Extended Disk BASIC

User's Manual

Processor Technology

SECTIONS

1

2

3

4

5

6

7

Extended Disk BASIC User's Manual

Describes DBASIC, Release 1.1

Processor Technology Corporation

7100 Johnson Industrial Drive
Pleasanton, CA 94566
Telephone (415) 829-2600

APPENDICES

1

2

3

4

5

6

Second Edition, First Printing, August, 1978
Manual Part No 727037

IMPORTANT NOTICE

TABLE OF CONTENTS

APPENDICES

SECTION 1

1. INTRODUCTION

Extended BASIC is a special adaptation of BASIC (Beginner's all-purpose Symbolic Instruction Code) for use with the Processor Technology Disk Operating System (PTDOS) and Helios II Disk Memory 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 Extended BASIC are:

*Fully-formatted output to a variety of devices
*Many function subprograms, including mathematical, string, and video functions
*Program and data storage on floppy disk
*Full eight-digit precision
*User-defined functions on one or more lines
*Calculator mode for immediate answers
*Moving-cursor editing on video displays
*Complete capability for string handling
*Functions and statements for communicating with any number of input/output channels
*Ability to view memory locations, change values, and branch to absolute addresses
*DATA files
*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 enters
        20 INPUT X <CR>                          the lines of a program,
        30 LET Y = X^3 <CR>                      each followed by a
        40 PRINT "X CUBED IS ";X^3 <CR>  carriage return.
        DEL 30 <CR>                     -The user deletes line 30.
        LIST <CR>                       -The user tells BASIC to
                                         list what has been typed.
```

BASIC:	10 PRINT "WHAT IS THE VALUE OF X" 20 INPUT X 40 PRINT "X CUBED IS ";X^3	-BASIC list all but line 30, which was deleted.
User:	RUN <CR>	-The user tells BASIC to execute the program.
BASIC: User: BASIC:	WHAT IS THE VALUE OF X ?3 <CR> X CUBED IS 27 READY	-The user types 3 in response to the ? prompt.

1.1. HOW TO USE THIS BOOK

This book is intended as a description of this particular version of BASIC, namely Extended Disk 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 Extended 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).

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 upper- and lowercase characters to differentiate between characters to be typed literally and terms indicating the types of information to be inserted. 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 elipses indicate an indefinite number of arguments.

Optional parts of command and statement forms are enclosed in braces. 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.

1.2. SYSTEM REQUIREMENTS

Extended BASIC must be used in conjunction with the Processor Technology Disk Operating System (PTDOS), software which is normally used on the Helios II Disk Memory System. Besides the hardware required to support PTDOS, BASIC requires additional memory from location 100 (Hex) to an address which depends on how BASIC is initialized. (See Section 2.1.) During initialization, you will have the opportunity to delete portions of the BASIC interpreter that may not be of use to you, thereby saving the memory they would have occupied for program storage. The last address used by the BASIC interpreter is given during the initialization process, but an additional amount of memory is required for program and data storage. Overall, at least 24K is recommended.

The current BASIC program begins at the address immediately above BASIC itself, an address which varies with initialization as described above.

BASIC includes video display driver software. During initialization it is possible to select an option which sends all output from BASIC to another PTDOS file, which could be a device driver file, instead of to the video display driver. If the video display driver will be used, it is necessary to have compatible video display circuitry, normally the video display circuitry in the Sol Terminal Computer or a VDM-1 video display module. This circuitry is based around a 1K page of system memory addressed at CC00, which holds the characters to be displayed.

SECTION 2

2. ELEMENTS OF BASIC USAGE

Before writing and working with BASIC programs, you have to know how to get into the BASIC environment and the rules for using BASIC. This section presents the fundamentals of BASIC usage.

2.1. HOW TO INITIALIZE AND LEAVE BASIC

Extended BASIC is provided on a floppy diskette in a version compatible with the Processor Technology Disk Operating System (PTDOS). It is assumed in the instructions below that you have a Helios II Disk Memory System or other computer which is using PTDOS. 24K of random access memory starting at address 0 is recommended.

Extended Disk BASIC is stored on diskette under the name DBASIC. Although BASIC can be used directly from this disk file, it is more convenient to load in DBASIC from diskette, perform some initialization procedures, and store the initialized version under a new file name for normal usage. A procedure file named MAKBASIC is supplied on your BASIC disk to help accomplish this initialization. To use it, put a PTDOS system disk in unit 0 and the BASIC disk in unit 1, and follow these steps:

1) In PTDOS command mode, type:

```
DO MAKBASIC/1
```

This will clear memory and load in the uninitialized DBASIC from diskette. BASIC will begin execution and a copyright notice will appear, and then the message: SIZING MEMORY. At this time BASIC scans the memory locations above BASIC to determine how much space is available for program and data storage.

2) After a brief delay, the message:

```
LAST AVAILABLE MEMORY LOCATION (HEX) IS nnnn
```

appears, where nnnn is a memory address in hexadecimal notation. If an address appears which is lower than expected, it may be due to a bad memory address or the existence of read only memory at location nnnn + 1.

3) Next, a question appears:

DELETE MATRIX OPERATIONS?

Now type Y for yes or N for no. If you type Y, the part of BASIC which performs matrix operations will be removed from the version you are initializing, making more memory available for programs and data. If you type N, a new message will appear; go to step 5 below.

4. If you typed Y the following additional message will appear:

DELETE EXTENDED FUNCTIONS?

Again type Y or N to remove or not remove an additional part of BASIC which performs trigonometric functions and certain other extended functions. The following functions cannot be used if Y is typed: SIN, COS, TAN, EXP, ATN, LOG, LOG1Ø, and the exponentiation operator ^.

5. After Y or N is typed, this message appears:

PTDOS OUTPUT or INTERNAL VDM DRIVER? (I,P)

If you want all output generated by BASIC to go to BASIC's video display driver software, type I. (See Section 1.2 for the hardware requirements of this software.) If you want all of BASIC's output to go to the PTDOS Command Interpreter console output file, type P. (The file used for console output may be changed with the PTDOS SETOUT command. See Section 1.8 of the PTDOS Manual.) Once you are in BASIC, you can still change the output file with the SET OF statement.

6. After I or P is typed, the following message will appear:

SAVE FROM yyyy TO xxxx

where yyyy and xxxx represent addresses in hexadecimal form. Depending on how you answered the questions above, the newly initialized form of BASIC will now occupy the addresses between yyyy and xxxx. A working copy of BASIC (named BASIC) is now IMAGEd onto the disk in unit Ø. Since BASIC is stored as an image file, you can now enter BASIC by simply typing BASIC <CR> in PTDOS command mode. Be sure to keep a safe copy of DBASIC

in case a version that is initialized differently is needed. If want a different version, simply repeat the above procedure. You can see the PTDOS commands which accomplished the initialization by typing the command PRINT MAKBASIC/1 in PTDOS.

After you are done working in BASIC, you can exit to PTDOS command mode by simply typing BYE <CR>. Any program that was in BASIC, and options you may have selected, will remain intact. Note that many PTDOS commands not marked [SAFE], and other operations can write over BASIC in memory.

If you are in PTDOS command mode, you can return to BASIC, provided it is still intact in memory, by typing:

EXEC 1ØØ <CR>

Using this mode of re-entry preserves any program that was in BASIC, as well as various options you may have selected when you were last in BASIC. If you re-enter BASIC by typing its name as a command, you clear any program that was in BASIC, and set various other options to their default values.

If you have a BASIC program stored as a PTDOS file, 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.

When system power is first applied, and you BOOTLOAD PTDOS, a DO file called START.UP, containing a series of commands, is automatically executed. If you want your system to "come up" in BASIC, or you would like to run a BASIC program on bootloading, you can enter an appropriate command line the START.UP file.

2.2. 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. For more information type BASIC RUNME/1 <CR>.

2.3. 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.

2.4. DESCRIPTION OF BASIC STATEMENTS

A statement is preceded by a line number which must be an integer between 1 and 65000. 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.

2.4.1. Constants

A constant is a quantity that has a fixed value. In Extended 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:

	Examples
Integer	1, 4000, 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:

```
"Illinois"
"The answer is"
```

Strings are discussed in more detail in section 5.

2.4.2. Variables

A variable is an entity that can be assigned a value. In Extended 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 variables:

```
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, Dec. 9"
```

2.4.3. 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:

^	exponentiate
*	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.

2.5. 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
```

2.6. THE CALCULATOR MODE OF BASIC

In unit 2.2, 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 Extended 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 in the box containing the statement form.

Without exception, all number 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

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.

3.1. CREATING A PROGRAM

To create a program, simply type the 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 by the SET LL = length command or statement. For more information about line length errors, see "LL" in Appendix 3.

It is not necessary to enter the 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 statement
         LIST <CR>                                   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:

DEL	Deletes the current character and shifts the remainder of the line to the left.
← (Left Arrow)	Moves the cursor one position to the left.
→ (Right Arrow)	Moves the cursor one position to the right.
REPEAT	Moves the cursor rapidly through the line when used with the left or right arrows. Also causes repetition of any key held down at the same time.
MODE SELECT	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.
RETURN	Terminates the line. The line remains as it appeared when the RETURN key was typed.
LINE FEED	Terminates the line. All characters to the right of the cursor are erased.
↑ (Up Arrow)	Initiates the insert mode. When you type characters in the insert mode, they are inserted at current cursor position, and the rest of the line is moved to the right.
↓ (Down Arrow)	Terminates the insert mode.
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.

3.2. 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 displays the program. DELETE and SCRATCH are used to erase statements. REN lets you automatically renumber statements. The EDIT command makes the line editor available.

LIST Command

General forms:

```
LIST            List the entire program.
LIST n          List statement number n.
LIST n1,        List statement number n1 through the end
                of the program.
LIST ,n2        List all statements from the first through
                statement number n2.
LIST n1,n2      List statements numbered n1 through n2.
     | |
     | |__Last in a series of statement numbers
     |__First in a series of statement numbers
```

Examples:

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

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. For example:

```
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
```

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

MODE key	Aborts listing
Space bar	Causes a pause in the listing. Striking it again causes the listing to resume.
1 through 9	Changes the speed at which material is displayed.

DEL Command

```
General forms:

  DEL             Delete all statements.
  DEL n           Delete statement number n.
  DEL nl,         Delete all statements from nl through the
                  end of the program.
  DEL ,n2         Delete all statements from the first through
                  statement n2.
  DEL nl,n2       Delete statement numbers nl through n2.
       | |
       | |__Last in a series of statement numbers
       |__First an a series of statement numbers

Examples:

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

The DEL command deletes the indicated statements. For 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
```

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

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

SCRATCH Command

```
General form:

  SCR{ATCH}        Delete the entire program and clear all
                   variable definitions.
Examples:
  SCR <CR>
  SCRATCH <CR>
```


The SCRATCH command deletes the entire program and clears all variable definitions established during previous program runs or by statements executed in the calculator mode. For example:

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

REN Command

```
General forms:

  REN            Renumber all statements. The first statement
                 will be numbered 10 and subsequent statement
                 numbers will be increments of 10.
  REN n          Renumber all statements. The first statement
                 will be numbered n and subsequent statement
                 numbers will be increments of 10.
  REN n,i        Renumber all statements. The first statement
      | |        will be numbered n and subsequent statement
      | |        numbers will be increments of i.
State-| |__integer increment
ment
number

Examples:

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

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

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

```
General form:

  EDIT n            Edit statement number n.
       |
       |__Statement number

Example:

  EDIT 150 <CR>
```

The EDIT command displays the line to be edited on the line in which the EDIT command was typed, and the following line. EDIT then enters a mode that allows changes to the line using any of the following special keys:

Key	Effect in EDIT Mode
DEL	Deletes the current character and shifts the remainder of the line to the left.
← (Left Arrow)	Moves the cursor one position to the left.
→ (Right Arrow)	Moves the cursor one position to the right.
REPEAT	Moves the cursor rapidly through the line when used with a <- or ->.
CONTROL-X	Cancels the line being typed, and positions the cursor on a new line. The cancelled line remains on the screen.
MODE SELECT	Terminates the edit leaving the line as it was.
RETURN	Terminates the edit leaving the line as it appears on the screen.
LINE FEED	Terminates the edit deleting all characters to the right of the cursor.
↑ (Up Arrow)	Initiates the insert mode. When you type characters in the insert mode, they are inserted at the current cursor position and the rest of the line is moved to the right.
↓(Down Arrow)	Terminates the insert mode.

For example:

