicated statements in increasing numerical order. It automatically formats the text of the statements, indenting and adding spaces where appropriate.

EXAMPLES:

```
User:   10 FOR I = 1 TO 100 <CR>
        30 NEXT I <CR>
        20 PRINT I^2 <CR>
        LIST <CR>
          10 FOR I=1 TO 100
          20   PRINT I^2
          30 NEXT I
```

EXAMPLES:

```
        LIST 100,150 <CR>
        LIST 50, <CR>
```

NOTES: You can control the display of material using the following keys:

| | |
|---|---|
| Control-C | Aborts listing |
| Control-S | Causes a pause in the listing. Striking any key causes the listing to resume. |

**LLIST** **LLIST**

FUNCTION: To list the indicated program statements on the printer.

FORMAT-1:

LLIST [n]

FORMAT-2:

LLIST [n1,n2]

RULES:

1. The command LLIST will list the entire program on the systems printer.

2. The command LLIST n will list the statement number n.

3. The command LLIST n1, will list the statement number n1 through the end of the program.

4. The command LLIST ,n2 will list all the statements from the first through the statement number n2.

5. The command LLIST n1,n2 will list statements numbered n1 through n2.

6. The LLIST command lists the indicated statements in increasing numerical order. It automatically formats the text of the statements, indenting and adding spaces where appropriate.

EXAMPLES:

```
User:   10 FOR I = 1 TO 100 <CR>
        30 NEXT I <CR>
        20 PRINT I^2 <CR>
        LLIST <CR>
          10 FOR I=1 TO 100       on the printer
          20   PRINT I^2
          30 NEXT I
```

NOTES: You can control the list of material using the following keys:

| | |
|---|---|
| Control-C | Aborts listing |
| Control-S | Causes a pause in the listing. Striking any key causes the listing to resume. |

**DEL** **DEL**

FUNCTION: To delete the indicated statements.

FORMAT-1:

```
DEL [n]
```

FORMAT-2:

```
DEL [n1,n2]
```

RULES:

1. The command DEL will delete all the statements in the program.

2. The command DEL n will delete statement number n.

3. The command DEL n1, will delete all statements from n1 through the end of the program.

4. The command DEL ,n2 will delete all statements from the first through statement n2.

5. The command DEL n1,n2 will delete statement numbers n1 through n2.

6. It should also be noted that entering a statement number, followed by a carriage return will also delete that statement.

EXAMPLE:

```
User:   100 LET A = 100 <CR>
        110 INPUT X,Y,Z <CR>
        120 PRINT (X+Y+Z)/A <CR>
        DEL 110, <CR>
        LIST <CR>
BASIC:    100 LET A=100
```

EXAMPLES:

```
        DEL ,150 <CR>
        DEL 75,90 <CR>
```

NOTES: Also, entering a line number that is not followed by a statement deletes a line.

EXAMPLE:

```
User:   100 <CR>
        LIST 100 <CR>
BASIC:  Ready            Line 100 has been deleted.
```

**SCRATCH** **SCRATCH**

FUNCTION: To delete the entire program and clear all variable definitions.

FORMAT:

SCR
SCRATCH

RULES:

1. The SCRATCH command deletes the entire program and clears all variable definitions established during previous program run or by statements executed in the calculator mode.

EXAMPLE:

| | | |
|---|---|---|
| User: | A = 100 <CR> | A receives a value of 100. |
| | PRINT A <CR> | |
| BASIC: | 100 | Prints the assigned value for A. |
| User: | SCR <CR> | The SCR command clears variables. |
| | PRINT A <CR> | |
| BASIC: | 0 | A's value is now 0. |
| User: | LIST <CR> | The SCR command has deleted all statements previously existing in the BASIC environment. |

**REN** **REN**

FUNCTION: To renumber all statements of the program.

FORMAT-1:

REN [n]

FORMAT-2:

REN [n,i]

RULES:

1. The command REN will renumber all statements. The first statement will be numbered 10 and subsequent statement numbers will increments of 10.

2. The command REN n will renumber all statements. The first statement will be numbered n and subsequent statements numbers will be increments of 10.

3. The command REN n,i will renumber all statements. The first statement will be numbered n and subsequent statement numbers will be increments of integer i.

4. The REN command renumbers all statements of the program as indicated, maintaining the correct order and branches in the program.

EXAMPLE:

```
REN <CR>
REN 100,5 <CR>
```

EXAMPLE:

```
User:   10 INPUT A,B <CR>
        20 PRINT "A*B IS ",A*B <CR>
        30 GO TO 10 <CR>
        REN 100
        LIST <CR>
          100 INPUT A,B
          110 PRINT "A*B IS ",A*B
          120 GO TO 100
```

Notice in line 120 that GO TO 10 has been changed to GO TO 100. IF line 30 had been GO TO 50, thus referring a line number which does not exist in the program to be renumbered, GO TO 50 would have been changed to GO TO 0, and an error message would have been printed. All references to non-existant line numbers will be changed to 0 before an error message is given.

**EDIT** **EDIT**

FUNCTION: To make available a new set of commands that can be used to create and alter text and program files.

FORMAT:

EDIT [n]

NOTES:

In EDIT the cursor may be positioned anywhere on the screen, lines may be scrolled up and down, and characters and entire lines may be inserted or deleted. There are also provisions for searching the file for strings, and for moving quickly to any one-tenth portion of the file from 0 to 9.

If you enter the EDIT command specifying line number (n), then that line number will be at the top of the Screen with the cursor set to it.

If you enter the EDIT command without a line number (n) specified, the first page of the file in memory is displayed, with the cursor at line one and position one (column 0). If the existing text is not enough to fill the screen, the remaining portion of the screen will be filled with blanks.

The next few pages tell how to go about changing a file by using control characters.

Below is a list of the command keys used by the EDITor. A more complete description of each command is given after the list. These are default commands that will be used unless modified during a configuration process by the program BASKEY. Other control codes or special character sequences may be substituted, thus allowing use of special keys on some terminals. The specific memory locations to be changed will be found in the file NVBASIC.PRN.

**COMMAND KEY LIST**

CONTROL KEYS

| | | |
|---|---|---|
| CONTROL | E | - move cursor up one line |
| CONTROL | X | - move cursor down one line |
| CONTROL | S | - move cursor left one character |
| CONTROL | D | - move cursor right one character |
| CONTROL | V | - toggle insert character mode; ON/OFF |
| CONTROL | G | - delete character under cursor |
| CONTROL | N | - insert line above cursor |
| CONTROL | Y | - delete line |
| CONTROL | Q | - move cursor to upper left corner of screen |
| CONTROL | Z | - move file up one line |
| CONTROL | W | - move file down one line |
| CONTROL | C | - scroll file up one-half page |
| CONTROL | R | - scroll file down one-half page |
| CONTROL | A | - move cursor to a mid line, column 1 |
| CONTROL | F | - initiate string search mode |
| CONTROL | L | - continue search for string |
| CONTROL | K | - exit from the editor or editor text search |
| TAB | | - move cursor to next tab position (CNTL-I) |
| RETURN | | - insert line below cursor (CONTROL-M) |
| LINE FEED | | - position cursor one line down (CONTROL-J) |
| BACKSPACE | | - backspace and erase a character |
| DELETE | | - (or RUBOUT) same action as backspace |

When you leave EDIT mode by pressing ctrl-K, the program you have prepared resides in memory and is ready to RUN. But it has not been saved. If you wish it to be stored in a disk file for later use, you must type "SAVE <file name>" before leaving BASIC.

## DETAILED COMMAND DESCRIPTION

### Cursor Positioning Commands

The keys S,D,E,X form a diamond on the input keyboard. When pressed simultaneously with the 'CTRL' (control) key, they move the cursor as indicated below:

```
CONTROL E  move cursor up one line
CONTROL X  move cursor down one line
CONTROL S  move cursor left one character
CONTROL D  move cursor right one character
CONTROL A  move cursor to mid screen, column 1
CONTROL Q  move cursor to home position, upper left
```

NOTE: Moving the cursor does not change the text.

### Screen Scroll Commands

Screen scroll commands are provided to allow the file to be "rolled" through the screen area until the desired file line is reached.

```
CONTROL Z   scroll up one line
CONTROL W   scroll down one line
CONTROL C   scroll up one-half page
CONTROL R   scroll down one-half page
```

(Please note that BASKEY.COM program can be used to change the scrolling from one half page to a full page scrolling.)

### Direct File Positioning Commands

In addition to cursor positioning controls, the EDITor offers a way of searching for a specific string of text within your file. The search (find) command is CONTROL F.

```
CONTROL F   editor text search
```

When you type control F, the last line of the display is cleared and the normal video reversed for this line. If an extra line is available, such as a 25th line on some terminals, this line is activated and used. The cursor is placed at the first position in the line. At this point the EDITor is waiting for you to enter either: 1) An input line consisting of one or more characters, or 2) a single digit. The input is terminated by a carriage return.

1. Character entry:

Any occurrence of the string entered, regardless of preceding or following characters, will represent a find. Therefore, only enough characters to define the desired text uniquely need be supplied. As an example, "the qu" can be used to locate a line in the file containing "the quick brown fox."

Upon receiving a carriage return, the EDITor searches the file, beginning one line below the current cursor position, until a string match is made or until the end of the file is reached. At the beginning of a file, the search begins at the first line. If a match is found, the EDITor positions the line containing the match at the top of the screen.

CONTROL L continue search

If you wish to continue searching for text matches after having left edit text search, pressing CONTROL L will cause continued searching for the string that was last designated. The EDITor resumes the search at the first line following that in which the cursor resides at the time of the command and continues until a match is made or until the end of file is reached. This command may be given as often as is desired.

2. Digit entry:

If you enter a digit from 0 to 9 in the command line, the file will be scrolled so that the top line on the video display screen marks the end of that tenth of the file which corresponds to the number entered. Thus, if the number is 5, the file will be positioned at the half-way point. If 0 is entered, the file will be positioned at the beginning. ESCAPE will cause an exit from editor text search and a return to EDIT mode.

**File Modification Commands**

CONTROL V character insert mode switch (on-off-on....)

Normal file characters input from the terminal are placed in the file in either of two modes. These modes, normal and insert, are alternately selected using the insert mode control.

When insert mode is OFF (default mode when EDITor is entered), each character that you type replaces what was formerly at the current cursor position, and the cursor moves to the right one place. When insert mode is ON, characters are actually inserted BEFORE the current cursor position, moving the character at that location, and any characters to the right of it, one position to the right. The cursor also advances one position. A line contains a

maximum number of characters, so you may begin to lose text that is pushed off the screen by the insertion.

CONTROL G delete character

The delete character command removes the character at the current cursor position and moves each character to the right of the cursor one position to the left.

CONTROL N insert line command

The insert line command puts a new blank line at the present cursor position, and moves each subsequent line of the file one row down. The cursor is moved to the first character position of the new line. Use this command to insert a new line "above" the current line.

CONTROL Y delete line command

This control removes the current cursor line from the file.

CONTROL T blank remaining line

This control deletes all characters to the right of the current cursor position (on the cursor line). The cursor appears at the beginning of the next line in the file.

CARRIAGE RETURN (CONTROL M) scroll up & insert line

Carriage return scrolls up one line and inserts a blank line in the file. The cursor is moved to the first character position of the new line. The new line is automatically numbered one greater than the previous line number, if the new line number does not already exist in the program. If the new line does exist, the program should be renumbered to make room for the new line. Use RETURN to insert a line 'BELOW' the current cursor position. No characters on the current line are deleted. The exception to this rule is that, if a file contains fewer than one page of text, RETURN will open a new blank line below the last line of text but will not scroll the file.

TAB (CONTROL I) horizontal tab

When TAB is pressed, the cursor will move to the next column divisible by eight (columns 8, 16, 32, ...).

CONTROL K exit from editor to BASIC.

When CONTROL K is struck the editor mode is terminated and control is returned to BASIC. The changes can then be tested by typing RUN.

**RUN** **RUN**

FUNCTION: To execute all or part of the current program.

FORMAT:

RUN [n]

RULES:

1. The command RUN will execute all of the current program.

2. The command RUN n will execute the current program beginning with statement number n.

3. If no statement number is specified, the command clears all variables and then executes the program.

4. If a statement number is indicated, the command executes the program beginning with that statement number, but does not clear the variable definitions first.

5. When a program is executed with the RUN command, BASIC interprets each of the statements sequentially, then it carries out the instructions.

6. If BASIC encounters a problem during any of these steps, it prints a message describing the error. The meanings of BASIC error messages are given in Appendix 3.

7. During execution a program can be interrupted by pressing Control-C keys. This is true weather the program is running correctly, is in a loop, or is waiting of input. No information is lost and you can continue execution by giving the CONT command.

8. When a program run terminates for any reason, all open files are closed.

EXAMPLE:

```
User:   10 LET A = 10, B = 20, C = 30 <CR>
        20 PRINT A^2*B-C <CR>
        30 STOP <CR>
        40 PRINT A^2*(B-C) <CR>
        RUN <CR>
BASIC:   1970
        STOP IN LINE 30          The STOP statement
                                 interrupts the program.
User:   RUN 40 <CR>
BASIC:   -1000                   Notice that A, B, and C
        Ready                    still have the values
                                 assigned in statement 10.
```

**CONT** **CONT**

Function: To continue program execution.

FORMAT:

CONT

RULES:

1. The CONT statement continues the execution of a program that was interrupted by Control-C Keys or stopped by the execution of a STOP statement.

2. If you edit any part of a program after interrupting execution, all variable definitions are lost. Thus you cannot stop a program's execution, change a statement in that program, and then CONTinue execution or print variable names.

3. When a program run is terminated for any reason, all open files are closed, which also could interfere with subsequent CONTinuation.

## CLEAR CLEAR

FUNCTION: To erase the definitions of all variables and leave the program intact.

FORMAT:

CLEAR

RULES:

1. The CLEAR command clears all variable definitions but does not erase the statements of the current program.

2. If CLEAR is used as a statement, all open files will be closed.

EXAMPLE:

```
User:   10 A=10, B=20,C=30 <CR>
        20 STOP <CR>
        30 PRINT A,B,C <CR>
        RUN <CR>
BASIC:  STOP IN LINE 20

User:   RUN 30
BASIC:   10      20       30        The variables have the
        Ready                       values assigned in line
                                    10.
User:   CLEAR <CR>
        RUN 30 <CR>
BASIC:   0       0        0        Variable definitions
        Ready                       have been cleared.

User:   LIST <CR>
        10 A=10,B=20,C=30           The program remains
        20 STOP                     intact.
        30 PRINT A,B,C
```

## HANDLING PROGRAM FILES ON DISKETTE

Once you have created and tested a program you can save it on diskette for future use. The commands described in this section can be used to save the program on diskette, read it as a file, read and automatically execute it, or read the program and append it to the statements currently in BASIC. Additional commands allow you to kill files or make a listing of all files of a specified type.

### Text and Semi-Compiled Modes of Program Storage

The four commands involved in storing and retrieving programs from diskette are SAVE, GET, APPEND, and XEQ. Only SAVE has optional parameters T, for text mode of storage, or C, for semi-compiled mode of storage. In text mode, the current program is saved literally, as the program would appear when listed. If a program may be used with other versions of BASIC, or other editors, it should be saved in this form. In semi-compiled mode, the program is partially compiled, and is stored on diskette in a condensed form which saves space, allows programs to be recorded and accessed faster. The semi-compiled program may not be intelligible to other versions of BASIC, however, and cannot be manipulated in a meaningful way by other editors.

### Commands for Handling Diskette Program Files

Most of the commands for manipulating diskette program files, which are described next, use the following general form:

COMMAND file name

The file name is the name of a CP/M file, and subject to the same conditions as any other CP/M file. The file name can be from one to eight alphanumeric characters. Two extra characters can prefix the file name which specify the diskette drive unit to be used in the command.

If the file name is given alone in the command, without a unit specification, the default unit (usually A) is used. CP/M allows the user to change which unit will be the default unit.

**SAVE** **SAVE**

FUNCTION: To save the current program on a diskette file.

FORMAT:

SAVE file-name [,mode]

RULES:

1. The SAVE command writes the current program on a disk file and labels the file with the specified name.

2. If the diskette already contains a file of the specified name, that file will be overwritten.

3. The mode option can be either T or C. T specifies that the verbatem text of your program is to be saved. And C specifies that a semi-compiled version of the program is to be saved. C (semi-compiled) is the default option and need not be specified.

4. The C (semi-compiled) version is more efficient, loads more quickly, can be saved more quickly, might be dependent on version of BASIC in use, and cannot be edited by external editors.

5. The T (text) version is recognizable as a sequence of BASIC statements, can be edited by editors outside BASIC, and is independent of the version of BASIC in use.

6. For programs you intend to preserve and use frequently, it is best to save in both modes: in text mode to preserve complete documentation and enable compatibility with other editors, and in semi-compiled form for rapid loading.

7. Both the T and C modes, create a CP/M file with the file extension type of (.BAS), if no file extension is explicitly given.

EXAMPLE:

```
User:   10 PRINT "ENTER INTEREST RATE" <CR>
        20 INPUT R <CR>
        25 S = 1 <CR>
        30 FOR I = 1 TO 100 <CR>
        40 S = S + S*R <CR>
        50 IF S >= 2 THEN 70 <CR>
        60 NEXT I <CR>
        70 PRINT "INVESTMENT DOUBLES IN ",I;"YEARS" <CR>
        SAVE INV <CR>
BASIC:   (Records the program on diskette)
        Ready
```

**GET** **GET**

FUNCTION: To read the specified file from disk.

FORMAT:

GET file-name

RULES:

1. The GET command searches the directory for the specified file, then reads the file making the program contained on it available in BASIC.

2. Any statements residing in BASIC before the file was read are lost.

3. The GET command determines whether the file was SAVEd in text or semi-compiled form and acts accordingly.

4. The file extension (.BAS) will be used if now file extension is explicitly given.

EXAMPLE:

```
User:   LIST <CR>          BASIC generates no listing--there
                           are no statements residing in BASIC
        GET INV <CR>
BASIC:   (Reads the file from diskette)
        Ready

User:   LIST <CR>
BASIC:  10 PRINT "ENTER INTEREST RATE"
        20 INPUT R                          BASIC now contains
        25 S = 1                            the program that
        30 FOR I = 1 TO 100                 was read from
        40 S = S + S*R                      diskette.
        50 IF S >= 2 THEN 70
        60 NEXT I
        70 PRINT "INVESTMENT DOUBLES IN ",I;"YEARS"
```

**XEQ** **XEQ**

FUNCTION: To read the specified file from diskette and execute the program contained in it.

FORMAT:

XEQ file-name

RULES:

1. The XEQ command reads the specified file, making the program contained on it available in BASIC, and begins execution.

2. Any statements residing in BASIC before the file was read are lost.

3. The file extension (.BAS) will be used if no file extension is explicitly given.

EXAMPLE:

```
User:   XEQ INV <CR>
BASIC:  ENTER INTEREST RATE      BASIC begins execution of
        ?                        the program contained on
                                 the file INV.
```

**APPEND** **APPEND**

FUNCTION: To read the specified file from disk and merge the program contained on it with the statements already residing in BASIC.

FORMAT:

APPEND file-name

RULES:

1. The APPEND command searches the directory for the named file. Without erasing the statements currently in BASIC, it reads the file and merges the statements found there with the existing statements.

2. The line numbers of statements from the appended file determine their positions with respect to the statements already in BASIC.

3. If a line number from the file is the same as that of a statement residing in BASIC, the statement from the file replaces the previous statement.

4. Only T (text) files can be appended.

5. The file extension (.BAS) will be used if no file extension is explicitly given.

EXAMPLES:

```
User:   LIST <CR>
BASIC:  10 LET X=0
        20 PRINT "ENTER Y AND Z"
        30 INPUT Y,Z

User:   APPEND PART2 <CR>
BASIC:  (Reads the file from diskette)
        Ready

User:   LIST <CR>
        10 LET X=0                  Now BASIC contains the
        20 PRINT "ENTER Y AND Z"    statements read from
        30 INPUT Y,Z                diskette as well as
       100 A1=X+Y+Z                 the original statements.
       110 A2=X+Y-Z
       120 A3=X-Y+Z
       130 PRINT A1,A2,A3
```

**KILL** **KILL**

FUNCTION: To kill (erase) the named file.

FORMAT:

KILL file-name

RULES:

1. KILL performs operations that may be thought of as "erasing" the named file: the file name is removed from the directory, and the space used by the file is made available for other files.

EXAMPLE:

```
'KILL JEC.BAS
KILL A:USELESS.CCC
```

**CAT** **CAT**

FUNCTION: To display a catalog of files of the specified type on the specified diskette drive unit.

FORMAT:

CAT [u:][type.ext]

RULES:

1. CAT reads the directory of the specified unit, or the default unit if none is specified, and prints a listing of files of the specified type.

2. The CAT command will also show is a file has been marked as $R/O or $SYS by the CP/M STAT program. If a file is both, the $SYS is the only attribute shown.

3. If [u:] is omitted, then the default drive is used.

4. If [type.ext] is omitted, then *.* will be used and all files will be displayed.

# SECTION 4

## A BEGINNER'S SET OF BASIC STATEMENTS

You can write BASIC programs for a multitude of mathematical and business applications using just the statements described in this section. This section tells how to assign values to variables, perform data input and output, stop a program, control the sequence in which statements are executed, and make logical decisions. These include the simpler BASIC concepts. After you have become familiar with the statements presented in this section, read Section 5 to learn about the more advanced BASIC concepts.

**REM** **REM**

FUNCTION: To allow comments within a program.

FORMAT:

REM [any series of characters]

RULES:

1. The REM statement allows you to insert comments and messages within a program. It is a good practice to include remarks about the purpose of a program and how to use it.

2. The REM statement has no effect on program execution.

EXAMPLE:

```
 10 REM - THIS PROGRAM COMPUTES THE TOTAL INTEREST
 20 REM - ON A TEN-YEAR LOAN
 30 REM
 40 REM - TO USE IT YOU MUST SUPPLY THE PRINCIPAL
 50 REM - AND THE INTEREST RATE
 60 REM
 70 PRINT "ENTER THE PRINCIPAL"
 80 INPUT P
    .
    .
    .
200 PRINT "THE INTEREST IS ";I
```

**LET** **LET**

FUNCTION: To assign the value of an expression to a variable.

FORMAT-1:

[LET] variable = expression

FORMAT-2:

[LET] variable1=expression1, variable2=expression2, ...

CALCULATOR MODE.

RULES:

1. The LET statement evaluates an expression and assigns its value to a variable.

2. The variable may be a numeric or string variable and the value of the expression can be a number or a character string.

3. The value of the expression and the variable must be the same type.

4. The equal sign is not a mathematical "equals" operator. It is an assignment operator. Thus A = A + B assigns to A the previous value of A plus the value of B.

5. The word "LET" is optional; LET X=1 is equivalent to X=1.

EXAMPLE:

```
10 LET A=0, B=100, C$="FIRST"
20 PRINT A, C$
30 A = A + B, C$ = "SECOND"
40 PRINT A, C$
```

## GETTING DATA INTO AND OUT OF THE PROGRAM

A program must read and write information to communicate with a user. Using the INPUT and PRINT statements is the simplest way to have your program perform input and output.

The INPUT statement reads data typed at the terminal. The form of the PRINT statement described next displays information at the terminal's display device. Using these two statements, you can make your program converse with a user at the terminal.

**INPUT** **INPUT**

FUNCTION: To read one or more values from the terminal and assign them to variables.

FORMAT-1:

```
INPUT var1 [,var2, ...]
```

FORMAT-2:

```
INPUT "message", var1 [,var2, ...]
```

RULES:

1. The INPUT statement accepts one or more values entered at the terminal and assigns them in order to the specified variables.

2. The values entered must agree with the type of variable receiving the value.

3. When an INPUT statement is executed, BASIC requests values from the terminal by printing a question mark or, in the case of format-2, the message.

4. You may enter one or more values after the question mark, but not more than one are required by the INPUT statement.

5. If you enter several values on one line, they must be separated by commas.

6. BASIC prompts for additional value with two question marks until all values required by the INPUT statement have been entered.

7. If the message used is "" (no message) then the normal ? prompt is surpressed.

8. If a comma is placed in the statement after the word INPUT, then the carriage return and line feed will be supressed when the user depresses the carriage return key. In this way the next message printed by BASIC may appear on the same line.

9. If an INPUT statement requests input for a numeric variable, and the user's response contains and inappropriate character, the message INPUT ERROR, RETYPE appears, and the

user is given another chance to type appropriate values. If the ERRSET statement is in effect, no message is given, but an IN error message is made availabe through the ERR(0) function.

EXAMPLES:

```
10 PRINT "ENTER VALUES FOR A, B, C, & D "
20 INPUT A,B,C,D
30 PRINT "A*B/C*D IS ";A*B/C*D
```

When executed, this program accepts data from the terminal as follows:

```
User:   RUN <CR>
BASIC:  ENTER VALUES FOR A, B, C, & D
User:   ?5.7 <CR>          The user types values in response
        ??8.9, 7.4 <CR>    to BASIC's ? prompt. Notice that
        ??10.5 <CR>        one or more can be typed per line.
BASIC:  A*B/C*D IS 71.981757
```

When a message is included in the INPUT statement, that message is displayed as a prompt before data is accepted from the terminal. For example:

```
User:   10 INPUT "WHAT IS YOUR NAME? ", N$ <CR>
        20 PRINT "HI ";N$ <CR>
        RUN <CR>
BASIC:  WHAT IS YOUR NAME? SUE <CR>  -The user types SUE in
        HI SUE                       response to the prompt.
```

The next examples illustrates supressing the carriage return line feeds by using a comma. See line 10.

```
User:   10 INPUT, "GIVE A VALUE TO BE SQUARED: ",A
        20 PRINT " *"; A; " ="; A*A
        RUN <CR>
BASIC:  GIVE A VLAUE TO BE SQUARED: 3 * 3 = 9
```

The user typed only 3 <CR> as input; BASIC completed the line.

**PRINT** **PRINT**

FUNCTION: To display information to the terminal.

FORMAT-1:

```
PRINT [exp1, exp2, ...]
```

FORMAT-2:

```
PRINT [exp1; exp2; ...]
```

CALCULATOR MODE.

RULES:

1. The PRINT statement displays the value of each expression at the terminal.

2. Each expression is displayed in order and the separation between one value and the next is determined by the separator used.

3. If a comma is used as a separator, each value is printed at the left of a field of 14 character positions.

4. If a semicolon is used between two expressions, the second is printed one space after the first.

5. The output of each PRINT statement begins on a new line unless the statement ends with a separator. In this case, the next PRINT statement will cause values to be displayed on the same line and the separator will determine the position at which the cursor (or print head) will remain.

