MOTOROLA

M68BAS(D3)

M6800
BASIC INTERPRETER

Reference Manual

SYSTEMS

MICROSYSTEMS

M68BAS(D3)

JANUARY 1980

M6800

BASIC INTERPRETER

REFERENCE MANUAL

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

Third Edition
©Copyright 1980 by Motorola Inc.
Second Edition February 1979

TABLE OF CONTENTS

Page

CHAPTER 1. GENERAL DESCRIPTION

1.1 INTRODUCTION 1-1
1.2 EXORciser-RESIDENT BASIC INTERPRETER. 1-1
1.3 FEATURES 1-2
1.4 PROGRAM STRUCTURE 1-2
1.4.1 Statements 1-2
1.4.2 Commands 1-3
1.4.3 Functions 1-3
1.5 DATA REPRESENTATION 1-3
1.5.1 Numbers 1-3
1.5.1.1 Numeric Variables 1-4
1.5.1.2 Numeric Constants 1-4
1.5.2 Strings 1-4
1.5.2.1 String Variables 1-4
1.5.2.2 Literals 1-5
1.6 OPERATING MODES 1-5
1.6.1 Command Mode 1-5
1.6.2 Execution Mode 1-5

CHAPTER 2. PROGRAM PREPARATION

2.1 LOADING BASIC 2-1
2.1.1 Loading MDOS BASIC 2-1
2.1.2 Loading Tape BASIC 2-2
2.2 ORIGINATING AND CHANGING A BASIC PROGRAM . . . 2-2

CHAPTER 3. STATEMENTS

3.1 INTRODUCTION 3-1
3.2 ASSIGNMENT STATEMENTS 3-1
3.2.1 LET Statement 3-1
3.2.2 Arithmetic Operator Symbols 3-1
3.2.3 Operator Heirarchy 3-2
3.2.4 String Concatenation 3-2
3.3 DECLARATION STATEMENTS 3-3
3.3.1 DIM Statement 3-3
3.4 CONTROL STATEMENTS 3-3
3.4.1 FOR and NEXT Statements 3-3
3.4.2 STOP Statement 3-4
3.4.3 END Statement 3-5
3.4.4 GOTO Statement 3-5
3.4.5 GOSUB Statement 3-5
3.4.6 RETURN Statement 3-5
3.4.7 ON Statement 3-5
3.4.8 IF and THEN Statements 3-6
3.4.9 USER Function 3-6
3.5 INPUT/OUTPUT STATEMENTS 3-6
3.5.1 INPUT Statement 3-7
3.5.2 DATA and READ Statements 3-8
3.5.3 RESTORE Statement 3-9
3.5.4 PRINT Statement 3-9
3.5.5 POKE Statement 3-10
3.5.6 PEEK Function 3-10
3.5.7 LINE Statement 3-11
3.5.8 DIGITS Statement 3-11

3.6 DOCUMENTATION STATEMENTS 3-11
3.6.1 REM Statement 3-11
3.7 DEBUG STATEMENTS 3-11
3.7.1 TRACE ON Statement 3-12
3.7.2 TRACE OFF Statement 3-12
3.7.3 PATCH Statement 3-12

CHAPTER 4. COMMANDS

4.1 INTRODUCTION 4-1
4.2 SYSTEM COMMANDS 4-1
4.2.1 LOAD Command 4-1
4.2.2 APPEND Command 4-1
4.2.3 SAVE Command 4-1
4.2.4 EXIT Command 4-1
4.2.5 LIST Command 4-2
4.3 PROGRAM EXECUTION COMMANDS 4-2
4.3.1 RUN Command 4-3
4.3.2 CONT Command 4-3
4.3.3 NEW Command 4-3
4.3.4 BREAK KEY Command 4-3
4.4 EDITING COMMANDS 4-3
4.4.1 CONTROL X Command (Cancel) 4-4
4.4.2 RUBOUT or DEL Command (Backspace) 4-4
4.5 SIZING COMMANDS 4-4

CHAPTER 5. FUNCTIONS

5.1 INTRODUCTION 5-1
5.2 CONTROL FUNCTIONS 5-1
5.2.1 TAB Function 5-1
5.2.2 POS Function 5-1
5.3 DATA FUNCTIONS 5-1
5.3.1 RND Function 5-1
5.3.2 INT Function 5-2
5.3.3 ABS Function 5-2
5.3.4 SGN Function 5-2
5.3.5 PEEK Function 5-2
5.4 CHARACTER STRING FUNCTIONS 5-3
5.4.1 LEN Function 5-3
5.4.2 ASC Function 5-3
5.4.3 CHR$ Function 5-3
5.4.4 VAL Function 5-3
5.4.5 STR$ Function 5-4
5.4.6 LEFT$ Function 5-4
5.4.7 RIGHT$ Function 5-4
5.4.8 MID$ Function 5-4
5.5 TRANSCENDENTAL FUNCTIONS 5-5
5.6 USER DEFINED FUNCTIONS 5-5
5.6.1 DEF Function 5-5
5.6.2 USER Function 5-6

CHAPTER 6. DISK FILE I/O (MDOS ONLY)

6.1 INTRODUCTION 6-1
6.2 OPEN Statement 6-1
6.3 CLOSE Statement 6-2
6.4 RESTORE Statement 6-2
6.5 INPUT Statement 6-3
6.6 PRINT Statement 6-3

6.7 EXAMPLES OF DISK FILE I/O 6-3
6.7.1 Creation of a Data File 6-3
6.7.2 Reading a Data File 6-4
6.8 End of File Detection 6-4
6.9 Updating a Data File 6-4
6.10 Deleting a Data File 6-5
6.11 Data File Data Representation 6-5