```
User:   10 PRINT "ENTER Q, Y, AND Z" <CR>
        20 INT X,Y,Z <CR>
        EDIT 10 <CR>
BASIC:  10 PRINT "ENTER Q, Y, AND Z"
User:   (Positions the cursor over Q and types X <CR>).
        LIST 10 <CR>
BASIC:     10 PRINT "ENTER X, Y, AND Z"
User:   EDIT 20 <CR>
BASIC:     20 INT X,Y,Z
User:   (Positions the cursor over T and strikes the up arrow
        key.  In insert mode he then types PU.  A line feed
        terminates the edit.)
        LIST <CR>
BASIC      20 INPUT X,Y,Z
```

3.3. EXECUTING A PROGRAM

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

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.

During execution a program can be interrupted by pressing the MODE key. This is true whether the program is running correctly, is in a loop, or is waiting for input. No information is lost and you can continue execution by giving the CONT command.

RUN Command

```
General forms:

  RUN               Execute the current program.
  RUN n             Execute the current program beginning with
      |             statement number n.
      |__Statement number

Examples:

  RUN <CR>
  RUN 100 <CR>
```

The RUN command executes all or part of the current program. If no statement number is specified, the command clears all variables and then executes the program. If a statement number is indicated, the command executes the program beginning with that statement number, but does not clear the variable definitions first. For 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 still have
         READY                the values assigned in statement 10.
```

The Mode Select key may be used to abort a running program. When a program run terminates for this or any reason, all open files are closed.

CONT Command

General form:	
CONT	Continue execution.
Example:	
CONT <CR>	

The CONT command continues the execution of a program that was interrupted by the MODE key or stopped by the execution of a STOP statement (STOP is documented on page 4-9.) For example:

```
User:    RUN <CR>         The user executes a program that computes
BASIC:    1               and prints the squares of numbers 1
          4               through 100.
          9
          16
User:    MODE                The user presses the MODE key to inter-
BASIC:   STOP IN LINE 70     rupt execution.
User:    CONT <CR>           The CONT command continues execution of
BASIC:    25                 the program.
          36
          49
           .
           .
           .
```

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

CLEAR Command

General form:	
CLEAR	Erases the definitions of all variables and leaves the program intact.
Example:	
CLEAR <CR>	

The CLEAR command clears all variable definitions but does not erase the statements of the current program. If CLEAR is used as a statement, all open files will be closed.

For 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 <CR>
BASIC:    10            20            30    The variables have the
         READY                              values assigned in line
User:    CLEAR <CR>                         10.
         RUN 30 <CR>
BASIC:    0             0             0     Variable definitions
         READY                              have been cleared.
User:    LIST <CR>
BASIC:    10 A=10,B=20,C=30                 The program remains
          20 STOP                           intact.
          30 PRINT A,B,C
```

3.4. 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.

3.4.1 Text and Semi-Compiled Modes of Program Storage

Four commands involved in storing and retrieving programs from diskette: SAVE, GET, APPEND, and XEQ, all have optional parameters T, for text mode of storage, or C, for semi-compiled mode of storage. (APPEND does not offer the semi-compiled option.) 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, and allows programs to be recorded and accessed faster. The semi-compiled program may not intelligible to other versions of BASIC, however, and cannot be manipulated in a meaningful way by other editors.

3.4.2 Commands for Handling Diskette Program Files

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

COMMAND file name

The file name is the name of a PTDOS file, and subject to the same conditions as any other PTDOS file. The file name can be from one to eight alphanumeric characters. Two extra characters may be added to the end of the file name (making the name up to 10 characters long) which specify the diskette drive unit to be used in the command. Helios II Disk Memory systems can be configured with up to eight units, each of which can contain a diskette. The minimum, and most popular system, has two units called 0 and 1. The characters to add are the slash (/) and a digit from 0 to 1 or 0 to 7, depending on how many units your system has. Note that blanks are not allowed: MYFILE/1 is an acceptable filename. Note that the unit specification does not refer to a particular diskette, but to any diskette which is inserted in the specified unit. If the diskette containing MYFILE were in unit 0, it would be called MYFILE/0, or just MYFILE (if 0 is the default unit). If the same diskette were inserted in unit 1, the same file would be called MYFILE/1.

If the file name is given alone in the command, without a unit specification, the default unit (usually 0) is used. PTDOS allows the user to change which unit will be the default unit.

In all, BASIC can create and manipulate files of four types. Besides types C and T (see above), data files use types R and S. The commands in BASIC for handling these four types, use the four letters T, C, R, and S to specify the types. BASIC uses PTDOS system calls in working with these files, and all four types will appear in the PTDOS directory of the diskette on which they are written, and will be listed by the PTDOS FILES command. However, the file type designation in the PTDOS directory is different from the type letters used by BASIC. Below is a table which relates the two type designations:

BASIC Type	PTDOS Type	Description
T	06	Text-type program files
C	05	Semi-compiled program files
R	08	Random access data files
S	07	Serial access data files

With certain limitations, a file of any type created in BASIC may be manipulated from within PTDOS, and vise versa. In particular, it is convenient to create or edit BASIC programs

in text form with the PTDOS editor. This editor has features which are not available in the BASIC editor. The line length of such programs should be limited to 64 characters, however, to avoid truncation.

If a PTDOS file contains the appropriate contents, BASIC can be used to manipulate the file, no matter what type is assigned.

SAVE Command

```
General form:

  SAVE  file name  {,mode}   Save the current program on a
            |          |     diskette file and label it with
  1 to 8 ___|          |     the specified file name.
  characters{/unit}    |
                       |__T or C
Examples:

  SAVE  SUMS  <CR>
  SAVE  ADDRESSX/1 ,T <CR>
```

The SAVE command writes the current program on a disk file and labels the file with the specified name. If the diskette already contains a file of the specified name, that file will be overwritten. SAVEd files are "information-protected". They will not by shown by the simple PTDOS FILES command.

The T and C options let you specify that the text of your program is to be saved or that a semi-compiled version of the program is to be saved. C (semi-compiled) is the default option and need not be specified. If a file of the given name already exists on diskette, but is of a different type (T or C) than given, an error message will be displayed. Semi-compiled program files are name- and attribute-protected as well as information-protected.

In deciding whether to save your program in text or in semi-compiled form, keep the following factors in mind:

Semi-compiled	versus	Text
-Is more efficient		-Is recognizable as a sequence of BASIC statements
-Loads more quickly		-Can be edited by editors outside BASIC
-Can be saved more quickly		-Is independant of the version of BASIC in use
-Might be dependent on the version of BASIC in use		
-Cannot be edited by external editors		

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

For 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 Command

```
General form:

  GET  file name            Read the specified file from
                            diskette.
 1 to 8
 characters{/unit}

Examples:

  GET  SUMS  <CR>
  GET  AN33/2 <CR>
```

The GET command searches the directory for the specified file, then reads the file making the program contained on it available in BASIC. Any statements residing in BASIC before the file was read are lost. The GET command determines whether the file was SAVEd in text or semi-compiled form and acts accordingly.

The command below retrieves a program file named FAC from unit 1

```
        GET FAC/1
```

An example of an interchange using the GET command follows:

```
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 the
         25 S=1                          program that was read
         30 FOR I=1 TO 100               from diskette.
         40 S=S+S*R
         50 IF S >= 2 THEN 70
         60 NEXT I
         70 PRINT "INVESTMENT DOUBLES IN ",I;"YEARS"
```


XEQ Command

```
General form:

  XEQ file name             Read the specified file from
                            diskette and execute the pro-
  1 to 8                    gram contained in it.
  characters{/unit}

Examples:

  XEQ SQR <CR>
  XEQ STRAIGHT/1  <CR>
```

The XEQ command reads the specified file, making the program contained on it available in BASIC, and begins execution. Any statements residing in BASIC before the file was read are lost. For example:

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

APPEND Command

```
General form:

  APPEND file name          Read the specified file from disk
                            and merge the program contained on
     1 to 8                 it with the statements already re-
     characters{/unit}      siding in BASIC.

Example:

  APPEND PROG2/1
```

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. The line numbers of statements from the appended file determine their positions with respect to the statements already in BASIC. 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. If no unit is specified in the command, the default unit is used.

Note: Only text files can be appended.

For example:

```
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>
BASIC:     10 LET X=0
           20 PRINT "ENTER Y AND Z"
           30 INPUT Y,Z
          100 A1=X+Y+Z
          110 A2=X+Y-Z
          120 A3=X-Y+Z
          130 PRINT A1,A2,A3
```

Now BASIC contains the statements read from diskette as well as the original statements.

KILL Command

General Form:

KILL file name | 1 to 8 characters{/unit}	Kill the named file, as if it were erased.

Examples:

```
KILL JEC
KILL USELESS/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 returned to the Free Space Map for use by other files. (Actually the file is not literally erased, but the space used is made available for re-use.)

CAT Command

General Form:

CAT {/unit} {type}	Displays a catalog of files of the the specified type on the specified diskette drive unit.

Examples:

```
CAT
CAT /1
CAT /4T
```

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. There are four possible types:

T	Text type program files
C	Semi-compiled program files (default)
S	Serial data files
R	Random data files

If no type is specified, files of type C only will be listed. The PTDOS FILES command performs a function similar to the BASIC CAT command, but it also allows all files on a given diskette to be listed in one command.

Text-type program files and serial access data files can be created outside BASIC and used within BASIC provided that program lines are not longer than 132 characters. However, to be CATalogued, a program file must be given the type 06 and a serial access data file must be given the type 07. Semi-compiled program files and random access data files may be created only within BASIC.

SECTION 4

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 extended BASIC concepts.

REM Statement

```
General Form:

  REM {any series of characters}  Has no effect on program
                                  execution.

Examples:

  10 REM
 100 REM: THIS PROGRAM COMPUTES INCOME TAX
```

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. For 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
```

```
General forms:                                        calculator

  {LET} variable = expression        Assigns the value of the
                                     expression to the variable.
  {LET} variable1 = expression1, variable2 = expression2, ...

Examples:

  10 LET X = 100.50
 100 A1=12.7, A2=5.4, A3=50
 200 LET M$ = "SHREVEPORT"
```

The LET statement evaluates an expression and assigns its value to a variable. The variable may be a numeric or string variable and the value of the expression can be a number or a character string. The value of the expression and the variable must be the same type. For example:

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

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

4.1. 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 below displays information at the terminal's display device. Using these two statements, you can make your program converse with a user at the terminal.


```
General forms:                                          calculator

   INPUT var1, var2, ...       Reads one or more values from the
                               terminal and assigns them to var1,
variable                       var2, etc.

   INPUT "message", var1, var2, ... Prints the message, then
                                    reads values from the ter-
            any characters          and assigns them to var1,
                                    var2, etc.
Examples:

   10 INPUT X
  100 INPUT "WHAT IS THE VALUE OF S",S
  200 INPUT, A1, A2, A3, N, T$, Y
```

The INPUT statement accepts one or more values entered at the terminal and assigns them in order to the specified variables. The values entered must agree with the type of variable receiving the value.

When an INPUT statement is executed, BASIC requests values from the terminal by printing a question mark or the message, if you have specified one. You may enter one or more values after the question mark, but not more than are required by the INPUT statement. If you enter several values on one line, they must be separated by commas. BASIC prompts for additional value with two question marks until all values required by the INPUT statement have been entered. For example:

```
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.
```

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

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

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

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

If an INPUT statement requests input for a numeric variable, and the user's response contains an 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 available through the ERR(0) function.

PRINT Statement

```
General forms:                                              calculator

  PRINT                      Skips one line.
  PRINT exp                  Displays the value of exp.
  PRINT exp1, exp2, ...      Displays the values of exp1, exp2,
                             etc., each filling 14 columns.
  PRINT exp1; exp2; ...      Displays the values of exp1, exp2,
           |                 etc.
           |__exp is a numerical
              or string expression

Examples:

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

The PRINT statement displays information at the terminal. The information displayed is the value of each expression. It is displayed in order and the separation between one value and the next is determined by the separator used. If a comma is used as a separator, each value is printed at the left of a field of 14 character positions. If a semicolon is used between two expressions, the second is printed one space after the first. For example:

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


The output from 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. For example:

```
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 <CR>
BASIC:    1 2 3             4
         NEXT LINE
         READY
```

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 preceded by a back slash (\). A few of the special control characters and their functions are:

Control M	-	Carriage return
Control J	-	Line feed
Control K	-	Home cursor and clear screen
Control N	-	Home cursor

Printing other control characters displays special symbols on the screen, as shown in the Sol manual.