6. The following expressions can be used in a PRINT statement for further control over the position of output:

| | |
|---|---|
| TAB(exp) | Causes the cursor to move horizontally to the character position given by the value of exp (any numerical expression.) This function may only be used in a PRINT statement. |
| "\c" | Prints the control character c. Printing certain control characters performs a function on the video display. Note that the character is preceeded by a back slash (\). A few of the special control |

characters and their functions are:

Control M - Carriage Return
Control J - Line feed

7. While the PRINT statement is executing and values are being output, it is possible to interrupt the printing by depressing the Control-S on the keyboard. Depressing any key will cause printing to resume.

8. More complex forms of the PRINT statement are covered in Section 5.

EXAMPLES:

```
User:   10 PRINT 5, 10, 15; 20 <CR>
        RUN <CR>
BASIC:  5             10            15 20
```

```
User:    5 LET A1 = 1, A2 = 2, A3 =3, A4 = 4 <CR>
        10 PRINT A1;A2; <CR>
        20 PRINT A3,A4 <CR>
        30 PRINT "NEXT LINE" <CR>
         RUN
BASIC:  1 2 3           4
         NEXT LINE
         Ready
```

```
         10 PRINT TAB(I); "DECIMAL";TAB(I+30);"ENGLISH"
        100 PRINT X,"\J", Y, "\J", Z
```

Statement 10 above prints ENGLISH 30 columns beyond DECIMAL; 100 prints the values of X, Y, and Z, each on a new line.

```
         10 PRINT X
        100 PRINT "THE SUM IS "; A+B+C+D
        200 PRINT X,Y,Z;A,B/X;L$
```

**LPRINT** **LPRINT**

FUNCTION: To list information to the systems printer.

FORMAT-1:

LPRINT [exp1, exp2, ...]

FORMAT-2:

LPRINT [exp1; exp2; ...]

CALCULATOR MODE.

RULES:

1. The LPRINT statement lists the value of each expression at the systems printer.

2. Each expression is listed in order and the separation between one value and the next is determined by the separator used.

3. If a comma is used as a separator, each value is printed at the left of a field of 14 character positions.

4. If a semicolon is used between two expressions, the second is printed one space after the first.

5. The output of each LPRINT statement begins on a new line unless the statement ends with a separator. In this case, the next LPRINT statement will cause values to be listed on the same line and the separator will determine the position at which the print head will remain.

6. The following expressions can be used in a LPRINT statement for further control over the position of output:

| | |
|---|---|
| TAB(exp) | Causes the cursor to move horizontally to the character position given by the value of exp (any numerical expression.) This function may only be used in a LPRINT statement. |
| "\c" | Prints the control character c. Printing certain control characters performs a function on the printer. Note that the character is preceeded by a back slash (\). A few of the special control |

characters and their functions are:

Control M - Carriage Return
Control J - Line feed

7. While the LPRINT statement is executing and values are being output, it is possible to interrupt the printing by depressing the Control-S on the keyboard. Depressing any key will cause printing to resume.

8. The LPRINT statement can also have the same function as the PRINT FORMATTED statement described later.

EXAMPLES:

```
User:   10 LPRINT 5, 10, 15; 20 <CR>
        RUN <CR>
BASIC:  5             10            15 20
```

## RETRIEVING DATA FROM WITHIN A PROGRAM

You can place data in a BASIC program using the DATA statement and access it as needed using the READ statement. The RESTORE statement allows you to start reading data again from the first DATA statement or from a specified DATA statement. The TYP(0) function allows you to determine the type of data to be read from the DATA statement corresponding to the next READ statement.

Data may also be stored as diskette data files. This subject is covered in Section 5.

**READ** **READ**

FUNCTION: To read one or more values from DATA statements and store them in variables.

FORMAT:

```
READ var1 [,var2, ...]
```

RULES:

1. The READ statement reads one or more values from one or more DATA statements and assigns the values to specified variables.

2. The value read must be the same type as the variable it is assigned to.

EXAMPLES:

```
 10 READ X2
100 READ A1, A2, A3, M$
```

**DATA** **DATA**

FUNCTION: To specify one or more values that can be read by a READ statement.

FORMAT:

DATA constant1 [, constant2, ...]

RULES:

1. The DATA statement is used with the READ statement to assign values to variables.

2. The values listed in one or more DATA statements are read sequentially by the READ statement.

EXAMPLE:

```
 10 DATA 47.12
100 DATA "ALPHA",400,"BETA",22.6,"GAMMA",74.4
```

or

```
User:   10 DATA 44.2,76.4,18.9 <CR>
        20 DATA 100,47.8,11.25 <CR>
        30 READ A,B,C,D <CR>
        40 PRINT "SUM IS "; A+B+C+D <CR>
        50 READ X,Y <CR>
        60 PRINT "SUM IS "; X+Y <CR>
        RUN <CR>
BASIC   SUM IS 239.5              (44.2+76.4+18.9+100)
        SUM IS 59.05              (47.8+11.25)
        Ready
```

**TYP(0)** **TYP(0)**

FUNCTION: To indicate the data type of the next DATA item.

FORMAT:

```
TYP(0)
```

RULES:

1. The TYP(0) statement returns values 1, 2, or 3, depending on the type of the next DATA item which will be read by the next READ statement.

2. The values returned are:

```
1 = numeric data
2 = string data
3 = data exhausted
```

3. TYP(0) does not work for file READ.

EXAMPLE:

```
10 IF TYP(0) = 3 THEN 30
20 READ X
```

The example above skips a READ statement if the data in the corresponding DATA statement is exhausted.

**RESTORE** **RESTORE**

FUNCTION: To reset the pointer in the DATA statement so that the next value read will be the first value in the first DATA statement.

FORMAT:

RESTORE [n]

RULES:

1. The n represents a statement number.

2. The RESTORE statement lets you change the reading sequence in DATA statements.

3. You can start over or move to a particular DATA statement.

EXAMPLE:

```
User:   10 READ X,Y,Z <CR>
        20 PRINT X+Y+Z <CR>
        30 RESTORE 70 <CR>
        40 READ X,Y,Z <CR>
        50 PRINT X+Y+Z <CR>
        60 DATA 100 <CR>
        70 DATA 200,300 <CR>
        80 DATA 400 <CR>
        RUN <CR>
BASIC:  600                     (100+200+300)
        900                     (200+300+400)
        Ready

        10 RESTORE
       100 RESTORE 50
```

**ON...RESTORE** **ON...RESTORE**

FUNCTION: To specify the line from which the next data statement will be read.

FORMAT:

ON exp RESTORE n1 [,n2,...]

RULES:

1. The ON...RESTORE statement lets you specify the line from which the next data statement will be read.

2. The next READ statement will start reading from the DATA statement selected.

EXAMPLE:

```
10 READ X, Y, Z, A, B, C
20 ON X-Y RESTORE 100, 110, 120
.
.                          The first two values read
.                          determine which line will
100 DATA 4,1,0,4,7,2       be read next.
110 DATA 3,2,7,2,8,1
120 DATA 2,0,3,0,2,2
```

## STOPPING OR DELAYING EXECUTION

There are two ways to stop execution of a program from within the program. The END statement ends execution of a program. The STOP statement stops execution and displays a message telling where execution stopped. The CONT command can be used to resume execution at the next statement. However, any time a program run terminates due to STOP, END, the Control-C Keys, or an error, all open files are closed. The PAUSE statement can be used to delay execution of the statement following it, for a period of .1 second to 1.82 hours on a 2 MHZ 8080.

**END** **END**

FUNCTION: To terminate the execution of a program.

FORMAT:

END

RULES:

1. The END statement terminates execution of a program.

2. All open files will be closed.

EXAMPLE:

```
10 INPUT "WHAT IS THE DIAMETER ", D
20 PRINT "THE CIRCUMFERENCE IS "; 3.1416*D
30 END
40 PRINT "THE AREA IS "; 3.1416*(D/2)^2
```

When the RUN command is given, only the first three lines of this program are executed. Statement 40 can be executed with the command:

```
RUN 40 <CR>
```

**STOP** **STOP**

FUNCTION: To stop program execution.

FORMAT:

STOP

RULES:

1. The STOP statement stops execution of a program.
2. All open files are close.
3. A message is displayed as follows:

STOP IN LINE n

where n is the line number of the STOP statement.

4. Execution can be continued with the CONT command.

EXAMPLE:

```
User:   LIST <CR>
BASIC:  10 INPUT "WHAT IS THE DIAMETER? ",D
        20 PRINT "THE CIRCUMFERENCE IS ";3.1416*D
        30 STOP
        40 PRINT "THE AREA IS ";3.1416*(D/2)^2

User:   RUN <CR>
BASIC:  WHAT IS THE DIAMETER? 2 <CR> -The user enters 2 for
        THE CIRCUMFERENCE IS 6.2832   the diameter.
        STOP IN LINE 30

User:   CONT <CR>                    -The CONT command
BASIC:  THE AREA IS   3.1416          continues execution
                                      with the next statement.
```

**PAUSE** **PAUSE**

FUNCTION: Causes a pause before execution of the following statement.

FORMAT:

PAUSE nexpr

RULES:

1. The nexpr may be from 1 to 65535.

2. The argument nexpr is first evaluated, and truncated to a positive integer between 1 and 65535.

3. A pause of approximately nexpr tenths of seconds then occurs before the next statement in the program is executed.

4. If nexpr has a value less than 1, it will be truncated to zero and no pause will occur.

5. If nexpr has a value greater or equal to 65536 an error message will appear.

6. The precise duration of the pause is controlled by the clock rate of the microprocessor.

7. Of course multiple PAUSE statements or a loop can create a pause of any length.

EXAMPLE:

```
PAUSE 100          gives a pause of 10 seconds
                   on a 2 MHZ 8080.
```

## EXECUTION CONTROL

The statement described next will allow you to control the order in which statements are executed. With the GO TO and ON...GO TO statements you can branch to a different part of the program. The FOR and NEXT statements let you repeatedly execute a set of statements a specified number of times.

**GO TO** **GO TO**

FUNCTION: To transfer control to another part of the program.

FORMAT:

GO TO n

RULES:

1. The GO TO statement causes a specified statement to be the next statement executed.

2. The statement number (n) should be either greater than or less than the number of the GO TO statement.

EXAMPLE:

```
10 PRINT "ENTER A VALUE FOR X"
20 INPUT X
30 PRINT "X SQUARED IS ";X^2
40 GO TO 10
```

When executed, this program repeats statements 10 through 40 over and over. To escape such an infinite loop, strike the Control-C keys. For example:

```
User:   RUN <CR>
BASIC:  ENTER A VALUE FOR X

user:   ?10 <CR>
BASIC:  X SQUARED IS 100
        ENTER A VALUE FOR X

User:   ?5 <CR>
BASIC:  X SQUARED IS 25
        ENTER A VALUE FOR X
        ? (The user strikes the Control-C keys)
        STOP IN LINE 20
```

**ON...GO TO** **ON...GO TO**

FUNCTION: To depart from the normal sequence of statements.

FORMAT:

ON exp GO TO nl [,n2,...]

RULES:

1. Transfer of control passes to nl is exp is 1, n2 if exp is 2, etc.

2. The ON...GO TO statement lets you branch to one of several statements numbers depending on the value of an expression.

3. If the value of the expression is not an integer, BASIC truncates it to an integer.

4. If there is no statement number corresponding to the value of the expression or truncated expression, the next line is executed.

EXAMPLE:

```
User:   LIST <CR>
BASIC:  10 INPUT "ENTER VALUES FOR X AND Y ",X,Y
        20 PRINT "TYPE 1 TO ADD AND 2 TO SUBTRACT X FROM Y"
        30 INPUT N
        40 ON N GOTO 60,70
        50 GO TO 10
        60 PRINT "THE SUM IS ";X+Y:GO TO 10
        70 PRINT "THE DIFFERENCE IS ";Y-X:GO TO 10

User:   RUN <CR>
BASIC:  ENTER VALUES FOR X AND Y ?23.6,98.04 <CR>
        TYPE 1 TO ADD AND 2 TO SUBTRACT X FROM Y

User:   ?2 <CR>
BASIC:  THE DIFFERENCE IS  74.44
        ENTER VALUES FOR X AND Y ?234,89 <CR>
        TYPE 1 TO ADD AND 2 TO SUBTRACT X FROM Y

User:   ?1.9 <CR>
BASIC:  THE SUM IS 323
        ENTER VALUES FOR X AND Y ? The user types Control-C
        STOP IN LINE 10             keys to escape the loop
```

**FOR** **FOR**

FUNCTION: To execute a set of statements an indicated number of times.

FORMAT:

```
FOR var = expl TO exp2 [STEP i]
 .
 .
 .
NEXT [var]
```

RULES:

1. The FOR and NEXT statements allow you to execute a set of statements an indicated number of times.

2. The variable specified in the FOR and (optionally) NEXT statements increases in value at each repetition of the loop.

3. The variable's first value is expl, subsequent values are determined by adding 1 (or i, if specified) to the previous value, and the final value of the variable is exp2.

4. If the starting value is greater than the ending value in the FOR statement, the statements in the loop are not executed.

5. After var reaches its final value and the loop is executed the last time, the next sequential statement is executed.

6. The value of a variable specified in a FOR statement can be changed within the loop, affecting the number of times the loop will be executed.

EXAMPLES:

```
 5 S=1
10 FOR I=1 TO 10
20 S=S*I
30 PRINT I;" FACTORIAL IS ";S
40 NEXT I
50 PRINT "THE LOOP IS FINISHED AND I = ";I
```

When executed, this program prints the factorials of 1

through 10 as follows:

```
User:   RUN <CR>
BASIC:  1 FACTORIAL IS 1
        2 FACTORIAL IS 2
        3 FACTORIAL IS 6
        4 FACTORIAL IS 24
        5 FACTORIAL IS 120
        6 FACTORIAL IS 720
        7 FACTORIAL IS 5040
        8 FACTORIAL IS 40320
        9 FACTORIAL IS 3628800
        10 FACTORIAL IS 3628800
        THE LOOP IS FINISHED AND I = 10
        Ready
```

The next FOR loop will be executed only once because I is set to its final value during the first pass through the loop.

```
10 FOR I=100 TO 50 STEP -5
20 PRINT I
30 LET I=50
40 NEXT I
```

You can include FOR/NEXT loops within other FOR/NEXT loops provided you do not overlap parts of one loop with another.

```
10 FOR A=1 TO 3
20 FOR B=A TO 30
30 PRINT A*B                    is legal
40 NEXT B
50 NEXT A

10 LET Y =10
20 FOR X=1 TO Y
30 FOR Z=Y TO 1 STEP -2         is not legal
40 PRINT X+Y
50 NEXT X
60 NEXT Z
```

NOTE: A GO TO or ON...GO TO statement should not be used to enter or exit a FOR loop. Doing so may produce a fatal error. Use the EXIT statement, described next, to exit a FOR loop.

**EXIT** **EXIT**

FUNCTION: To transfer control to a statement outside the the current FOR/NEXT loop.

FORMAT:

EXIT n

RULES:

1. The EXIT statement allows escape from a FOR/NEXT loop.

2. The statement number n will be executed next.

3. Only the current FOR/NEXT loop is terminated; if it is nested in others, they will not be terminated.

EXAMPLE:

```
100 FOR I = 1 TO N
110 FOR J = 1 TO I
120 C =C+1
130 IF C> 100 THEN EXIT 300
 .
 .
200 NEXT J
201 IF C< 100 THEN EXIT 300
202 NEXT I
250 END
300 PRINT "MORE THAN 100 ITERATIONS"
```

**ON...EXIT** **ON...EXIT**

FUNCTION: To escape the current FOR/NEXT loop depending on the value of an expression.

FORMAT:

ON exp EXIT n1, n2 [,...]

RULES:

1. The ON... EXIT statement lets you escape the current FOR/NEXT loop to a statement determined by the value of an expression.

2. If exp or its truncated value corresponds to a statement number following EXIT, the current FOR/NEXT loop is terminated and control is transferred to that statement.

3. If exp does not correspond to a statement number following EXIT, the ON...EXIT statement is ignored.

4. The value of exp must correspond to the position of a statement number in the list following EXIT, not to the value of the statement number itself.

EXAMPLE:

```
 10 FOR I = 1 TO 9
 20 READ S
 30 ON S+4 EXIT 500,600,700
  .
  .
100 NEXT I
110 DATA 1,4,3,6,4,7,9,4,-1
115 DATA 4,3,7,5,4,3,4,6,-2
120 DATA 4,9,4,0,4,5,7,8,-3
```

The program above operates as follows: When a value of S is read, it is added to 4 and the result is truncated to an integer. If this integer is +1, the current FOR/NEXT loop is terminated and statement 500 is executed; if the integer is +2, statement 600 is executed; if the integer is +3, statement 700 is executed. If the integer is not +1, +2, or +3, the ON...EXIT statement is ignored.

## EXPRESSION EVALUATION

An expression is any combination of constants, variables, functions, and operators that has a numerical or string value. An expression is evaluated by performing operations on quantities preceding and/or following an operator. These quantities are called operands. Examples of some expression and their operands and operators are:

| Operand | Operator | Operand |
|---|---|---|
| X | + | Y |
| A | OR | B |
| I | ^ | 2 |
| | NOT | X |

The NOT operator precedes an operand. All other operators join two operands.

When BASIC evaluates an expression, it scans from left to right. It performs higher-order operations first, and the results become operands for lower-order operations. For example:

| | |
|---|---|
| A-B > C | The value of A-B becomes an operand for the > operator. |

Thus, operators act on expressions.

The order of evaluation for all BASIC operators is as follows:

| | |
|---|---|
| Highest | - (unary negation)<br>^<br>NOT<br>* /<br>> >= = <> <= <<br>AND |
| Lowest | OR |

where operators on the same line have the same order, and are evaluated from left to right.

You can enclose parts of a logical expression in parentheses to change the order of evaluation. Expressions in parentheses are evaluated first.

BASIC operators are divided into four types: arithmetic, string, logical, and relational.

**Arithmetic Operators**

The arithmetic operators act on numerical operands as follows:

| | |
|---|---|
| ^ | exponentiate |
| * | multiply |
| / | divide |
| + | add |
| - | subtract |

The results are numerical.

Note: BASIC evaluates X*X faster than it does X^2. Evaluation of X*X*X is about the same speed as X^3. Remember that (-X)^Y is not allowed, and that -X^Y is equivalent to (-X)^Y, since unary negation preceedes exponentiation.

**STRING OPERATOR**

The plus operator can be used to concatenate string constants or variables, or expressions. No blanks are added. For example:

```
User:   PRINT "BAR" + "tok" <CR>
BASIC:  BARtok
```

**RELATIONAL OPERATORS**

A relational operator compares the values of two expressions as follows:

expression1 relational operator expression2

The result of a relational operation has a numerical value of 1 or 0 corresponding to a logical value of true or false.

The relational operators are:

| Operator | Meaning |
|---|---|
| = | Equal to |
| <> | Not equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |

The following expressions with relational operators are evaluated for A1 = 1, A2 =2, X =3, and Y = 4:

| | Logical Value | Numerical Value |
|---|---|---|
| A1 > A2 | false | 0 |
| A1 <= A2 | true | 1 |
| X + Y/4 <> 7 | true | 1 |
| X = Y | false | 0 |

**LOGICAL OPERATORS**

The results of a logical operation has a numerical value of 1 or 0, which corresponds to a logical value of true or false. The logical operators AND and OR join two expressions with the following results:

| | |
|---|---|
| expression1 AND expression2 | True only if both expression1 and expression2 are true; otherwise false. |
| expression1 OR expression2 | False only if expression1 and expression2 are false; otherwise true. |

The following expressions are evaluated for A = 1, B = 2, and C = 3:

| | Logical value | Numerical value |
|---|---|---|
| C > B AND B > A | true | 1 |
| C > B AND A = B | false | 0 |
| C = B AND B = A | false | 0 |
| C > B OR B > A | true | 1 |
| C > B OR A = B | true | 1 |
| A > C OR A = C | false | 0 |

The logical operator NOT reverses the logical value of the expression it precedes. For example, if A, B, and C have the values shown above, the values of logical expression using the NOT operator are as follows:

| | Logical value | Numerical value |
|---|---|---|
| NOT (C > A) | false | 0 |
| NOT (A = B) | true | 1 |
| NOT C | false (C is true because it has a nonzero value.) | 0 |

**Logical and Relational Operations in Algebraic Computations**

The numerical value resulting from a logical or relational operation can be used in algebraic computations as shown in the example that follows.

The program below counts the number of 3's in 100 values read from DATA statements:

```
 10 FOR I = 1 TO 100
 20 READ A
 30 LET X = X + (A =3)    When A = 3, X is increased
 40 NEXT I                by 1.
 50 PRINT "OF 100 VALUES ";X;" WERE THREE'S"
100 DATA 1,5,4,7,8,9,9,2,3,4,5,3,2,6,7,8,9,3
110 DATA 4,6,7,4,6,8,2,3,8,4,6,9,6,0,4,0,3,1
  .
  .
  .
```

**EVALUATING EXPRESSIONS IN IF STATEMENTS**

The IF statement evaluates an expression and decides on an action based on the truth or falsity of that expression. The IF statement determines the logical value of a statement as follows:

| Numerical Value | Logical Value |
|---|---|
| 0 | false |
| nonzero | true |

Some examples of expression evaluations in IF statements are:

IF A > B THEN ......
A > B has a value of true (1) or false (0).

IF A THEN ..........
A has a value of true (nonzero) or false (0).

IF A AND B THEN.......
A and B each have a value of true (nonzero) of false (0). A AND B is true only if both A and B are nonzero.

IF A < B > C THEN.....
An expression is evaluated from left to right for operators of the same order. In this example, A < B has a value of true (1) or false (0). That value is then compared to C. (1 or 0) > C is either true (1) or false (0).
Warning: This is not the way to Compare B with A and C. For such a comparison, use the AND operator:

```
IF A < B AND B > C THEN...
```

IF A = B = C THEN...
A = B has a value of true (1) or false (0). That value is then compared to C. (1 or 0) = C is either true (1) or false (0).
Warning: this is not the way to test for

the equivalence of A, B, and C. For such a test, use the AND operator:

```
IF A = B AND B = C THEN...
```

IF A = B + C THEN ...

The arithmetic operation is performed first, giving a value for B+C. Then A is either equal to that value (true or 1) or not equal to that value (false or 0).

**IF** **IF**

FUNCTION: To evaluate a logical expression and then take action based on its value.

FORMAT-1:

```
IF exp THEN n
```

FORMAT-2:

```
IF exp THEN n1 ELSE n2
```

RULES:

1. The IF statement evaluates a logical expression and then takes action based on its value.

2. A true value causes the statement number or statement(s) following THEN to be executed next.

3. If there is an ELSE clause, a false value for exp causes the statement number or statement(s) following ELSE to be executed next.

4. Execution continues with the statement following the IF statement, provided control has not been transferred elsewhere.

EXAMPLE:

```
 10 INPUT "WHAT IS THE TAXABLE INCOME? $",I
 20 IF I <= 2000 THEN T = .01*I : GO TO 200
 30 IF I <= 3500 THEN T = 20 + .02*I : GOTO 200
 40 IF I <= 5000 THEN T = 50 + .03*I : GOTO 200
 50 IF I <= 6500 THEN T = 95 + .04*I : GOTO 200
 60 IF I <= 9500 THEN T = 230 +.06*I : GOTO 200
 70 IF I <= 11000 THEN T =320 +.07*I : GOTO 200
 80 IF I <= 12500 THEN T =425 +.08*I : GOTO 200
 90 IF I <= 14000 THEN T =545 +.09*I : GOTO 200
100 IF I <= 15500 THEN T =680 +.1*I : GOTO 200
110 T = 830 + .11*I
200 PRINT "THE TAX IS $";T
```

# SECTION 5

## ADVANCED BASIC

The statements described in this section make NEVADA BASIC's more powerful features available for use:

1. With subroutines and functions, you can define activities that will be performed when a simple call is made or when a function name is specified.

2. By using string functions and statements, you can manipulate character data.

3. With dimensioned variables, you can set aside storage to quickly and easily manipulate large volumes of data.

4. Using the diskette storage and retrieval commands and statements, you can save data for later use.

5. With the formatting capabilities of the PRINT statement, you can control the appearance of numeric output.

6. Using time and space constraints in the INPUT statement, you can control the response to an INPUT prompt.

7. Through cursor-controlling statements and functions, you can draw on the screen.

8. Calling upon commands as statements in a program, you can set systems characteristics, leave BASIC, and delete the program.

9. With the error control statements, you can predetermine a course of action if an error should occur in a program.

## SUBROUTINES

If you have a particular task that must be performed several times during the execution of a program, you can write a subroutine to perform that task and then simply activate the subroutine at the appropriate time. When a subroutine is called from any point in the program, the statements of the subroutine are executed and then control returns to the statement following the calling statement. Variables are not reset or redefined before or after a subroutine's execution.

In NEVADA BASIC, subroutines are called by specifying the first statement number of the routine in a GOSUB or ON...GOSUB statement. Control returns to the statement after the calling statement when a RETURN statement is encountered.

**GOSUB** **GOSUB**

FUNCTION: To execute a subroutine.

FORMAT:

GOSUB n

RULES:

1. The GOSUB statement causes immediate execution of the subroutine starting at the specified statement number.

2. After the subroutine has been executed control returns to the statement following the GOSUB statement.

3. Calls to subroutines can be included within a subroutine. NEVADA BASIC allows any level of nested subroutines.

4. Nested subroutines are executed in the order in which they are entered.

EXAMPLE:

```
100 P = 2000, Y = 5, R = .06
110 GOSUB 200
120 PRINT "THE PRINCIPAL AFTER 5 YEARS IS "; P
 .
 .
200 REM: This subroutine finds the principal after
210 REM: Y years on an R% investment of P dollars.
220 FOR N = 1 TO Y
230 P = P + R*P
240 NEXT N
250 RETURN
```

EXAMPLE-2:

```
100 GOSUB 200
110 PRINT A
 .
 .
200 FOR I = 1 TO R
210 IF I = R GOSUB 370
220 A = A + X^2
230 NEXT I
240 RETURN
 .
 .
370 INPUT "J=",J
  .
  .
430 RETURN
```

execution of this subroutine is interrupted when I=R. After the subroutine at 370 is executed, statements 220 - 240 are executed and control returns to statement 110.

This subroutine is executed before the execution of the subroutine at 200 is complete.

**RETURN** **RETURN**

FUNCTION: To transfer control to the statement following the GOSUB or ON...GOSUB statement that called the subroutine.

FORMAT:

RETURN

RULES:

1. The RETURN statement causes the exit from a subroutine.

2. When a GOSUB or ON...GOSUB statement transfers control to a set of statements ending with a RETURN statement, the line number of the calling statement is saved and control is returned to that line plus one when the RETURN statement is encountered.