APPENDIX A. GENERAL CONVERSION CHARTS A-1
APPENDIX B. ASCII TO DECIMAL CONVERSION B-1
APPENDIX C. ENHANCING PROGRAM OPERATION C-1
APPENDIX D. MINIMIZING MEMORY REQUIREMENTS D-1
APPENDIX E. ERROR MESSAGES E-1
APPENDIX F. ROM-RESIDENT BASIC INTERPRETER F-1

CHAPTER 1

GENERAL DESCRIPTION

1.1 INTRODUCTION

This manual provides detailed information for the EXORciser- and ROM-resident BASIC Interpreters. Chapters 1-6 discuss the EXORciser-resident BASIC Interpreter. Since both Interpreters are basically identical, these chapters can be used to support the ROM-resident BASIC Interpreter. Differences are noted and references made to Appendix F.

1.2 EXORciser-RESIDENT BASIC INTERPRETER

The EXORciser-resident BASIC Interpreter provides a high-level programming language widely used for general purpose and business-related applications. BASIC uses English-like statements and familiar mathematical notations which make it one of the easiest computer languages to learn and use. The system consists of the BASIC Interpreter program software and one of the following hardware configurations:

MDOS DISK SYSTEM

EXORciser
20K Bytes of Memory (Minimum)
EXORdisk II or III Drive Unit
ASCII Terminal
MDOS Floppy Disk Operating System

NON-DISK SYSTEM

EXORciser
8K Bytes of Memory (Minimum)
ASCII Terminal
Digital Cassette or Paper Tape Reader/Punch

The minimum configuration may be expanded to include up to 56K bytes of memory and four disk drives for a total of over 2 million bytes of on-line disk storage capability. In addition, a variety of hard copy line printers and CRT terminals may be used. Throughout this manual, it has been assumed that a console terminal having both keyboard and printer is being used with an MDOS floppy disk operating system. If a CRT terminal is used, the words "console printer" may be replaced by the words "CRT display".

Any differences between the MDOS version and tape version of BASIC are noted. ROM version differences are described in Appendix F.

1.3 FEATURES

The features of this BASIC Interpreter are:

All mathematical operations are performed in BCD (Binary Coded Decimal) arithmetic for maximum accuracy.

Nine significant digit precision.

User programs may be saved and loaded from cassette, paper tape, or floppy disk.

Most arithmetic functions and transcendentals are implemented as directly executable subprograms.

String variables and arrays are permitted. Most of the popular string functions are provided.

Most program statements may be executed in the direct mode (no statement numbers required for immediate execution).

Users can call machine language programs from the BASIC program.

Data file disk I/O may be performed (MDOS only).

Limited editing of programs without leaving BASIC.

1.4 PROGRAM STRUCTURE

A user program written in BASIC consists of various statements. The following paragraphs provide a brief description, while chapters referenced within each paragraph provide an in-depth description.

1.4.1 Statements

Every statement must begin with a statement number, followed by the statement directive and statement argument, and terminated by a carriage return (CR). Six types of statements are used in BASIC: Input/Output, Declaration, Control, Assignment, Documentation, and Debug. These statements are described in Chapter 3. The following rules apply to all BASIC statements:

Every statement must have a statement number ranging from 1 to 9999. DO NOT USE STATEMENT NUMBER 0.

Statement numbers are used to sequentially order the program statements and to allow transfer of control.

In any program, a statement number may only be used once.

A previously entered line may be changed by entering the same statement number along with the desired new statement. Entering a statement number immediately followed by a carriage return deletes that line and statement number.

Statements do not need to be entered in numeric sequence, since the BASIC interpreter automatically maintains a sequential ordering of statements.

A statement may NOT contain more than 72 characters, including spaces and statement number.

Unless used within a character string, spaces are ignored by BASIC. Although spaces may be inserted within a statement to improve readability, the use of spaces increases a program's memory requirements and execution time. Numbers MUST NOT contain embedded spaces. All examples presented in this manual include spaces.

Example: 110 LET A = B + (3.5 * 5E2)

is exactly equivalent to:

110LETA=B+(3.5*5E2)

1.4.2 Commands

BASIC provides a set of commands which are used for program loading, editing, and execution. In addition, most of the BASIC statements may be used as commands, thereby operating as a calculator. A complete description of each BASIC command may be found in Chapter 4.

1.4.3 Functions

Functions may be either intrinsic or user defined.

Intrinsic functions are program functions which are inherent to the BASIC interpreter itself and do not require a separate program for execution. Three types of intrinsic functions are available to the user: Control, Data, and Mathematical. The intrinsic and user defined functions of BASIC are described in Chapter 5.

1.5 DATA REPRESENTATION

All data used with BASIC must be represented in accordance with the paragraphs of this section.

1.5.1 Numbers

The range of numbers capable of being represented in BASIC is: 1.0E-99 to 9.99999999E+99. The "E" represents exponent; thus, E-99 equals 10 to the -99th power and E+99 equals 10 to the 99th power. BASIC provides for nine

significant digits of accuracy and truncates numbers representing more than nine significant digits. Numbers may be entered and displayed in one of three formats: integer, decimal, or exponential.

Example: 153 (integer)
153.34 (decimal)
1.5334E+2 (exponential)

1.5.1.1 Numeric Variables

Numeric variables may be named by any single alphabetic character or any single alphabetic character followed by a single numeric digit (0 through 9).

Example: A, B, C, A3, B4, or C0

Numeric variables may also be used to represent arrays. In this case, the one or two dimension element number follows the variable name in parentheses. See 3.3.1 for the DIM statement and array information.