"\\" — Print a single backslash character (\).

For example:

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

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

While the PRINT statement is executing and values are being output, it is possible to interrupt the printing by depressing the space bar on the keyboard. Depressing the space bar a second time will cause printing to resume. The speed of printing may be controlled with the number keys 1 through 9, key 1 giving the fastest speed. The SET DS = nexpr command also controls speed of printing to the video display, but has the additional effect of controlling all output to the screen, whether or not it was generated by a PRINT statement. More complex forms of the PRINT statement are covered in Section 5.5 and 5.6.

4.2. 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(Ø) 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 5.5.

READ Statement

General form:

```
  READ var1, var2, ...     Reads one or more values from DATA
                           statements and stores them in var1,
variable                   var2, etc.
```

Examples:

```
 1Ø READ X2
1ØØ READ A1, A2, A3, M$
```

The READ statement reads one or more values from one or more DATA statements and assigns the values to specified variables. The value read must be the same type as the variable it is assigned to.

An example of a program using the READ statement is shown in the explanation of the DATA statement.

DATA Statement

General form:

```
  DATA constant1, constant2, ...   Specifies one or more
                                   values that can be read by
            number                 a READ statement.
            or string
```

Examples:

```
 1Ø DATA 47.12
1ØØ DATA "ALPHA",4ØØ,"BETA",22.6,"GAMMA",74.4
```

The DATA statement is used with the READ statement to assign values to variables. The values listed in one or more DATA statements are read sequentially by the READ statement. For example:


```
User:   1Ø DATA 44.2,76.4,18.9 <CR>
        2Ø DATA 1ØØ,47.8,11.25 <CR>
        3Ø READ A,B,C,D <CR>
        4Ø PRINT "SUM IS "; A+B+C+D <CR>
        5Ø READ X,Y <CR>
        6Ø PRINT "SUM IS "; X+Y <CR>
        RUN <CR>
BASIC:  SUM IS  239.5         (44.2 + 76.4 + 18.9 + 1ØØ)
        SUM IS  59.Ø5         (47.8 + 11.25)
        READY
```

TYP(Ø) Function

General Form:

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

Value	Type
1	numeric data
2	string data
3	data exhausted

Example:

```
1Ø IF TYP(Ø) = 3 THEN 3Ø
2Ø READ X
```

When the TYP(Ø) function is encountered the program looks ahead to the next data item in the DATA statement corresponding to the next READ statement. A value of 1, 2 or 3 is returned depending on the type of this data item. The argument Ø must appear. The example above skips a READ statement if the data in the corresponding DATA statement is exhausted. TYP(Ø) does not work for file READ.

RESTORE Statement

```
General forms:

  RESTORE                 Resets the pointer in the DATA state-
                          ments so that the next value read will
                          be the first value in the first DATA
                          statement.
 RESTORE n                Resets the pointer in the DATA state-
         |                ments so that the next value read will
         |_statement      be the first value in the DATA state-
           number         ment at or after line n.

Examples:

  10 RESTORE
 100 RESTORE 50
```

The RESTORE statement lets you change the reading sequence in DATA statements. You can start over or move to a particular DATA statement. For 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
```

ON...RESTORE Statement

```
General form:

 ON exp RESTORE n1,n2,...   If the value of exp is 1,
    |              | |      restores to statement n1, if it
    |              |_|state- is 2, restores to statement n2,
    |_numerical       ment   etc.
      expression      number

Examples:

  10 ON A+3 RESTORE 150
 100 ON R RESTORE 200, 300, 350
```

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

will start reading from the DATA statement selected. For example:

```
 10 READ X, Y, Z, A, B, C
 20 ON X-Y RESTORE 100, 110, 120
  .
  .                                  The first two value 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
```

4.3. STOPPING OR DELAYING EXECUTION

There are two ways to stop execution of a program from within the program. The END statement ends the 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 Mode Select key, or an error, all open files are closed. The PAUSE statement can be used to delay execution of the following statement for a period of .1 second to 1.82 hours.

END Statement

```
General form:

  END                          Terminates execution.

Example:

 100 END
```

The END statement terminates execution of a program, and closes all files that were open. For 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 Statement

```
General form:

  STOP                     Stops program execution.

Example:

 100 STOP
```

The STOP statement stops execution of a program, closes all open files, and displays the message:

    STOP IN LINE n

where n is the line number of the STOP statement. Execution can be continued with the CONT command. For 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 con-
BASIC:  THE AREA IS  3.1416            tinues execution with the
                                       next statement.
```

PAUSE Statement

```
General Form:

    PAUSE nexpr          Causes a pause before execution of
                         the following statement of duration
                         nexpr tenths of seconds. nexpr may
                         be from 1 to 65535.

Example:

    PAUSE 100            Gives a pause of 10 seconds.
```

The argument nexpr is first evaluated, and truncated to a positive integer between 1 and 65535. A pause of of approximately nexpr tenths of seconds then occurs before the next statement in the program is executed. If nexpr has a value less than 1, it will be truncated to zero and no pause will occur. If nexpr has a value greater or equal to 65536 an error message will appear. The precise duration of the pause is controlled by the clock rate of the microprocessor. In a Sol Terminal Computer with the standard 2.045 MHz jumper installed, the delays will be

approximately as given above. If the clock rate is faster or slower, the pause will be proportionately shorter or longer. The maximum delay is 65535 tenths of seconds, or approximately 1.82 hours. Of course multiple PAUSE statements or a loop can create a pause of any length.

4.4. EXECUTION CONTROL

The statements described in this unit 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 Statement

```
General forms:

  GO TO n                    Transfers control to statement
  GOTO n                     number n.
       |___statement
           number

Example:

  10 GO TO 150
```

The GO TO statement causes a specified statement to be the next statement executed. The statement number can be either greater than or less than the number of the GO TO statement. For 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 MODE key. 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 MODE key)
        STOP IN LINE 20
```

ON...GO TO Statement

```
General forms:

  ON exp GO TO n1, n2, ...      Executes statement n1 next if
  ON exp GOTO n1, n2, ...       exp is 1, executes n2 next if
                                exp is 2, etc.
                     statement
       numerical     number
       expression

Examples:

  10 ON X GO TO 30, 100
 100 ON A+2 GOTO 10,50,150
```

The ON...GO TO statement lets you branch to one of several statement numbers depending on the value of an expression. If the value of the expression is not an integer, BASIC truncates it to an integer. If there is no statement number corresponding to the value of the expression or truncated expression, the next line is executed.

For 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 GOTO 10
         60 PRINT "THE SUM IS ";X+Y:GOTO 10
         70 PRINT "THE DIFFERENCE IS ";Y-X:GOTO 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>          (1.9 is truncated to 1 by BASIC.)
BASIC:   THE SUM IS  323
         ENTER VALUES FOR X AND Y ? (The user strikes the MODE
         STOP IN LINE 10             key to escape the loop.)
```


FOR and NEXT Statements

```
General form:

  FOR var = exp1 TO exp2 {STEP i}
   .                                  numerical
   .                                  expressions
   .       numerical variable
   .
  NEXT {var}                  The statements between FOR and NEXT
                              are executed repeatedly as the value
        same                  of var increases from exp1 to exp2
        variable              in steps of 1 or in steps of i, if
        as used in            present.
        FOR statement
```

The FOR and NEXT statements allow you to execute a set of statements an indicated number of times. The variable specified in the FOR and (optionally) NEXT statements increases in value at each repetition of the loop. Its first value is exp1, subsequent values are determined by adding 1 (or i, if specified) to the previous value, and the final value of the variable is exp2. If the starting value is greater than the ending value in the FOR statement, the statements in the loop are not executed.

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

```
 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  362880
          10 FACTORIAL IS  3628800
         THE LOOP IS FINISHED AND I =  10
         READY
```

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. For example:

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

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

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

```
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 on the next page, to exit a FOR loop.

EXIT Statement

```
General form:

  EXIT n                Transfers control to statement n and
       |                terminates the current FOR/NEXT loop.
       |_statement
         number

Example:

  10 EXIT 75
```

The EXIT statement allows escape from a FOR/NEXT loop. It causes the specified statement to be executed next and terminates the current FOR/NEXT loop. Only the current FOR/NEXT loop is terminated; if it is nested in others, they will not be terminated. In this respect the statement functions differently than its counterpart in Extended Cassette BASIC. Here is a sample program using EXIT:


```
.
.
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 Statement

```
General form:

  ON exp EXIT n1, n2, ...        If the truncated value of exp
     |       |   |   |           is 1, exits to statement n1,
     |       |___|___|statement  if exp is 2, exits to state-
     |_numerical      numbers    ment n2, etc. Otherwise the
       expression                statement is ignored.

Examples:

   10 ON I EXIT 110,150
  100 ON A+B*C EXIT 300, 320, 130
```

The ON...EXIT statement lets you escape the current FOR/NEXT loop to a statement determined by the value of an expression. 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. If it does not, the ON...EXIT statement is ignored. Note that 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. For 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.

4.5. 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 expressions 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:

$\boxed{A - B}$ > 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)
   |            ^
   |            NOT
   |            *    /
   |            +    -
   |            >    >=    =    <>    <=    <
   V            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 an parentheses to change the order of evaluation. Expressions in parentheses are evaluated first. For example:

```
X^2 + 1 AND A > B OR C = D
                                First
                                Second
                                Third
                                Fourth
                                Fifth
```

```
X^2 + 1 AND (A > B OR C = D)
                                First
                                Second
                                Third
                                Fourth
                                Fifth
```


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

4.5.1. 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.

4.5.2. 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
```

4.5.3. Relational Operators

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

expression 1   relational operator   expression2

The result of a relational operation has a numerical value of 1 or Ø 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	Ø
A1 <= A2	true	1
X + Y/4 <> 7	true	1
X = Y	false	Ø

4.5.4. Logical Operators

The result of a logical operation has a numerical value of 1 or Ø, 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	Ø
C = B AND B = A	False	Ø
C > B OR B > A	True	1
C > B OR A = B	True	1
A > C OR A = C	False	Ø

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	Ø
NOT (A = B)	True	1
NOT C	False (C is true because it has a nonzero value.)	Ø

4.5.5. 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 by 1.
 40 NEXT I
 50 PRINT "OF 100 VALUES ";X;" WERE THREE'S"
100 DATA 1,5,4,6,7,8,9,9,2,3,4,5,3,2,6,7,8,9,3
110 DATA 4,6,7,4,6,8,2,3,8,4,6,9,6,0,4,0,3,1,3
  .
  .
  .
```

4.5.6. 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
Ø	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 (Ø).