3. A RETURN statement will terminate as may FOR/NEXT loops as necessary to return to the calling GOSUB statement.

4. RETURN statements can be used at any desired exit point in a subroutine.

EXAMPLE:

```
 10 GOSUB 50
  .
  .
 50 X = 700
 60 FOR I = 1 TO X
  .
  .
 90 RETURN
100 NEXT I
```

EXAMPLE-2:

```
 10 X = 100
 20 FOR I = 1 TO X
  .
  .
100 GO SUB 150
  .
  .
150 INPUT X,Y,Z
160 IF X = 0 THEN RETURN
  .
  .
200 RETURN
210 NEXT I
```

**ON...GOSUB** **ON...GOSUB**

FUNCTION: To execute a subroutine, if an expression is true.

FORMAT:

ON exp GOSUB nl [,n2,...]

RULES:

1. The ON...GOSUB executes the subroutine beginning with statement nl if the value of exp is 1, executes the subroutine beginning with statement n2 if exp is 2, etc.

2. The ON...GOSUB evaluates, then truncates the expression (exp).

3. If the truncated value of exp is less than 1 or greater than the number of statements specified, BASIC executes the next line.

4. After the subroutine has been executed, control is transferred to the statement following the ON...GOSUB statement.

EXAMPLE:

```
  5 INPUT "ENTER TWO NUMBERS ",X,Y
 10 PRINT "DO YOU WANT TO ADD (1), SUBTRACT (2),"
 20 PRINT "MULTIPLY (3), OR DIVIDE (4) THE NUMBERS"
 30 INPUT I
 40 ON I GOSUB 100,200,300,400
 50 PRINT "THE ANSWER IS ";A
 60 END
100 A = X+Y
110 RETURN
200 A = X-Y
210 RETURN
300 A = X*Y
310 RETURN
400 A = X/Y
410 RETURN
```

## FUNCTIONS

Functions are similar to subroutines in that they perform a task that may be required several times in a program. They differ in that functions can be used in expressions. For example:

```
10 LET A = SQR(176) + B
```

SQR is the square root function and 176 is its argument. When statement 10 is executed, BASIC computes the square root of 176 and assigns the value to SQR(176); then B is added and the sum is assigned to A.

SQR is one of many functions supplied by NEVADA BASIC. Others are presented on the pages that follow.

Besides the functions supplied by BASIC, you can create your own one-line or multi-line functions using statements described in this section.

**ABS** **ABS**

FUNCTION: To obtain the absolute value of an expression.

FORMAT:

ABS(exp)

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
200 IF ABS(X^2-Y^2) > 10 THEN 250
```

**EXP** **EXP**

FUNCTION: To raise the constant e to the power of an expression.

FORMAT:

```
EXP(exp)
```

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
10 LET X = EXP(X) - LOG(Y)
```

**INT** **INT**

FUNCTION: To obtain the integer portion of an expression.

FORMAT:

```
INT(exp)
```

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
100 PRINT "THE ANSWER IS "; INT(A*B)
```

**LOG** **LOG**

FUNCTION: To obtain the natural logarithm of an expression.

FORMAT:

LOG(exp)

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
10 LET X = EXP(X) - LOG(Y)
```

**LOG10** **LOG10**

FUNCTION: To obtain the logarithm base 10 of an expression.

FORMAT:

LOG10(exp)

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
10 LET X = LOG10(X)
```

**RND** **RND**

FUNCTION: To obtain a random number between 0 and 1.

FORMAT:

RND(exp)

RULES:

1. The exp may be 0, -1, or n.

2. This function behaves as if a table of random numbers were available, and an entry in the table were returned. The selection of which entry in the table is returned depends on the argument:

| Argument | Value returned |
|---|---|
| 0 | Returns the next entry in the table |
| -1 | Returns the first entry, and resets the table printer to the first entry |
| n | Returns the entry following n |

3. Although the random numbers generated are between 0 and 1, numbers in any range may be obtained with an appropriate expression. The following example gives random integers between 1 and 99:

EXAMPLE:

```
20 X = INT(RND(0)*100)
```

**SQR** **SQR**

FUNCTION: To obtain the square root of an expression.

FORMAT:

SQR(exp)

RULES:

1. The expression (exp) must be positive.

EXAMPLE:

```
10 LET A = SQR (176) + B
```

**SGN** **SGN**

FUNCTION: To obtain the sign of the value of an expression.

FORMAT:

SGN(exp)

RULES:

1. The expression (exp) must be a numerical expression.

2. The value is 1 if positive, -1 if negative, and 0 if zero.

EXAMPLE:

```
10 LET A = SGN(B)
```

**SIN** **SIN**

FUNCTION: To obtain the sine of an expression in radians.

FORMAT:

```
SIN(exp)
```

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
10 PRINT "THE SIN OF   Y   IS "; SIN(Y)
```

**COS** **COS**

FUNCTION: To obtain the cosine of and expression in radians.

FORMAT:

COS(exp)

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
100 LET R = SIN (A)^ + COS(A)^2
```

**TAN** **TAN**

FUNCTION: To obtain the tangent of an expression in radians.

FORMAT:

TAN(exp)

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
200 IF TAN(14.7) < 1 THEN 400
```

**ATN** **ATN**

FUNCTION: To obtain the arctangent of an expression in radians.

FORMAT:

ATN(exp)

RULES:

1. The expression (exp) must be a numerical expression.

EXAMPLE:

```
200 IF ATN(14.7) < 1 THEN 400
```

## USER-DEFINED FUNCTIONS

You can define your own functions making them available for use in the current program. A function's value is determined by operations on one or more variables. For example, the definition below determines that any time FNA is specified with two values, it will compute the sum of the squares of those values:

```
10 DEF FNA(X,Y) = X*X+Y*Y
```

(X*X and Y*Y are used instead of X^2 and Y^2 because the * operator is faster than the ^ operator for squaring numbers.)

The function defined in statement 10 can be used as follows:

```
100 A = 50, B = 25
110 PRINT FNA(A,B)
```

When executed, statement 110 will print 50 squared + 25 squared or 3125.

**DEF FN** **DEF FN**

FUNCTION: To allow the user to create a single-line or multi-line function.

FORMAT-1:

```
DEF FNvar (var1, var2, ...) = exp
```

FORMAT-2:

```
DEF FNvar (var1, var2, ... )
 .
 .
RETURN exp
 .
 .
FNEND
```

RULES:

1. FORMAT-1 defines a one-line function that evaluates exp based on the values of var1, var2, etc.

2. FORMAT-2 defines a multi-line function that evaluates exp based on the values of var1, var2, etc.

3. The variables and expression used to define a single-line or multi-line function can be either numeric or string. However, the variables and expression must agree in type. That is, if you are defining a numeric function, use a numerical variable in the function's name and return a numeric value as the value of the expression. The same is true for string functions. See example-2 below.

4. In multi-line function definitions, the value returned is the value of the expression on the same line as the RETURN statement.

5. RETURN statements can be used to exit multi-line function definitions as desired.

6. Each definition must end with a FNEND statement.

EXAMPLE-1:

```
        10 DEF FNX(A,B,C) = A*B/SIN(C)

        100 DEF FNA1(R,S)
        110 X=0
        120 FOR I = 1 TO R
        130 X = X + R*S
        140 NEXT I
        150 RETURN X
        160 FNEND
```

EXAMPLE-2:

```
 10 DEF FNA1(U) = SIN(U) + COS(U)
100 DEF FNA1$(U) = "NON"+U$
200 DEF FNZ(X$) = VAL(X$(2,4))
```

EXAMPLE-3:

```
100 DEF FNL(A,B,X,Y)
110 S = 0
120 FOR I = 1 TO X
130 S= S+X*Y
140 NEXT I
150 IF A > B THEN RETURN S-A  -The value of FNL will be S-A
160 RETURN S-B                -The value of FNL will be S-B
170 FNEND
```

In the above example, the variable names listed in parentheses after FNL in line 100 are called formal parameters. In user-defined functions, all formal parameters are locally defined within the function; if any statement in the function modifies the value of a variable which is also a formal parameter, the value of that variable outside the function will NOT be changed. This is true for numerical variables only, not strings, arrays or matrices.

EXAMPLE-4:

```
 1 Q = 40
10 DEF FNA1(X,Y,Z)
20 X = X+1, Q = X+Y, Z = Q/3
25 S = 4
30 RETURN Z
40 FNEND
50 X = 1, Y = 2, Z = 3
60 PRINT FNA1(X,Y,Z), X, Y, Z, Q, S
RUN
  1   1   2   3   3   4
Ready
```

Note that the values of X, Y, and Z, outside the function were not changed by line 20 which is inside the function. Note also that Q, which was not a formal parameter, WAS changed by line 20. Variable S, introduced within the function, retains its value outside the function.

**FNvar** **FNvar**

FUNCTION: To evaluate a user-defined function with the same name and assign the computed value to itself.

FORMAT:

FNvar(var1, var2, ... )

NOTES:

1. The FNvar function call evaluates a user-defined function with the same name and assigns the computed value to itself.

EXAMPLE:

```
 10 PRINT FNX(A,B)
100 A1 = FNA1(X1,X2,X3)
```

EXAMPLE-2:

```
 10 DEF FNB(I,J)
 20 FOR X = 1 TO I
 30 FOR Y = 1 TO J
 40 Z = Z+Y
 50 NEXT Y
 60 NEXT X
 70 RETURN Z
 80 FNEND
 90 LET U = 2, V = 3
100 PRINT FNB(U,V)                -function call
```

The above program prints 12 (1 + 2 + 3 summed twice). If X and Y were already defined in the main program, this function will change their values.

## CHARACTER STRINGS

A character string is simply a sequence of ASCII characters treated as a unit. NEVADA BASIC performs operations with strings as it does with numbers. The string operations use string constants, string variables, string expressions, and string functions.

### String Constants

You have encountered string constants earlier in this text. THE ANSWER IS, in the statement below, is a string constant:

```
10 PRINT "THE ANSWER IS ";X+Y
```

A string constant is indicated in a program by enclosing the characters of the string in quotation marks. However no quotation marks are used when entering a string value from the terminal. Quotation marks cannot be included as part of a string constant.

The size of a string constant is limited only by its use in the program and the memory available.

Some examples of string constants are:

```
"JULY 4, 1776"
"Dick's stereo"          a string with no charaacters
"APT #"                  is called the null string.
""
```

In NEVADA BASIC all lowercase characters are automatically converted to uppercase, except for characters in strings or REM statements. Lowercase characters in strings can be entered from or displayed on terminals having lowercase capability.

For example:

```
INPUT S$ This string has UPPER- and lowercase characters.
PRINT S$ This string has UPPER- and lowercase characters.
```

Teletypes print lowercase characters as their uppercase equivalents. If you have a terminal without lowercase capability, refer to the terminal's users guide to find out how it treats lowercase characters.

Control Characters can be included in a string. They may be entered by pressing the control key and the character, simultaneously, if the character has no immediate function; or control characters can be typed as \c where c is the character. When a control character is printed, the symbol

for the character is displayed, or if it has a function, the character's function is performed,. For Example:

```
10 PRINT "ALPHA \M\JBETA \M\JGAMMA"
```

prints the following when executed because the function of control-M is carriage return and the function of control-J is line feed:

```
ALPHA
BETA
GAMMA
```

To print a single backslash, use this form: \\".

**String Variables**

A string variable is a variable that can be assigned a string value. To distinguish it from a numerical variable, it's symbol is a single letter followed by a dollar sign; or, a letter, digit, and then a dollar sign. For example: A$, S$, C0$ Z2$.

A string variable can contain one to ten characters unless it's maximum size has been declared as a value larger than 10 in a DIM statement.

The assignment statement assigns values to string variables as it does with numerical variables. For example:

```
 10 LET A$ = "MISSOURI"
100 S$ = A$
200 R$ = "BOX #", T$ = "Address"
```

**String Expressions**

String expressions can include string constants, string variables and any of the string functions described later. In addition, they may include the + operator, which means "concatenate", when used with strings. For example:

```
PRINT "ARGO"+NAUT"    prints ARGONAUT

S$ = "REASON"
PRINT S$ + "ABLE"     prints REASONABLE
```

String expressions are treated like numerical expressions in the LET, INPUT, READ, DATA, and PRINT statements. For Example:

```
 5 PRINT "WHAT IS THE SOURCE OF THE DATA"
10 INPUT S$
```

```
 20 IF S$ = "DATA" THEN 70
 30 INPUT X$, Y$, Z$
 40 PRINT "THE LAST VALUE READ WAS ";Z$
 60 END
 70 READ X$, Y$, Z$
 80 GO TO 40
100 DATA "FIRST", "SECOND", "THIRD"
```

The treatment of strings in logical expressions differs from that of numbers as follows:

1. Strings can be compared using relational operators only within IF statements.

2. No logical operators are allowed in string expressions.

When strings are compared in an IF statement, they are compared one character at a time, left to right. If two strings are identical up to the end of one of them, the shorter is logically smaller. The characters are compared according to their ASCII representations (see Appendix 4). Examples are :

| | | |
|---|---|---|
| "ASCII" | is greater than | "073234" |
| "ALPHA" | is greater than | "AL" |
| "94.28" | is greater than | "# and name" |

The program below shows how an IF statement can be used to compare string values:

```
 10 INPUT "WHAT RANGE OF NAMES DO YOU WANT?",A$,Z$
 20 FOR I = 1 TO 35
 30 READ S$
 40 IF S$ < A$ THEN 60          Notice that 40 and 50
 50 IF S$ <= Z$ THEN PRINT S$   cannot be combined
 60 NEXT I                      because logical
100 REM                         operators are not
110 REM                         allowed.
120 "SMITH,JB", "RONSON,CH" "PEAL,JP", ADAMS,J"
 .
 .
```

**DIM** **DIM**

FUNCTION: To specify the maximum size of a string that can be contained in a variable.

FORMAT:

DIM var (n)

RULES:

1. The DIM statement for strings declares the maximum size of a string variable. The maximum size is specified as an integer between 1 and the amount of memory available.

2. The actual length of the variable at any time is determined by the size of the string currently assigned to it.

3. If a string value with more characters than allowed by the DIM statement is assigned to a variable, the rightmost characters are truncated.

EXAMPLE:

```
10 DIM S$ (12)
20 LET S$ = "ALPHA IS THE FIRST SERIES"
30 PRINT S$
```

When executed, this program prints "ALPHA IS THE", the first 12 characters of the string constant.

**SEARCH** **SEARCH**

FUNCTION: Searches exp2 for the first occurance of expl and sets var to the number of the position at which it is found or 0 if it is not found.

FORMAT:

SEARCH expl, exp2, var

RULES:

1. The SEARCH statement evaluates expl and looks for that string as all or part of the value of exp2.

2. If it is found, its location is given by var.

3. If expl is not found the value of var is 0.

EXAMPLE:

```
10 LET X$ = "ANOTHER"
20 LET Y$ = "THE"
30 SEARCH Y$, X$, A
40 PRINT A
```

When executed, this program prints 4 as the value of A because THE begins at the fourth position of ANOTHER.

**FILL** **FILL**

FUNCTION: Fills the string or substring with a copy of the first character in the string expression.

FORMAT:

FILL string, string expression

RULES:

1. The FILL statement fills a string specified by a string variable or a substring specified by a substring function with a series of characters identical to the character specified by the string expression.

2. If the string expression yields a string containing more than one character, only the first is used.

3. The expression must yield at least one character.

4. One way of displaying a table or other pattern of characters is to use a string variable which represents one line of output. Appropriate elements of the string are then filled with the characters to be displayed.

5. Elements of the string variable that should not show characters may be FILLED with blanks. A blank may be represented as CHR(32) or " ".

6. The FILL statement may also be used as a command.

EXAMPLE:

```
 1 DIM A$(5)
10 FILL A$, "XYZ"
20 PRINT A$
RUN
XXXXX
```

## STRING FUNCTIONS

The functions described next deal with characters and character strings. The substring function lets you extract or alter part of a string. The LEN function gives the current length of a character string. The ASC and CHR functions perform conversions between characters and their USACII codes. The VAL and STR functions convert numbers to strings and vice versa.

**var-substring** **var-substring**

FUNCTION-1: To extract characters from a string variable.

FORMAT-1:

```
var(exp1, exp2)
```

FORMAT-2:

```
var(exp1)
```

RULES:

1. The substring function extracts part of a string allowing that section to be altered or used in expressions.

2. The portion of a string to be extracted is indicated by subscripts between 1 and n, where n is the total number of characters in the string.

3. Expressions may be used which yield a value for the subscripts, provided that the value is greater than 1 and less than the number of characters in the string plus two.

4. Noninteger subscript expressions are truncated to integers.

```
USER:  LET A$ = "HORSES" <CR>
       PRINT A$(4, 6) <CR> SES  Characters 4 through the
                                end of the string are
                                extracted.
```

5. If the subscripts specify a substring not contained within the string it refers to, an error message appears. For example, statements 20 and 30 below result in errors:

```
10 LET X$ = "TERMINAL"
20 LET Y$ = X$ (1,9)
30 LET Z$ = X$ (7,10)
```

6. Substrings can be used to change characters within a larger string as shown in the example below:

```
USER:   100 A$ = "abcdefgh" <CR>
        200 A$(3,5) = "123" <CR>
        300 PRINT A$ <CR>
        RUN <CR>
BASIC:  ab123fgh
```

7. A string may be used as if it were like an array of

subscripted strings.

EXAMPLE:

```
 10  REM.
 20  REM.  CONSTANTS
 30  REM.
 40  LET  L1=6:  REM.  LENGTH OF SUBSTRING
 50  LET  N1=5:  REM.  NUMBER OF SUBSTRING
 60  REM.
 70  DIM  S$(L1*N1)
 80  REM.
 90  REM.  PACK ALL SUB-STRINGS INTO A$
100  REM.
110  FOR I=1 TO N1
120  READ I$
130  LET S$=S$+I$
140  NEXT I
150  REM.
160  REM.  PRINT SUBSTRING OF S$ USING INDEX OF
170  REM.  LOOP FOR POINTER INTO S$
180  REM.
190  FOR I=1 TO 5
200  PRINT S$(I*L1-(L1-1), I*L1)
210  NEXT I
220  END
230  REM. NOTE:  ALL SUBSTRINGS ARE THE SAME LENGTH
240  DATA "APPLE", "BANANA", "FIGS", "MELON", "PEAR"
```

**LEN** **LEN**

FUNCTION: Finds the number of characters in a string.

FORMAT:

LEN (var)

RULES:

1. The LEN function supplies the current length of the specified string (var). The current length is the number of characters assigned to the string, not the dimension of the string.

EXAMPLE:

```
10 DIM S$ (15)
15 LET S$ = "COW"
20 PRINT LEN (S$)
```

When executed, this program prints 3, the length of the string COW.

**ASC** **ASC**

FUNCTION: To supply the USASCII code for the first character in a string expression.

FORMAT:

ASC(exp)

RULES:

1. The ASC function performs conversions between characters and their USASCII equivalents.

2. ASC returns the USASCII code for a character whose value is given by a string expression.

EXAMPLE:

```
10 Z = ASC("A")
20 PRINT Z      will print 41
```

**CHR** **CHR**

FUNCTION: To supply the character whose USASCII code is given by an expression.

FORMAT:

CHR(exp)

RULES:

1. The CHR function performs conversions between characters and their USASCII equivalents.

2. CHR returns a character whose USASCII code is given by the value of a numerical expression.

EXAMPLE:

```
10 PRINT CHR(41)        will print "A"
```

**VAL** **VAL**

FUNCTION: VAL(exp) supplies the numerical value of the string whose value is given by an expression.

FORMAT:

VAL(exp)

RULES:

1. The VAL function performs conversions between decimal numbers and strings that can be converted to numbers.

2. The VAL function evaluates the string argument as a number. Evaluation stops on the first character which is not legal in an arithmetic constant.

EXAMPLE:

```
10 Z = VAL("1+3+5")
20 PRINT Z              will print 9
```

**STR** **STR**

FUNCTION: STR(exp) supplies the string value of the number whose value is given by an expression.

FORMAT:

STR(exp)

RULES:

1. The STR function performs conversions between decimal numbers and strings that can be converted to numbers.

2. The STR function produces a string that represents the result of its argument, based on the current default number printing format set by a PRINT statement.

EXAMPLE:

```
10 LET X$ = "33.4"
20 A = 76.5 + VAL(X$)
30 PRINT STR(A)
```

When executed, this program adds 33.4 to 76.5 and assigns the value, 109.9, to A. Then the STR function converts A to a string and prints the string "109.9".

```
USER:   PRINT %#10F3 <CR>
        PRINT STR(100.01) <CR>

BASIC:  100.010     Note the use of the 10 character field

USER:   PRINT %#$C
        PRINT STR (99999999)

BASIC:  $99,999,999     Note the use of the dollar sign ($)
                        and commas (,) as specified in the
                        first PRINT statement.

USER:   PRINT VAL("99,999,999)  This statement will result
                                an IN error due to the $.

     PRINT VAL("99,999,99")  Evaluation will stop the
                             first comma:

BASIC:  99
```

## DIMENSIONED VARIABLES

You can assign many values to a single variable name by allowing additional space for that variable. Such a group of values is called an array and each individual value is an element of that array. The values can be referred to by using subscripts with the variable name. For example, if A1 is an array with 10 elements, individual elements of A1 can be referred to as follows:

```
A1(1)          refers to      the first element.
A1(2)          refers to      the second element.
A1(10)         refers to      the last element.
```

An array can have more than one dimension as in the following two-dimensional, 4 by 3 array:

```
10             15             30
 8.2            7.4            8.6
11.4            4.0            15
 8             11              8.4
```

A two-dimensional array is referred to as a matrix. The elements in the example above are referred to by using two subscripts. For example, if the name of the preceding array is T:

```
T(1,1) = 10
T(1,2) = 15
T(1,3) = 30
T(2,1) = 8.2
T(4,3) = 8.4
```

To assign additional space to a variable name so that it can contain an array of values, you must dimension it with the DIM statement. The number of dimensions is determined by the number of subscripts specified in the DIM statement.

**DIM** **DIM**

FUNCTION: To define an array with one or more dimensions.

FORMAT-1:

```
DIM var (expl, exp2,...)
```

FORMAT-2:

```
DIM varl (expl,exp2,...), var2 (exp3,exp4,...),...
```

RULES:

1. The DIM statement allots space for an array with the specified variable name.

2. The number of dimensions in the array equals the number of expressions in parentheses following the variable name.

3. The number of elements in the array is the product of the expressions.

4. Elements of an array are referred to as follows:

```
var(expl, exp2,...)
```

5. String dimension expressions can be included as well.

EXAMPLE:

```
 10 DIM R(5,5)
 20 FOR I = 1 TO 5
 30 FOR J = 1 TO 5          These statements store 25
 40 READ R(I,J)             values in matrix R.
 50 NEXT J
 60 NEXT J
 70 INPUT "WHICH ELEMENT?",A,B
 80 PRINT R(A,B)
100 DATA 7.2, 8.4, 9.4, 8.6, 7.2
110 DATA 3.4, 3.7, 3.8, 9.5, 7.8
120 DATA 7.7,2.1,3.2,5.4,5.3,7.6,5.3,6.4,2.1,2.0
130 DATA 4.8, 9.7, 8.6, 8.2, 11.4
```

When executed, this program prints the requested elements as shown below:

```
User:   RUN <CR>
BASIC:  WHICH ELEMENT?  2,3, <CR>
        3.8
User:   RUN <CR>
BASIC:  WHICH ELEMENT?  3,2  <CR>
        2.1
```

The amount of storage necessary for a given array is given by:

```
9 + (dimension1) * (dimension2) * (dimension3)...
```

The amount of storage that can be assigned to a variable is determined by the total storage available to BASIC. The memory limit for BASIC can be changed using the command:

```
SET ML = numeric expression
```

To find out how much free storage you have left at any time, use the FREE(0) function, which prints the number of bytes of space left for program and variables. For example:

```
PRINT FREE (0) <CR>
2960
```

## USING DISKETTE FILES FOR DATA STORAGE

The statements described next allow you to store data on diskette, retrieve it, and perform other manipulations.

A data file is a collection of data items stored on diskettes under one file name. The user may create, manipulate, or destroy a file. Structurally, a file consists of a set of uniformly sized blocks of disk space. The physical block structure is controled by the operating system. There is no limit to the number of blocks in a file, except for diskette capacity.

Data stored in diskette files is more permanant than data stored in variables, arrays, or DATA statements. Once data is placed in a file, it can be changed only by a series of special statements designed to change it. Data stored in variables and arrays disappears if the memory containing it is overwritten or if the systems power is turned off or fails. The capacity of diskette files is much greater than the amount of system memory which could be made available for the data.

Data in diskette files can be accessed in three ways: serial access, serial access with spacing, and random access, each progressively more complex. File READ and file PRINT statements of all three types are available.

In serial access files, data is read or printed as a sequential list of items. Each PRINT statement prints items on the file where the last READ or PRINT statement left off. To read the file, the file is "rewound" to the beginning, and read item by item until the desired items are found, as if the data were stored on magnetic tape. Serial access with spacing is similar to serial access, except that items may be read forward or backward. It is also possible to skip over items in either direction. Random access files have a fundamentally different structure than serial files, described later in this section.

All programs which use diskette data files must request access to the file ("open" the file) with the FILE statement, before any reading or writing takes place. The maximum number of files which can be open at one time is limited. Access to open files may be concluded with the CLOSE statement.

Two forms of the FILE statement are described below: one for opening serial access files, and one for opening random access files.

For each open file there is a pointer in the file called the

file cursor, which keeps track of where the last access ended. Each open file also has an EOF function which keeps track of the last operation performed on the file.

Statements which print data into diskette files can include format elements, as described in Section 5, which do not get printed into the file, but control the format in which the data is printed.

The syntax of most data file statements includes the key word, followed by a series of arguments, separated by commas. Optional arguments are shown enclosed in {braces}. When the commas separating such arguments are not enclosed within the braces, they themselves must be included within the command, even if the argument is not included. This is to "hold the argument's place", when other arguments will follow. If commas are included within the braces, they may be omitted along with the argument. However, no commas are needed after the last argument; the statement does not need to end in a comma.

Two forms of the FILE statement and three forms each for PRINT and READ are described below. Actually there is only one highly general form each for FILE, PRINT, and READ statements, but presentation of the general forms would be hard to understand. The PRINT and READ statements can include a non- zero expression for cursor displacement ("spacing"), or a non- zero expression for a record number (in which case the file is a random access file). Since spacing is used in serial access files but not random access files, and record numbers are used in random access but not serial access files, the expression for one or the other must equate to zero. When the syntax descriptions below allow for "an expression which, if present, must equate to zero", this is the reason.

With certain limitations, data and program diskette files created in BASIC may be manipulated from within CP/M, and CP/M may be used to create files for use in BASIC. See Section 3 for a discussion of this subject and for information about file names for use in the statements below.

**FILE-SERIAL** **FILE-SERIAL**

FUNCTION: To open or create a serial file.