Example: A(5), B(67,2), C(1), A3(14), or C0(7)

1.5.1.2 Numeric Constants

A numeric constant (that is, a non-varying quantity) is composed of one or more numeric digits, with or without a decimal point.

Example: 153, 34.52, .554, or 132E-2

1.5.2 Strings

Alphanumeric data (alphabetic letters and numerals) and certain punctuation and special characters may be represented in BASIC in several ways. Strings refer to this type of data as opposed to numeric data. String data is represented and stored in ASCII form while numeric data is stored in BCD form.

1.5.2.1 String Variables

String variables are named by any single alphabetic character followed by a dollar sign ($). String variables may contain a MAXIMUM of 18 characters, including spaces. Each string variable will be allocated 18 bytes of memory. Each character in a string is represented internally in ASCII.

Example: A$, B$, C$, or W$

String variables may also be used to represent arrays. The one or two dimension element number follows the variable name in parentheses. See 3.3.1 for the DIM statement and array information.

Example: A$(3), B$(21,2), or C$(1)

1.5.2.2 Literals

A literal is the string equivalent of a numeric constant. It is composed of one or more characters contained within quotation (") marks.

Example: "LITERAL STRING", "5", "--*(#$5"

1.6 OPERATING MODES

Two operating modes are used during entering and executing BASIC programs: Command and Execution.

1.6.1 Command Mode

The system operates in the command mode when it is ready for the user to enter the next program statement or BASIC command. The system indicates the command mode to the user by printing the # character as a prompt.

1.6.2 Execution Mode

When in execution mode, BASIC processes the user program, executing one statement at a time.

CHAPTER 2

PROGRAM PREPARATION

2.1 LOADING BASIC

The BASIC Interpreter is available in three types of storage media: ROM, paper tape, or diskette. (Refer to Appendix F for ROM-Resident BASIC Interpreter differences.

The procedure required to load the BASIC interpreter into memory is dependent upon the type of storage media used. If cassette or paper tape is used, then loading is controlled by the EXbug monitor. If BASIC is stored on diskette, the interpreter is loaded by typing the command BASIC followed by the input file name and the output file name (if required) on the disk operating system command line. This is covered in paragraphs 2.1.1 and 2.1.2.

After the BASIC interpreter is loaded, the word READY is printed on the console printer followed by the # character on the next line. This indicates that the interpreter is ready to accept input (either program statements or commands) from the console keyboard. After each successive program statement is entered, the system will prompt the user by printing a # character. This indicates that the statement has been entered. After a command is entered, the word READY will again be printed, followed by the # character to indicate that the command has been executed and the system is ready for input of the next program statement or command.

2.1.1 Loading MDOS BASIC

This example shows the interactive procedure used to start the BASIC interpreter on an MDOS disk system.

```
*E MDOS
=BASIC INFILE (or BASIC INFILE,OUFIL)

MDOS BASIC 2.xx
COPYRIGHT(C)- 19xx

READY
#
```

In the MDOS system, if INFILE doesn't exist, one will be created. If INFILE exists, it will be read into the BASIC user program buffer where it may be run or changed. A second file name may be specified and the changed copy of the program saved in it upon exiting.

2.1.2 Loading Tape BASIC

This example shows the interactive procedure used to start the BASIC interpreter on a system using paper tape or cassette.

```
*E LOAD
SGL/CONT S
BASIC8
*E 100;G
M6800 BASIC 1.xx
COPYRIGHT (C) - 19xx

READY
#
```

2.2 ORIGINATING AND CHANGING A BASIC PROGRAM

After the BASIC interpreter has been loaded into memory, the system is ready to accept commands or statements from the console keyboard. For example, the following program may be entered:

```
READY
#10 REM DEMONSTRATION
#20 PRINT "ENTER A NUMBER"
#30 INPUT N
#40 LET S=N*N
#50 PRINT
#60 PRINT "THE SQUARE OF ";N;
#70 PRINT "IS ";S
#80 STOP
```

To insert a statement between two others, a statement number that falls between the other two followed by the statement to be inserted is typed. The following example shows an inserted statement:

```
#45 REM THIS INSERTED BETWEEN 40 AND 50
```

To replace a statement, the statement number of the old statement is typed, followed by the new statement and a carriage return. For example, to change statement number 40 above:

```
#40 LET S=N©2
```

Each line entered must be terminated by a carriage return. BASIC automatically advances the printer to the start of the next line when a carriage return is recognized.

To execute the program at this point, the RUN command should be entered on the console keyboard. Chapter 4 explains this and other commands used to save, list, or edit a program.

CHAPTER 3

STATEMENTS

3.1 INTRODUCTION

A BASIC program consists of one or more statements which perform some action. BASIC statements may be divided into six categories: Assignment, Declaration, Control, Input/Output, Documentation, and Debug.

These statements are stored in memory when entered and are not executed until an execute command is given. Chapter 4 explains in detail the various commands. However, when BASIC is in the command mode (waiting for user input), most statements may be entered without statement numbers. BASIC will execute these statements immediately. This is referred to as the "direct mode" of execution. Since statements entered in the direct mode are normally of a mathematical nature, the user obtains immediate results to problems without requiring iterative (repeated) program procedures. The direct mode is sometimes referred to as the "calculator" mode.

3.2 ASSIGNMENT STATEMENTS

Assignment statements are used to assign values to variables and allow the concatenation of string variables. The following paragraphs list and explain the assignment statements.