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

IF A AND B THEN.....
: A and B each have a value of true (nonzero) of false (Ø). 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 (Ø). That value is then compared to C. (1 or Ø) > C is either true (1) or false (Ø).
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 (Ø). That value is then compared to C. (1 or Ø) = C is either true (1) or false (Ø).
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 Statement

```
General forms:                                  calculator (if no
                                                statement number is
                                                specified)
  IF exp THEN n             If the value of exp is true, execute
                            statement n next.

        n is a statement number in all of the
        forms shown here
  exp is a logical expression
  in all of the forms shown here

  IF exp THEN n1 ELSE n2 If the value of exp is true, execute
                            statement n1 next; otherwise execute
                            statement n2 next.
  IF exp THEN statement1 : statement2 : ...
                            If the value of exp is true then
                            execute the specified statement(s).
  IF exp THEN statement1 : statement2 : ...
         ELSE statement3 : statement4 : ...
                            If the value of exp is true then
                            execute the statement(s) following
                            THEN; otherwise execute the state-
                            ment(s) following ELSE.
                            Note:  The ELSE must appear on the
                            same line as the IF.
  IF exp THEN n ELSE statement1 : statement2 : ...
                            If the value of exp is true, execute
                            statement n next; otherwise execute
                            the statement(s) following ELSE.
  IF exp THEN statement1 : statement2 : ... ELSE n
                            If the value of exp is true then
                            execute the statement(s) following
                            THEN; otherwise execute statement n
                            next.

Examples:

  10 IF A > B THEN 250
 100 IF A=C AND NOT B THEN PRINT "ERROR":GO TO 350
 200 IF X1 OR Y2 THEN 750 ELSE 305
 300 IF NOT R THEN INPUT "R=",R ELSE 700
```

The IF statement evaluates a logical expression and then takes action based on its value. A true value causes the statement number or statement(s) following THEN to be executed next. If there is an ELSE clause, a false value for exp causes the statement number or statement(s) following ELSE to be executed next. Execution continues with the statement following the IF statement, provided control has not been transferred elsewhere.

In the example below, IF statements are used to create an automatic tax table:

```
 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 : GO TO 200
 40 IF I <= 5000 THEN T = 50 + .03*I : GO TO 200
 50 IF I <= 6500 THEN T = 95 + .04*I : GO TO 200
 60 IF I <= 9500 THEN T = 230 + .06*I : GO TO 200
 70 IF I <= 11000 THEN T = 320 + .07*I : GO TO 200
 80 IF I <= 12500 THEN T = 425 + .08*I : GO TO 200
 90 IF I <= 14000 THEN T = 545 + .09*I : GO TO 200
100 IF I <= 15500 THEN T = 680 + .1*I : GO TO 200
110 T = 830 + .11*I
200 PRINT "THE TAX IS $";T
```

SECTION 5

5. ADVANCED BASIC

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

*With subroutines and functions, you can define activities that will be perfomed when a simple call is made or when a function name is specified.
*By using string functions and statements, you can manipulate character data.
*With dimensioned variables, you can set aside storage to quickly and easily manipulate large volumes of data.
*Using the diskette storage and retrieval commands and statements, you can save data for later use.
*With the formatting capabilities of the PRINT statement, you can control the appearance of numeric output.
*Using time and space constraints in the INPUT statement, you can control the response to an INPUT prompt.
*Through cursor-controlling statements and functions, you can draw on the screen.
*Calling upon commands as statements in a program, you can set system characteristics, leave BASIC, and delete the program.
*With the error control statements, you can predetermine a course of action if an error should occur in a program.

5.1. 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 exection.

In Extended 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.

```
General form:

  GOSUB n                  Executes the subroutine beginning
        |                  at statement n.
        |_statement number

Example:

  10 GOSUB 270
```

The GOSUB statement causes immediate execution of the subroutine starting at the specified statement number. After the subroutine has been executed control returns to the statement following the GOSUB statement. For 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                                  ]  Sub-
230 P = P + R*P                                     ]  routine
240 NEXT N                                          ]
250 RETURN                                          ]
```

Calls to subroutines can be included within a subroutine. Extended BASIC allows any level of nested subroutines. Nested subroutines are executed in the order in which they are entered. For example:

```
    .
    .
100 GOSUB 200
110 PRINT A
    .
    .
200 FOR I = 1 TO R        ]  Execution of this subroutine is
210 IF I = R GOSUB 370    ]  interrupted when I=R.  After the
220 A = A + X^2           ]  subroutine at 370 is executed,
230 NEXT I                ]  statements 220 - 240 are executed
240 RETURN                ]  and control returns to statement 110
    .
    .
370 INPUT "J=",J          ]  This subroutine is executed before
    .                     ]  the execution of the subroutine at
    .                     ]  200 is complete.
430 RETURN                ]
```



```
General form:

  RETURN                   Transfers control to the statement
                           following the GOSUB or ON...GOSUB
                           statement that called the subroutine.

Example:

  100 RETURN
```

The RETURN statement causes the exit of a subroutine. 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.

A RETURN statement will terminate as many FOR/NEXT loops as necessary to return to the calling GOSUB statement. RETURN statements can be used at any desired exit point in a subroutine. For example:

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

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

ON...GOSUB Statement

General form:

```
  ON exp GOSUB n1, n2, ...      Executes the subroutine begin-
      |          |  |           ning with statement n1 if the
      |          |__|state-     value of exp is 1, executes the
      |__numerical   ment       subroutine beginning with
         expression  number     statement n2 if exp is 2, etc.
```

Examples:

```
 1Ø ON X GOSUB 12Ø, 15Ø
1ØØ ON S+A/B GOSUB 5ØØ
2ØØ ON N GOSUB 9Ø, 5ØØ, 1Ø
```

The ON...GOSUB evaluates, then truncates the expression (exp). If the value is 1, the subroutine starting at statement n1 is executed; if 2, the subroutine starting at statement n2 is executed; etc. If the truncated value of exp is less than 1 or greater than the number of statements specified, BASIC executes the next line. After the subroutine has been executed, control is transferred to the statement following the ON...GOSUB statement. For example:

```
  5 INPUT "ENTER TWO NUMBERS ",X,Y
 1Ø PRINT "DO YOU WANT TO ADD (1), SUBTRACT (2),"
 2Ø PRINT "MULTIPLY (3), OR DIVIDE (4) THE NUMBERS"
 3Ø INPUT I
 4Ø ON I GOSUB 1ØØ,2ØØ,3ØØ,4ØØ
 5Ø PRINT "THE ANSWER IS ";A
 6Ø END
1ØØ A = X+Y
11Ø RETURN
2ØØ A = X-Y
21Ø RETURN
3ØØ A = X*Y
31Ø RETURN
4ØØ A = X/Y
41Ø RETURN
```

5.2. 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. After execution, the function itself has a value. For example:

```
1Ø LET A = SQR(176) + B
```

SQR is the square root function and 176 is its argument. When statement 1Ø 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 the many functions supplied by Extended 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 unit.

5.2.1. General Mathematical Functions

General forms:

```
  ABS(exp)            The absolute value of exp.
  EXP(exp)            The constant e raised to the power exp.
  INT(exp)            The integer portion of exp (truncation)
  LOG(exp)            The natural logarithm of exp.
  LOG1Ø(exp)          The logarithm base 1Ø of exp.
  RND(exp)            A random number between Ø and 1.
                      exp may be Ø, -1, or n.
  SQR(exp)            The square root of exp (exp must be
                      positive).
  SGN(exp)            The sign of the value of exp; 1 if
       |              positive, -1 if negative, Ø if zero.
       |__exp is a numerical
          expression in all
          these functions.
```

Examples:

```
 1Ø LET X = EXP(X) - LOG(Y)
1ØØ PRINT "THE ANSWER IS "; INT(A*B)
2ØØ IF ABS(X^2-Y^2) > 1Ø THEN 25Ø
```

The use of all these functions in a program is straightforward except for the RND function. 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
Ø	Returns the next entry in the table
-1	Returns the first entry, and resets the table pointer to the first entry
n	Returns the entry following n

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

```
3Ø X = INT(RND(Ø)*1ØØ)
```

5.2.2. Trigonometric Functions

```
General forms:

  SIN(exp)          The sine of exp radians.
  COS(exp)          The cosine of exp radians.
  TAN(exp)          The tangent of exp radians.
  ATN(exp)          The arctangent of exp; the answer is in
                    radians.
       exp is a numerical expression
       in all these functions.

Examples:

  10 PRINT "THE SIN OF ";Y;" IS ";SIN(Y)
 100 LET R = SIN(A)^2 + COS(A)^2
 200 IF ATN(14.7) < 1 THEN 400
```

5.2.3. 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.

The rest of this unit describes in detail how to define and use functions of one or more lines.


```
General forms:

  DEF FNvar(var1, var2, ... ) = exp
                                    expression
                          variable
                                    Defines a one-line function
                                    that evaluates exp based on
                                    the values of var1, var2, etc.

  DEF FNvar(var1, var2, ... )
     .                              variable
     .
  RETURN exp                        Defines a multi-line function
     .                              that evaluates exp based on
     .      expression              the values of var1, var2, etc.
  FNEND

Examples:

  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
```

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. Examples are:

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

In multi-line function definitions, the value returned is the value of the expression on the same line as the RETURN statement. RETURN statements can be used to exit multi-line function definitions as desired. Each definition must end with a FNEND statement. For example:

```
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. For example:

```
   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 Function Call

General form:

```
FNvar(var1, var2, ... )      Evaluates a user-defined
                             function.
      |____|_____|_variable
```

Examples:

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

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


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

This 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.

5.3. CHARACTER STRINGS

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

5.3.1. 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 characters
"APT #"              is called the null string.
""
```

In Extended 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 the character's function is performed if it has a function. 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 funtion of control-J is line feed:

```
ALPHA
BETA
GAMMA
```

To print a single backslash, use this form: "\\". For a list of symbols and functions of control characters, see Section VII of the Sol Systems Manual.

5.3.2. 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$, S$, CØ$, Z2$

A string variable can contain one to ten characters unless it's maximum size has been declared as a value larger than 1Ø in a DIM statment.

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

```
 1Ø LET A$ = "MISSOURI"
1ØØ S$ = A$
2ØØ R$ = "BOX #", T$ = "Address"
```

5.3.3. String Expressions

String expressions can include string constants, string variables, and any of the string functions described later in this unit. 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"
 1Ø INPUT S$
 2Ø IF S$ = "DATA" THEN 7Ø
 3Ø INPUT X$, Y$, Z$
 4Ø PRINT "THE LAST VALUE READ WAS ";Z$
 6Ø END
 7Ø READ X$, Y$, Z$
 8Ø GO TO 4Ø
1ØØ 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	"Ø73234"
"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:

```
 1Ø INPUT "WHAT RANGE OF NAMES DO YOU WANT? ",A$,Z$
 2Ø FOR I = 1 TO 35
 3Ø READ S$
 4Ø IF S$ < A$ THEN 6Ø          Notice that 4Ø and 5Ø cannot
 5Ø IF S$ <= Z$ THEN PRINT S$   be combined because logical
 6Ø NEXT I                      operators are not allowed.
1ØØ "Smith, J.B.", "Ronson, C.H.", "Peale J.P.", "Adams, J.Q."
  .
  .
```

String DIM Statement

General form:

```
DIM var(n)              Specifies the maximum size of a
     |  |__integer      string that can be contained in var.
     |__string          n is the maximum number of characters
```

Examples:

```
 1Ø DIM S$(2Ø)
1ØØ DIM A$(72),B$(55),C$(15)
```

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.

The actual length of the variable at any time is determined by the size of the string currently assigned to it. If a string value with more characters than allowed by the DIM statement is assigned to a variable, the rightmost characters are truncated. For 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 Statement

```
General form:

  SEARCH expl, exp2, var          Searches exp2 for the first
                                  occurance of expl and sets var
 string ___|_____|     |          to the number of the position
 expression  numeri-___|          at which it is found or 0 if
             cal variable         it is not found.

Examples:

  10 SEARCH "CAT",M$,N
 100 SEARCH A$, R$, I
```

The SEARCH statement evaluates expl and looks for that string as all or part of the value of exp2. If it is found, its location is given by var. For 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.

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

FILL Statement

```
General form:

   FILL string, string expression
          |
          |_string variable or substring function

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

  10 FILL A$, CHR(10)
 125 FILL X9$(10, 200), " "
```

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. If the string expression yields a string containing more than one character, only the first is used. The expression must yield at least one character. For example:

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

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

The FILL statement may also be used as a command.

5.3.4. String Functions

The functions described in this unit 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 USASCII codes. The VAL and STR functions convert numbers to strings and vice versa. Finally, the ERR(0) function gives the last error message to appear.