FORMAT:

FILE #n; name, {access}, {ag},,

RULES:

1. This form of the FILE statement must be used prior to any of the following file access statements:

(1) Serial File PRINT Statement
(2) Serial File PRINT with Spacing Statement
(3) Serial File READ Statement
(4) Serial File READ with Spacing Statement

2. The REWIND and CLOSE statements may also be used to manipulate the file after the FILE statement.

3. The FILE statement opens the file (makes it accessible to BASIC), assigns a file reference number for use in the above file access statements, and requests access for reading, printing or both.

4. If the named file does not already exist, this statement will create it, if the access requested was 2 or 3.

5. A file of a given name may be opened with more than one FILE statement, for different purposes, provided that different file reference numbers are assigned.

| Argument | Description |
|---|---|
| n | An expression which equates to a file reference number to be assigned. |
| name | A string literal ("A:FILE.TXT" for example) or string variable (A$ for example) which is the file's name. |
| access | An optional number 1, 2 or 3, which specifies what type of access is requested: |
| 1 | READ only. No subsequent PRINT statements. File must already exist. |
| 2 | PRINT only. No subsequent READ |

statements.
3 READ or PRINT statements.

If the access is not specified, type 3 access will be requested.

ag — An optional access granted variable. A Value of 1, 2 or 3 will be assigned to the variable by the FILE statement, in accordance with the access requested. If no ag variable is used, a comma must be inserted to hold the place. Note that an extra comma must be inserted here (since no record size is specified for Serial Access files).

6. The FILE statement sets the file cursor to the first item in the file and sets its EOF function to 1. (The EOF function is described at the end of this section.) The number of files open at one time is limited (see Section 5). Any open file may be closed with the CLOSE statement. Any termination of the run of a program closes all open files.

7. A given named file may be opened by more than one FILE statement, provided different file reference numbers are assigned.

8. All PRINT statements on the named file must use the first file reference number assigned. Second and subsequent FILE statements assign the value 1 (READ only) in the ag (access granted variable) which prevents printing.

9. Commas must be inserted to hold the places of items, which are not specified in the command, if there are items to follow. No commas are needed after the last item specified.

EXAMPLE:

```
 10 FILE #10; S$, 2,,, 1024
100 FILE #3-F; "file" + STR(3-F)
210 FILE #1; "X", 1, X
```

**FILE-RANDOM** **FILE-RANDOM**

FUNCTION: To open or create a random access file.

FORMAT:

FILE #n; name, {access}, {ag}, {rs},

RULES:

1. This form of the FILE statement is used to open or create a random access file as opposed to a serial access file.

2. The syntax is similar to the Serial Access FILE Statement above, except that an expression is included which specifies a record size.

3. The EOF function is set to 1, as with the Serial Access FILE statement.

4. A random access file contains sub-structures called records, each a uniformly sized collection of data. Statements which access a serial access file, must move sequentially through the file to find or print data, but the various records in a random access file may be accessed directly.

5. The rs (record size) expression specifies how many characters (bytes) can be stored in each record. BASIC actually uses two extra characters for each item (collection of characters) in a record.

6. If 3 items, each containing 30 characters, are printed in a record, BASIC will use 98 characters of the record. If no record size is specified, the statement becomes a Serial Access FILE statement, described above.

7. Every FILE statement used with a random access file must include the rs (record size) argument and each FILE statement which refers to the same named file must specify an rs expression which yields the same value. BASIC cannot maintain the file structure unless this rule is observed.

8. The Random File READ and PRINT statements, described later in this section, include an extra argument which specifies which record will be accessed.

EXAMPLE:

```
FILE #25; "X",,, 200
```

**PRINT-SERIAL** **PRINT-SERIAL**

FUNCTION: To print values sequentially on the referenced file, starting at the current file cursor position.

FORMAT:

PRINT #n; elel {,ele2} {,ele3}...

RULES:

1. A previous FILE statement must have already opened the file; n is the file reference number that was assigned by that FILE statement.

2. elel, ele2, etc., are general expressions which result in numerical or string values to be printed on the file. elel, ele2, etc., may also be format elements.

3. The expressions are printed sequentially forward on the file, starting at the current file cursor position. If this statement is the first statement after the opening FILE statement to use the file, the beginning file cursor position will be at the end-of-file. Otherwise, the file cursor will be where it was left by the last file READ or file Print statement.

4. After a statement of this form, the Serial File READ (without spacing) statement cannot be used on the file. This statement leaves the file cursor positioned at the end of the 1st data item printed. The EOF function for the file is set to 3 (last was PRINT).

EXAMPLE:

```
User:  LIST <CR>
        10 FILE #3; "EMP", 2
        20 DIM S$ (30)
        30 PRINT "ENTER EMPLOYEE NAMES AND SS #'S"
        40 INPUT S$
        50 IF S$ = "END" THEN CLOSE #3: END
        60 PRINT #3; S$
        70 GO TO 40
       RUN <CR>

BASIC:  ENTER EMPLOYEE NAMES AND SS #'S
User:   ?John Dixon 343338749 <CR>
        ?Alfred Dill 322679494 <CR>
            .
            .          Periodically there is a pause while
        ?END <CR>      data is written on a diskette file.
BASIC:  Ready
```

**PRINT-SPACING** **PRINT-SPACING**

FUNCTION: The file cursor of the referenced file is displaced by d, and the values of elel, ele2, etc., are sequentially printed on the file.

FORMAT:

```
PRINT #n, {re}, d; elel {,ele2}...
```

NOTES:

| Argument | Description |
|---|---|
| n | The file reference number assigned when the file was previously opened in a FILE statement. |
| re | An optional expression for record size, if present, must evaluate to zero. Record size may be other than zero only if n specifies a random access file. |
| d | The desired file cursor displacement from its present position. d may range from -65535 to +65535 inclusive. A displacement of 1 prints the next item in the file. A displacement of -1 re-prints the last item accessed. If the displacement d is zero, the file cursor is not moved and the statement functions exactly like the Serial File PRINT statement (without spacing) above. |
| elel, ele2, etc. | General expressions which result in numerical or string values to be printed on the file. These expressions may also be format elements as described in Section 5. One or more expressions may be present. |

RULES:

1. This statement is the same as Serial File PRINT described above, except that the file cursor may be moved before printing. The file which will be printed on must be already opened by a FILE statement.

2. If this type of PRINT statement is the first statement executed on the file, the file cursor will be at the end-of-file.

3. The displacement d will then move the file cursor relative to the end of file. Otherwise the file cursor will be wherever it was left by the last file READ or file PRINT statement.

4. Overprinting old items with larger or smaller items may damage the file structure. For this reason, numerical formatting, is recommended to ensure uniform numerical fields for all items.

5. If strings are printed, some "padding" may be needed to keep a new string the same size as the last item in that position.

6. You must take care to maintain the file structure.

7. This form of the PRINT statement sets the EOF function to 35.

EXAMPLE:

```
 10 PRINT#3,0,-5;A;B,S$,"CONST",%Z10F3,74.8+B*C
100 PRINT #1,, X-4; X(I); Y(J)
```

**PRINT-RANDOM** **PRINT-RANDOM**

FUNCTION: To position the file cursor of the referenced random access file.

FORMAT:

PRINT #n, record {,d}; ele1 {,ele2}...

NOTES:

| Argument | Description |
|---|---|
| n | The file reference number assigned when the file was previously opened in a FILE statement. That FILE statement must have defined the file as a random access file, by the inclusion of the rs argument which specifies record size. |
| record | An expression which evaluates to a record number in the file, or zero, where the file cursor will be placed prior to printing. The expression must not exceed the total number of records in the file plus one; the file cursor cannot be positioned beyond the first nonexistent record. If the expression evaluates to zero, this statement will function exactly like the Serial File PRINT statement. |
| d | An expression for cursor displacement. Since this form of the PRINT statement does not use cursor displacement, this expression must equate to zero, if present. |
| ele1, ele2, etc. | General expression, which result in numerical or string values to be printed on the file or format elements as described in Section 5.6. One or more of either type of element may be present. |

RULES:

1. The file to be printed on must be a random access file and it must be opened by a prior FILE statement. The file cursor is positioned to the beginning of the specified record and the values of elel, ele2, etc., are printed in the record.

2. The EOF function is set to 35.

3. If the sum of the total length of all expressions to be printed, plus the number of such items, is greater than the record size of the file, a record overflow error message is printed and the program run is terminated.

4. If the example PRINT statement above is executed on a file containing three records, then record four will be created and the listed items will be printed into it.

5. The Serial File PRINT statement may also be used to print on a random access file. However, the Serial File PRINT with Spacing statement, may not be used.

EXAMPLE:

```
PRINT #F, 4; "HELLO HUMAN!", "?QUE PASA?"
```

**READ-SERIAL** **READ-SERIAL**

FUNCTION: Items from the referenced file are read and assigned.

FORMAT:

READ #n; var1 {,var2} {,var3}...{statement1: statement2...}

RULES:

1. A FILE statement must have previously opened the file with type 1 or type 3 access.

2. The READ statement reads items, starting at the current file cursor position and assigns them as the values of the variables.

3. One or more variables may be present.

4. The number of values read is equal to the number of variables present in the statement.

5. If this is the first statement which accesses the referenced file after the FILE statement which opened it, reading will begin at the first element of the file. Otherwise, reading will begin from where the file cursor was left by the last access.

6. The statement itself leaves the file cursor positioned just after the last data item read.

7. The EOF function is set to 2.

8. The optional statement(s) is executed only if an end of file is encountered.

EXAMPLE:

```
 10 FILE #1; "VAL",1
 20 DIM A(500)
 30 FOR I = 1 TO 500
 40 READ #1;A(I) : EXIT 200     --Only if the end of
 50 NEXT I                      the file is reached
                                before 500 values are
                                read is statement 200
200 PRINT I;"VALUES READ FROM VAL"        executed.
```

**READ-SPACING** **READ-SPACING**

FUNCTION: To position the file cursor

FORMAT:

```
READ #n, {rn,} d; var1 {,var2}...
     {statement1:  statement2...}
```

NOTES:

| Argument | Description |
|---|---|
| n | The file reference number assigned when the file was previously opened in a FILE statement. |
| rn | An optional expression for record number. Since this form of the READ statement accesses only serial access files, this expression must equate to zero if present. |
| d | The desired file cursor displacement from its present position before reading takes place. d may range from -65535 to +65535 inclusive. A displacement of +1 reads the next item from the file. A displacement of -1 re-reads the last item accessed. If the displacement equates to zero, the file cursor is not moved and the statement functions exactly like the Serial File READ (without spacing) statement above. |
| var1, var2 etc. | Each variable in this list will receive values, unless the end of file (EOF) is reached first, in which case any following optional statements are executed. |

RULES:

1. This statement is the same as Serial File READ (without spacing) except that the file cursor may be moved before reading.

2. A FILE statement must have previously opened the file with type 1 or type 3 access.

3. The file cursor is displaced by d items and enough items are read to fill the variables given.

4. If this type of READ statement is the first statement executed on the file, after the FILE statement, reading will begin with the first item in the file, or the displacement d will move the file cursor relative to the first item.

5. Otherwise, the file cursor will be wherever it was left by the last access. This statement itself leaves the file cursor positioned just after the last item read.

6. The EOF function is set to 18.

**READ-RANDOM** **READ-RANDOM**

FUNCTION: The file cursor of the referenced random access file is positioned to the specified record.

FORMAT:

```
READ #n, rn {,d}; varl {,var2}...
     {statement1:  statement2...}
```

NOTES:

| Argument | Description |
|---|---|
| n | The file reference number assigned when the file was previously opened in a FILE statement. The file must be open with type 1 or 3 access, the FILE statement must have defined the file as a random access file, by the inclusion of the rs argument that specifies record size. |
| rn | An expression which evaluates to a record number in the file, or zero, where the file cursor will be placed prior to reading. The expression must not exceed the total number of records in the file plus one; the file cursor cannot be positioned beyond the end-of-file mark. If the expression evaluates to zero, this statement will function exactly like the Serial File READ statement. |
| d | An optional expression for file cursor displacement. Since the file cursor is displaced by the record expression but not by the file cursor displacement expxression in this form of the READ statement, d must equate to zero if present. |
| varl, | Names of variables which will receive the values read. Enough values will be read to fill all variables present unless the record is exhausted first, in which case any following optional statements are executed. |

RULES:

1. The file to be read must be a random access file and opened by a FILE statement with type 1 or 3 access.

2. The file cursor is positioned to the beginning of the specified record and the values are read into var1, var2, etc., until all variables are filled or the record is exhausted.

3. The Serial File READ statement may also be used to read from a random access file. However, the Serial File READ with Spacing statement may not be used.

4. If the end-of-file (EOF) mark is read, the file cursor wil be left at the end of the file and the EOF function will be set to 38 (last was READ EOF).

5. If the end of the current record is encountered, the file cursor will be left pointing to the first item in the next record and the EOF function will be set to 37 (last was end-of-record).

EXAMPLE:

```
 10 READ #Q, R9, 0; X, Y, Z$  :PRINT "EOF" :END
120 READ #3-F, FNA(X); R9, R8, L$, P
```

**REWIND** **REWIND**

FUNCTION: To rewind the specified files.

FORMAT:

```
REWIND #n1,#n2,...
```

RULES:

1. The REWIND statement positions the file cursors of the referenced files to the first data item in the files.

2. If the EOF function for a file is 3, meaning that the last access was Serial File Print (without spacing), the REWIND statement will end-file the file before REWINDING it.

EXAMPLE:

```
 10 REWIND #3
100 REWIND #I-1,#5
```

**CLOSE** **CLOSE**

FUNCTION: To close the specified files.

FORMAT:

```
CLOSE #n1,#n2, ...
```

RULES:

1. The CLOSE statement makes the specified files unavailable for reading or writing.

2. They cannot be accessed again until another FILE statement requests access.

3. If the EOF function for a file is 3 at the time of the CLOSE, the CLOSE statement will end-file the file at the current cursor position.

4. All the data items after the file cursor are "erased".

EXAMPLE:

```
110 FILE #1; "NAMES", 2
120 PRINT "1; N$
   .                    Here file "1 refers to a
                        file called NAMES.
200 CLOSE "1
210 FILE #1; "SALS", 2
   .                    Here file #1 refers to a
   .                    file called SALS.
```

**PURGE** **PURGE**

FUNCTION: To erase (kill) a file.

FORMAT:

PURGE string

RULES:

1. The file whose name is defined by the string expression is erased.

**EOF** **EOF**

FUNCTION: Supplies the status of the specified file.

FORMAT:

EOF(file number)

RULES:

1. Every diskette data file which has been opened with a FILE statement has an associated End-Of-File (EOF) function.

2. The EOF function supplies the current status of the specified file as follows:

| VALUE OF EOF | MEANING |
|---|---|
| 0 | File number was not assigned |
| 1 | Last operation was FILE |
| 2 | Last operation was READ |
| 3 | Last operation was PRINT |
| 4 | Last operation was REWIND |
| 5 | Last operation was READ EOR (end of record) |
| 6 | Last operation was READ EOF (end of file) |
| 18 | Last operation was Serial File READ with Spacing |
| 19 | Last operation was Serial File PRINT with Spacing |
| 34 | Last operation was Random File READ |
| 35 | Last operation was Random File PRINT |
| 37 | Last operation was Random File READ EOR |
| 38 | Last operation was Random File READ EOF |

EXAMPLE:

```
 10 PRINT EOF(2)
100 IF EOF(I) = 4 THEN 150
```

## CONTROLLING THE FORMAT OF NUMERIC OUTPUT

This section gives additional material about the PRINT statement which prints on the user's terminal or standard output device. Forms of the PRINT statement which print on diskette files are covered in the preceding section, but format elements, as described in this section, may be included in file PRINT statements.

In Section 4 the PRINT statement was described in its simplest form, in which the output is automatically formatted. Additional format specifiers may be added to the PRINT statement which give great control over the format.

**PRINT-FORMATTED** **PRINT-FORMATTED**

FUNCTION: To send information to the console.

FORMAT-1:

PRINT exp, exp,..format element,exp, exp,..

FORMAT-2:

PRINT ele, ele, ele; ele..

RULES:

1. The general form consists of zero or more expressions to be printed according to default format, followed by a format element, followed by one or more expressions to be printed according to the format specified in the format element.

2. The same PRINT statement can also contain additional format elements which control additional expressions which follow them.

3. The format element produces no printed results of its own; it controls the form in which subsequent numbers are printed.

4. A format element controls only the expressions following in the same PRINT statement, up to the next format element, if any.

5. Using a special format option it is possible to redefine the default format used in all following PRINT statements which contain expressions not controlled by a specific format element.

6. A format element has the general form: %{format options} {format specifier}. The percent sign % is required and distinguishes the format element from an expression to be printed.

7. Format options, which are not required, add special features such as commas and define the default format.

8. The format specifier, also not required, defines:

   1) The number of columns to be occupied by a PRINTED expression (field width),

2) The type of number to be printed: integer, floating point or exponential and

3) The number of places to the right of the decimal point to be printed.

9. The following format options are available:

| Option | Purpose |
|---|---|
| $ | Places a dollar sign $ in front of the number. |
| C | Places commas (,) every three places as required, for example: 3,456,789.00 |
| Z | Suppresses trailing zeros after the decimal point. |
| + | Places a plus sign + in front of all positive numbers. (A minus sign - is always printed in front of negative numbers.) |
| # | Sets the format element containing it as a new default format used by subsequent PRINT statements, as well as by expressions immediately following. |
| D | Resets the format to the current default. Since the default format is already defined, this option is used alone only: %D is the complete format element. |

10. Only one format specifier may appear in a format element.

Format specifiers have the following four forms:

| Specifier | Format |
|---|---|
| nI | Integer. Numbers will be printed in a field of width n. n must be between 1 and 26. If the value to be printed is not an integer, an error message will be printed. |
| nFm | Floating Point. Numbers will be printed in a field of width n, with m digits to the right of the decimal point. n must be between 1 and 26 and m must be between 1 and n. Trailing zeros are printed to fill width m, unless the Z option is specified. If the specified field cannot hold all the digits in the value to be printed, the value is rounded up to fit. |

nEm Exponential. Numbers will be printed in a field of width n, with m digits to the right of the decimal point. At the end of the field five characters will be printed containing the letter E, a plus or minus sign, and space for an exponent of one to three digits. The exponent may range from -126 to +126. One and only one digit is printed to the left of the decimal point. The field width n must be at least 7 to contain one significant digit plus the 5 characters of the exponential notation. n must be from 7 to 26 and m must be from 0 to n. Here is an example of a number printed in 10E3 format: 1.234E-123. If the specified field cannot hold all the digits in the value to be printed, the value is rounded up to fit.

none Free Format. If a format element consisting of a percent sign alone is used, the format will become the free format as used in the simple unformatted PRINT statement. In free format, integer, floating point or exponential, format is automatically selected depending on the value of the number to be printed and a field width sufficient to hold all the digits of the number is used. The format options may be added to free format by using a percent sign followed by one or more format options with no format specifier.

11. The field width n in the format specifiers above must be large enough to hold all the characters to be printed, including signs, decimal points, commas, dollar signs and exponents.

12. If the field width is larger than necessary to contain all the characters to be printed, extra blank spaces are added to the left of the printed characters to fill the field. (In its exponent.) Extra field width can be used to create columns of printer output spaced at desired intervals.

13. If semicolons are used to separate the format elements and expressions in a PRINT statement, the field widths given in the format specifiers will be adjoining in the output. This does not mean that numbers printed will have no spaces between; that depends on whether the number fills its field.

14. If commas are used to separate the format elements and expressions, there may be extra space added between the fields. The total width of the output is tabulated at fixed 14-character intervals.

15. If a given number has not used the full 14-characters,

the field for the next number will begin at the next 14-character interval. In other words, if field widths of 14 or less are used, the numbers will appear in 14-character columns.

16. If field widths of 15 to 26 are used, the numbers will appear in 28-character columns. A mixture of semicolon and comma separators may be used to give variable spacing.

17. Normally, after a PRINT statement has been executed, the cursor or print head moves to the beginning of the next line, so that the output from the next PRINT statement appears on a new line.

18. If a semicolon is used at the end of a PRINT statement, the return of the cursor or print head is inhibited so that the output from the next PRINT statement will appear on the same line.

19. If a comma is used at the end, the cursor or print head advances to the beginning of the next 14-character interval, as when commas separate elements within the PRINT statement.

EXAMPLE:

```
10 PRINT A; %C8I; SQR(2 + C); %#10F3
20 PRINT %Z5F1; ((A=12) AND B), %D, A, B,
30 PRINT %; A(1, 1); "next is"; B(2,2)
```

MONETARY FORM:

```
%$C11F2
```

Examples of output:

```
$200.00          $9,983.00
 $35.34        $100,000.00
```

SCIENTIFIC FORM:

```
%Z15E7
```

Examples of output:

```
1.1414    E+  2
9.4015687E-104
3.        E+  0
```

(How format elements can interact)

```
10 PRINT %#$C11F2;
```

This statement sets the monetary form given above as the new default format.

```
20 PRINT A, 42.3, P/I
```

The values of these expressions will be printed according the de-

fault format in statement 10.

```
30 PRINT B9; %+26F8; P, I; %D; P/I
```

B9 will be printed according to statement 10. %+26F8 sets a new format for P and I which follow it. %D resets the format to the default of statement 10. P/I is printed accordingly.

## CONTROLLED INPUT

You can include parameters in the INPUT statement to control the number of characters that can be entered from the terminal and the time allowed to enter them. This feature is useful when you want only certain types of answers to questions, or when testing someone's ability to answer quickly.

**INPUT** **INPUT**

FUNCTION: Enters values from the terminal and assigns them.

FORMAT:

```
INPUT{,}(#chars,t)var1,var2,...
INPUT{,}(#chars,t)"message",var1,var2,...
```

RULES:

1. The controlled INPUT statement lets you specify how many characters can be entered and how much time is allowed for response.

2. As soon as #chars characters have been typed, BASIC generates a carriage return and accepts no more characters.

3. If the user takes more than t tenths of a second to respond, BASIC assumes a carriage return was typed.

4. If the optional comma follows INPUT the cursor will remain where the user left it after typing his response, instead of moving to a new line.

5. If the value of #chars is 0, as many as 131 characters can be entered. If the value of t is 0, the user has an infinite amount of time to respond.

EXAMPLE:

```
 5 DIM A$(3)
10 FOR X = 1 TO 9
20 FOR Y = 1 TO 9
30 PRINT X;"*";Y;" = "
40 INPUT (3, 100) A$
42 IF A$=""THEN PRINT"YOU ARE SURE SLOW!":GO TO 30
45 A = VAL(A$)
50 IF A <> X*Y THEN PRINT "TRY AGAIN":GO TO 30
60 NEXT Y
70 NEXT X
```

When executed, this program accepts a three-character answer from the user and waits 10 seconds for a response. If the user does not respond within 10 seconds, the message YOU ARE SURE SLOW is printed. If the user types the wrong response, the message TRY AGAIN is printed.

## ERROR CONTROL

BASIC detects many kinds of errors. Normally, if an error occurs, BASIC will print one of the error messages listed in Appendix 3. However, using the error-control statements described below, you can tell BASIC to execute another statement in the program instead. The ERR(0) function gives a string containing the last error message provided by BASIC.

**ERRSET** **ERRSET**

FUNCTION: Statement n will be executed if any error occurs, cancelling the last ERRSET statement.

FORMAT:

ERRSET n

RULES:

1. The ERRSET n statement lets you determine that statement n will be executed when any error occurs. The error could be an error that would normally result in one of the error messages listed in Appendix 3.

2. If an error does occur and the ERRSET n statement does cause a transfer to statement n, before statement n is executed, the ERRSET statement itself is cancelled (as if an ERRCLR statement were executed.)

3. The transfer to statement n clears all current FOR/NEXT loops, GOSUBS and user-defined function calls (as if a CLEAR statement was executed.)

6. However, if the ERRSET statement is executed again, it will again set the error trap statement n, as if the ERRSET was encountered for the first time.

EXAMPLE:

```
10 ERRSET 75
```

**ERRCLR** **ERRCLR**

FUNCTION: To clear the last ERRSET statement.

FORMAT:

ERRCLR

RULES:

1. The ERRCLR statement cancels the most recent ERRSET statement.

2. If a statement executed after an ERRCLR statement produces an error, BASIC will print a standard error message (See Appendix 3), rather than going to statement n.

3. However, if the ERRSET statement is executed again, it will again set the error trap statement n, as if the ERRSET was encountered for the first time.

EXAMPLE:

```
 10 ERRSET 75
100 ERRCLR
```

**ON..ERRSET** **ON..ERRSET**

FUNCTION: Establishes which statement will be executed in the event of an error.

FORMAT:

```
ON exp ERRSET n1, n2, ...
```

RULES:

1. The ON...ERRSET allows you to conditionally determine which statement will be executed if an error occurs.

2. Once an error has occurred, the ON...ERRSET statement is no longer in effect, as if an ERRCLR statement had been executed.

EXAMPLE:

```
 10 ON I ERRSET 105, 250, 400
100 ON A-J ERRSET 50, 300
```

**ERR(0)** **ERR(0)**

FUNCTION: Returns a string consisting of the last error message from BASIC.

FORMAT:

ERR(0)

RULES:

1. The ERR(0) function returns a USASCII string constant containing the last error message which appeared on the user's terminal.

2. If the ERRSET statement kept the error message from appearing, then the string contains the error message which would have appeared.

3. The argument 0 must be given. Since error messages can take two forms: "XX ERROR", or "XX ERROR IN LINE 00000", care must be used in comparing the ERR(0) string to other strings.

4. The first two characters in the error message are sufficient to identify which error has occurred and may be used in comparisons.

5. In the example below, the error message string is stored in string variable A$, then the first two characters of A$ are compared with "NI" (not implemented). If there is a match, then a message appears on the terminal.

6. Similar statements can be used to branch to special routines when certain errors occur.

7. If the error detected was a CP/M error, ERR(0) will return "FS ERROR".

EXAMPLE:

```
10 A$ = ERR(0)
20 IF A$1,2="NI" THEN PRINT "DELETED FUNCTION USED"
```

**FREE** **FREE**

FUNCTION: To provide the amount of free storage available.

FORMAT:

FREE(0)

NOTES:

1. To find out how much free storage you have left at any time, use the FREE(0) function, which prints the number of bytes of space left for program and variables.

EXAMPLES:

```
PRINT FREE (0) <CR>
2960
```

**SYST** **SYST**

FUNCTION: Returns miscellaneous systems information.

FORMAT:

SYST (EXP)

NOTES:

1. EXP can have the following values.

1 The Control-C key can be used to abort a running program. This feature can be disabled by the SYSTEM 5 statement. The SYST(1) function returns the value of 1 if the program user typed the Control-C key while its abort function was disabled by a SYSTEM 5 statement. Once the value of 1 is read, it is cleared. A subsequent SYST(1) will return 0, unless the user type a Control-C again.