3.2.1 LET Statement

The LET statement is used to assign a value to a variable. The use of the word LET is optional. The following example shows typical uses of the LET statement:

```
110 LET C=33.5
120 LET M=(C+11)/3
130 D=M+C+50
140 C$="YES"
```

The equal sign used in the LET statement does not mean equivalence as normally used in the mathematical sense. Instead, it represents a replacement operator. When used in this context, the equal sign means: replace the value of the variable name on the left with the value of the expression on the right.

3.2.2 Arithmetic Operator Symbols

The arithmetic operator symbols used to form arithmetic expressions are shown below. No two operators may occur in sequence (with one exception) and no operator may ever be assumed.

Thus A++B and (A+3)(B-4) are not valid arithmetic expressions. The only exception to this rule is in the use of exponents. In this case, A^-5 is valid, meaning that A is raised to the -5 power. A negative value cannot be exponentiated, therefore C in the expression C^2 must be positive.

- ^ Exponentiation (raising to a power)
- \- Unary (Negate, requires only one operand)
- * Multiplication
- / Division
- \+ Addition
- \- Subtraction

3.2.3 Operator Heirarchy

Operator heirarchy (priority) controls the order of evaluation of an arithmetic expression. This heirarchy is listed below showing the highest first:

```
( ) Parenthesis
^
- (unary)
* or /
+ or -
```

Note that in the heirarchy shown above, arithmetic operators contained within parentheses are executed first, followed by non-parenthetical operators. Some typical examples are shown below:

```
130 M=(X*Y)/(3+A)
160 K=-2^2
```

In the first example (statement number 130), the terms within the parentheses will be evaluated first. Therefore, (X*Y) and (3+A) will be evaluated, then the result of the first divided by the second. Even though the + of the second term is lower heirarchy than the division, it is executed first since it is enclosed in parentheses. In statement 160, K will be -4 since the exponentiation is performed before the unary operation.

3.2.4 String Concatenation

String concatenation is used to join together two or more string variables. Even though a string variable may contain up to 18 characters, it may be desirable to combine several strings for printing. The string concatenation symbol is the + sign. The example which follows illustrates string concatenation:

```
110 A$="THE "
120 B$="QUICK "
130 C$="BROWN "
140 D$="FOX "
150 E$="JUMPED "
160 F$=A$+B$+C$
170 G$=D$+E$
180 PRINT F$;G$
```

the results of executing statement number 180 would be:

```
THE QUICK BROWN FOX JUMPED
```

3.3 DECLARATION STATEMENTS

Declaration statements are used to specify (declare) the attributes of various variables within BASIC. One declaration statement is used in this version of BASIC, the DIM statement.

3.3.1 DIM Statement

The DIM statement allocates memory space for an array. One or two dimensions are allowed. The maximum array size is 255 by 255 elements. (Of course, the real maximum is less since insufficient memory exists for any array 255 by 255.) If an array is not explicitly defined by a DIM statement prior to its use, then it is implicitly assumed to be an array of 10 elements (or 10 by 10 if two elements are used) when it is first referenced in a program. The array size can only be declared once in a program, either explicitly or implicitly. The following examples show the various uses of this statement:

```
630 DIM A(35),B4(12),J9(200)
640 DIM K$(15,10),W(10,50)
650 DIM L(Q)                Note in this case, Q must have been
                            previously given a value before its
                            use in the DIM statement.
```

3.4 CONTROL STATEMENTS

Control statements are used to alter the normal sequential progression of program statement execution. Control statements can be used to transfer control to another part of a program, terminate execution, or control iterative loops. The following paragraphs list and explain each of the control statements.

3.4.1 FOR and NEXT Statements

The FOR and NEXT statements are used in conjunction with each other to establish program loops. A loop causes the execution of one or more statements for a specified number of times.

sn FOR vc=init TO end STEP val

where sn is the statement number
vc is the counter variable
init is the initial value
end is the final value
val is the step or increment value

sn NEXT vc
where sn and vc are as described above.

The FOR statement consists of three parts: FOR, TO, and STEP. The FOR portion specifies the counter variable and its initial value. The TO portion specifies the final counter value at which the loop will be done. The optional STEP indicates the size of the change to the counter for each interation through the loop. (STEP will be 1 if omitted). Subsequently, each of the statements following the FOR statement is executed until the corresponding NEXT statement is encountered. Although expressions are permitted for the initial, final, and step values in the FOR statement, these expressions will be evaluated only once -- upon initially processing the FOR statement.

The NEXT statement is located at the end of the loop. When the NEXT statement is encountered, the variable specified by the FOR statement is incremented by the value of the STEP. If the resultant value is less than the TO value, the next statement executed will be the statement immediately following the FOR statement starting the loop. If the value is equal to or greater than the TO value, then program execution continues on to the statement following the NEXT.

When the statement following the NEXT statement is executed, the counter will be equal to the value used the last time through the loop. It is not possible to use the same variable for the counter of two different loops if the loops are nested (one loop within another loop). The counter variable K7 would have a value of 9.25 after passing out of the loop shown below:

```
110 FOR K7=0.5 TO 10 STEP 1.25
120 INPUT X
130 PRINT K7,X,X/2.75
140 NEXT K7
```

A FOR-NEXT loop will always be executed at least once, even if the TO value is less than the initial value with a positive STEP. The STEP value may be either positive or negative.

The variable name in the NEXT statement must be the same as the variable name for the corresponding FOR statement.

Control may be transferred out of the FOR-NEXT loop. However, certain precautions should be taken. First, if control is transferred out of the loop and then returned to the loop, execution will continue normally. If control is transferred out of the loop and NOT RETURNED, then the nesting level is not reduced and could cause an error due to too many levels of nesting.

The maximum number of nesting levels is 8.