Substring Function

```
General forms:

  var(exp1, exp2)                  Extracts characters exp1 through
                                   exp2 of the string contained in
           subscript               var.
     string  expressions
     variable

  var(exp1)                        Extracts characters exp1 through
                                   the last character of var.

Examples:

  10 LET S$ = X$(2,4)
 100 LET A$(1,3) = "NON"
 200 INPUT X$(Q-7)
 300 LET I$ = L$ + M$(1,5)
```

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

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)
```

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
```

A string may be treated as if it were like an array of subscripted strings, as shown in the example below:


```
 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 Function

```
General form:

  LEN(var)                        Finds the number of characters in
                                  the string currently contained in
       string                     var.
       variable

Examples:

  10 PRINT LEN(S$)
 100 IF LEN(X1$) > 10 THEN 75
```

The LEN function supplies the current length of the specified string. The current length is the number of characters assigned to the string, not the dimension of the string. For 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 and CHR Functions

```
General forms:

  ASC(exp)                 Supplies the USASCII code for the
      |                    first character in the string ex-
      |__string ex-        pression exp.
         pression

  CHR(exp)                 Supplies the character whose
       |                   USASCII code is given by exp.
       |__numerical
          expression

Examples:

  10 LET I = ASC("%")
 100 LET I$ = CHR(70)
 200 IF ASC(X$) = 65 THEN PRINT "A"
```

The ASC and CHR functions perform conversions between characters and their USASCII equivalents. ASC returns the USASCII code for a character whose value is given by a string expression and CHR returns a character whose USASCII code is given by the value of a numerical expression. A table of USASCII codes is presented in Appendix 4.

VAL and STR Functions

```
General forms:

  VAL(exp)               Supplies the numerical value of the
      |                  string whose value is given by exp.
      |__string expression
         that can be converted
         to a decimal number

  STR(exp)               Supplies the string value of the
       |                 number whose value is given by exp.
       |__numerical
          expression

Examples:

  10 X = I * VAL(J$)
 100 PRINT VAL(A$)
 200 IF VAL(A$) = 13.2 THEN END
 300 X$ = A$ + STR(I)
```

The VAL and STR functions perform conversions between decimal numbers and strings that can be converted to numbers. For 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".

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. For example:

```
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.
```

The VAL function evaluates the string argument as a number. Evaluation stops on the first character which is not legal in an arithmetic constant as described in Section 2.3.1. For example:

```
User:    PRINT VAL("$99,999,999")   This statement will result
                                    an IN error due to the $.
         PRINT VAL("99,999,999")    Evaluation will stop at the
                                    first comma:
BASIC:   99
```

ERR(0) Function

```
General Form:

   ERR(0)                     Returns a string consisting of
                              the last error message.

Example:

   10 A$ = ERR(0)
   20 IF A$(1,2)= "NI" THEN PRINT " DELETED FUNCTION USED!"
```

The ERR(0) function returns a USASCII string constant containing the last error message which appeared on the user's terminal. If the ERRSET statement kept the error message from appearing, then the string contains the error message which would have appeared. 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. The first two characters in the error message are sufficient to identify which error has occurred, and may be used in comparisons. In the example above, the error message string is stored in string variable A$, then the first two characters of of A$ are compared with "NI" (Not Implemented). If there is a match, then a message appears on the terminal telling the user he used a version of BASIC which was initialized without a funtion which his program used. Similar statements can be used to branch to special routines when certain errors occur.

5.4. 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.


```
General forms:

  DIM var(expl,exp2,...)         Defines an array with one or
       |     |    |              more dimensions.  The size of
       |     |____|numerical     the array is (expl*exp2*...)
numer-_|          expression     elements.
cal variable

  DIM var1(expl,exp2,...),var2(exp3,exp4,...),...

                                 Defines one or more arrays.
                                 String dimension expressions can
                                 be included as well.
Examples:

  10 DIM A(100)
 100 DIM A1(4,5),I(L,M-L),J(2,3,10)
 200 DIM X(100),S$(72),Y(I,J,K)
```

The DIM statement allots space for an array with the specified variable name. The number of dimensions in the array equals the number of expressions in parentheses following the variable name. The number of elements in the array is the product of the expressions.

Elements of an array are referred to as follows:

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

For 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 I
 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).....etc.

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(Ø) function, which prints the number of bytes of space left for program and variables. For example:

```
PRINT FREE(Ø) <CR>
296Ø
```

5.5. USING DISKETTE FILES FOR DATA STORAGE

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

A data file is a collection of data items stored on diskette 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 block structure is of no interest to the BASIC programmer, except that a file divided into large blocks can be read or written faster than a file of small blocks. For this reason, BASIC allows the programmer to specify a file's block size. 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 dissappears if the memory containing it is overwritten or if the system 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. The PTDOS CONFIGR command can be used to change this limit. Each open file requires a certain amount of PTDOS buffer memory, which must be addressed for this use. With the correct configuration and adequate memory, up to 16 files may be open at once. 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 5.6, 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 ommited 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 PTDOS, and PTDOS may be used to create files for use in BASIC. See Section 3.4.2 for a discussion of this subject, and for information about file names for use in the statements below.

Serial Access FILE Statement

```
General form:

  FILE #n; name ,{access},{ag},,{bs}
           |    |         |       |    |  |_block size expression
           |    |         |       |    |
           |    |         |       |    |_(no record size expression)
           |    |         |       |
           |    |         |       |_access granted variable
           |    |         |
           |    |         |_access requested (1, 2, or 3)
           |    |
           |    |_string or string expr. containing file name
           |
           |_file reference number

                      Opens or creates a file of the given
                      name, assigning a file reference number
                      for later use, and requesting read,
                      write, or read/write access.

Examples:

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

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

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

The FILE statement opens the file (makes it acessible to BASIC), assigns a file reference number for use in the above file access statements, and requests access for reading, printing, or both. If the named file does not already exist, this statement will create it, if the access requested was 2 or 3. 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.

Here is a description of the arguments of the FILE statement:

n An expression which equates to a file reference number to be assigned.

name A string literal ("thisfile/1" 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.
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.)

bs An expression which specifies the block size to be used by the file. If no block size is specified, and this statement creates the file, a default block size of 256 bytes will be used. bs must be in the range of 1 to 4095 bytes.

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 5.5). Any open file may be closed with the CLOSE statement. Any termination of the run of a program closes all open files.

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

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. (See Section 5.5.)

Random Access FILE Statement

```
General form:

  FILE #n; name, {access},{ag},{rs},{bs}
                                        |_block size expression
   file_|      |       |        |    |
   reference   |       |        |    |_record size expression
   number      |       |        |
               |       |        |_access granted variable
               |       |
               |       |_access requested (1, 2, or 3)
               |
               |_string or string variable containing file name

                    Opens or creates a random access file of
                    the given name, assigning a file reference
                    number for later use, and requesting read,
                    write, or read/write access.

Example:

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

This form of the FILE statement is used to open or create a random access file as opposed to a serial access file. The syntax is similar to the Serial Access FILE Statement above, except that an expression is included which specifies a record size. The EOF function is set to 1, as with the Serial Access FILE statement.

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.

The rs (record size) expression specifies how many characters (bytes) can be stored in each record. BASIC actually uses one extra character for each item (collection of characters) in a record. For example, if 3 items, each containing 30 characters are printed in a record, BASIC will use 93 characters of the record. If no record size is specified, the statement becomes a Serial Access FILE statement, described above.

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.

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

Serial File PRINT Statement

```
General form:

  PRINT #n; ele1 {,ele2} {,ele3}...
           |   |_____|_______|_____|_expressions to print
           |                          or format elements
           |_file reference number

              The values of the expressions are printed
              sequentially on the referenced file, starting
              at the current file cursor position

Examples:

   10 PRINT #Q; "the answer is", 2 + SQR(N)
  100 PRINT #3; 100, 100 + X, 100 + X * 100
  230 PRINT #45; %C10F3, X, Y, A, %D, Q
```

A previous FILE statement must have already opened the file; n is the file reference number that was assigned by that FILE statement. ele1, ele2, etc., are general expressions which result in numerical or string values to be printed on the file. ele1, ele2, etc., may also be format elements. (See Section 5.6) 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. 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 last data item printed. The EOF function for the file is set to 3 (last was PRINT).

For 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 data
         ?END <CR>       is written on the diskette file.
BASIC:   READY
```

Serial File PRINT Statement with Spacing

```
General form:

  PRINT #n, {re}, d; ele1 {,ele2}...
                                     expression(s) to print

                      expression for file cursor displacement

                 expression for record number-if present, must
                 equate to zero in this form of PRINT

            file reference number

                 The file cursor of the referenced file is
                 displaced by d, and the values of ele1, ele2,
                 etc., are sequentially printed on the file

Examples:

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

Here is a description of the arguments of the Serial File PRINT with spacing statement:

- **n** — The file reference number assigned when the file was previously opened in a FILE statement.
- **re** — An optional expression for record size, which, 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.
- **ele1, 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 5.6. One or more expressions may be present.

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 alreay opened by a FILE statement. If this type of PRINT statement is the first statement executed on the file, the file cursor will be at the end-of-file. 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. Overprinting old items with larger or smaller items may damage the file structure. For this reason, numerical formatting as described in Section 5.6 is recommended, to ensure uniform numerical fields for all items. If strings are printed, some "padding" may be needed to keep a new string the same size as the last item in that position. You must take care to maintain the file structure.

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

Random File PRINT Statement

```
General Form:

  PRINT #n, record {,d}; ele1 {,ele2}...
                                          expression(s)
   file                                   to print
   reference
   number                expresssion for cursor displacement
                         if present, must equate to zero

                 expression which equate to a record number

                 The file cursor of the referenced random
                 access file is positioned to the specified
                 record, and the values of ele1, ele2, etc.,
                 are printed on the file.

Example:

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

Here is a description of the arguments of the Random File PRINT statement:

- **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 nonexistant 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.

elel, ele2, etc. — General expressions 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.

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. The EOF function is set to 35.

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.

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.

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.

Serial File READ Statement

General form:

```
   READ #n; varl {,var2}... {:statementl: statement2: ...}
        |     |_________|___variable names which
        |                   receive the read values
        |_file reference number
```

Items from the referenced file are read, beginning at the current file cursor position, and assigned to varl, var2, etc. The optional statement(s) is executed only if an end of file is encountered.

Examples:

```
 10 READ #2; X, Y, Z, A, B
100 READ #1; S(I) : PRINT "EOF": EXIT 200
```

A FILE statement must have previously opened the file with type 1 or type 3 access. The READ statement reads items, starting at the current file cursor position, and assigns them as the values of the variables. One or more variables may be present. The number of values read is equal to the number of variables present in the statement. If this is the first statement which accesses the referenced file after the FILE statement which opened it, reading will begin at first element of the file. Otherwise, reading will begin from where the file cursor was left by the last access. The statement itself leaves the file cursor postionned just after the last data item read. The EOF function is set to 2.

For 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 the
 50 NEXT I                           file is reached before
                                     500 values are read is
200 PRINT I;" VALUES READ FROM VAL"  statement 200 executed.
```

Serial File READ with Spacing

General form:

```
READ #n, {rn,} d; varl {,var2}...{statementl: statement2...}
     |     |    |    |_______|variable names
file_|     |    |             which receive the values read
reference  |    |
number     |    |_ expression for file cursor displacement
           |
           |_expression for record number
             if present, must equate to zero
```

The file cursor of the referenced file is displaced by d, and items from the file are read and assigned to varl, var2, etc. The optional statement(s) is executed only if an end of file is encountered.

Examples:

```
 10 READ #2,, 54; X(1), X(2), X(3), X(4)
120 READ #X, 0, -10; X$, Z
```

Here is a description of the arguments:

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.

varl, 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.

This statement is the same as Serial File READ (without spacing) except that the file cursor may be moved before reading. A FILE statement must have previously opened the file with type 1 or type 3 access. The file cursor is displaced by d items and enough items are read to fill the variables given. 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. 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. The EOF function is set to 18.

Random File READ Statement

```
General form:
                     Ø
READ #n, rn {,d}; var1 {,var2}...{statement1: statement2...}
          |     |        |        |      |  variable name(s)
  file____|     |        |        |______|__which receive values
  reference     |        |
  number        |        |__expression for cursor displacement
                |           if present, must equate to zero
                |
                |__expression which equates to a record number

                   The file cursor of the referenced random
                   access file is positioned to the specified
                   record, and items are read into the variables.
                   The optional statement list is executed only
                   if an end of file is encountered.
Examples:

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

Here is a description of the arguments:

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 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 expression in this form of the READ statement, d must equate to zero, if present.

var1, — 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.