2. Returns last control character sent. Returns 128 is none sent since last call.

3 Returns the time left from a timed input statement.

4. Returns the count left from a count input.

EXAMPLE:

```
10 A = SYST(1)
```

**SYSTEM** **SYSTEM**

FUNCTION: Special system functions.

FORMAT:

SYSTEM (EXP)

NOTES:

1. EXP can have the following values.

0 = disk reset
3 = control character echo on
4 = control character echo off
5 = control-c on (enable)
6 = control-c off (disable)
7 = control character off
8 = control character on
9 = close all files
11 = system reset

## COMMANDS CAN BE STATEMENTS AND STATMENTS COMMANDS

There are a number of commands that can be included in programs as statements. Most commands that can be statements are used for system control. The SET commands set system characteristics and the BYE and SCRATCH commands let you leave BASIC or erase your program. The Calculator Mode of BASIC, shows how statements may be directly executed without being in a program. Appendix 1, the command and statement summary, lists which commands may be used as statements and which statements as commands.

### THE SET COMMANDS

The SET Commands let you determine system characteristics. Each Set command except SET ML can be used as a statement in a program. Three SET commands related to diskette data files are covered in Section 5. Other SET commands are:

| | |
|---|---|
| SET LL = exp | Sets the ouput line length to exp. LL is initially set to 64. |
| SET ML = exp | Sets the memory limit. BASIC will not use addresses higher than exp for program or data storage. Cannot be used as a program statement. BASIC initially uses all available memory. |
| SET CP = exp | Sets the character polarity: white characters on black rectangles, or black characters on white. If exp is zero, characters will appear in normal polarity as set by the video display circuitry. If exp is other than zero, characters appear in opposite polarity. Can be used as a program statement. Initially 0. |
| SET CM = exp | Sets the cursor mode. If exp is zero, the cursor will not appear. If exp is other than zero, the cursor will appear. Can be used as a program statement. |

Note: SET CP and CM are terminal dependant. Not all terminals support these functions.

EXAMPLE:

User: 10 SET LL = 10 <CR>

```
        20 PRINT "THE LINE IS TOO LONG" <CR>
        RUN <CR>

BASIC:  THE LINE I
        S TOO LONG
```

**BYE AND SCRATCH COMMANDS**

The BYE and SCRATCH command can be used a statement, so you can exit BASIC from a program or erase the current program. For example:

```
        10 PRINT "NOW I'M HERE"
        20 PRINT "NOW I'M NOT"
        30 SCRATCH
```

When executed, this program prints:

```
        NOW I'M HERE
        NOW I'M NOT
```

and then erases itself.

**CURSOR CONTROL**

You can control the position of the cursor or use it to draw on the screen using the CURSOR statement and other devices described in this unit. The current horizontal position of the cursor or print head is given by the POS(0) function.

**CURSOR** **CURSOR**

FUNCTION: To position the cursor.

FORMAT:

CURSOR {expl}{,exp2}

RULES:

1. You can use the CURSOR statement to position the cursor and then use a PRINT statement to display a character or characters in that position.

2. You can also print any of the control characters which has an effect on the screen.

EXAMPLE:

```
10 PRINT "\K"
20 FOR I = .1 TO 3.14 STEP .1
30 LET X = SIN (I)
40 CURSOR I*10,X*10
50 PRINT "*"
60 NEXT I
```

Appendix 4 contains a table of ASCII codes.

**ERASE** **ERASE**

FUNCTION: To clear the CRT screen and home the Cursor.

FORMAT:

ERASE

NOTES:

1. The CRT screen is cleared and the cursor is set to the first line and first position.

**POS (0)** **POS (0)**

FUNCTION: Returns a number between 0 and 131, representing the current horizontal position of the cursor or print head.

FORMAT:

POS(0)

RULES:

1. In Nevada BASIC a line of output from the PRINT statement can be up to 132 characters long. The character positions are numbered 0 to 131 starting from the left.

2. After a PRINT statement and after some other types of operations, the cursor on the video display (or the print head if the output device is a printer or teletype) is left in a new position.

3. The value of the POS(0) function is a number between 0 and 131, representing the current position of the cursor (or print head).

4. If the SET LL = exp command or statement has limited the line length to less than 132 characters, the value returned by the POS(0) function will be limited to the new value.

5. Line length varies with output device. The video display of the Sol Terminal Computer has a line length of 64 characters, but if a line longer than 64 characters is printed, some of the extra characters will be automatically printed on a new line.

6. In the example below, the number of characters remaining on the line (63 - POS(0)) is compared with a string A$ which will be printed.

7. If the string will not fit on the remainder of the line, the statement PRINT is executed which positions the cursor on the beginning of a new line.

EXAMPLE:

```
10 IF (63 - POS(0)) < LEN(A$) THEN PRINT
```

## MACHINE LEVEL INTERFACE

One of the functions of BASIC is to isolate the user from the operations and requirements of the specific computer on which he is working. BASIC does all interpreting and executing of commands and programs on whatever computer is in use and the user is free to concentrate only on the logical flow of his program. He can ignore matters such as the absolute locations of his program and data in memory and the flow of input and output through ports. This isolation could prevent the user from dealing with programs not written in BASIC and from interfacing with other hardware and software, if special tools were not available within BASIC for doing so.

BASIC provides three tools for addressing absolute memory locations and three tools for using I/O ports. The POKE statement stores data in a specified memory address, while the PEEK function reads data from a specified address. The CALL function transfers program control to a routine outside of BASIC. The OUT statement places a value in a specified I/O port, while the INP function reads a value from a specified port. The WAIT statement delays program execution until a specified value appears in a port.

Remember that BASIC assumes all numeric expressions are decimal, so all addresses and port numbers must be converted to decimal before use. Appendix 5 contains a table for conversion between hexadecimal and decimal numbers.

In the descriptions of syntax which follow, "numerical expression between 0 and 255" may be interpreted to mean "any expression allowed in BASIC, which when evaluated, yields a decimal value between 0 and 255."

**POKE** **POKE**

FUNCTION: To write to a memory location.

FORMAT:

POKE expl, exp2

RULES:

1. The POKE statement place a value between 0 and 255 in a specified memory address.

2. Since the 8080/8085/Z80 microprocessor can address 65,536 memory locations, this value is set as a limit to the value of expl.

3. The value of exp2 is converted to a one-byte binary value.

EXAMPLE:

```
10 POKE 4095, 11
```

**OUT** **OUT**

FUNCTION: To write to an I/O port.

FORMAT:

OUT expl, exp2

RULES:

1. The OUT statement place a value between 0 and 255 in a specified I/O port.

2. Since the 8080/8085/Z80 microprocessor has 256 ports, this value is set as a limit to the value of expl.

3. The value of exp2 is converted to a one-byte binary value.

EXAMPLE:

```
100 OUT 248, 0
```

**PEEK** **PEEK**

FUNCTION: To supply the numerical value contained in the specified memory location.

FORMAT:

PEEK(exp)

RULES:

1. The PEEK function returns a value equal to the contents of a memory location.

2. Since the 8080/8085/Z80 processor can address 65,536 memory locations, this value is set as a limit to the value of exp.

3. One byte is retrieved and its value interpreted as a number between 0 and 255.

EXAMPLE:

```
10 X = PEEK(4095)
```

**INP** **INP**

FUNCTION: To supply the numerical value contained in the specified I/O port.

FORMAT:

INP(exp)

RULES:

1. The INP function returns a value equal to the contents of an I/O port exp.

2. Since the 8080/8085/Z80 processor has 256 I/O ports, this value is set as a limit to the value of exp.

3. One byte is retrieved and its value interpreted as a number between 0 and 255.

EXAMPLE:

```
100 Y = INP(249)
```

**LOAD** **LOAD**

FUNCTION: Loads the named CP/M (.OBJ) file into memory and places its starting address in var, if present.

FORMAT:

LOAD string {, var}

RULES:

1. The LOAD statement loads a CP/M (.OBJ) file.

2. If var is present, the file's starting address is placed in it.

3. The file may not be loaded below the "first protected memory address" set upon initialization.

4. The first protected address may be changed with the BASIC SET ML command. This statement may be used as a command.

5. However, in a command, "string" must be the actual file name and not a string.

6. The CALL function may be used to execute the loaded image, with the value of var used for expl.

EXAMPLE:

```
100 LOAD X$, Y
 35 LOAD "GUN"
```

**CALL** **CALL**

FUNCTION: Invokes a machine language program.

FORMAT:

CALL(expl{, exp2})

RULES:

1. The CALL function invokes a machine language program that begins at address expl.

2. If exp2 is given, it must be present as a two byte binary value in the D and E registers of the 8080 when control is transferred.

3. A return address is placed on the 8080 stack, so that a RET or equivalent return instructions at the end of the machine language program may return control to the BASIC program that invoked it.

4. The routine may place a value in the H and L registers to become the value of the CALL function.

5. Since H and L consist of 16 bits, the value returned will consist of a positive integer between 0 and 65535.

**WAIT** **WAIT**

FUNCTION: Program execution pauses for a value given.

FORMAT:

WAIT expl, exp2, exp3

RULES:

1. When a WAIT statement is executed, program execution pauses until a certain value is present in I/O port expl.

2. To determine this value, exp2, exp3 and the value in port expl are converted to one-byte binary values. Each bit in the selected port is "ANDed" with the corresponding bit of exp2.

3. If the result is equal to exp3, program execution continues at the next statement.

4. If the result is not equal to exp3, the program continues to wait for the specified value.

5. Depressing the Control C key will escape from a WAIT statement.

6. Exp2 and the logical AND operation provide a way to mask at the selected port bits which are not of current interest.

NOTES:

1. Assume, for example, that you want a program to wait, until bit 7 at port F8 (hexadecimal) becomes a 1.

2. First look in Appendix 5 and find that the decimal value for F8 is 248, so the first part of the statement is WAIT 248,...

3. Next, create an eight bit binary mask, with only the bit of interest, bit 7, set to 1: 10000000.

4. Note that a 0 results when a 0 in the mask is ANDed with either 0 or 1 from the selected port. Thus the mask has zeros for all the "don't care" bits.

5. The decimal value for 10000000 binary is 128, so the WAIT statement now consists of WAIT 248, 128,...

6. The value from the port is ANDed with the mask and compared for equivalence with exp3.

7. Since the mask 128 or 10000000 sets the last seven bits of of the incoming value from the port to zero, the last seven bits of exp3 must also be zero to achieve a match.

8. You are waiting for bit 7 from the port to become 1. Since you "care" about this bit, bit 7 of the mask is also one, and the result of the AND operation is also one.

9. Thus bit 7 of exp3 should be 1 and the entire byte will be 10000000. Converted to decimal, this value is 128.

10. The complete statement is WAIT 248, 128, 128.

## MATRIX OPERATIONS

A matrix variable is a numeric variable which has been dimensioned with the DIM statement for two dimensions. A branch of mathematics deals with the manipulation of matrices according to special rules. Nevada BASIC contains an extension, described in this section, which allows programs to be written involving matrix calculations according to these special rules. No attempt is made here to present the mathematics of matrices; a prior background is assumed.

Since a matrix has two dimensions, any element is located by two positive integers. One of these integers may be thought of as representing rows and the other columns in a table of values. A three (row) by five (column) matrix arranged as a table and containing real constants is shown below:

| | five columns | | | | |
|---|---|---|---|---|---|
| | 3.1 | 4.6 | 7.0 | 3.1 | 0.0 |
| three rows | 3.1 | 9.9 | 0.0 | 7.2 | 0.0 |
| | 4.4 | 1.9 | 5.6 | 3.3 | 0.0 |

Before any calculations are made involving matrix variables, the program must first declare the variables to be matrices in a dimension satement.

EXAMPLE:

```
10 DIM A(10, 2), B9(1, B+C),...
```

Here, numeric variable A is given dimensions of 10 rows by 2 columns and numeric variable B9 is given dimensions of A rows by B+C columns. Any valid BASIC expression may be used as a dimension. Simple variables and matrices of the same name may co-exist in the same program. The matrix A, declared in the example above, is independent of the variable A which has not been dimensioned. Matrix B9 is therefore given a first dimension equal to the value of numeric variable A, not the number of elements in matrix A.

EXAMPLE:

```
100 DIM C(5, A(9, 1))
```

Matrix C is given 5 rows and a number of columns equal to the value of matrix element A(9, 1). The memory space needed for the 8-digit version to dimension a matrix is given by the following expression.

EXAMPLE:

9 + ((first dimension) * (second dimension) * 6)

Since a matrix such as A may co-exist with a variable A in the program, care must be taken to distinguish the two in program statements. In general, A always refers to the variable, while matrix A must have subscripts (A(I, J)).

Matrix elements may be manipulated by all the methods given in earlier sections of this manual. The program below, for example, adds corresponding elements of matrices X and Y into matrix Z.

EXAMPLE:

```
10 DIM X(5, 5), Y(5, 5), Z(5, 5)
20 FOR I = 1 TO 5
30 FOR J = 1 TO 5
40 Z(I, J) = X(I, J) + Y(I, J)
50 NEXT J
60 NEXT I
```

In this respect a matrix can be treated like any multi-dimensional array. This section presents a special group of statements which can manipulate entire matrices in one statement, as compared to the example program above, while it has the effect of adding two matrices, actually deals with individual matrix elements, one at a time. These special statements all begin with MAT (for matrix). MAT identifies the statement as one dealing with matrices, so within such a statement it is not necessary to include subscripts.

EXAMPLE:

```
10 MAT Z = X + Y
```

The statement accomplishes the same addition process as the program example above, but in only one statement. Note the effect of the same statement without the initial "MAT".

EXAMPLE:

```
10 Z = X + Y
```

Here the value of X+Y would be assigned to variable Z.

In the descriptions of matrix manipulations which follow, mvar is used to refer to a matrix variable. Shape is used to refer to correspondance in dimensions. The matrix defined by DIM A(5,2) has the same shape as the matrix defined by DIM B9(5,2), but the matrix defined by DIM C(3,4) has a different shape. A matrix defined by DIM D(2,5) is said to have dimensions opposite those of matrices A and B9.

**MATRIX INITIALIZATION**

The following three statements may be used to define or redefine the contents of a matrix:

| | |
|---|---|
| MAT mvar = ZER | Sets every element in matrix mvar to zero. |
| MAT mvar = CON | Sets every element in matrix mvar to one. |
| MAT mvar = IDN | Sets the matrix to an identity matrix. mvar must have equal dimensions for rows and columns. |

## MATRIX COPY

If two matrices have the same shape, the values in one may be assigned to the corresponding elements of the other with a statement of the form:

```
MAT mvar1 = mvar2
```

If the matrices in this statement have a different shape, the values will be assigned only where there are corresponding elements with the same subscript.

EXAMPLE:

```
10 DIM A(5, 5), B(10, 2)
20 MAT A = B
```

Here the values in the first five rows of B will be assigned to the five rows of A, but only the first two columns of A will receive new values since B has only two columns. The elements in A which have no corresponding elements in B will retain their original value.

## SCALAR OPERATIONS

Each element of a matrix may be added, subtracted, multiplied or divided by an expression and placed into a matrix of the same shape, using a statement of the form shown.

**SCALER** **SCALER**

FUNCTION: Each element may be arithmetic by an expression of a matrix.

FORMAT:

MAT mvar1 = mvar2 op (expr)

RULES:

1. A statement performs the same scalar operation on each element of a matrix. mvar1 and mvar2 must have identical dimensions.

2. The parentheses around expr are required.

3. Matrix elements such as A(5,4) may appear in expr, but not entire matrices.

4. If mvar1 and mvar2 are the same matrix, the resulting new elements will be placed in the old matrix.

EXAMPLE:

```
10 MAT A = B * (2.3356)
20 MAT C = D / (2.35 * C(I, J) + SIN(X))
30 MAT E = E + (1)
```

## MATRIX ARITHMETIC OPERATIONS

A matrix may be added, subtracted or multiplied (but not divided) by another matrix, and the result placed in a third matrix. A statement of the following general form is used:

```
MAT mvar3 = mvar1 op mvar2
```

Differing rules apply, depending on the arithmetic operator used. In addition and subtraction, mvar1, mvar2 and mvar3 must all have the same shape.

In multiplication:

1. mvar 3 must not be the same matrix as mvar1 or mvar2. No check is made to insure this rule is adhered to. If it is broken, unpredictable results will occur.

2. The first dimension (rows) of mvar3 must be the same as the first dimension of mvar1.

3. The second dimension (columns) of mvar3 must be the same as the second dimension of mvar2.

4. The second dimension (columns) of mvar1 must equal the first dimension (rows) of mvar2.

## MATRIX FUNCTIONS

Two matrix functions may be used to place the inverse or transpose of a matrix into another matrix.

**INVERSE** **INVERSE**

FUNCTION: Places the inverse of mvar2 into mvar1.

FORMAT:

```
MAT mvar1 = INV (mvar2)
```

RULES:

1. mvar1 and mvar2 must not be the same matrix.

2. In both functions, mvar1 and mvar2 must have equal dimensions.

3. No check is made to insure that mvar1 is not the same matrix as mvar2. If they are the same, unpredictable results will occur.

4. As with all functions, the argument must be within parentheses.

EXAMPLE:

```
20 MAT C = INV(D9)
```

**TRANSPOSE** **TRANSPOSE**

FUNCTION: Places the transpose of mvar2 into mvar1. mvar1 and mvar2 must have opposite dimensions.

FORMAT:

```
MAT mvar1 = TRN (mvar2)
```

RULES:

1. mvar1 and mvar2 must not be the same matrix.

2. In both functions, mvar1 and mvar2 must have equal dimensions.

3. No check is made to insure that mvar1 is not the same matrix as mvar2. If they are the same, unpredictable results will occur.

4. As with all functions, the argument must be within parentheses.

EXAMPLE:

```
10 MAT A = TRN(B)
```

## REDIMENSIONING MATRICES

The total number of elements in a matrix is the product of its two dimensions. In any MAT statement, a matrix may be given new dimensions, as long as the number of elements is not increased. The new dimensions are assigned merely by giving the new dimensions in parentheses following the matrix variable name.

EXAMPLE:

```
10 DIM A(20, 20)
20 MAT B = A(25, 5) + 1
```

Here matrix A is redimensioned from 20 by 20 to 25 by 5 and put in matrix B.

To understand how the elements of the original matrix are reassigned by the new dimensions, consider how the matrix initially dimensioned DIM X(2,3) is reorganized by including new subscripts X(3,2). Let us number the original elements:

```
1  2  3
4  5  6
```

Visualize these same elements in an equivalent linear array (as they are actually stored in the computer's memory):

```
1 2 3 4 5 6
```

When the matrix is given new dimensions, elements are taken row by row from this equivalent linear array. When the last element of the first row is filled, the first element of the second row is filled and so forth. Here is the resulting arrangement:

```
1 2
3 4
5 6
```

If there are more elements in the original matrix than in the new matrix, elements at the end of the equivalent linear array are not assigned to the new matrix, but remain available if another redimension should increase the size. A redimension may only be done in a MAT statement and may not be done in a second DIM statement.

The following attempted redimension will not work:

```
DIM A(10, 10)
      .
      .
      .
DIM A(5, 5)
```

A matrix variable may appear in a DIM statement only once. The example above violates this rule.

## BASIC COMMAND AND STATEMENT SUMMARY

Minimum keyword abbreviations are underlined. An abbreviation must be followed by a period. Functions and some commands and statements do not have abbreviations. An S following a command description means it may be also used as a statement; a C following a statement means it may be used as a command.

### COMMANDS

| Command | Description |
|---|---|
| APPEND file<br>-- | Reads a program stored on a diskette file and appends it to the current program. |
| BYE<br>- | Leaves BASIC and returns to CP/M. S |
| CAT {/unit}{type} | Displays a catalog of BASIC program or diskette data files, from the specified disk drive unit, of type T, or C. |
| CLEAR<br>-- | Erases all variable definitions. S |
| CONT<br>-- | Continues execution of a program stopped with the MODE key or by a STOP statement. |
| DEL | Deletes all statements. |
| DEL n | Deletes statement n. |
| DEL n1, n2 | Deletes statements n1 through n2. |
| DEL n1, | Deletes statements n1 through the last statement. |
| DEL ,n2 | Deletes the first statement through statement n2. Note space before comma. |

| | |
|---|---|
| EDIT n<br>-- | Allows the edit of statement n. |
| GET file<br>-- | Reads a diskette file program of type C or T for execution later. |
| KILL file<br>- | Kills the named program file. |
| LIST<br>-- | Lists the entire program. |
| LIST n<br>-- | Lists statement n. |
| LIST n1, n2<br>-- | Lists statements n1 through n2. |
| LIST n1,<br>-- | Lists statements n1 through the last statement. |
| LIST ,n2<br>-- | Lists the first statement through statement n2. |
| LLIST<br>-- | Lists the entire program. |
| LLIST n<br>-- | Lists statement n. |
| LLIST n1, n2<br>-- | Lists statements n1 through n2. |
| LLIST n1,<br>-- | Lists statements n1 through the last statement. |
| LLIST ,n2<br>-- | Lists the first statement through statement n2. |
| REN | Renumbers the statements starting with 10 in increments of 10. |
| REN n | Renumbers the statements starting with n in increments of 10. |
| REN n,1 | Renumbers the statements starting with n in increments of 1. |
| RUN<br>-- | Clears all variable definitions and executes the program begin- |

| | |
|---|---|
| | ning with the first line. |
| RUN n<br>-- | Executes the program beginning with statement n and does not clear variable definitions. |
| SAVE file {,C}{,T}<br>-- | Saves the current program on a diskette file of the name indicated. C saves the program in semi-compiled format or T saves the program in text format. The default is C. |
| SCRATCH<br>-- | Deletes the entire program and clears all variable definitions. S |
| SET CM=exp | If exp equates to zero, the video cursor will not appear; if exp is non-zero, it will appear. S |
| SET CP=polarity | If the polarity expression is zero, video characters will appear in normal polarity; if non-zero, characters will appear in reverse video. S |
| SET LL=length | Sets the line length for BASIC output to the value specified. S |
| SET ML=size | Sets the memory limit for BASIC to the number of bytes specified. |
| XEQ file<br>- | Reads and executes a diskette file program of type C or T. |

## STATEMENTS

| Statement | Description |
|---|---|
| CLOSE #file number1, #file number2,... | Closes the specified files so that they cannot be accessed unless another FILE statement requests access. |
| CURSOR {L}{,C} | Moves the cursor to line L, position C, on the screen. If L or C is omitted, its value from the last CURSOR statement is used. C |
| DATA constant1, constant2,... | Specifies numerical or string constants that can be read by the READ statement. |
| DEF FNvariable(variable1, variable2,...)=expression | Defines a one-line function that evaluates an expression based on the values of the variables in parantheses. |
| DEF FNvariable(variable1, variable2,...) | Defines a multi-line function that executes statements following, using the values of the variables in parentheses in calculations. |
| RETURN expression | and,when a RETURN statement is encountered, returns the value of the expression on the same line. |
| FNEND | FNEND ends the function on definition. |
| DIM variable(dimension1, dimension2,...) | Defines a multi-dimensional numerical array with the number of dimensions specified. |

DIM string variable (size)
--
Declares the number of characters that can be contained in the specified string variable.

END
-
Terminates execution of the program.

ERRCLR
----
Clears the error trap line number set by the most recent ERRSET statement. C

ERRSET n
--
When an error occurs, BASIC executes statement n next. C

EXIT n
--
Escapes from and terminates the current FOR/NEXT loop. Statement n is executed next.

FILE #n; name, {access}, {ag}, {rs}, {bs}
--
Opens or creates a random access diskette data file, or if the rs expression is absent or equates to zero, opens or creates a serial access file. File reference number n is assigned to the named file for use in later statements. An access is requested: 1 for READ only, 2 for PRINT only and 3 (default) for either. If the variable ag is present, it receives the access granted. If rs is present, it specifies the record size of a random access file.

FILL string, string expression

Fills a string variable or substring function with a copy of the first character, which the first string expression yields. C

FNEND
--
Ends a function definition.

FOR variable = expression1 TO expression2 {step interval}
- .
. .
. .
NEXT {variable}
-
The value of expression1 is assigned to the variable, then the statements between FOR and NEXT are executed repeatedly until the variable equals expression2. After each iteration, the variable is incremented by 1 or by

the STEP interval if given.

GOSUB n
---
Executes the subroutine beginning at statement number n. Execution continues with the statement following the GOSUB statement.

GOTO n
-
Transfers control to statement number n.

IF expression THEN n
-
Executes statement n if the value of the expression is true; otherwise, executes the next statement in sequence.

IF expression THEN n1 ELSE n2
- -- --
Executes statement n1 if the value of the expression is true; otherwise, executes statement n2.

IF expression THEN statement1:statement2:...
- --
Executes statement1, statement2, etc., if the value of the expression is true; otherwise, executes the next statement in sequence. C

IF expression THEN statement1:statement2:...
- --
ELSE statement3:...
--
Executes the statements following THEN if the value of the expression is true; otherwise, executes the statements following ELSE. C

IF expression THEN n ELSE statement1:statement2:...
- -- --
Executes statement n if the value of the expression is true; otherwise, executes the statements following ELSE.

IF expression THEN statement1:statement2:...ELSE n
- -- --
Executes the statements following THEN if the value of the expression is true; otherwise, executes statement n.

```
INPUT variable1, variable2,...
--
```

Accepts values from the terminal and assigns them to variable1, variable2, etc. C

```
INPUT "message", variable1, variable2,...
--
```

Displays the message as a prompt and then accepts values from the terminal, assigning them to variable1, variable2, etc. C

```
INPUT (characters, time) variable1, variable2,...
--
```

Accepts values from the terminal and assigns them to variable1, variable2, etc.. The user can only type the number of characters indicated and has time (in tenths of a second) to respond.

```
INPUT (characters, time) "message", variable1, variable2,...
--
```

Displays the message as a prompt and them accepts values form the terminal, assigning them to variable1, variable2, etc.. The user can only type the number of characters indicated in parentheses and has time (in tenths of second) to respond.

```
{LET} variable = expression1 {,variable2=expression2}...
-
```

Assigns the value of each expression to the corresponding variable. The word LET may be absent. C

```
LOAD string {,var}
--
```

Loads the CP/M type (.OBJ) file, whose name is given by the string expression, into memory. The variable receives its starting address. The file may be executed with the CALL function. C

```
LPRINT ele {,ele,ele...}{,}
-
```

Prints numerical or string expression elements according to format elements. Commas or semicolons may separate elements or terminate the LPRINT statement.

MAT mvar=ZER Sets every element in matrix
- mvar to zero. C

MAT mvar=CON Sets every element in matrix
- mvar to one. C

MAT mvar=IDN Sets the matrix to an identity
- matrix. C

MAT mvar1=mvar2 Copies matrix variable 1 into
- matrix variable 2. C

MAT mvar1=mvar2 op mvar2
-

Performs the same scalar operation
on each element of matrix variable 2.
op is + - * or / C

MAT mvar3=mvar1 op mvar2
-

Adds, subtracts or multiplies matrix
variable 1 by matrix variable 2.
op is + - or * C

MAT mvar1=TRN(mvar2)
-

Places the transpose of matrix
variable 2 into matrix variable 1. C

MAT mvar1=INV(mvar2)
-

Places the inverse of matrix
variable 2 into matrix variable 1. C

mvar(expression1, expression2)

Matrix mvar may be redimensioned
by including the new dimensions
expression1 and expression2 after
the matrix variable name in a MAT
statement.