3.4.2 STOP Statement

The STOP statement causes the program to stop executing and return to the command level of BASIC. This statement differs from the END statement in that it causes the message "STOP xxxx" to be printed (where xxxx is the statement number of the STOP statement). In addition, the program can be restarted by a CONT command.

3.4.3 END Statement

The END statement causes the program to stop executing and return to the command level of BASIC. In this version of BASIC, the END statement may appear more than once in a program or need not appear at all.

3.4.4 GOTO Statement

The GOTO statement causes the program to jump to the statement number specified. The program then continues at the new statement number. In the example below, the GOTO statement will cause statement 230 to be skipped.

```
220 GOTO 300
230 LET A=77
300 PRINT A
```

3.4.5 GOSUB Statement

The GOSUB statement causes the program to execute a subroutine. A subroutine is a sequence of statements that perform some task and may be called one or more times in a program. Upon completion of a subroutine, its RETURN statement causes control to be passed to the statement immediately after the GOSUB which called the subroutine. Subroutine nesting (one subroutine calling another) is limited to 8 levels.

In this example, both the GOSUB and RETURN statements are shown:

```
Main Program:
110 INPUT X
120 GOSUB 210
130 PRINT Y
140 STOP

Subroutine:
210 Y=(X+3)/3.1415926
220 RETURN
```

3.4.6 RETURN Statement

The RETURN is used at the logical end of a subroutine to return to the statement directly following the GOSUB statement. The RETURN statement must be the last logical statement of the subroutine. It may appear more than once within a subroutine.

3.4.7 ON Statement

The ON statement transfers the program to another statement or subroutine based upon the value of the expression following the ON statement. The expression will be evaluated, truncated to an integer (whole number), and the program then transferred to the statement number corresponding in position to the expression value.

```
110 ON N GOTO 200,300,550,710
800 ON (K+5)/3 GOSUB 450,7790
```

In the above examples, if N were evaluated as 3, the program would go to statement number 550. If K were 1, the subroutine at statement number 7790 would be performed. The expression must evaluate to a positive value less than 256. If N were 5 or greater in the above example, then an error message would be printed on the console during execution of the program.

3.4.8 IF and THEN Statements

The IF and THEN statements are used to control the execution sequence via the testing of specific conditions. Various relational operators are used with the IF statement to determine the specific conditions that will cause the program to execute the statement following the THEN.

The relational operators listed below are used:

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

If the relational expression specified in the IF statement is true, then the statement following the THEN is executed. However, if the relational expression is false, program execution continues with the statement following the IF statement. If a statement number appears after the THEN, it is taken as an implied GOTO statement and control will be transferred to the specified statement number.

```
Example: 20 IF A<B THEN 150
         30 IF A=B+4 THEN PRINT "VALUE=";A
         40 IF K3>C3 THEN K3=0
         50 IF A$="YES" THEN 500
```

3.4.9 USER Function

The USER function acts as a form of a control statement which allows calls to a user-written, non-BASIC subroutine. With the USER function, operations or calculations written in assembly or other language may be performed. A description of the USER function is found in section 5.6.2.

3.5 INPUT/OUTPUT STATEMENTS

Input/Output statements are used to control the flow of data between the EXORciser system and the peripherals. The following paragraphs list and explain each of the input/output statements. Refer to Chapter 6 for disk file I/O operations available in MDOS BASIC. The following table lists I/O device assignments:

```
# 1   = console (default)
# 2   = line printer
# 3-9 = disk file (refer to Chapter 6)
```

3.5.1 INPUT Statement

The INPUT statement permits the user to enter data from the console keyboard or disk data file during program execution. The following example presents two typical INPUT statements:

120 INPUT X	means to input one numeric value (X) from the console keyboard. The number may be integer, decimal, or exponential.
130 INPUT X$	means to input one string variable (X$) from the console keyboard.

The string in the above example may consist of alphanumeric and special characters up to a length of 18. If a blank (space) or comma is used within the string, the string must be enclosed in double quotes.

Example: "ADAMS, BEN"

The double quote (") itself cannot be input as a string character.

The INPUT statement may have more than one variable specified after it. This is called a "list", and may consist of a mixture of numeric variable names and string variable names. In addition, the first item following the INPUT may be a literal which will print on the console. All list items must be separated by commas.

When an INPUT statement occurs in the program, a question mark (?) character is printed on the console printer (unless disk file input is being used). In response to this prompt, the user types in the requested data, separated by commas, and followed by a carriage return. Numbers and strings can be entered in response to an INPUT statement. If insufficient data is entered by the user, the system will once again prompt with a question mark. If no data is entered, the prompt will be repeated. If non-numeric data is entered into a numeric variable, the system will prompt with the word RE-ENTER. The example below illustrates the method of entering both numeric variables and string variables using the INPUT statement. Note that each of the variables are separated by a comma.

140 INPUT A$,X,Y,Z	Multiple inputs must be entered. If the expected number of values are not entered, the system will prompt with another "?" character.
150 INPUT "ENTER B",X	Prints the message enclosed within quotation marks as a prompt to the user. A "?" character is then printed directly after the message. The system then awaits the entered value. This value is stored in variable X. A semicolon is not permitted in place of the comma here.

If only string variables remain unsatisfied in an INPUT list, a carriage return will terminate the INPUT and the remaining string variables will be null. This facilitates entering data which may or may not cross the 18 character boundaries of the string variables.

3.5.2 DATA and READ Statements

The DATA and READ statements are used with each other to provide a method of assigning internally stored initial values to variables.

The DATA statement sequentially loads each value specified in the argument field (the field of data directly following the DATA directive) into the data buffer. Each value within the argument field is separated from the next by a comma. If string variables are used within the argument field, they do not need to be enclosed within quotation marks unless they contain a comma within the string. All DATA statements, regardless of where they occur in the program, cause the DATA argument to be combined into one list.