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

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.

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

REWIND Statement

```
General form:

  REWIND #n1,#n2,...          Rewinds the specified files.
          |___|_numerical expression which equates to a
                file reference number as defined in a
                previous FILE statement.

Examples:

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

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

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. This means that all items following the file cursor position at the time of the REWIND will be erased and the space that they occupied will be made available for other files.

CLOSE Statement

```
General form:

  CLOSE #n1,#n2, ...        Closes the specified files.
         |___|_numerical expression which equates to a file
               reference number as defined in a previous
               FILE statement.

Examples:

  10 CLOSE #3
 100 CLOSE #I-J,#I+J,#1
```

The CLOSE statement makes the specified files unavailable for reading or writing. They cannot be accessed again until another FILE statement requests access. For 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.
```


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. This means that all of the data items after the file cursor are "erased", and the space they occupied will be made available for other files.

File PURGE Statement

```
General form:

  PURGE string {,var}
         |       |_variable to accept error code
         |
         |_a string expression which equates to a file name

           The named file is KILLed.  If PTDOS gives an
           error code, it is placed in the variable.
```

The file whose name is defined by the string expression is KILLed as described in Section 1.5 of the PTDOS manual. If PTDOS returns an error code, its value is placed in the variable.

SET XI Statement

```
General form:

   SET XI = n
            |_the file reference number assigned by a
              prior FILE statement

              Loads the index block of the referenced file
              into memory

Examples:

   10 SET XI = 2
  120 SET XI = ((2 + F9 - SGN (FNA (E2))) + 1
```

The index block (See PTDOS Manual for explanation) of the referenced file is loaded into memory. This allows PTDOS to access the index block faster than from diskette, and therefore speeds execution of Random File READ and PRINT statements. THIS STATEMENT CANNOT BE USED AS A COMMAND.

SET FB Command

```
General form:

   SET FB = expression

                  If the expression is zero, static file
                  buffering will be used.  If non-zero,
                  dynamic file buffering will be used.

Examples:

   SET FB = Ø
   SET FB = SGN (2ØØ - RND (Ø) * 4ØØ / X)
```

See the PTDOS manual for a discussion of buffering. This command can be used as a statement.

SET OF Command

```
General forms:

   SET OF = string
               |___string expression for file name

                  Sets the output file.  Instead of appearing
                  on the terminal or normal output device, all
                  output will be sent to the named file.

   SET OF = #exp
                |_numeric expression for file number

                  Sets the output file to the open PTDOS file
                  whose number is the value of exp.
Examples:

   SET OF= "SLAT12"
   A$ = "STAT12" :  SET OF = A$
   SET OF = #4
   SET OF = #(N + 3)
```

This statement diverts all output from BASIC, including everything which would normally appear on the video display to the named file. If the named file is a device file (See PTDOS Manual), the output will appear on the related device. In the second form, the file exp must be already open. If exp equates to zero, BASIC's video display driver will be used for output. The # before exp must be contained in the statement, and not supplied by the expression itself. For example:

```
     SET OF = #(N + 1)         is correct.
     A$ ="#2":  SET OF = A$    is incorrect.
```

This command may also be used as a statement.

EOF Function

```
General form:

  EOF(file number)      Supplies the status of the specified
                        file.
          |__numerical expression for a file reference
             number assigned in a prior FILE statement.

Examples:

  1Ø PRINT EOF(2)
 1ØØ IF EOF(I) = 4 THEN 15Ø
```

Every diskette data file which has been opened with a FILE statement has an associated End-Of-File (EOF) function. The EOF function supplies the current status of the specified file as follows:

VALUE OF EOF	MEANING
Ø	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

In BASIC you can control the relays in the Sol Terminal Computer or on a CUTS module which are normally used to control the drive motors in cassette recorders. These relays may be used to start and stop audio tapes, or for controlling other equipment. The TUOFF and TUON commands turn the relays off and on. Their forms are:

```
     TUOFF          Turn off both relays
     TUON exp       Turn on relay exp

             |__numerical expression
```

The TUOFF and TUONN commands can also be used as statements in a program. Their forms as statements is given below.

TUOFF and TUON Statements

```
General forms:

  TUOFF                    Turns off both tape motor control
                           units.
  TUON exp                 Turns on tape motor control unit
       |__numerical        exp.  exp must evaluate as 1 or 2.
          expression

Examples:

  10 TUON 1
 100 TUOFF
 200 TUON K-1
```

The TUOFF and TUON statements let you turn on and off the cassette recorder motors. They actually control two reed relays, which have isolated low-power contacts appearing at J8 and J9 of the Sol Terminal Computer, or J1 and J2 of the CUTS module. The closure of these contacts under program control can be used for general purpose control applications, provided that that extra power handling circuitry using relays or semiconductors is used when necessary. The reed relay contacts are SPST and will handle .5 Amp, 100 VDC, with a maximum of 10 watts for a resistive load.

For example:

```
10 DIM A(100)
20 TUON 1
30 FOR I = 1 TO 100
40 READ A(I)
50 IF A(I) = 0 THEN TUOFF : END
60 NEXT I
70 DATA 1,2,3,4,5,6,7,8,9,0
```

5.6 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 4.1 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.

Formatted PRINT Statement

```
General Form:

   PRINT exp, exp,...format element, exp, exp,...
             |____|__expressions not      |____|__expressions
                     affected by the              affected by the
                     format element               format element

 Or more generally:

   PRINT ele, ele, ele; ele....
           |    |    |  |__commas or semicolons may separate
           |    |    |     elements
           |____|____|__elements consisting of:
                         numeric expressions,
                         string expressions, format elements,
                         or TAB functions.

 Examples:

   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)
```

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. The same PRINT statement can also contain additional format elements which control additional expressions which follow them. The format element produces no printed results of its own; it controls the form in which subsequent numbers are printed. A format element controls only the expressions following in the same PRINT statement, up to the next format element, if any. Using a 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.

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

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.ØØ
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.

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 can not 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 Ø to n. Here is an example of a number printed in 1ØE3 format: 1.234E-123. If the specified field can not 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.

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. 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 exponential format, blanks are added between the number and its exponent.) Extra field width can be used to create columns of printed output spaced at desired intervals. 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 in between; that depends on whether the number fills its field.

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. 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. 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.

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. 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, 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.

Here are some examples of useful format elements:

MONATARY FORM:

```
%$C11F2
 ||  |
 ||  |_floating point form, eleven characters in width,
 ||    with two of those characters to the right of the
 ||    decimal point
 ||
 ||____commas will separate every three digits
 |
 |_____dollar signs will be printed in front of each number
```

Examples of output:

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

SCIENTIFIC FORM:

```
%Z15E7
 |  |
 |  |_Exponential notation, fifteen characters in width,
 |    with seven of those characters to the right of the
 |    decimal point
 |
 |____Trailing zeros will be suppressed
```

Examples of output:

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

The sample program segment below illustrates how format elements can interact:

```
10 PRINT %#$C11F2;    This statement sets the monatary 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 default
                      format in statement 10.

30 PRINT B9; %+26F8; P, I; %D; P/I
                      B9 will be printed according to state-
                      ment 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.
```


5.7. 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.

Controlled INPUT Statement

```
General forms:

  INPUT{,} (#chars,t) var1, var2, ...    Enters values from the
                                          terminal and assigns
 numerical ___|_______|   |____|____|     them to var1, var2,
 expressions                   |          etc.; however, only
                          variable(s)     #chars characters can
                                          typed by the user and
                                          the user has t tenths
                                          of a second to
                                          respond.

  INPUT{,} (#chars,t) " message", var1, var2, ...
                           |              Same as above, but a
                         string           message is printed as
                        constant          a prompt before timing
                        in quotes         begins.
Examples:

  10 INPUT (3,10) X
 100 INPUT, (20, 0) N$, A$
 200 INPUT (0 ,100) A, B, C
 300 INPUT, (10,300) "WHAT IS THE DATE?" ,D$
```

The controlled INPUT statement lets you specify how many characters can be entered and how much time is allowed for responce. As soon as #chars characters have been typed, BASIC generates a carriage return and accepts no more characters. If the user takes more than t tenths of a second to respond, BASIC assumes a carriage return was typed. 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.

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. For 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.

5.8. 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, and SYST(0) gives the number of the last PTDOS error.

ERRSET and ERRCLR Statements

```
General forms:

  ERRSET n           Determines that statement n will be
         |           executed if any error is detected by
         |_statement BASIC or PTDOS.
           number

  ERRCLR             Cancels the last ERRSET statement.

Examples:

  10 ERRSET 75
 100 ERRCLR
```

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, or a PTDOS error, as listed in the PTDOS manual. 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.) Also, the transfer to statement n clears all current FOR/NEXT loops, GOSUBs, and user-defined function calls (as if a CLEAR statement were executed.)

The ERRCLR statement cancels the most recent ERRSET statement. 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. However, if the ERRSET statement is executed again, it will again set the error trap statement n, as if the ERRSET were encountered for the first time.

ON...ERRSET Statement

```
General form:

  ON exp ERRSET n1, n2, ...      Establishes which statement
      |         |  |             will be executed in the event
      |_numeri- |__|__state-     of an error. If exp is 1,
    cal expression    ment       statement n1 is selected, if
                      number     exp is 2, statement n2, etc.

Examples:

  10 ON I ERRSET 105,250,400
 100 ON A-J ERRSET 50, 300
```

The ON...ERRSET allows you to conditionally determine which statement will be executed if an error occurs. Once an error has occurred, the ON...ERRSET statement is no longer in effect, as if an ERRCLR statement had been executed.

ERR(0) Function

```
General Form:

   ERR(0)                       Returns a string consisting of
                                the last error message from
                                BASIC.
Example:

   10 A$ = ERR(0)
   20 IF A$1,2 = "NI" THEN PRINT "DELETED FUNCTION USED"
```

The ERR(0) function returns a USASCII string constant containing the last error message which appeared on the user's terminal. If the ERRSET statement kept the error message from appearing, then the string contains the error message which would have appeared. 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. The first two characters in the error message are sufficient to identify which error has occurred, and may be used in comparisons. In the example above, the error message string is stored in string variable A$, then the first two characters of of A$ are compared with "NI" (not implemented). If there is a match, then a message appears on the terminal. Similar statements can be used to branch to special routines when certain errors occur. If the error detected was a PTDOS error, ERR(0) will return "FS ERROR". The SYST(0) function below can then be used to determine which PTDOS error occurred.

SYST Function

```
General form:

  SYST (exp)
        └numeric expression that evaluates to 0

        Returns the number of the last PTDOS error.

Example:

  10 LET X = SYST(0)
  20 PRINT "The last PTDOS error was",SYST(0)
```

The SYST function returns the number of the last PTDOS error that appeared on the user's terminal. If an ERRSET statement kept the error message from appearing, then SYST(0) returns the number of the message that would have appeared. SYST(0) can be used in conjunction with the ERR(0) function:

```
10 A$ = ERR(0)
20 IF A$(1,2) = "FS" THEN A$ = STR(SYST(0))
30 IF A$ = "4" THEN PRINT "PROTECTED FILE"
```

Note that the PTDOS errors, as listed in Section 6 of the first edition of the PTDOS manual, are given as hexadecimal values. SYST(0) returns their decimal equivalents.

5.9. COMMANDS CAN BE STATEMENTS AND STATEMENTS COMMANDS

There are a number of commands that can be included in programs as statements. You have already encountered two: the TUON and TUOFF commands. 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. Section 2.5, 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.