NEXT{variable} Ends a FOR loop.
-

ON expression ERRSET n1, n2,...
- --

If the value of the expression is 1,
sets n1 is the statement to be
executed when an error occurs;
if the value is 2, sets n2 is the
statement to be executed when an
error occurs; etc..

```
ON expression EXIT n1, n2,...
-             --
                If the value of the expression is 1,
                transfers control to statement n1 and
                terminates the currently active
                FOR/NEXT loop; if 2, transfers to
                statement n2; etc..

ON expression GOSUB n1, n2,...
-             ---
                If the value of the expression is 1,
                executes the subroutine starting at
                statement n1; if the value is 2,
                executes the subroutine starting
                at statement n2; etc..

ON expression RESTORE n1, n2,...
-             ---
                If the value of the expression is 1,
                executes statement n1 next; if it is
                2, executes statement n2 next; etc..

ON expression RESTORE n1, n1,...
-             ---
                If the value of the expression is 1,
                resets the pointer in the DATA
                statements so that the next value
                read is the first data item in line
                n1; if it is 2, resets the pointer
                to n2; etc..

OUT port, value
--              Places the specified value in the
                indicated I/O port.   C

PAUSE nexpr     Delays further execution for nexpr
--              tenths of a second.

POKE location, value
--
                Places the specified value in the
                specified memory location.   C

PRINT ele {,ele,ele...}{,}
-
                Displays numerical or string expression
                elements according to format elements.
                Commas or semicolons may separate
                elements or terminate the PRINT
                statement.

PRINT #file number; ele {,ele, ele...}
-
                Sequentially prints the values of
```

numerical or string expression elements, according to format elements, onto the referenced diskette data file. C

PRINT #file number, {record} {,d}; elel {,ele2}...
-

If the file cursor displacement expression d is non-zero, the file cursor is displaced by d and the values of the element(s) are printed on a serial access diskette data file; or if the record number expression is non-zero, the file cursor is positioned to the given record number in a random access data file, and the values of the element(s) are printed.

PURGE string
--

Kills the diskette data file whose name is the value of a string expression.

READ variable1, variable2,...
-

Reads values from DATA statements and assigns them to variable1, variable2, etc..

READ #n;var1 {,var2}...{:statement1 :statement2}
-

Reads values from the specified file starting at the current file cursor position and assigns them to var1, var2, etc. If EOF is encountered, the optional statement(s) are executed.

READ #n,{rn}{,d};var1{,var2}...{:statement1 :statement2}
-

If the file curosr displacement expression d is non-zero, the file cursor is displaced by d and items from a serial access diskette data file are read and assigned to var1, var2, etc.; or if the record number expression rn is non-zero, the file cursor is positioned to the given record number in a random access data file, and items are read into the variables. If EOF is encountered, the optional statement(s) are executed.

REM any series of characters

The characters appear in the program as remarks. The statement has no effect on execution.

RESTORE {n}
---
Resets the pointer in the DATA statements to the beginning. If n is present, the pointer is set to the first data item in statement n.

RETURN
---
Returns from a subroutine.

RETURN exp
---
Returns from a function. The value returned is exp.

REWIND "file numberl, #file number2,...
---
Rewinds the specified files.

SEARCH string expressionl, string expression2, variable
--
Searches the second string for the first occurance of the first string specified. The variable is set equal to the character position at which the first string was found. If is is not found, the variable is set equal to zero.

STOP
-
Terminates execution of the program and prints "STOP IN LINE n", where n is the line number of the STOP statement.

WAIT expl, exp2, exp3
-
The next statement is not executed until the value in port expl, ANDed with exp2, is equal to exp3.

XEQ file
-
Reads the program from the specified diskette file and begins execution. The file name is a string expression so it must be enclosed in quotation marks if given directly.

**APPENDIX 2** **APPENDIX 2**

## BASIC FUNCTION SUMMARY

In the function forms below, which are arranged alphabetically, n represents a numeric expression and s represents a string expression. Function names may not be abbreviated.

| Function | Value Returned |
|---|---|
| ABS(n) | The absolute value of the numerical expression n. |
| ASC(s) | The USASCII code for the string expression s. Only the first character of the string is interpreted. |
| ATN(n) | The arctangent of the numerical expression n in radians. |
| CALL(address{,parameter}) | The value in HL. CALL places a return address on the 8080 stack, calls the routine at the specified memory address and optionally passes the value of a parameter in the DE register. The routine may return a value in HL, which becomes the value of the CALL function. |
| CHR(n) | The character whose USASCII code is the value of numerical expression n. |
| COS(n) | The cosine of n in radians. |
| EOF(file number) | The status of the specified file.<br>0 file number was not assigned<br>1 last operation was FILE<br>2 last operation was READ<br>3 last operation was PRINT<br>4 last operation was REWIND<br>5 last operation was READ EOR (end of record)<br>6 last was READ EOF (end of file)<br>18 last was Serial FIle READ with Spacing<br>19 last was Serial File PRINT with |

| | |
|---|---|
| | Spacing |
| 34 | last was Random File READ |
| 35 | last was Random File PRINT |
| 37 | last was Random File READ EOR |
| 38 | last was Random FIle READ EOF |

ERR(0) A string containing the last error message.

EXP(n) The constant e raised to the power n.

FNvariable(variable1, variable2, ...)

The value of used-defined function FNvariable. variable1, variable2, etc. are arguments.

FREE(0) The number of bytes of space left available in BASIC for program and variables.

INP(exp) Supplies the numerical value contained in I/O port exp. Exp is between and 255.

INT(n) Truncates n to its integer part.

LEN(name) The number of characters in the string variable whose name is specified.

LOG(n) The natural logarithm of n.

LOG10(n) The logarithm base 10 of n.

PEEK(n) The value contained in memory location n.

POS(0) The current position of the cursor (0 - 131).

RND(exp) A random number between 0 and 1. exp = 0, -1 or n.

SGN(n) The sign of the value of n; 1 if positive, -1 if negative, 0 if n is zero.

SIN(n) The sine of n in radians.

SQR(n) The square root of n.

STR(n) The character representation of the value of n.

SYST(n) Returns systems information.

TAB(n) Moves the cursor or print head horizontally to character position n. Use only in a PRINT statement.

TAN(n) The tangent of n in radians.

TYP(0) A value representing the type of data that will be read from the DATA statement corresponding to the next READ statement: 1 for numeric data, 2 for string data, or 3 for data exhausted.

VAL(s) The numerical value of the string s. The value of s must be convertible to a legal numerical constant.

string variable (exp1{,exp2})

Characters exp1 through exp2 of the specified string if exp2 is present. Characters exp1 through the end of the specified string if exp2 is omitted.

numerical variable (n1{, n2, ...})

An element of an array with the specified name. The element's position is given by n1, n2, etc..

APPENDIX 3 APPENDIX 3

## ERROR MESSAGES

All errors are fatal and stop the execution of the program or command causing the error, unless an ERRSET statement is in effect. If the error occurs while writing data on a file or saving a program, some information may be lost. Errors are arranged below alphabetically by error message.

| Message | Meaning | What to Do |
|---|---|---|
| AC | Access error. An attempt has been made to access a file in the wrong mode (read, write or read/write). | Check the File Statement requesting access. Change the access mode if it is incorrect. |
| AM | Argument error. A function has been called with the wrong number or type of arguments. | Review the function's definition in Appendix 2 or in your program if it is a user-defined function. |
| BC | Bad semi-compiled file. | Check your hardward. Re-create file. |
| BV | Bad CP/M version. | Must be version 2.0 or greater. |
| CA | Cannot append. The file indicated in the last APPEND command is the wrong type. It must be a text format file. | SAVE the file in text format. |
| CC | Can't convert. The last VAL function attempted to determine the value of a string which did not contain a number. | Provide a string which contains a number. Study the program logic. |
| CL | File close error. | |
| CS | Control stack error. Possible causes are:<br>-RETURN without a prior GOSUB<br>-Incorrect FOR/NEXT nesting<br>-Too many nested FOR loops | List the statements surrounding the error causing statement and check the logical flow. Execute just a few statements at a |

| | | |
|---|---|---|
| | -Too many nested function calls | time and list variable values to find out where things go wrong. |
| DD | Double definition. An attempt has been made to define a function with a name that is already defined. | Rename the function. |
| DI | Direct execution error. The statement last typed cannot be executed in calculator mode. | Give the statement a line number and execute it as all or part of a program. |
| DM | Dimension error. A dimension statement contains a variable name that is already dimensioned or cannot be dimensioned. | Rename the dimensioned variable. Make sure the variable name is valid. |
| DZ | Divide by zero error. An expression in the last statement attempted to divide by zero. | Set the value of the divisor to a non-zero number before dividing. |
| FD | Format definition error or file declaration error. The last PRINT statement contained a bad format definition, the last statement referring to a file number specified an undeclared file, or the last FILE statement could not declare the file as requested. | Either check the format definition against the documentation under "Formatted PRINT Statement" or find the most recent FILE statement and verify its syntax and the file number declared. |
| FM | Format error. A field definition in the last formatted PRINT statement is not large enough or it is too large. | Use the PRINT statement in calculator mode to determine the size of the value to be printed. Adjust the field declaration accordingly. |
| FN | File name error. A filename is too short, too long or contains illegal characters. | Check for spelling errors or use a different name. |
| FO | Field overflow. An attempt has been made to print a number larger than Nevada BASIC'S numerical field size. | Display values used to compute the number. Trace the source of the overflow in reverse order through the program. |

| | | |
|---|---|---|
| FP | Floating Point error. BASIC cannot handle numbers greater than 10 to the 126th power or less than 10 to the -126th power. | No solution. |
| FS | File Structure error. A CP/M error occurred. | Check diskette for damage. |
| IN | INput error. The ERRSET statement is in effect and non-numeric input was given to a numeric INPUT statement. | Rerun the program, using appropriate input. |
| IS | Internal stack error. An expression was too complex to evaluate. | Divide the expression into parts, using assignment statements. |
| KI | Kill attempted on an open file. | Close file and then retry. |
| LL | Line too long. The next line to be listed is too long for BASIC. It cannot be edited or saved in the text mode. | If you don't know the number of the next line to be listed, renumber the program and give the LIST command again. Replace the long line with shorter lines. You cannot list the long line, so you must reconstruct its meaning from the context of the surrounding statements. |
| LN | Line number reference error. A statement referred to a line that does not exist. | List the area of the program around the line referred to. Find the correct line number and revise the reference. |
| MD | Matrix Dimension Error. Dimensions are incompatible with the operation attempted. | Redimension the matrix or restructure the operation. |
| MP | Memory Protect error. An attempt was made to overwrite BASIC or the current BASIC program. This error can be produced by the LOAD command/statement. | Check image file load address if the LOAD statement was used. |

| | | |
|---|---|---|
| MS | Matrix Singular Error. The operation attempted cannot be performed on a singular matrix. | The operation cannot be performed on the data in the given matrix. |
| NA | Not available. A command is not presently available. | Don't use offending command. |
| NC | Not CONTinuable. The current program, if any, cannot be CONTinued. | Make sure a BASIC program is ready to run. You cannot CONTinue after editing a program, using the CLEAR command,etc. |
| NI | Command or function not implemented. | |
| NP | No program. BASIC was instructed to act on the current program and none exists. | Type the program or read it from diskette. |
| OB | Out of bounds. The argument or parameter given is not within the range of the function or command last executed. | Display the values of the argumnts or parameters used. If they seem reasonable, look up the definition of the function or the command. |
| OP | File open error. | |
| RD | Read error from file. | File may not exist or bad data. |
| RO | Record overflow. An attempt was made to write more items into a record of a random access date file than the record could hold. | Write the extra items into a new record, write less items per record, or rewrite the file with a new record size. |
| RW | Rewind error. | |
| SN | Syntax error. The statement or command last executed was constructed incorrectly. | Check the syntax of the command or statement in Appendix 1. |
| SO | Storage overflow. There is insufficient storage to complete the last operation. | Use the FREE command to find out how much storage is left. Use SET ML to change the memory limit for |

| | | |
|---|---|---|
| | | BASIC. |
| TY | Type error. The variable or function name appearing in the last statement is the wrong type. The types are string variable, simple variable, dimensioned variable and function. | Check the names of functions and dimensioned variables. Make sure the operation is appropriate for the type of data indicated. |
| UD | Undimensioned matrix. A variable name was used which was not previously. | DIMension the matrix in an earlier DIM statement. |
| UR | Unresolved line number reference. During a RENumber command, a control transfer statement referred to a nonexistent line number. | Look for typos in the program. Unresolved references will have been changed to 0. |
| WT | File write error. | Disk may be full. |

## APPENDIX 4 TABLE OF ASCII CODES (Zero Parity) APPENDIX 4

| Paper tape 123.4567P | Upper Octal | Octal | Decimal | Hex | Character | | |
|---|---|---|---|---|---|---|---|
| . | 0000 | 000 | 0 | 00 | ctrl @ | NUL | |
| •. | 0004 | 001 | 1 | 01 | ctrl A | SOH | Start of Heading |
| •. | 0010 | 002 | 2 | 02 | ctrl B | STX | Start of Text |
| •• . | 0014 | 003 | 3 | 03 | ctrl C | ETX | End of Text |
| •. | 0020 | 004 | 4 | 04 | ctrl D | EOT | End of Xmit |
| •. | 0024 | 005 | 5 | 05 | ctrl E | ENQ | Enquiry |
| •. | 0030 | 006 | 6 | 06 | ctrl F | ACK | Acknowledge |
| •. | 0034 | 007 | 7 | 07 | ctrl G | BEL | Audible Signal |
| .• | 0040 | 010 | 8 | 08 | ctrl H | BS | Back Space |
| •.• | 0044 | 011 | 9 | 09 | ctrl I | HT | Horizontal Tab |
| •.• | 0050 | 012 | 10 | 0A | ctrl J | LF | Line Feed |
| •• .• | 0054 | 013 | 11 | 0B | ctrl K | VT | Vertical Tab |
| •.• | 0060 | 014 | 12 | 0C | ctrl L | FF | Form Feed |
| •.• | 0064 | 015 | 13 | 0D | ctrl M | CR | Carriage Return |
| •.• | 0070 | 016 | 14 | 0E | ctrl N | SO | Shift Out |
| •.• | 0074 | 017 | 15 | 0F | ctrl O | SI | Shift In |
| . • | 0100 | 020 | 16 | 10 | ctrl P | DLE | Data Line Escape |
| • . • | 0104 | 021 | 17 | 11 | ctrl Q | DC1 | X On |
| •. • | 0110 | 022 | 18 | 12 | ctrl R | DC2 | Aux On |
| •• . • | 0114 | 023 | 19 | 13 | ctrl S | DC3 | X Off |
| •. • | 0120 | 024 | 20 | 14 | ctrl T | DC4 | Aux Off |
| •. • | 0124 | 025 | 21 | 15 | ctrl U | NAK | Negative Acknowledge |
| •. • | 0130 | 026 | 22 | 16 | ctrl V | SYN | Synchronous File |
| •. • | 0134 | 027 | 23 | 17 | ctrl W | ETB | End of Xmit Block |
| .•• | 0140 | 030 | 24 | 18 | ctrl X | CAN | Cancel |
| •.•• | 0144 | 031 | 25 | 19 | ctrl Y | EM | End of Medium |
| •.•• | 0150 | 032 | 26 | 1A | ctrl Z | SUB | Substitute |
| •• .•• | 0154 | 033 | 27 | 1B | ctrl [ | ESC | Escape |
| •.•• | 0160 | 034 | 28 | 1C | ctrl \ | FS | File Separator |
| •.•• | 0164 | 035 | 29 | 1D | ctrl ] | GS | Group Separator |
| •.•• | 0170 | 036 | 30 | 1E | ctrl ^ | RS | Record Separator |
| •.•• | 0174 | 037 | 31 | 1F | ctrl _ | US | Unit Separator |
| . • | 0200 | 040 | 32 | 20 | Space | | |
| • . • | 0204 | 041 | 33 | 21 | ! | | |
| •. • | 0210 | 042 | 34 | 22 | " | | |
| •• . • | 0214 | 043 | 35 | 23 | # | | |
| •. • | 0220 | 044 | 36 | 24 | $ | | |
| •. • | 0224 | 045 | 37 | 25 | % | | |
| •. • | 0230 | 046 | 38 | 26 | & | | |
| •. • | 0234 | 047 | 39 | 27 | ' | | |
| .•• | 0240 | 050 | 40 | 28 | ( | | |
| •.•• | 0244 | 051 | 41 | 29 | ) | | |
| •.•• | 0250 | 052 | 42 | 2A | * | | |
| •• .•• | 0254 | 053 | 43 | 2B | + | | |
| •.•• | 0260 | 054 | 44 | 2C | , | | |
| •.•• | 0264 | 055 | 45 | 2D | - | | |
| •.•• | 0270 | 056 | 46 | 2E | . | | |
| •.•• | 0274 | 057 | 47 | 2F | / | | |
| .•• | 0300 | 060 | 48 | 30 | 0 | | |
| •. •• | 0304 | 061 | 49 | 31 | 1 | | |
| •. •• | 0310 | 062 | 50 | 32 | 2 | | |
| •• . •• | 0314 | 063 | 51 | 33 | 3 | | |
| •. •• | 0320 | 064 | 52 | 34 | 4 | | |
| •. •• | 0324 | 065 | 53 | 35 | 5 | | |
| •. •• | 0330 | 066 | 54 | 36 | 6 | | |
| •. •• | 0334 | 067 | 55 | 37 | 7 | | |
| .••• | 0340 | 070 | 56 | 38 | 8 | | |
| •.••• | 0344 | 071 | 57 | 39 | 9 | | |
| •.••• | 0350 | 072 | 58 | 3A | : | | |
| •• .••• | 0354 | 073 | 59 | 3B | ; | | |
| •.••• | 0360 | 074 | 60 | 3C | < | | |
| •.••• | 0364 | 075 | 61 | 3D | = | | |
| •.••• | 0370 | 076 | 62 | 3E | > | | |
| •.••• | 0374 | 077 | 63 | 3F | ? | | |

## TABLE OF ASCII CODES *(Cont'd)* (Zero Parity)

| Paper tape 123.4567P | Upper Octal | Octal | Decimal | Hex | Character | |
|---|---|---|---|---|---|---|
| . ● | 0400 | 100 | 64 | 40 | @ | |
| ● . ● | 0404 | 101 | 65 | 41 | A | |
| ● . ● | 0410 | 102 | 66 | 42 | B | |
| ●● . ● | 0414 | 103 | 67 | 43 | C | |
| ●. ● | 0420 | 104 | 68 | 44 | D | |
| ● ●. ● | 0424 | 105 | 69 | 45 | E | |
| ●●. ● | 0430 | 106 | 70 | 46 | F | |
| ●●●. ● | 0434 | 107 | 71 | 47 | G | |
| .● ● | 0440 | 110 | 72 | 48 | H | |
| ● .● ● | 0444 | 111 | 73 | 49 | I | |
| ● .● ● | 0450 | 112 | 74 | 4A | J | |
| ●● .● ● | 0454 | 113 | 75 | 4B | K | |
| ●.● ● | 0460 | 114 | 76 | 4C | L | |
| ● ●.● ● | 0464 | 115 | 77 | 4D | M | |
| ●●.● ● | 0470 | 116 | 78 | 4E | N | |
| ●●●.● ● | 0474 | 117 | 79 | 4F | O | |
| . ●● | 0500 | 120 | 80 | 50 | P | |
| ● . ●● | 0504 | 121 | 81 | 51 | Q | |
| ● . ●● | 0510 | 122 | 82 | 52 | R | |
| ●● . ●● | 0514 | 123 | 83 | 53 | S | |
| ●. ●● | 0520 | 124 | 84 | 54 | T | |
| ● ●. ●● | 0524 | 125 | 85 | 55 | U | |
| ●●. ●● | 0530 | 126 | 86 | 56 | V | |
| ●●●. ●● | 0534 | 127 | 87 | 57 | W | |
| .●● ● | 0540 | 130 | 88 | 58 | X | |
| ● .●● ● | 0544 | 131 | 89 | 59 | Y | |
| ● .●● ● | 0550 | 132 | 90 | 5A | Z | |
| ●● .●● ● | 0554 | 133 | 91 | 5B | [ | |
| ●.●● ● | 0560 | 134 | 92 | 5C | \ | |
| ● ●.●● ● | 0564 | 135 | 93 | 5D | ] | |
| ●●.●● ● | 0570 | 136 | 94 | 5E | ^ | |
| ●●●.●● ● | 0574 | 137 | 95 | 5F | _ | |
| . ●● | 0600 | 140 | 96 | 60 | ` | |
| ● . ●● | 0604 | 141 | 97 | 61 | a | |
| ● . ●● | 0610 | 142 | 98 | 62 | b | |
| ●● . ●● | 0614 | 143 | 99 | 63 | c | |
| ●. ●● | 0620 | 144 | 100 | 64 | d | |
| ● ●. ●● | 0624 | 145 | 101 | 65 | e | |
| ●●. ●● | 0630 | 146 | 102 | 66 | f | |
| ●●●. ●● | 0634 | 147 | 103 | 67 | g | |
| .● ●● | 0640 | 150 | 104 | 68 | h | |
| ● .● ●● | 0644 | 151 | 105 | 69 | i | |
| ● .● ●● | 0650 | 152 | 106 | 6A | j | |
| ●● .● ●● | 0654 | 153 | 107 | 6B | k | |
| ●.● ●● | 0660 | 154 | 108 | 6C | l | |
| ● ●.● ●● | 0664 | 155 | 109 | 6D | m | |
| ●●.● ●● | 0670 | 156 | 110 | 6E | n | |
| ●●●.● ●● | 0674 | 157 | 111 | 6F | o | |
| . ●●● | 0700 | 160 | 112 | 70 | p | |
| ● . ●●● | 0704 | 161 | 113 | 71 | q | |
| ● . ●●● | 0710 | 162 | 114 | 72 | r | |
| ●● . ●●● | 0714 | 163 | 115 | 73 | s | |
| ●. ●●● | 0720 | 164 | 116 | 74 | t | |
| ● ●. ●●● | 0724 | 165 | 117 | 75 | u | |
| ●●. ●●● | 0730 | 166 | 118 | 76 | v | |
| ●●●. ●●● | 0734 | 167 | 119 | 77 | w | |
| .●●●● | 0740 | 170 | 120 | 78 | x | |
| ● .●●●● | 0744 | 171 | 121 | 79 | y | |
| ● .●●●● | 0750 | 172 | 122 | 7A | z | |
| ●● .●●●● | 0754 | 173 | 123 | 7B | { | |
| ●.●●●● | 0760 | 174 | 124 | 7C | \| | |
| ● ●.●●●● | 0764 | 175 | 125 | 7D | } | |
| ●●.●●●● | 0770 | 176 | 126 | 7E | ~ | Prefix |
| ●●●.●●●● | 0774 | 177 | 127 | 7F | DEL | Rubout |

APPENDIX 5

# HEXADECIMAL-DECIMAL INTEGER CONVERSION TABLE

The table appearing on the following pages provides a means for direct conversion of decimal integers in the range of 0 to 4095 and for hexadecimal integers in the range of 0 to FFF.