The READ statement sequentially accesses the data buffer and assigns the values previously stored (by the DATA statement) to the variables specified in the argument field of the READ statement. Each variable specified by the READ statement is separated by a comma.

When using READ statements, both numeric and string variables may be used intermixed. However, if this is done, they must be used in the same sequence to match the type of data in the DATA statements. An example using both numeric and string variables follows:

```
160 DATA 10,20,30,45.5,"TEST, ONE",1.2345,6.7E8,END
170 READ A,B,C,D,E$,F,G4,F$
```

Each time a READ statement is encountered in the program, the next available value from the data buffer is assigned to the variables used as arguments of the READ. If there are more variables in the READ statement than values remaining in the data buffer, an error message would be generated. It is permissible to have more values in the DATA statement than variables in the READ statement.

The following example shows DATA and READ statements and the equivalent method of assigning values to variables using the LET statement:

```
210 DATA 1,2,3,4,5,6,7,8,9,END
220 DATA EXIT
230 READ A,B,C,D,E,F,G,H,I
240 READ Z$,X$
```

is the equivalent of:

```
210 LET A=1
220 LET B=2
230 LET C=3
240 LET D=4
250 LET E=5
260 LET F=6
270 LET G=7
280 LET H=8
290 LET I=9
300 LET Z$="END"
310 LET X$="EXIT"
```

3.5.3 RESTORE Statement

The RESTORE statement resets the data buffer pointer, which is advanced by the execution of READ statements, to the first position in the data buffer. This permits the first argument of the first DATA statement to be reused for the next READ statement. The example below shows how the RESTORE statement can be used and also presents the equivalent of the example if LET statements were used:

```
400 DATA 1,2,3
410 READ A,B
420 RESTORE
430 READ C,D
```

is the equivalent of:

```
400 LET A=1
410 LET B=2
420 LET C=1
430 LET D=2
```

3.5.4 PRINT Statement

The PRINT statement enables the user to print the value of expressions, literal strings, or variables on the console printer, line printer, or to a disk data file. The various variables or literal strings of the PRINT statement may be separated by using either a comma (,) or a semicolon (;). If commas are used, each variable or literal string will be separated into 16-character zones. If more is to be printed than line length permits, the remaining variables or strings are printed on the following line or lines. See LINE statement, section 3.5.7.

If semicolons are used to separate the variables or literal strings, then no separation is used for strings or one space is provided after numeric values. The semicolon used at the end of the PRINT statement inhibits the normal carriage return/line feed function.

To print to a line printer, the use of the form PRINT #2 is required. Refer to Chapter 6 for disk I/O information.

Various uses of the PRINT statement are shown and explained in the following examples:

710 PRINT	Skips a line.
720 PRINT A,B,C	Prints the value of A, B, and C separated into 16 character zones. If the semicolon (;) were used, the A, B, and C values would be printed with only one space between them.
730 PRINT "LITERAL STRING"	Prints the string characters contained within the quotation marks.
740 PRINT #2,"VALUES A & B=";A,B	Prints the string characters contained within the quotation marks on the line printer directly followed by the values for A and B.

The PRINT statement may also be used to print the results of expressions, such as PRINT A+B/SQR(C-D).

3.5.5 POKE Statement

The POKE statement stores a specified DECIMAL value into the specified DECIMAL memory location. The format for this statement is:

sn POKE(ma,nv)

where "sn" is the statement number, "ma" is the decimal memory address, and "nv" is the decimal numeric value from 0 to 255 inclusive.

CAUTION! This statement should be used with extreme care. Using an incorrect memory address may change the BASIC program, the BASIC interpreter, or some other memory location causing unpredictable results.

3.5.6 PEEK Function

The PEEK(X) function provides the decimal value contained in the decimal memory address specified by "X" in the function. The number specified by X can be a constant or a variable.

Example: 170 LET A=PEEK(B)

In the above example, A will now contain the decimal value of data found in memory located at decimal address equal to the value of B.

3.5.7 LINE Statement

The LINE statement specifies the number of character positions capable of being printed on a single line by the program. If the printer position is within the last 25% of the line length specified with this statement and a "space" character is to be printed, a carriage return and line feed operation will be performed to move the printer to the start of the next line.

This is done to prevent separation of a word or number between two lines. However, if this feature is not desired, it can be effectively inhibited by setting the total line length to a value which is at least 133% of the desired line length.

```
Example: 810 LINE=60
```

would mean a 60 character line length. The first "space" after the first 45 characters would cause the automatic carriage return/line feed operation.

3.5.8 DIGITS Statement

The DIGITS statement establishes the number of digits printed to the right of the decimal point. Digits in excess of the number specified by the DIGITS statement will be truncated (cut off, not rounded). If the value to be printed contains fewer digits than the number specified by the DIGITS statement, the remaining positions will be zero-filled. If DIGITS=0 is specified, the value will be printed in the default floating point mode (omitting non-significant zeros and un-needed decimal point). The maximum value which can be specified for the digits is 9.

```
Example: 130 DIGITS=4
         140 A=123.456789
         150 PRINT A
```

The above example will print as: 123.4567

3.6 DOCUMENTATION STATEMENTS

Documentation statements are used to add comments to a BASIC program. Only one documentation statement is used in BASIC - the REM statement.

3.6.1 REM Statement

The REM statement (from REMark) causes the comment on that line to be printed during a LIST of the program only. It is not executed and not printed during execution. The REM statement may contain up to 72 characters, including the statement number and spaces.