5.9.1. 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 5.5. Other SET commands are:

SET DS = exp1 {,exp2} — Sets the video display speed to exp. The larger the value of exp, the slower the display speed. The initial value is 0. If exp2 is present, it determines the availability of the video display speed and spacebar-stop controls during output:

exp2	result
0	normal controls
1	no space bar stopping
2	no speed control
3	neither

SET LL = exp — Sets the output 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.

numeric expressions

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.

Examples:

```
User:  10 SET LL = 10 <CR>
       20 PRINT "THE LINE IS TOO LONG" <CR>
       RUN <CR>
BASIC: THE LINE I
       S TOO LONG
```

5.9.2. BYE and SCRATCH Commands

The BYE and SCRATCH commands can be used as 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.

5.10. 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 Statement

General form:

```
CURSOR {expl}{,exp2}{,exp3}
```

numerical expressions

Moves the cursor to line expl and and character exp2. If either is ommitted, its value from the last CURSOR statement is used. Expl can be any number 1 through 16, and expl can be any number 1 through 64. If exp3 is present, the USASCII character whose value is exp3 will be placed at the current cursor position.

Examples:

```
 10 CURSOR I,J,42
100 CURSOR FNA(L)
200 CURSOR ,X*Y
300 CURSOR ,,45
```

You can use the CURSOR statement to position the cursor and then use exp3 or a PRINT statement to display a character in that position. You can also print any of the control characters which has an effect on the screen such as \K, which clears The screen. These special characters are described in Section 4.1. The last example prints a hypen at a position determined by last executed CURSOR statement, since both expl and exp2 are are absent, and 45 is the decimal USASCII code for hypen. Appendix 4 contains a table of USASCII codes.

For 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
```


POS(0) Function

General Form:

POS(0)	Returns a number between 0 and 131, representing the current horizontal postion of the cursor or print head.

Example:

```
10 IF (63 - POS(0)) < LEN(A$) THEN PRINT
```

In Extended 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. 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. The value of the POS(0) function is a number between 0 and 131 representing the current position of the cursor (or print head). 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.

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. In the example above the number of characters remaining on the line (63 - POS(0)) is compared with a string A$ which will be printed. 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.

SECTION 6

6. 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 Ø and 255" may be interpreted to mean "any expression allowed in BASIC, which, when evaluated, yields a decimal value between Ø and 255."

6.1 WRITING TO A PORT OR MEMORY LOCATION

POKE and OUT Statements

```
General forms:

  POKE exp1, exp2      The value exp2 is stored in memory
        |     |        location exp1.
        |     |
        |     |__numerical expression between 0 and 255
        |
        |__numerical expression from 0 to 65535

  OUT exp1, exp2       The value exp2 is sent to I/O port exp2
        |     |__numerical expressions between 0 and 255
        |
        |__numerical expression between 0 and 255

Examples:

  10 POKE 4095, 11
 100 OUT 248, 0
```

The POKE and OUT statements place a value between 0 and 255 in a specified memory address or I/O port. Since the 8080 microprocessor can address 65,536 memory locations, and has 256 ports, these values are set as limits to the value of exp1. The value of exp2 is converted to a one-byte binary value.

PEEK and INP Functions

```
General forms:

   PEEK(exp)        Supplies the numerical value contained
        |           in memory location exp
        |
        |__numerical expression between 0 and 65535

    INP(exp)        Supplies the numerical value contained
        |           in I/O port exp
        |
        |__numerical expression between 0 and 255

Examples:

   10 X = PEEK(4095)
  100 Y = INP(249)
```


The PEEK and INP functions return values equal to the contents of memory location or I/O port exp. Since the 8080 processor can address 65,536 memory locations, and has 256 I/O ports, these values are set as limits to the value of exp. One byte is retrieved and its value interpreted as a number between 0 and 255.

Image File LOAD Statement

```
General form:

   LOAD string {, var}
          |
          |__a string expression which equates to a PTDOS
             file name

                Loads the named PTDOS image file into
                memory, and places its starting address
                in var, if present.
Example:

   100 LOAD X$, Y
    35 LOAD "GUN"
```

The LOAD statement loads a PTDOS image file. If var is present, the file's starting address is placed in it. The image file may not be loaded below the "first protected memory address" set upon initialization. The first protected address may be changed with the PTDOS CONFIGR command, or the BASIC SET ML command. This statement may be used as a command. However, in a command, "string" must be the actual file name and not a string. The CALL function (below) may be used to execute the loaded image, with the value of var used for exp1.

CALL Function

```
General form:

   CALL(exp1{, exp2})     Calls a routine at address exp1,
         |      |         passing optional exp2 in registers
         |      |         D and E, and optionally returning
         |      |         a value in H and L registers.
         |      |
         |      |__numerical expression between 0 and 65535
         |
         |__numerical expression between 0 and 65535

Examples:

    10 X = CALL(34579)
   100 PRINT CALL(18026, 59)
```

The CALL function invokes a machine language program that begins at address exp1. 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. 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. The routine may place a value in the H and L registers to become the value of the CALL function. Since H and L consist of 16 bits, the value returned will consist of a positive integer between 0 and 65535.

WAIT Statement

```
General form:

   WAIT exp1, exp2, exp3    Wait until the the value in port
            |      |    |   exp1 ANDed with exp2, is equal to
            |      |    |   to exp3
            |      |____|_numerical expressions, 0 to 255
            |
            |_numerical expression for port, 0 to 255

Example:

   WAIT 248, 128, 128
```

When a WAIT statement is executed, program execution pauses until a certain value is present in I/O port exp1. To determine this value, exp2, exp3, and the value in port exp1 are converted to one-byte binary values. Each bit in the selected port is "ANDed" with the corresponding bit of exp2. If the result is equal to exp3, program execution continues at the next statement. If the result is not equal to exp3, the program continues to wait for the specified value. Depressing the MODE SELECT key will escape from a WAIT statement.

Exp2 and the logical AND operation provide a way to mask at the selected port bits which are not of current interest. Assume for example that you want a program to wait until bit 7 at port F8 (hexadecimal) becomes a 1. (In a Sol, port F8 contains the status of the serial communications channel, and if bit 7 is 1, it means that the UART transmit buffer is empty. The program is going to transmit a character out the serial communications channel, but we need to wait until the UART is empty before placing a new character in the port.*)

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,... Next, create an eight bit binary mask, with only the bit of interest, bit 7, set to 1: 10000000. 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. The decimal value for 10000000 binary is 128, so the WAIT statement now consists of WAIT 248, 128,... The value from the port is ANDed with the mask and compared

for equivalence with exp3. Since the mask 128 or 10000000 sets the last seven bits of the incoming value from the port to zero, the last seven bits of exp3 must also be zero to achieve a match. 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. Thus bit 7 of exp3 should be 1, and the entire byte will be 10000000. Converted to decimal, this value is 128. The complete statement is WAIT 248, 128, 128.

* WAIT cannot be used to monitor the keyboard status port of a Sol Terminal Computer.

7. 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. Extended 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				
three rows	3.1	4.6	7.0	3.1	0.0
	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 statement. For example:

```
10 DIM A(10, 2), B9(A, 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 independant 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. In the statement:

```
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 to dimension a matrix is given by the following expression:

```
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.

```
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 which, 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. For example, the statement

```
10 MAT Z = X + Y
```

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":

```
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.

7.1. 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.

7.2. 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. For 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.

7.3. 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 below:

Scalar Operations

```
General Form:

    MAT mvar1 =  mvar2 op (expr)
                       |   |
                       |   |_any expression
                       |
                       |_arithmetic operator (+ - * /)

Examples:

    10 MAT A = B * (2.3356)
    20 MAT C = D / (2.35 * C(I, J) + SIN(X))
    30 MAT E = E + (1)
```

A statement of this form performs the same scalar operation on each element of a matrix. mvar1 and mvar2 must have identical dimentions. The parentheses around expr are required. Matrix elements such as A(5,4) may appear in expr, but not entire matrices. If mvar1 and mvar2 are the same matrix, as in the last example, the resulting new elements will be placed in the old matrix.

7.4. 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
                  └arithmetic operator (+ - *)
```

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. mvar3 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.

7.5. MATRIX FUNCTIONS

Two matrix functions may be used to place the inverse or transpose of a matrix into another matrix.

Inverse and Transpose Functions

General Forms:

MAT mvar1 = TRN (mvar2)	Places the transpose of mvar2 into mvar1. mvar1 and mvar2 must have opposite dimensions.
MAT mvar1 = INV (mvar2)	Places the inverse of mvar2 into mvar1

Examples:

```
10 MAT A = TRN(B)
20 MAT C = INV(D9)
```

mvar1 and mvar2 must not be the same matrix. In both functions, mvar1 and mvar2 must have equal dimensions. No check is made to insure that mvar1 is not the same matrix as mvar2. If they are the same, unpredictable results will occur. As with all functions, the argument must be within parentheses.

7.6 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. For 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 orignal matrix are reasigned 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 redimention 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.

APPENDIX 1

Extended BASIC Command and Statement Summary and Index

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	Page
APPEND file --	Reads a program stored on a diskette file and appends it to the current program.	3-13
BYE -	Leaves BASIC and returns to PTDOS. S	2-3
CAT {/unit}{type}	Displays a catalog of BASIC program or diskette data files, from the specified disk drive unit, of type T, C, S,or R.	3-14
CLEAR ---	Erases all variable definitions. S	3-9
CONT --	Continues execution of a program stopped with the MODE key or by a STOP statement.	3-8
DEL	Deletes all statements.	3-4
DEL n	Deletes statement n.	3-4
DEL nl, n2	Deletes statements nl through n2.	3-4
DEL nl,	Deletes statements nl through the last statement.	3-4
DEL ,n2	Deletes the first statement through statement n2. Note space before comma.	3-4
EDIT n --	Allows the edit of statement n.	3-6
GET file --	Reads a diskette file program of type C or T for execution later.	3-12
KILL file -	Kills the named program file.	3-14
LIST --	Lists the entire program.	3-3
LIST n --	Lists statement n.	3-3

LIST n1, n2 --	Lists statements n1 through n2.	3-3
LIST n1, --	Lists statements n1 through the last statement.	3-3
LIST ,n2 --	Lists the first statement through statement n2.	3-3
REN	Renumbers the statements starting with 10 in increments of 10.	3-5
REN n	Renumbers the statements starting with n in increments of 10.	3-5
REN n,i	Renumbers the statements starting with n in increments of i.	3-5
RUN --	Clears all variable definitions and executes the program beginning with the first line.	3-7
RUN n --	Executes the program beginning with statement n and does not clear variable definitions.	3-7
SAVE file {,C}{,T} --	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.	3-11
SCRATCH --	Deletes the entire program and clears all variable definitions. S	3-4, 5-45
SET CM=exp	If exp equates to zero, the video cursor will not appear; if exp is non-zero, it will appear. S	5-45
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	5-45
SET DS=speed	Sets the video display speed to the value indicated. S	5-45
SET FB=exp	If exp equates to zero, static diskette file buffering will be used; if non-zero, dynamic buffering will be used.	5-34
SET LL=length	Sets the line length for BASIC output to the value specified. S	5-45
SET ML=size	Sets the memory limit for BASIC to the number of bytes specified.	5-45
SET OF=file	Sets the output file. All output from BASIC is sent to the named file. S	5-34

SET XI=n	See the statement summary.	5-33
TUOFF	Turns off both tape motor relays. S	5-36
TUON unit# -	Turns on the specified tape motor relay. S	5-36
XEQ file -	Reads and executes a diskette file program of type C or T.	3-13

STATEMENTS

Statement	Description	Page
CLOSE #file number1, #file number2, ... -	Closes the specified files so that they cannot be accessed unless another FILE statement requests access.	5-32
CURSOR {L}{,C}{,B} --	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. If B is present, the USASCII character whose value is B will be placed at the new cursor position. C	5-46
DATA constant1, constant2, ... -	Specifies numerical or string constants that can be read by the READ statement.	4-6
DEF FNvariable(variable1, variable2,...) = expression --	Defines a one-line function that evaluates an expression based on the values of the variables in parentheses.	5-8
DEF FNvariable(variable1, variable2,...) -- . . RETURN expression ---. . FNEND --	Defines a multi-line function that executes statements following using the values of the variables in parentheses in calculations, and, when a RETURN statement is encountered, returns the value of the expression on the same line. FNEND ends the function definition. --	5-7
DIM variable(dimension1, dimension2, ...) --	Defines a multi-dimensional numerical array with the number of dimensions specified. C	5-19
DIM string variable (size) --	Declares the number of characters that can be contained in the specified string variable. C	5-11
END -	Terminates execution of the program.	4-9