To convert numbers above those ranges, add table values to the figures below:

| Hexadecimal | Decimal | Hexadecimal | Decimal |
|---|---|---|---|
| 01 000 | 4 096 | 20 000 | 131 072 |
| 02 000 | 8 192 | 30 000 | 196 608 |
| 03 000 | 12 288 | 40 000 | 262 144 |
| 04 000 | 16 384 | 50 000 | 327 680 |
| 05 000 | 20 480 | 60 000 | 393 216 |
| 06 000 | 24 576 | 70 000 | 458 752 |
| 07 000 | 28 672 | 80 000 | 524 288 |
| 08 000 | 32 768 | 90 000 | 589 824 |
| 09 000 | 36 864 | A0 000 | 655 360 |
| 0A 000 | 40 960 | B0 000 | 720 896 |
| 0B 000 | 45 056 | C0 000 | 786 432 |
| 0C 000 | 49 152 | D0 000 | 851 968 |
| 0D 000 | 53 248 | E0 000 | 917 504 |
| 0E 000 | 57 344 | F0 000 | 983 040 |
| 0F 000 | 61 440 | 100 000 | 1 048 576 |
| 10 000 | 65 536 | 200 000 | 2 097 152 |
| 11 000 | 69 632 | 300 000 | 3 145 728 |
| 12 000 | 73 728 | 400 000 | 4 194 304 |
| 13 000 | 77 824 | 500 000 | 5 242 880 |
| 14 000 | 81 920 | 600 000 | 6 291 456 |
| 15 000 | 86 016 | 700 000 | 7 340 032 |
| 16 000 | 90 112 | 800 000 | 8 388 608 |
| 17 000 | 94 208 | 900 000 | 9 437 184 |
| 18 000 | 98 304 | A00 000 | 10 485 760 |
| 19 000 | 102 400 | B00 000 | 11 534 336 |
| 1A 000 | 106 496 | C00 000 | 12 582 912 |
| 1B 000 | 110 592 | D00 000 | 13 631 488 |
| 1C 000 | 114 688 | E00 000 | 14 680 064 |
| 1D 000 | 118 784 | F00 000 | 15 728 640 |
| 1E 000 | 122 880 | 1 000 000 | 16 777 216 |
| 1F 000 | 126 976 | 2 000 000 | 33 554 432 |

HEXADECIMAL - DECIMAL INTEGER CONVERSION TABLE (Continued)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 |
| 010 | 0016 | 0017 | 0018 | 0019 | 0020 | 0021 | 0022 | 0023 | 0024 | 0025 | 0026 | 0027 | 0028 | 0029 | 0030 | 0031 |
| 020 | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 | 0039 | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 |
| 030 | 0048 | 0049 | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 | 0056 | 0057 | 0058 | 0059 | 0060 | 0061 | 0062 | 0063 |
| 040 | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 0070 | 0071 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 |
| 050 | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 | 0088 | 0089 | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 |
| 060 | 0096 | 0097 | 0098 | 0099 | 0100 | 0101 | 0102 | 0103 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 | 0110 | 0111 |
| 070 | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 |
| 080 | 0128 | 0129 | 0130 | 0131 | 0132 | 0133 | 0134 | 0135 | 0136 | 0137 | 0138 | 0139 | 0140 | 0141 | 0142 | 0143 |
| 090 | 0144 | 0145 | 0146 | 0147 | 0148 | 0149 | 0150 | 0151 | 0152 | 0153 | 0154 | 0155 | 0156 | 0157 | 0158 | 0159 |
| 0A0 | 0160 | 0161 | 0162 | 0163 | 0164 | 0165 | 0166 | 0167 | 0168 | 0169 | 0170 | 0171 | 0172 | 0173 | 0174 | 0175 |
| 0B0 | 0176 | 0177 | 0178 | 0179 | 0180 | 0181 | 0182 | 0183 | 0184 | 0185 | 0186 | 0187 | 0188 | 0189 | 0190 | 0191 |
| 0C0 | 0192 | 0193 | 0194 | 0195 | 0196 | 0197 | 0198 | 0199 | 0200 | 0201 | 0202 | 0203 | 0204 | 0205 | 0206 | 0207 |
| 0D0 | 0208 | 0209 | 0210 | 0211 | 0212 | 0213 | 0214 | 0215 | 0216 | 0217 | 0218 | 0219 | 0220 | 0221 | 0222 | 0223 |
| 0E0 | 0224 | 0225 | 0226 | 0227 | 0228 | 0229 | 0230 | 0231 | 0232 | 0233 | 0234 | 0235 | 0236 | 0237 | 0238 | 0239 |
| 0F0 | 0240 | 0241 | 0242 | 0243 | 0244 | 0245 | 0246 | 0247 | 0248 | 0249 | 0250 | 0251 | 0252 | 0253 | 0254 | 0255 |
| 100 | 0256 | 0257 | 0258 | 0259 | 0260 | 0261 | 0262 | 0263 | 0264 | 0265 | 0266 | 0267 | 0268 | 0269 | 0270 | 0271 |
| 110 | 0272 | 0273 | 0274 | 0275 | 0276 | 0277 | 0278 | 0279 | 0280 | 0281 | 0282 | 0283 | 0284 | 0285 | 0286 | 0287 |
| 120 | 0288 | 0289 | 0290 | 0291 | 0292 | 0293 | 0294 | 0295 | 0296 | 0297 | 0298 | 0299 | 0300 | 0301 | 0302 | 0303 |
| 130 | 0304 | 0305 | 0306 | 0307 | 0308 | 0309 | 0310 | 0311 | 0312 | 0313 | 0314 | 0315 | 0316 | 0317 | 0318 | 0319 |
| 140 | 0320 | 0321 | 0322 | 0323 | 0324 | 0325 | 0326 | 0327 | 0328 | 0329 | 0330 | 0331 | 0332 | 0333 | 0334 | 0335 |
| 150 | 0336 | 0337 | 0338 | 0339 | 0340 | 0341 | 0342 | 0343 | 0344 | 0345 | 0346 | 0347 | 0348 | 0349 | 0350 | 0351 |
| 160 | 0352 | 0353 | 0354 | 0355 | 0356 | 0357 | 0358 | 0359 | 0360 | 0361 | 0362 | 0363 | 0364 | 0365 | 0366 | 0367 |
| 170 | 0368 | 0369 | 0370 | 0371 | 0372 | 0373 | 0374 | 0375 | 0376 | 0377 | 0378 | 0379 | 0380 | 0381 | 0382 | 0383 |
| 180 | 0384 | 0385 | 0386 | 0387 | 0388 | 0389 | 0390 | 0391 | 0392 | 0393 | 0394 | 0395 | 0396 | 0397 | 0398 | 0399 |
| 190 | 0400 | 0401 | 0402 | 0403 | 0404 | 0405 | 0406 | 0407 | 0408 | 0409 | 0410 | 0411 | 0412 | 0413 | 0414 | 0415 |
| 1A0 | 0416 | 0417 | 0418 | 0419 | 0420 | 0421 | 0422 | 0423 | 0424 | 0425 | 0426 | 0427 | 0428 | 0429 | 0430 | 0431 |
| 1B0 | 0432 | 0433 | 0434 | 0435 | 0436 | 0437 | 0438 | 0439 | 0440 | 0441 | 0442 | 0443 | 0444 | 0445 | 0446 | 0447 |
| 1C0 | 0448 | 0449 | 0450 | 0451 | 0452 | 0453 | 0454 | 0455 | 0456 | 0457 | 0458 | 0459 | 0460 | 0461 | 0462 | 0463 |
| 1D0 | 0464 | 0465 | 0466 | 0467 | 0468 | 0469 | 0470 | 0471 | 0472 | 0473 | 0474 | 0475 | 0476 | 0477 | 0478 | 0479 |
| 1E0 | 0480 | 0481 | 0482 | 0483 | 0484 | 0485 | 0486 | 0487 | 0488 | 0489 | 0490 | 0491 | 0492 | 0493 | 0494 | 0495 |
| 1F0 | 0496 | 0497 | 0498 | 0499 | 0500 | 0501 | 0502 | 0503 | 0504 | 0505 | 0506 | 0507 | 0508 | 0509 | 0510 | 0511 |
| 200 | 0512 | 0513 | 0514 | 0515 | 0516 | 0517 | 0518 | 0519 | 0529 | 0521 | 0522 | 0523 | 0524 | 0525 | 0526 | 0527 |
| 210 | 0528 | 0529 | 0530 | 0531 | 0532 | 0533 | 0534 | 0535 | 0536 | 0537 | 0538 | 0539 | 0540 | 0541 | 0542 | 0543 |
| 220 | 0544 | 0545 | 0546 | 0547 | 0548 | 0549 | 0550 | 0551 | 0552 | 0553 | 0554 | 0555 | 0556 | 0557 | 0558 | 0559 |
| 230 | 0560 | 0561 | 0562 | 0563 | 0564 | 0565 | 0566 | 0567 | 0568 | 0569 | 0570 | 0571 | 0572 | 0573 | 0574 | 0575 |
| 240 | 0576 | 0577 | 0578 | 0579 | 0580 | 0581 | 0582 | 0583 | 0584 | 0585 | 0586 | 0587 | 0588 | 0589 | 0590 | 0591 |
| 250 | 0592 | 0593 | 0594 | 0595 | 0596 | 0597 | 0598 | 0599 | 0600 | 0601 | 0602 | 0603 | 0604 | 0605 | 0606 | 0607 |
| 260 | 0608 | 0609 | 0610 | 0611 | 0612 | 0613 | 0614 | 0615 | 0616 | 0617 | 0618 | 0619 | 0620 | 0621 | 0622 | 0623 |
| 270 | 0624 | 0625 | 0626 | 0627 | 0628 | 0629 | 0630 | 0631 | 0632 | 0633 | 0634 | 0635 | 0636 | 0637 | 0638 | 0639 |
| 280 | 0640 | 0641 | 0642 | 0643 | 0644 | 0645 | 0646 | 0647 | 0648 | 0649 | 0650 | 0651 | 0652 | 0653 | 0654 | 0655 |
| 290 | 0656 | 0657 | 0658 | 0659 | 0660 | 0661 | 0662 | 0663 | 0664 | 0665 | 0666 | 0667 | 0668 | 0669 | 0670 | 0671 |
| 2A0 | 0672 | 0673 | 0674 | 0675 | 0676 | 0677 | 0678 | 0679 | 0680 | 0681 | 0682 | 0683 | 0684 | 0685 | 0686 | 0687 |
| 2B0 | 0688 | 0689 | 0690 | 0691 | 0692 | 0693 | 0694 | 0695 | 0696 | 0697 | 0698 | 0699 | 0700 | 0701 | 0702 | 0703 |
| 2C0 | 0704 | 0705 | 0706 | 0707 | 0708 | 0709 | 0710 | 0711 | 0712 | 0713 | 0714 | 0715 | 0716 | 0717 | 0718 | 0719 |
| 2D0 | 0720 | 0721 | 0722 | 0723 | 0724 | 0725 | 0726 | 0727 | 0728 | 0729 | 0730 | 0731 | 0732 | 0733 | 0734 | 0735 |
| 2E0 | 0736 | 0737 | 0738 | 0739 | 0740 | 0741 | 0742 | 0743 | 0744 | 0745 | 0746 | 0747 | 0748 | 0749 | 0750 | 0751 |
| 2F0 | 0752 | 0753 | 0754 | 0755 | 0756 | 0757 | 0758 | 0759 | 0760 | 0761 | 0762 | 0763 | 0764 | 0765 | 0766 | 0767 |

HEXADECIMAL - DECIMAL INTEGER CONVERSION TABLE (Continued)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 300 | 0768 | 0769 | 0770 | 0771 | 0772 | 0773 | 0774 | 0775 | 0776 | 0777 | 0778 | 0779 | 0780 | 0781 | 0782 | 0783 |
| 310 | 0784 | 0785 | 0786 | 0787 | 0788 | 0789 | 0790 | 0791 | 0792 | 0793 | 0794 | 0795 | 0796 | 0797 | 0798 | 0799 |
| 320 | 0800 | 0801 | 0802 | 0803 | 0804 | 0805 | 0806 | 0807 | 0808 | 0809 | 0810 | 0811 | 0812 | 0813 | 0814 | 0815 |
| 330 | 0816 | 0817 | 0818 | 0819 | 0820 | 0821 | 0822 | 0823 | 0824 | 0825 | 0826 | 0827 | 0828 | 0829 | 0830 | 0831 |
| 340 | 0832 | 0833 | 0834 | 0835 | 0836 | 0837 | 0838 | 0839 | 0840 | 0841 | 0842 | 0843 | 0844 | 0845 | 0846 | 0847 |
| 350 | 0848 | 0849 | 0850 | 0851 | 0852 | 0853 | 0854 | 0855 | 0856 | 0857 | 0858 | 0859 | 0860 | 0861 | 0862 | 0863 |
| 360 | 0864 | 0865 | 0866 | 0867 | 0868 | 0869 | 0870 | 0871 | 0872 | 0873 | 0874 | 0875 | 0876 | 0877 | 0878 | 0879 |
| 370 | 0880 | 0881 | 0882 | 0883 | 0884 | 0885 | 0886 | 0887 | 0888 | 0889 | 0890 | 0891 | 0892 | 0893 | 0894 | 0895 |
| 380 | 0896 | 0897 | 0898 | 0899 | 0900 | 0901 | 0902 | 0903 | 0904 | 0905 | 0906 | 0907 | 0908 | 0909 | 0910 | 0911 |
| 390 | 0912 | 0913 | 0914 | 0915 | 0916 | 0917 | 0918 | 0919 | 0920 | 0921 | 0922 | 0923 | 0924 | 0925 | 0926 | 0927 |
| 3A0 | 0928 | 0929 | 0930 | 0931 | 0932 | 0933 | 0934 | 0935 | 0936 | 0937 | 0938 | 0939 | 0940 | 0941 | 0942 | 0943 |
| 3B0 | 0944 | 0945 | 0946 | 0947 | 0948 | 0949 | 0950 | 0951 | 0952 | 0953 | 0954 | 0955 | 0956 | 0957 | 0958 | 0959 |
| 3C0 | 0960 | 0961 | 0962 | 0963 | 0964 | 0965 | 0966 | 0967 | 0968 | 0969 | 0970 | 0971 | 0972 | 0973 | 0974 | 0975 |
| 3D0 | 0976 | 0977 | 0978 | 0979 | 0980 | 0981 | 0982 | 0983 | 0984 | 0985 | 0986 | 0987 | 0988 | 0989 | 0990 | 0991 |
| 3E0 | 0992 | 0993 | 0994 | 0995 | 0996 | 0997 | 0998 | 0999 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 |
| 3F0 | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 |
| 400 | 1024 | 1025 | 0126 | 0127 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 |
| 410 | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 |
| 420 | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 |
| 430 | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 |
| 440 | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 |
| 450 | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 |
| 460 | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 |
| 470 | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 |
| 480 | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 |
| 490 | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 |
| 4A0 | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 |
| 4B0 | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 |
| 4C0 | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 |
| 4D0 | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 |
| 4E0 | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 |
| 4F0 | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 |
| 500 | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 | 1288 | 1289 | 1290 | 1291 | 1291 | 1293 | 1294 | 1295 |
| 510 | 1296 | 1297 | 1298 | 1299 | 1399 | 1301 | 1302 | 1303 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 |
| 520 | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 | 1329 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 |
| 530 | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 |
| 540 | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 | 1352 | 1353 | 1354 | 1355 | 1356 | 1367 | 1358 | 1359 |
| 550 | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 |
| 560 | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 |
| 570 | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 |
| 580 | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 | 1416 | 1417 | 1418 | 1419 | 1429 | 1421 | 1422 | 1423 |
| 590 | 1324 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 |
| 5A0 | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 |
| 3B0 | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 | 1463 | 1464 | 1465 | 1466 | 1467 | 1468 | 1469 | 1470 | 1471 |
| 5C0 | 1472 | 1473 | 1474 | 1475 | 1476 | 1477 | 1478 | 1479 | 1480 | 1481 | 1482 | 1483 | 1484 | 1485 | 1486 | 1487 |
| 5D0 | 1488 | 1489 | 1490 | 1491 | 1492 | 1493 | 1494 | 1495 | 1496 | 1497 | 1498 | 1499 | 1500 | 1501 | 1502 | 1503 |
| 5E0 | 1504 | 1505 | 1506 | 1507 | 1508 | 1509 | 1510 | 1511 | 1512 | 1513 | 1514 | 1515 | 1516 | 1517 | 1518 | 1519 |
| 5F0 | 1520 | 1521 | 1522 | 1523 | 1524 | 1515 | 1526 | 1527 | 1528 | 1529 | 1530 | 1531 | 1532 | 1533 | 1534 | 1535 |

HEXADECIMAL - DECIMAL INTEGER CONVERSION TABLE (Continued)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 600 | 1536 | 1537 | 1538 | 1539 | 1540 | 1541 | 1542 | 1543 | 1544 | 1545 | 1546 | 1547 | 1548 | 1549 | 1550 | 1551 |
| 610 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 | 1559 | 1560 | 1561 | 1562 | 1563 | 1564 | 1565 | 1566 | 1567 |
| 620 | 1568 | 1569 | 1570 | 1571 | 1572 | 1573 | 1574 | 1575 | 1576 | 1577 | 1578 | 1579 | 1580 | 1581 | 1582 | 1583 |
| 630 | 1584 | 1585 | 1586 | 1587 | 1588 | 1589 | 1590 | 1591 | 1592 | 1592 | 1594 | 1595 | 1596 | 1597 | 1598 | 1599 |
| 640 | 1600 | 1601 | 1602 | 1603 | 1604 | 1605 | 1606 | 1607 | 1608 | 1609 | 1610 | 1611 | 1612 | 1613 | 1614 | 1615 |
| 650 | 1616 | 1617 | 1618 | 1619 | 1620 | 1621 | 1622 | 1623 | 1624 | 1625 | 1626 | 1627 | 1628 | 1629 | 1630 | 1631 |
| 660 | 1632 | 1633 | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 | 1640 | 1641 | 1642 | 1643 | 1644 | 1645 | 1646 | 1647 |
| 670 | 1648 | 1649 | 1650 | 1651 | 1652 | 1653 | 1654 | 1655 | 1656 | 1657 | 1658 | 1659 | 1660 | 1661 | 1662 | 1663 |
| 680 | 1664 | 1665 | 1666 | 1667 | 1668 | 1669 | 1670 | 1671 | 1672 | 1673 | 1674 | 1675 | 1676 | 1677 | 1678 | 1679 |
| 690 | 1680 | 1681 | 1682 | 1683 | 1684 | 1685 | 1686 | 1687 | 1688 | 1689 | 1690 | 1691 | 1692 | 1693 | 1694 | 1695 |
| 6A0 | 1696 | 1697 | 1698 | 1699 | 1700 | 1701 | 1702 | 1703 | 1704 | 1705 | 1706 | 1707 | 1708 | 1709 | 1710 | 1711 |
| 6B0 | 1712 | 1713 | 1714 | 1715 | 1716 | 1717 | 1718 | 1719 | 1720 | 1721 | 1722 | 17231 | 1724 | 1725 | 1726 | 1727 |
| 6C0 | 1728 | 1729 | 1730 | 1731 | 1732 | 1733 | 1734 | 1735 | 1736 | 1737 | 1738 | 1739 | 1740 | 1741 | 1742 | 1743 |
| 6D0 | 1744 | 1745 | 1746 | 1747 | 1748 | 1749 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 | 1759 |
| 6E0 | 1760 | 1761 | 1762 | 1763 | 1764 | 1765 | 1766 | 1767 | 1768 | 1769 | 1770 | 1771 | 1772 | 1773 | 1774 | 1775 |
| 6F0 | 1776 | 1777 | 1778 | 1779 | 1780 | 1781 | 1782 | 1783 | 1784 | 1785 | 1786 | 1787 | 1788 | 1789 | 1790 | 1791 |
| 700 | 1792 | 1793 | 1794 | 1795 | 1796 | 1797 | 1798 | 1799 | 1800 | 1801 | 8102 | 1803 | 1804 | 1805 | 1806 | 1807 |
| 710 | 1808 | 1809 | 1810 | 1811 | 1812 | 1813 | 1814 | 1815 | 1816 | 1817 | 1818 | 1819 | 1820 | 1821 | 1822 | 1823 |
| 720 | 1824 | 1825 | 1826 | 1827 | 1818 | 1829 | 1830 | 1831 | 1832 | 1833 | 1834 | 1835 | 1836 | 1837 | 1838 | 1839 |
| 730 | 1840 | 1841 | 1842 | 1843 | 1844 | 1845 | 1846 | 1847 | 1848 | 1849 | 1850 | 1851 | 1852 | 1853 | 1854 | 1855 |
| 740 | 1856 | 1857 | 1858 | 1859 | 1860 | 1861 | 1862 | 1863 | 1864 | 1865 | 1866 | 1867 | 1868 | 1869 | 1870 | 1871 |
| 750 | 1872 | 1873 | 1874 | 1875 | 1876 | 1877 | 1878 | 1879 | 1880 | 1881 | 1882 | 1883 | 1884 | 1885 | 1886 | 1887 |
| 760 | 1888 | 1889 | 1890 | 1891 | 1892 | 1893 | 1894 | 1895 | 1896 | 1897 | 1898 | 1899 | 1900 | 1909 | 1902 | 1903 |
| 770 | 1904 | 1905 | 1906 | 1907 | 1908 | 1909 | 1910 | 1911 | 1912 | 1913 | 1914 | 1915 | 1916 | 1917 | 1918 | 1919 |
| 780 | 1920 | 1921 | 1922 | 1923 | 1924 | 1925 | 1926 | 1927 | 1928 | 1929 | 1930 | 1931 | 1932 | 1933 | 1934 | 1935 |
| 790 | 1936 | 1937 | 1938 | 1939 | 1940 | 1941 | 1942 | 1943 | 1944 | 1945 | 1946 | 1947 | 1948 | 1949 | 1950 | 1951 |
| 7A0 | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 | 1966 | 1967 |
| 7B0 | 1968 | 1969 | 1970 | 1971 | 1972 | 1973 | 1974 | 1975 | 1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 |
| 7C0 | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 |
| 7D0 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
| 7E0 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 |
| 7F0 | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 |
| 800 | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 |
| 810 | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 |
| 820 | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 |
| 830 | 2096 | 2097 | 2098 | 2099 | 2100 | 2101 | 2102 | 2103 | 2104 | 2105 | 2106 | 2107 | 2108 | 2109 | 2110 | 2111 |
| 840 | 2112 | 2113 | 2114 | 2115 | 2116 | 2117 | 2118 | 2119 | 2120 | 2121 | 2122 | 2123 | 2124 | 2125 | 2126 | 2127 |
| 850 | 2128 | 2129 | 2130 | 2131 | 2132 | 2133 | 2134 | 2135 | 2136 | 2137 | 2138 | 2139 | 2140 | 2141 | 2142 | 2143 |
| 860 | 2144 | 2145 | 2146 | 2147 | 2148 | 2149 | 2150 | 2151 | 2152 | 2153 | 2154 | 2155 | 2156 | 2157 | 2158 | 2159 |
| 870 | 2160 | 2161 | 2162 | 2163 | 2164 | 2165 | 2166 | 2167 | 2168 | 2169 | 2170 | 2171 | 2172 | 2173 | 2174 | 2175 |
| 880 | 2176 | 2177 | 2178 | 2179 | 2180 | 2181 | 2182 | 2183 | 2184 | 2185 | 2186 | 2187 | 2188 | 2189 | 2190 | 2191 |
| 890 | 2192 | 2193 | 2194 | 2195 | 2196 | 2197 | 2198 | 2199 | 2200 | 2201 | 2202 | 2203 | 2204 | 2205 | 2206 | 2207 |
| 8A0 | 2208 | 2209 | 2210 | 2211 | 2212 | 2213 | 2214 | 2215 | 2216 | 2217 | 2218 | 2219 | 2220 | 2221 | 2222 | 2223 |
| 8B0 | 2224 | 2225 | 2226 | 2227 | 2228 | 2229 | 2230 | 2231 | 2232 | 2233 | 2234 | 2235 | 2236 | 2237 | 2238 | 2239 |
| 8C0 | 2240 | 2241 | 2242 | 2243 | 2244 | 2245 | 2246 | 2247 | 2248 | 2249 | 2250 | 2251 | 2252 | 2253 | 2254 | 2255 |
| 8D0 | 2256 | 2257 | 2258 | 2259 | 2260 | 2261 | 2262 | 2263 | 2264 | 2265 | 2266 | 2267 | 2268 | 2269 | 2270 | 2271 |
| 8E0 | 2272 | 2273 | 2274 | 2275 | 2276 | 2277 | 2278 | 2279 | 2280 | 2281 | 2282 | 2283 | 2284 | 2285 | 2286 | 2287 |
| 8F0 | 2288 | 2289 | 2290 | 2291 | 2292 | 2293 | 2294 | 2295 | 2296 | 2297 | 2298 | 2299 | 2300 | 2301 | 2302 | 2303 |

HEXADECIMAL - DECIMAL INTEGER CONVERSION TABLE (Continued)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 900 | 2304 | 2305 | 2306 | 2307 | 2308 | 2309 | 2310 | 2311 | 2312 | 2313 | 2314 | 2315 | 2316 | 2317 | 2318 | 2319 |
| 910 | 2320 | 2321 | 2322 | 2323 | 2324 | 2325 | 2326 | 2327 | 2328 | 2329 | 2330 | 2331 | 2332 | 2333 | 2334 | 2335 |
| 920 | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2342 | 2343 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 | 2351 |
| 930 | 2352 | 2353 | 2354 | 2355 | 2356 | 2357 | 2358 | 2359 | 2360 | 2361 | 2362 | 2363 | 2364 | 2365 | 2366 | 2367 |
| 940 | 2368 | 2369 | 2370 | 2371 | 2372 | 2373 | 2374 | 2375 | 2376 | 2377 | 2378 | 2379 | 2380 | 2381 | 2382 | 2383 |
| 950 | 2384 | 2385 | 2386 | 2387 | 2388 | 2389 | 2390 | 2391 | 2392 | 2393 | 2394 | 2395 | 3496 | 2397 | 2398 | 2399 |
| 960 | 2400 | 2401 | 2402 | 2403 | 2404 | 2405 | 2406 | 2407 | 2408 | 2409 | 2410 | 2411 | 2412 | 2413 | 2414 | 2415 |
| 970 | 2416 | 2417 | 2418 | 2419 | 2420 | 2421 | 2422 | 2423 | 2424 | 2425 | 2426 | 2427 | 2428 | 2429 | 2430 | 2431 |
| 980 | 2432 | 2433 | 2434 | 24351 | 2436 | 2437 | 2438 | 2439 | 2440 | 2441 | 2442 | 2443 | 2444 | 2445 | 2446 | 2447 |
| 990 | 2448 | 2449 | 2450 | 2451 | 2452 | 2453 | 2454 | 2455 | 2456 | 2457 | 2458 | 2459 | 2460 | 2461 | 2462 | 2463 |
| 9A0 | 2464 | 2465 | 2466 | 2467 | 2468 | 2469 | 2479 | 2471 | 2472 | 2473 | 2474 | 2475 | 2476 | 2477 | 2478 | 2479 |
| 9B0 | 2480 | 2481 | 2482 | 2483 | 2484 | 2485 | 2486 | 2487 | 2488 | 2489 | 2490 | 2491 | 2492 | 2493 | 2494 | 2495 |
| 9C0 | 2496 | 2497 | 2498 | 2499 | 2500 | 2501 | 2502 | 2503 | 2504 | 2505 | 2506 | 2507 | 2508 | 2509 | 2510 | 2511 |
| 9D0 | 2512 | 2513 | 2514 | 2515 | 2516 | 2517 | 2518 | 2519 | 2520 | 2521 | 2522 | 2523 | 2524 | 2525 | 2526 | 2527 |
| 9E0 | 2528 | 2529 | 2530 | 2531 | 2532 | 2533 | 2534 | 2535 | 2536 | 2537 | 2538 | 2539 | 2540 | 2541 | 2542 | 2543 |
| 9F0 | 2544 | 2545 | 2546 | 2547 | 2548 | 2549 | 2550 | 2551 | 2552 | 2553 | 2554 | 2555 | 2556 | 2557 | 2558 | 2559 |
| A00 | 2560 | 2561 | 2562 | 2563 | 2564 | 2565 | 2566 | 2567 | 2568 | 2569 | 2570 | 2571 | 2572 | 2573 | 2574 | 2575 |
| A10 | 2576 | 2577 | 2578 | 2579 | 2580 | 2581 | 2582 | 2583 | 2584 | 2585 | 2586 | 2587 | 2588 | 2589 | 2590 | 2591 |
| A20 | 2592 | 2593 | 2594 | 2595 | 2596 | 2597 | 2598 | 2599 | 2600 | 2601 | 2602 | 2603 | 2604 | 2605 | 2606 | 2607 |
| A30 | 2608 | 2609 | 2610 | 2611 | 2612 | 2613 | 2614 | 2615 | 2626 | 2617 | 2618 | 2619 | 2620 | 2621 | 2622 | 2623 |
| A40 | 2624 | 2625 | 2626 | 2627 | 2628 | 2629 | 2630 | 2631 | 2632 | 2633 | 2634 | 2635 | 2636 | 2637 | 2638 | 2639 |
| A50 | 2640 | 2641 | 2642 | 2643 | 2644 | 2645 | 2646 | 2647 | 2648 | 2649 | 2650 | 2651 | 2652 | 2653 | 2654 | 2655 |
| A60 | 2656 | 2657 | 2658 | 2659 | 2660 | 2661 | 2662 | 2663 | 2664 | 2665 | 2666 | 2667 | 2668 | 2669 | 2670 | 2671 |
| A70 | 2672 | 2673 | 2674 | 2675 | 2676 | 2677 | 2678 | 2679 | 2680 | 2681 | 2682 | 2683 | 2684 | 2685 | 2686 | 2687 |
| A80 | 2688 | 2689 | 2690 | 2691 | 2692 | 2693 | 2694 | 2695 | 2696 | 2697 | 2698 | 2699 | 2700 | 2701 | 2702 | 2703 |
| A90 | 2704 | 2705 | 2706 | 2707 | 2708 | 2709 | 2710 | 2711 | 2712 | 2713 | 2714 | 2715 | 2716 | 2717 | 2718 | 2719 |
| AA0 | 2720 | 2721 | 2722 | 2723 | 2724 | 2725 | 2726 | 2727 | 2728 | 2729 | 2730 | 2731 | 2732 | 2733 | 2734 | 2735 |
| Ab0 | 2736 | 2737 | 2738 | 2739 | 2740 | 2741 | 2742 | 2743 | 2744 | 2745 | 2746 | 2747 | 2748 | 2749 | 2750 | 2751 |
| AC0 | 2752 | 2753 | 2754 | 2755 | 2756 | 2757 | 2758 | 2759 | 2760 | 2761 | 2762 | 2763 | 2764 | 2765 | 2766 | 2767 |
| AD0 | 2768 | 2769 | 2770 | 2771 | 2772 | 2773 | 2774 | 2775 | 2776 | 2777 | 2778 | 2779 | 2780 | 2781 | 2782 | 2783 |
| AE0 | 2784 | 2785 | 2786 | 2787 | 2788 | 2789 | 2790 | 2791 | 2792 | 2793 | 2794 | 2795 | 2796 | 2797 | 2798 | 2799 |
| AF0 | 2800 | 2801 | 2802 | 2803 | 2804 | 2805 | 2806 | 2807 | 2808 | 2809 | 2810 | 2811 | 2812 | 2813 | 2814 | 2815 |
| B00 | 2816 | 2817 | 2818 | 2819 | 2820 | 2821 | 2822 | 2823 | 2824 | 2825 | 2826 | 2827 | 2828 | 2829 | 2830 | 2831 |
| B10 | 2832 | 2833 | 2834 | 2835 | 2836 | 2837 | 2838 | 2839 | 2840 | 2841 | 2842 | 2843 | 2844 | 2845 | 2846 | 2847 |
| B20 | 2848 | 2849 | 2850 | 2851 | 2852 | 2853 | 2854 | 2855 | 2856 | 2857 | 2858 | 2859 | 2860 | 2861 | 2862 | 2863 |
| B30 | 2864 | 2865 | 2866 | 2867 | 2868 | 2869 | 2870 | 2871 | 2872 | 2873 | 2874 | 2875 | 2876 | 2877 | 2878 | 2879 |
| B40 | 2880 | 2881 | 2882 | 2883 | 2884 | 2885 | 2886 | 2887 | 2888 | 2889 | 2890 | 2891 | 2892 | 2893 | 2894 | 2895 |
| B50 | 2896 | 2897 | 2898 | 2899 | 2900 | 2901 | 2902 | 2903 | 2904 | 2905 | 2906 | 2907 | 2908 | 2909 | 2910 | 2911 |
| B60 | 2912 | 2913 | 2914 | 2915 | 2916 | 2917 | 2918 | 2919 | 2920 | 2921 | 2922 | 2923 | 2924 | 2925 | 2926 | 2927 |
| B70 | 2928 | 2929 | 2930 | 2931 | 2932 | 2933 | 2934 | 2935 | 2936 | 2937 | 2938 | 2939 | 2940 | 2941 | 2942 | 2943 |
| B80 | 2944 | 2945 | 2946 | 2947 | 2948 | 2949 | 2950 | 2951 | 2952 | 2953 | 2954 | 2955 | 2956 | 2957 | 2958 | 2959 |
| B90 | 2960 | 2961 | 2962 | 2963 | 2964 | 2965 | 2966 | 2967 | 2968 | 2969 | 2970 | 2971 | 2972 | 2973 | 2974 | 2975 |
| BA0 | 2976 | 2977 | 2978 | 2979 | 2980 | 2981 | 2982 | 2983 | 2984 | 2985 | 2986 | 2987 | 2988 | 2989 | 2990 | 2991 |
| BB0 | 2992 | 2993 | 2994 | 2995 | 2996 | 2997 | 2998 | 2999 | 3000 | 3001 | 3002 | 3003 | 3004 | 3005 | 3006 | 3007 |
| BC0 | 3008 | 3009 | 3010 | 3011 | 3012 | 3013 | 3014 | 3015 | 3016 | 3017 | 3018 | 3019 | 3020 | 3021 | 3022 | 3023 |
| BD0 | 3024 | 3025 | 3026 | 3027 | 3028 | 3029 | 3030 | 3031 | 3032 | 3033 | 3034 | 3035 | 3036 | 3037 | 3038 | 3039 |
| BE0 | 3040 | 3041 | 3042 | 3043 | 3044 | 3045 | 3046 | 3047 | 3048 | 3049 | 3050 | 3051 | 3052 | 3053 | 3054 | 3055 |
| BF0 | 3056 | 3057 | 3058 | 3059 | 3060 | 3061 | 3062 | 3063 | 3064 | 3065 | 3066 | 3067 | 3068 | 3069 | 3070 | 3071 |