3.7 DEBUG STATEMENTS

Debug statements are used to assist the programmer in tracing through the execution of a program or to allow going back to EXbug with all its diagnostic powers. The three statements of this type are described in the following paragraphs.

3.7.1 TRACE ON Statement

The TRACE ON statement causes the console to print each statement number as the statement is executed. This statement provides a valuable debug tool, since the user can easily follow the course of the program as it is being run. The statement number is printed in brackets [ ] to avoid confusion with other printout.

3.7.2 TRACE OFF Statement

The TRACE OFF statement turns off the debug trace feature.

3.7.3 PATCH Statement

The PATCH statement allows returning to EXbug from within a BASIC program. The EXbug features are now available to the user. When the user desires to return to BASIC, the EXbug command ;P is typed on the console keyboard, and the system will return to the BASIC program at the step following the PATCH statement, or to the command mode of BASIC if PATCH was used as a command. Refer to Appendix F (RETURNING TO MINIBUG/MICROBUG) for ROM-Resident BASIC Interpreter differences.

CAUTION: This will not work as described if the program counter pseudo register or the interrupt vectors are changed during the use of EXbug.

CHAPTER 4

COMMANDS

4.1 INTRODUCTION

BASIC provides a comprehensive set of commands which are used to control the operation of the interpreter. These commands permit such operations as program loading, editing, execution, and saving. All BASIC commands are typed without statement numbers and are executed immediately. The following paragraphs list the commands used in BASIC and describe the operation of each. Most statements discussed in chapter 3 may also be used as commands by omitting the statement number.

4.2 SYSTEM COMMANDS

System commands are used to immediately input or output data from or to a system peripheral device. The system commands are described in the following paragraphs.

4.2.1 LOAD Command

The LOAD command clears the memory and automatically loads into memory a BASIC program previously saved on either cassette or paper tape by the SAVE command. This command is NOT used for loading from diskette.

4.2.2 APPEND Command

The APPEND command operates in exactly the same manner as the LOAD command except that the memory is not cleared. APPEND adds to the existing program. This command is NOT used for loading from diskette.

4.2.3 SAVE Command

The SAVE command is used only when the user desires to make a copy of the program currently stored in memory, onto either cassette or paper tape. The SAVE command causes BASIC to produce the necessary control signals to the read/record mechanism or paper tape reader/punch. This command is NOT used to save programs on diskette (see EXIT). Refer to Appendix F for ROM-Resident BASIC Interpreter differences.

4.2.4 EXIT Command

The EXIT command is used in either of two ways, depending upon whether disk BASIC or tape BASIC is being used. If BASIC is stored on cassette or paper tape, the EXIT command returns control directly to EXbug. If BASIC is stored on diskette, the EXIT command first prompts the user with the message:

SAVE (Y/N)?

If the user enters a Y, then the program currently stored in memory will be copied onto the diskette under the new file name (or old file name if no new one was specified) specified on the original disk operating system command line (see 2.1) and then control is returned to the disk operating system.

If the user enters an N, control is returned directly to the disk operating system (MDOS).

4.2.5 LIST Command

The LIST command is used to print a listing of the current program on the console or line printer. The LIST command has two forms:

```
LIST [sn1][,sn2]
LIST #2[,sn1][,sn2]
```

The first form is used to direct the listing to the console while the second directs the listing to the line printer. If no statement numbers are specified, the entire program is listed. When only the first statement number (sn1) is specified, the single line with this number is printed. If both statement numbers (sn1,sn2) are specified, the lines corresponding to this inclusive range of numbers are printed.

The following examples present the various uses of this command.

LIST	list all statements
LIST #2	list all statements on line printer
LIST 320	list statement number 320
LIST #2,100,450	list all statements from statement number 100 to statement number 450 inclusive on the line printer.

Refer to Appendix F for ROM-Resident BASIC Interpreter differences.

4.3 PROGRAM EXECUTION COMMANDS

Program execution commands are used to execute previously entered program statements. BASIC uses four program execution commands: RUN, CONT, NEW, and the Break Key. These commands are listed and explained in the following paragraphs.

4.3.1 RUN Command

The RUN command causes the program currently stored in memory to begin execution starting at the first (lowest) statement number. The RUN command also resets all program parameters and initializes all variables to zero.

The "GOTO sn" can be used as a command in cases where it is desired to begin execution at statement number "sn" instead of the first statement, and/or where it is desired NOT to clear all variables at the start of execution.

4.3.2 CONT Command

The CONT command (CONTinue) causes the program to continue after a STOP statement or "break" key has been encountered in the program. The program will continue with the statement following the STOP statement or where interrupted by "break". This command cannot be used if an error was encountered in the program or if the program has been changed after a STOP statement or "break" key depression.

4.3.3 NEW Command

The NEW command causes the working storage area in memory to be cleared as shown in the example below. The effect of using this command is to erase all of the program currently in memory.

```
#LIST
10 REM DEMO OF NEW
20 STOP

#NEW
#LIST

#
```

4.3.4 BREAK KEY Command

Depressing the BREAK key on the console keyboard will cause the BASIC program to halt execution and enter the command mode, responding with the word READY and the # prompt sign. BASIC will now accept commands or allow the program to be modified. The BREAK key can be used to stop a LIST command before completion or to halt the execution of a program. Refer to Appendix F (CONTROL C COMMAND) for ROM-Resident BASIC Interpreter differences.

4.4 EDITING COMMANDS

Two commands are used to aid in editing a program being entered.

4.4.1 CONTROL X Command (Cancel)

Depressing the X key while the CONTROL key is held down will clear the current line buffer. The system will respond by printing the word DELETED and moving the printer to the start of a new line.