```
ERRCLR              Clears the error trap line number set by the
----                most recent ERRSET statement.  C                5-42

ERRSET n            When an error occurs, BASIC executes statement
--                  n next.  C                                      5-42

EXIT n              Escapes from and terminates the current FOR/
--                  NEXT loop. Statement n is executed next.        4-14

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 ac-
                    cess file.  If bs is present, it specifies the
                    block size (1 to 4095 bytes) to be used. 5-21, 2, 4

FILL string, string expression
                    Fills a string variable or substring function
                    with a copy of the first character which the
                    string expressions yields.      C              5-13

FNEND               Ends a function definition.
--                                                                  5-7

FOR variable = expression1 TO expression2 {STEP interval}
-  .               [The value of expression1 is assigned to the
   .               |variable, then the statements between FOR and
   .               |NEXT are executed repeatedly until the vari-
NEXT {variable}    |able equals expression2.  After each iteration
-                  |the variable is incremented by 1, or by the
                   [STEP interval if given.                         4-13

GOSUB n             Executes the subroutine beginning at statement
---                 number n. Execution continues with the
                    statement following the GOSUB statement.        5-2

GOTO n              Transfers control to statement number n.
-                                                                   4-11

IF expression THEN n
-             --    Executes statement n if the value of the ex-
                    pression is true; otherwise, executes the next
                    statement in sequence.                          4-20

IF expression THEN n1 ELSE n2
-             --      --
                    Executes statement n1 if the value of the ex-
                    pression is true; otherwise, executes state-
                    ment n2.                                        4-20
```



```
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     4-20

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      4-20

IF expression THEN n ELSE statement1:statement2:...
-             --     --
                    Executes statement n if the value of the ex-
                    pression is true; otherwise, executes the
                    statements following ELSE.                      4-20

IF expression THEN statement1:statement2:...ELSE n
-             --                            --
                    Executes the statements following THEN if the
                    value of the expression is true; otherwise,
                    executes statement n.                           4-20

INPUT variable1, variable2, ...
--                  Accepts values from the terminal and assigns
                    them to variable1, variable2, etc.  C           4-3

INPUT "message", variable1, variable2, ...
--                  Displays the message as a prompt and then
                    accepts values from the terminal, assigning
                    them to variable1, variable2, etc.  C           4-3

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 indica-
                    ted and has time (in tenths of a second) to
                    respond.                                        5-41

INPUT (characters, time) "message", variable1, variable2,...
--                  Displays the message as a prompt and then ac-
                    cepts values from 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 a
                    second) to respond.                             5-41

{LET} variable1 = expression1 {, variable2 = expression2}...
 -                  Assigns the value of each expression to the
                    corresponding variable. The word LET may be
                    absent.  C                                      4-2
```

LOAD string {,var}
-- Loads the PTDOS image 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 6-3

MAT mvar = ZER
- Sets every element in matrix mvar to zero. C 7-2

MAT mvar = CON
- Sets every element in matrix mvar to one. C 7-2

MAT mvar = IDN
- Sets the matrix to an identity matrix. C 7-2

MAT mvar1 = mvar2
- Copies matrix variable 1 into matrix variable 2. C 7-3

MAT mvar1 = mvar2 op (expr)
- Performs the same scalar operation on each element of matrix variable 2. op is + - * or / C 7-3

MAT mvar3 = mvar1 op mvar2
- Adds, subtracts, or multiplies matrix variable 1 by matrix variable 2. op is + - or * C 7-4

MAT mvar1 = TRN (mvar2)
- Places the transpose of matrix variable 2 into matrix variable 1. C 7-4

MAT mvar1 = INV (mvar2)
- Places the inverse of matrix variable 2 into matrix variable 1. C 7-4

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. 7-5

NEXT {variable}
- Ends a FOR loop. 4-13

ON expression ERRSET n1, n2, ...
- -- If the value of the expression is 1, sets n1 as the statement to be executed when an error occurs; if the value is 2, sets n2 as the statement to be executed when an error occurs; etc. 5-43

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. 4-15

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. 5-4

ON expression GO TO n1, n2, ...
- - If the value of the expression is 1, executes statement n1 next; if it is 2, executes statement n2 next; etc. 4-12

ON expression RESTORE n1, n2, ...
- --- 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. 4-8

OUT port, value
-- Places the specified value in the indicated I/O port. C 6-2

PAUSE nexpr
-- Delays further execution for nexpr tenths of a second. 4-10

POKE location, value
-- Places the specified value in the specified memory location. C 6-2

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. 4-4, 5-37

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 5-25

PRINT #file number, {record} {,d}; ele1 {,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. 5-26, 7

PURGE string {,var}
--
Kills the diskette data file whose name is the value of a string expression. The variable receives any PTDOS error message. 5-33

READ variable1, variable2, ...
-
Reads values from DATA statements and assigns them to variable1, variable2, etc. 4-6

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. 5-28

READ #n, {rn}{,d}; var1 {,var2}...{:statement1 :statement2}
-
If the file cursor 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. 5-29, 3Ø

REM any series of characters
The characters appear in the program as remarks. The statement has no effect on execution. 4-1

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. 4-8

RETURN

Returns from a subroutine. 5-3

RETURN exp

Returns from a function. The value returned is exp. 5-7

REWIND #file number1, #file number2, ...

Rewinds the specified files. 5-32

SEARCH string expression1, 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 it is not found, the variable is set equal to zero. 5-12

SET XI=number
Loads the index block of the referenced file into memory for fast access. 5-32

STOP
-
Terminates execution of the program and prints "STOP IN LINE n" where n is the line number of the STOP statement. 4-1Ø

WAIT exp1, exp2, exp3
-
The next statement is not executed until the value in port exp1, ANDed with exp2, is equal to exp3. 6-4

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. 3-13

APPENDIX 2

EXTENDED BASIC FUNCTION SUMMARY AND INDEX

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	Page
ABS(n)	The absolute value of the numerical expression n.	5-5
ASC(s)	The USASCII code for the string expression s. Only the first character of the string is interpreted.	5-16
ATN(n)	The arctangent of the numerical expression n in radians.	5-6
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.	6-3
CHR(n)	The character whose USASCII code is the value of numerical expression n.	5-16
COS(n)	The cosine of n in radians.	5-6
EOF(file number)	The status of the specified file. 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 was READ EOF (end of file) 18 last was Serial File READ with Spacing 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	5-35
ERR(0)	A string containing the last error message.	5-17, 5-43
EXP(n)	The constant e raised to the power n.	5-5
FNvariable(variable1, variable2, ...)	The value of user-defined function FNvariable. variable1, variable2, etc. are arguments.	5-7

FREE(Ø)	The number of bytes of space left available in BASIC for program and variables.	5-18
INP(exp)	Supplies the numerical value contained in I/O port exp. Exp is between Ø and 255.	6-2
INT(n)	Truncates n to its integer part.	5-5
LEN(name)	The number of character in the string variable whose name is specified.	5-15
LOG(n)	The natural logarithm of n.	5-5
LOG1Ø(n)	The logarithm base 1Ø of n.	5-5
PEEK(n)	The value contained in memory location n.	6-2
POS(Ø)	The current position of the cursor (Ø - 131).	5-47
RND(exp)	A random number between Ø and 1. exp = Ø, -1 or n.	5-5
SGN(n)	The sign of the value of n; 1 if positive, -1 if negative, Ø if n is zero.	5-5
SIN(n)	The sine of n in radians.	5-6
SQR(n)	The square root of n.	5-5
STR(n)	The character representation of the value of n.	5-16
SYST(n)	Return the number of the last PTDOS error, where n evaluates to Ø.	5-44
TAB(n)	Moves the cursor or print head horizontally to character position n. Use only in a PRINT statement.	4-5
TAN(n)	The tangent of n in radians.	5-6
TYP(Ø)	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.	4-7
VAL(s)	The numerical value of the string s. The value of s must be convertable to a legal numerical constant.	5-16

string variable (expl{,exp2})

Characters expl through exp2 of the specified string if exp2 is present. Characters expl through the end of the specified string if exp2 is ommitted. 5-14

numerical variable (nl{, n2, ...})

An element of an array with the specified name. The element's position is given by nl, n2, etc. 5-16

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.

There are many errors that PTDOS may print while BASIC is running. Such messages have the form: "PTDOS ERROR message" instead of "xx ERROR..." PTDOS errors are listed in Section 6 of the PTDOS manual.

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.
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.
CS	Control stack error. Possible causes are: -RETURN without a prior GOSUB -Incorrect FOR/NEXT nesting -Too many nested GOSUBs -Too many nested FOR loops -Too many nested function calls	List the statements surrounding the error-causing statement and check the logical flow. Execute just a few statements at a 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 nonzero 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 Extended 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 1Ø to the 126th power, or less than 1Ø to the -126th power.	No solution.
FS	File Structure error. A PTDOS error occurred.	Use the SYST(Ø) function to determine the error number.

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. A expression was too complex to evaluate.	Divide the expression into parts, using assignment statments.
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--for example commands to the video display driver are not available when output is set to another file.	Return output to the video driver, using SET OF=#Ø, or don't use the 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	Not implemented. An attempt was made to use matrix or trig functions which were deleted.	See Section 2.1.
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 arguments or parameters used. If they seem reasonable, look up the definition of the function or the command.
RO	Record overflow. An attempt was made to write more items into a record of a random access data 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.
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	Undimensionned matrix. A variable name was used which was not previously	DIMension the matrix in an earlier DIM statement.
UR	Unresolved linenumber reference. During a RENumber command, a control transfer statement referred to a nonexistant line number.	Look for typos in the program. Unresolved references will have been changed to Ø.

APPENDIX 4
TABLE OF ASCII CODES (Zero Parity)

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	?		

APPENDIX 4
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	[	shift K
•.•• •	0560	134	92	5C	\	shift L
• •.•• •	0564	135	93	5D	]	shift M
••.•• •	0570	136	94	5E	^	shift N
•••.•• •	0574	137	95	5F	_	shift O
. ••	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	}	Alt Mode
••.••••	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

	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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	[illegible]
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	[illegible]
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	[illegible]

	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

	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

	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

BIBLIOGRAPHY

1. An Introduction to Microcomputers, Volume 0, The Beginner's Book
   Adam Osborne and Associates, Inc. 1977

2. BASIC Programming in Real Time
   Don Cassel, 1942
   Reston Publishing Company, Inc. 1975

3. Simplified BASIC Programming
   Gerald A. Silver
   McGraw-Hill Co. 1974

4. Basic BASIC
   James S. Coan
   Hayden 1970

5. Advanced BASIC
   James S. Coan
   Hayden 1977

6. Problems for Computer Solution
   Fred Gruenberger and George Jaffray
   Wiley 1965

7. BASIC
   Samuel L. Marateck
   Academic Press 1975

8. Some Common BASIC Programs
   Lon Poole and Mary Borchers
   Adam Osborne & Associates, Inc. 1977

9. Game Playing with BASIC
   Donald D. Spencer
   Hayden Book Company, Inc. 1977

10. Game Playing with Computers
    Donald D. Spencer
    Hayden 1975

11. 101 BASIC Computer Games
    Digital Equipment Corporation 1975

12. Theory and Problems of Matrices
    Schaum's Outline Series
    Frank Ayres, Jr.
    McGraw-Hill Co. 1962

Processor Technology Corporation 7100 Johnson Industrial Drive, Pleasanton, CA 94566

Manual Number 727037