HEXADECIMAL - DECIMAL INTEGER CONVERSION TABLE (Continued)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C00 | 3072 | 3073 | 3074 | 3075 | 3076 | 3077 | 3078 | 3079 | 3080 | 3081 | 3082 | 3083 | 3084 | 3085 | 3086 | 3087 |
| C10 | 3088 | 3089 | 3090 | 3091 | 3092 | 3093 | 3094 | 3095 | 3096 | 3097 | 3098 | 3099 | 3100 | 3101 | 3102 | 3103 |
| C20 | 3104 | 3105 | 3106 | 3107 | 3108 | 3109 | 3110 | 3111 | 3112 | 3113 | 3114 | 3115 | 3116 | 3117 | 3118 | 3119 |
| C30 | 3120 | 3121 | 3122 | 3123 | 3124 | 3125 | 3126 | 3127 | 3128 | 3129 | 3130 | 3131 | 3132 | 3133 | 3134 | 3135 |
| C40 | 3136 | 3137 | 3138 | 3139 | 3140 | 3141 | 3142 | 3143 | 3144 | 3145 | 3146 | 3147 | 3148 | 3149 | 3150 | 3151 |
| C50 | 3152 | 3153 | 3154 | 3155 | 3156 | 3157 | 3158 | 3159 | 3160 | 3161 | 3162 | 3163 | 3164 | 3165 | 3166 | 3167 |
| C60 | 3168 | 3169 | 3170 | 3171 | 3172 | 3173 | 3174 | 3175 | 3176 | 3177 | 3178 | 3179 | 3180 | 3181 | 3182 | 3183 |
| C70 | 3184 | 3185 | 3186 | 3187 | 3188 | 3189 | 3190 | 3191 | 3192 | 3193 | 3194 | 3195 | 3196 | 3197 | 3198 | 3199 |
| C80 | 3200 | 3201 | 3202 | 3203 | 3204 | 3205 | 3206 | 3207 | 3208 | 3209 | 3210 | 3211 | 3212 | 3213 | 3214 | 3215 |
| C90 | 3216 | 3217 | 3218 | 3219 | 3220 | 3221 | 3222 | 3223 | 3224 | 3225 | 3226 | 3227 | 3228 | 3229 | 3230 | 3231 |
| CA0 | 3232 | 3233 | 3234 | 3235 | 3236 | 3237 | 3238 | 3239 | 3240 | 3241 | 3242 | 3243 | 3244 | 3245 | 3246 | 3247 |
| CB0 | 3248 | 3249 | 3250 | 3251 | 3252 | 3253 | 3254 | 3255 | 3256 | 3257 | 3258 | 3259 | 3260 | 3261 | 3262 | 3263 |
| CC0 | 3264 | 3265 | 3266 | 3267 | 3268 | 3269 | 3270 | 3271 | 3272 | 3273 | 3274 | 3275 | 3276 | 3277 | 3278 | 3279 |
| CD0 | 3280 | 3281 | 3282 | 3283 | 3284 | 3285 | 3286 | 3287 | 3288 | 3289 | 3290 | 3291 | 3292 | 3293 | 3294 | 3295 |
| CE0 | 3296 | 3297 | 3298 | 3299 | 3300 | 3301 | 3302 | 3303 | 3304 | 3305 | 3306 | 3307 | 3308 | 3309 | 3310 | 3311 |
| CF0 | 3312 | 3313 | 3314 | 3315 | 3316 | 3317 | 3318 | 3319 | 3320 | 3321 | 3322 | 3323 | 3324 | 3325 | 3326 | 3327 |
| D00 | 3328 | 3329 | 3330 | 3331 | 3332 | 3333 | 3334 | 3335 | 3336 | 3337 | 3338 | 3339 | 3340 | 3341 | 3342 | 3343 |
| D10 | 3344 | 3345 | 3346 | 3347 | 3348 | 3349 | 3350 | 3351 | 3352 | 3353 | 3354 | 3355 | 3356 | 3357 | 3358 | 3359 |
| D20 | 3360 | 3361 | 3362 | 3363 | 3364 | 3365 | 3366 | 3367 | 3368 | 3369 | 3370 | 3371 | 3372 | 3373 | 3374 | 3375 |
| D30 | 3376 | 3377 | 3378 | 3379 | 3380 | 3381 | 3382 | 3383 | 3384 | 3385 | 3386 | 3387 | 3388 | 3389 | 3390 | 3391 |
| D40 | 3392 | 3393 | 3394 | 3395 | 3396 | 3397 | 3398 | 3399 | 3400 | 3401 | 3402 | 3403 | 3404 | 3405 | 3406 | 3407 |
| D50 | 3408 | 3409 | 3410 | 3411 | 3412 | 3413 | 3414 | 3415 | 3416 | 3417 | 3418 | 3419 | 3420 | 3421 | 3422 | 3423 |
| D60 | 3424 | 3425 | 3426 | 3427 | 3428 | 3429 | 3430 | 3431 | 3432 | 3433 | 3434 | 3435 | 3436 | 3437 | 3438 | 3439 |
| D70 | 3440 | 3441 | 3442 | 3443 | 3444 | 3445 | 3446 | 3447 | 3448 | 3449 | 3450 | 3451 | 3452 | 3453 | 3454 | 3455 |
| D80 | 3456 | 3457 | 3458 | 3459 | 3460 | 3461 | 3462 | 3463 | 3464 | 3465 | 3466 | 3467 | 3468 | 3469 | 3470 | 3471 |
| D90 | 3472 | 3473 | 3474 | 3475 | 3476 | 3477 | 3478 | 3479 | 3480 | 3481 | 3482 | 3483 | 3484 | 3485 | 3486 | 3487 |
| DA0 | 3488 | 3489 | 3490 | 3491 | 3492 | 3493 | 3494 | 3495 | 3496 | 3497 | 3498 | 3499 | 3500 | 3501 | 3502 | 3503 |
| DB0 | 3504 | 3505 | 3506 | 3507 | 3508 | 3509 | 3510 | 3511 | 3512 | 3513 | 3514 | 3515 | 3516 | 3517 | 3518 | 3519 |
| DC0 | 3520 | 3521 | 3522 | 3523 | 3524 | 3525 | 3526 | 3527 | 3528 | 3529 | 3530 | 3531 | 3532 | 3533 | 3534 | 3535 |
| DD0 | 3536 | 3537 | 3538 | 3539 | 3540 | 3541 | 3542 | 3543 | 3544 | 3545 | 3546 | 3547 | 3548 | 3549 | 3550 | 3551 |
| DE0 | 3552 | 3553 | 3554 | 3555 | 3556 | 3557 | 3558 | 3559 | 3560 | 3561 | 3562 | 3563 | 3564 | 3565 | 3566 | 3567 |
| DF0 | 3568 | 3569 | 3570 | 3571 | 3572 | 3573 | 3574 | 3575 | 3576 | 3577 | 3578 | 3579 | 3580 | 3581 | 3582 | 3583 |
| E00 | 3584 | 3585 | 3586 | 3587 | 3588 | 3589 | 3590 | 3591 | 3592 | 3593 | 3594 | 3595 | 3596 | 3597 | 3598 | 3599 |
| E10 | 3600 | 3601 | 3602 | 3603 | 3604 | 3605 | 3606 | 3607 | 3608 | 3609 | 3610 | 3611 | 3612 | 3613 | 3614 | 3615 |
| E20 | 3616 | 3617 | 3618 | 3619 | 3620 | 3621 | 3622 | 3623 | 3624 | 3625 | 3626 | 3627 | 3628 | 3629 | 3630 | 3631 |
| E30 | 3632 | 3633 | 3634 | 3635 | 3636 | 3637 | 3638 | 3639 | 3640 | 3641 | 3642 | 3643 | 3644 | 3645 | 3646 | 3647 |
| E40 | 3648 | 3649 | 3650 | 3651 | 3652 | 3653 | 3654 | 3655 | 3656 | 3657 | 3658 | 3659 | 3660 | 3661 | 3662 | 3663 |
| E50 | 3664 | 3665 | 3666 | 3667 | 3668 | 3669 | 3670 | 3671 | 3672 | 3673 | 3674 | 3675 | 3676 | 3677 | 3678 | 3679 |
| E60 | 3680 | 3681 | 3682 | 3683 | 3684 | 3685 | 3686 | 3687 | 3688 | 3689 | 3690 | 3691 | 3692 | 3693 | 3694 | 3695 |
| E70 | 3696 | 3697 | 3698 | 3699 | 3700 | 3701 | 3702 | 3703 | 3704 | 3705 | 3706 | 3707 | 3708 | 3709 | 3710 | 3711 |
| E80 | 3712 | 3713 | 3714 | 3715 | 3716 | 3717 | 3718 | 3719 | 3720 | 3721 | 3722 | 3723 | 3724 | 3725 | 3726 | 3727 |
| E90 | 3728 | 3729 | 3730 | 3731 | 3732 | 3733 | 3734 | 3735 | 3736 | 3737 | 3738 | 3739 | 3740 | 3741 | 3742 | 3743 |
| EA0 | 3744 | 3745 | 3746 | 3747 | 3748 | 3749 | 3750 | 3751 | 3752 | 3753 | 3754 | 3755 | 3756 | 3757 | 3758 | 3759 |
| EB0 | 3760 | 3761 | 3762 | 3763 | 3764 | 3765 | 3766 | 3767 | 3768 | 3769 | 3770 | 3771 | 3772 | 3773 | 3774 | 3775 |

## HEXADECIMAL - DECIMAL INTEGER CONVERSION TABLE (Continued)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EC0 | 3776 | 3777 | 3778 | 3779 | 3780 | 3781 | 3782 | 3783 | 3784 | 3785 | 3786 | 3787 | 3788 | 3789 | 3790 | 3791 |
| ED0 | 3792 | 3793 | 3794 | 3795 | 3796 | 3797 | 3798 | 3799 | 3800 | 3801 | 3802 | 3803 | 3804 | 3805 | 3806 | 3807 |
| EE0 | 3808 | 3809 | 3810 | 3811 | 3812 | 3813 | 3814 | 3815 | 3816 | 3817 | 3818 | 3819 | 3820 | 3821 | 3822 | 3823 |
| EF0 | 3824 | 3825 | 3826 | 3827 | 3828 | 3829 | 3830 | 3831 | 3832 | 3833 | 3834 | 3835 | 3836 | 3837 | 3838 | 3839 |
| F00 | 3840 | 3841 | 3842 | 3843 | 3844 | 3845 | 3846 | 3847 | 3848 | 3849 | 3850 | 3851 | 3852 | 3853 | 3854 | 3855 |
| F10 | 3856 | 3857 | 3858 | 3859 | 3860 | 3861 | 3862 | 3863 | 3864 | 3865 | 3866 | 3867 | 3868 | 3869 | 3870 | 3871 |
| F20 | 3872 | 3873 | 3874 | 3875 | 3876 | 3877 | 3878 | 3879 | 3880 | 3881 | 3882 | 3883 | 3884 | 3885 | 3886 | 3887 |
| F30 | 3888 | 3889 | 3890 | 3891 | 3892 | 3893 | 3894 | 3895 | 3896 | 3897 | 3898 | 3899 | 3900 | 3901 | 3902 | 3903 |
| F40 | 3904 | 3905 | 3906 | 3907 | 3908 | 3909 | 3910 | 3911 | 3912 | 3913 | 3914 | 3915 | 3916 | 3917 | 3918 | 3919 |
| F50 | 3920 | 3921 | 3922 | 3923 | 3924 | 3925 | 3926 | 3927 | 3928 | 3929 | 3930 | 3931 | 3932 | 3933 | 3934 | 3935 |
| F60 | 3936 | 3937 | 3938 | 3939 | 3940 | 3941 | 3942 | 3943 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 |
| F70 | 3952 | 3953 | 3954 | 3955 | 3956 | 3957 | 3958 | 3959 | 3960 | 3961 | 3962 | 3963 | 3964 | 3965 | 3966 | 3967 |
| F80 | 3968 | 3969 | 3970 | 3971 | 3972 | 3973 | 3974 | 3975 | 3976 | 3977 | 3978 | 3979 | 3980 | 3981 | 3982 | 3983 |
| F90 | 3984 | 3985 | 3986 | 3987 | 3988 | 3989 | 3990 | 3991 | 3992 | 3993 | 3994 | 3995 | 3996 | 3997 | 3998 | 3999 |
| FA0 | 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 | 4008 | 4009 | 4010 | 4011 | 4012 | 4013 | 4014 | 4015 |
| FB0 | 4016 | 4017 | 4018 | 4019 | 4020 | 4021 | 4022 | 4023 | 4024 | 4025 | 4026 | 4027 | 4028 | 4029 | 4030 | 4031 |
| FC0 | 4032 | 4033 | 4034 | 4035 | 4036 | 4037 | 4038 | 4039 | 4040 | 4041 | 4042 | 4043 | 4044 | 4045 | 4046 | 4047 |
| FD0 | 4048 | 4049 | 4050 | 4051 | 4052 | 4053 | 4054 | 4055 | 4056 | 4057 | 4058 | 4059 | 4060 | 4061 | 4062 | 4063 |
| FE0 | 4064 | 4065 | 4066 | 4067 | 4068 | 4069 | 4070 | 4071 | 4072 | 4073 | 4074 | 4075 | 4076 | 4077 | 4078 | 4079 |
| FF0 | 4080 | 4081 | 4082 | 4083 | 4084 | 4085 | 4086 | 4087 | 4088 | 4089 | 4090 | 4091 | 4092 | 4093 | 4094 | 4095 |

APPENDIX 6 APPENDIX 6

## BIBLIOGRAPHY

AN INTRODUCTION TO MICROCOMPUTERS, VOLUME O, THE BEGINNER'S BOOK, Adam Osborne and Associates, Inc. 1977.

BASIC PROGRAMMING IN REAL TIME, Don Cassel, 1942, Reston Publishing Company, Inc. 1975.

SIMPLIFIED BASIC PROGRAMMING, Gerald A. Silver, McGraw-Hill Co., 1974.

Basic BASIC, James S. Coan, Hayden, 1970.

Advanced BASIC, James S. Coan, Hayden, 1977.

PROBLEMS FOR COMPUTER SOLUTION, Fred Gruenberger and George Jaffray, Wiley, 1965.

BASIC, Samuel L. Marateck, Academic Press, 1975.

SOME COMMON BASIC PROGRAMS, Lon Poole and Mary Borchers, Adam Osborne & Associates, Inc. 1977.

GAME PLAYING WITH BASIC, Donald D. Spencer, Hayden, 1975.

GAME PLAYING WITH COMPUTERS, Donald D. Spencer, Hayden, 1975.

101 BASIC COMPUTER GAMES, Digital Equipment Corporation, 1975.

THEORY AND PROBLEMS OF MATRICES, Schaum's Outline Series, Frank Ayres, Jr., McGraw-Hill Co., 1962.

NEVADA COBOL Application Packages Book1, Ellis C., ELLIS COMPUTING, 1980.

NEVADA EDIT, Ellis C., & Starkweather,J., ELLIS COMPUTING, 1982.

NEVADA COBOL, Ellis C., ELLIS COMPUTING, 1979.

Hogan, T., CPM Users Guide, Osborne, 1981.

NEVADA PILOT, Starkweather, J., ELLIS COMPUTING, 1981.

## CONFIGURING AN UNKNOWN TERMINAL

In many cases you will find the terminal you are using listed for a simple choice. Also, most of the newer machines emulate one of the listed terminals. For example the KAYPRO II emulates the Lear-Seigler ADM-3A. Other frequently emulated terminals are the Lear-Seigler ADM-31 and the SOROC IQ-120. So if your machine is not listed try these emulations first.

```
A>NVBASIC

NVBASIC VERSION 2.1 (0) CONFIGURING
COPYRIGHT (C) 1983 ELLIS COMPUTING, INC.
@ ANSI MODE TERMINAL
A ADVANTAGE
B APPLE COMPUTER, 40 COLUMN DISPLAY   (sends soroc IQ-120)
C APPLE COMPUTER + VIDEX 80 COLUMN BOARD    (soroc IQ-120)
D BEEHIVE 150 OR CROMENCO 3100
E COMMODORE 64
F FREEDOM 100
G HAZELTINE 1400 SERIES
H HAZELTINE 1500 SERIES
I HEATH H19/H89 OR ZENITH Z19/Z89
J HEWLETT-PACKARD 2621

Type a single letter to select terminal.
<Carriage Return> for more terminals

K IBM PERSONAL COMPUTER+ BABY BLUE CARD
L INFOTON I-100
M LEAR-SEIGLER ADM-3A
N LEAR-SEIGLER ADM-31
O MICROTERM ACT-IV
P OSBORNE I
Q PERKIN-ELMER 550 (BANTOM)
R PROCESSOR TECHNOLOGY SOL OR VDM
S SOROC IQ-120/140
T SUPERBRAIN
U TELEVIDEO 950
V TRS-80, MOD II (P. & T. CP/M)
W NONE OF THE ABOVE
Type a single letter to select terminal.
<Carriage Return> for more terminals W

Enter 2 digits for number of lines
in the display. 24

Enter 2 digits for number of characters
per line. ('U' will restart entries.) 80

Enter M for memory-mapped display (bank 0 only), T for
```

```
a serial-connected terminal.  (M or T)  T

Most terminals position the cursor from
a sequence of characters as follows:
  Lead characters, differing for
    different terminals.
  The line number, sometimes offset
  Sometimes separator characters
  The column number, sometimes offset
  Sometimes ending characters
On some terminals column is before line.
Does your terminal follow
this general pattern? (Y/N) Y

Enter no. of lead characters
for cursor positioning. 2

Enter the first lead character
in hex, e.g. '1B'. 1B
Enter the next lead character. 59

Enter the no. of line/col separator
characters. 0

Enter the no. of ending characters. 0

Enter offset to be added to line value.
  Enter 2 hex characters,e.g. 20 20

Enter offset added to column value. 20

Is column entered before line? (Y/N) N

Following controls will speed editing.
If control is not available, enter zero.

Enter no of characters to clear screen
and home the cursor to upper left. 2
Enter two hex characters for each. 1B45

Enter no. of characters to insert line
above cursor position. 2
Enter two hex characters for each. 1B4C

Enter the no. of characters to delete
the cursor line. 2
Enter two hex characters for each. 1B4D

---------------------------------------
Configuring of BASIC.COM is complete.

BASIC.COM saved on the default drive.
```

**Ellis Computing**
**3917 Noriega Street, San Francisco, CA 94122**

**SOFTWARE LICENSE AGREEMENT**

IMPORTANT: All Ellis Computing programs are sold only on the condition that the purchaser agrees to the following License.

ELLIS COMPUTING agrees to grant and the Customer agrees to accept on the following terms and conditions nontransferable and nonexclusive Licenses to use the software program(s) (Licensed Programs) herein delivered with this Agreement.

TERM:

This Agreement is effective from the date of receipt of the above-referenced program(s) and shall remain in force until terminated by the Customer upon one month's prior written notice, or by Ellis Computing as provided below.

Any License under this Agreement may be discontinued by the Customer at any time upon one month's prior written notice. Ellis Computing may discontinue any License or terminate this Agreement if the Customer fails to comply with any of the terms and conditions of this Agreement.

LICENSE:

Each program License granted under this Agreement authorizes the Customer to use the Licensed Program in any machine readable form on any single computer system (referred to as System). A separate license is required for each System on which the Licensed Program will be used.

This Agreement and any of the Licenses, programs or materials to which it applies may not be assigned, sublicensed or otherwise transferred by the Customer without prior written consent from Ellis Computing. No right to print or copy, in whole or in part, the Licensed Programs is granted except as hereinafter expressly provided.

PERMISSION TO COPY OR MODIFY LICENSED PROGRAMS:

The customer shall not copy, in whole or in part, any Licensed Programs which are provided by Ellis Computing in printed form under this Agreement. Additional copies of printed materials may be acquired from Ellis Computing.

The NEVADA BASIC Licensed Programs which are provided by Ellis Computing in machine readable form may be copied, in whole or in part, in machine readable form in sufficient

number for use by the Customer with the designated System, for back-up purposes, or for archive purposes. The original, and any copies of the Licensed Programs, in whole or in part, which are made by the Customer shall be the property of Ellis Computing. This does not imply that Ellis Computing owns the media on which the Licensed Programs are recorded.

The Customer agrees to reproduce and include the copyright notice of Ellis Computing on all copies, in whole or in part, in any form, including partial copies of modifications, of Licensed Programs made hereunder.

PROTECTION AND SECURITY:

The Customer agrees not to provide or otherwise make available the NEVADA BASIC Program including but not limited to program listings, object code and source code, in any form, to any person other than Customer or Ellis Computing employees, without prior written consent from Ellis Computing, except with the Customer's permission for purposes specifically related to the Customer's use of the Licensed Program.

DISCLAIMER OF WARRANTY:

Ellis Computing makes no warranties with respect to the Licensed Programs.

LIMITATION OF LIABILITY:

THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ELLIS COMPUTING BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF ELLIS COMPUTING HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

GENERAL:

If any of the provisions, or portions thereof, of this Agreement are invalid under any applicable statute or rule of law, they are to that extent to be deemed omitted. This is the complete and exclusive statement of the Agreement between the parties which supercedes all proposals, oral or written, and all other communications between the parties relating to the subject matter of this Agreement. This Agreement will be governed by the laws of the State of California.

## CORRECTIONS AND SUGGESTIONS

All suggestions and problems must be reported in writing. Please include samples if possible.

BASIC VERSION _____ SERIAL #_____

Operating system and version _____

Hardware configuration _____

ERRORS IN MANUAL:

SUGGESTIONS FOR IMPROVEMENTS TO MANUAL:

ERRORS IN BASIC:

SUGGESTIONS FOR IMPROVEMENT TO BASIC:

MAIL TO: Ellis Computing
3917 Noriega Street
San Francisco,CA 94122

FROM: NAME_____ DATE_____

ADDRESS_____

CITY,STATE,ZIP_____

PHONE NUMBER_____

If you wish a reply include a self-addressed postage-paid envelope. Thank you.