4.4.2 RUBOUT or DEL Command (Backspace)

Depressing the RUBOUT or DEL key on the terminal keyboard will delete the last character entered on the current line each time the key is depressed. The character just deleted will be echoed back. BASIC will prevent deletion past the start of the current line.

4.5 SIZING COMMANDS

Sizing commands are used to establish the maximum number of characters per line and the maximum number of digits to the right of the decimal point that the user will permit to be printed. Two sizing commands, LINE and DIGITS, are described as statements in sections 3.2.7 and 3.2.8.

CHAPTER 5

FUNCTIONS

5.1 INTRODUCTION

Intrinsic functions are program functions which are inherent to the BASIC Interpreter itself and therefore do not require a separate routine to be defined. Four types of intrinsic functions are included in BASIC: Control, Data, Character String, and Transcendental.

5.2 CONTROL FUNCTIONS

The intrinsic control functions permit the user to position the print head on the printer or to ascertain its position. The two control functions provided are TAB and POS.

5.2.1 TAB Function

The TAB function moves the print head to a specified print column. This function is similar to the typewriter function of the same name. If the print head is already past (to the right of) the position specified by the TAB function, the print head will not move.

The first print position (left side) is position number 1. The print head position may be expressed by a numeric variable or constant. More than one use of the TAB function may be made in a single print statement.

```
Example: 430 PRINT TAB(23);A;TAB(C7);E$
```

5.2.2 POS Function

The POS function returns the value of the present position of the print head. This is the reverse of the TAB function.

```
Example: 620 PRINT A;U$;W$;
         630 IF POS>66 THEN 900
         640 L7=POS
```

5.3 DATA FUNCTIONS

Data functions provide the user with simple access to certain useful operations.

5.3.1 RND Function

The RND(X) function produces a set of uniformly distributed pseudo-random numbers. If no argument is used with the function, it will be assumed to be zero. When a zero argument is used, the function will return a different

number in the range of 0.0000000 to 0.9999999 inclusive. If a number other than zero is used for the argument, then a specific random number will be returned each time (the same number each time).

```
Example: 320 N1=100*RND(0)
         330 N2=INT(RND*10)
```

In the above examples, N1 will be assigned a value between 0 and 99.99999 inclusive, and N2 will be 0 to 9 inclusive.

5.3.2 INT Function

The INT(X) function returns the largest integer less than or equal to the value of X, regardless of whether the value is positive or negative.

```
Example: 345 A=INT(3.14)
         350 B=-1.4
         355 C=INT(B)
```

After executing the above statements, A will equal 3 and C will equal -2.

5.3.3 ABS Function

The ABS(X) function returns the absolute value of X.

```
Example: 270 A=ABS(-7.3)
         280 B=88.3
         290 C=ABS(B)
```

After executing the above statements, A will equal 7.3 and C will equal 88.3.

5.3.4 SGN Function

The SGN(X) function returns the value of 1, 0, or -1 according to the sign of X (positive, zero value, or negative).

```
Example: 1130 A=5
         1140 B=-7
         1150 C=SGN(A)
         1160 D=SGN(B)
         1170 E=SGN(0)
```

After execution of the above statements, C will equal 1, D will equal -1, and E will equal 0.

5.3.5 PEEK Function

This function is described in paragraph 3.5.6.

5.4 CHARACTER STRING FUNCTIONS

The character string functions are used to perform various operations such as extracting substrings or conversion between string and numeric.

5.4.1 LEN Function

The LEN(X$) function returns the number of characters contained in the string specified in the argument.

```
Example: 820 LO=LEN("THE QUICK")
         830 B$="ABCDEF"
         840 L1=LEN(B$)
```

The value of LO will be 9 and the value of L1 will be 6 after the above statements are executed.

5.4.2 ASC Function

The ASC(X$) function returns the ASCII equivalent numeric value (in decimal) of the first character of the string in the argument.

```
Example: 330 B$="ABCDEF"
         340 V=ASC(B$)
         350 W=ASC("3")
```

V will have the value 65 (the ASCII decimal value of "A") and W will have 51 after execution of the above statements. See Appendix B for ASCII to decimal conversion.

5.4.3 CHR$ Function

The CHR$(X) function returns the single string character equivalent in value to the argument specified. This is the inverse of the ASC function.

```
Example: 880 B$=CHR$(65)
         890 W=51
         900 C$=CHR$(W)
```

B$ will now contain the single character "A" and C$ will contain "3". Note that this is a method which can be used to print the double quote (") character.

5.4.4 VAL Function

The VAL(X$) function returns the numeric value equivalent to the numeric string specified in the argument. The string must contain a valid number (alphabetic characters not permitted except in the form 0.123E-4).

```
Example: 1500 A$="123.45"
         1510 A=VAL(A$)
```

```
1520 B=VAL("-87.654")
```

A would equal 123.45 and B would equal -87.654 after execution of the above statements.

5.4.5 STR$ Function

The STR$(X) function returns the string representation of the numeric value specified by the argument.

```
Example: 770 A=123.45
         780 A$=STR$(A)
         790 B$=STR$(-87.654)
```

A$ would now contain 123.45 and B$ would contain -87.654. Note that this is the inverse of the VAL function.

5.4.6 LEFT$ Function

The LEFT$(X$,N) function returns a substring of characters from the string of characters specified by the argument (X$). The substring returned begins with the leftmost character of the string X$ and continues for a total of N characters.

```
Example: 400 A$="THE QUICK BROWN"
         410 T=7
         420 B$=LEFT$(A$,T)
         430 C$=LEFT$("ABCDEF",4)
```

The string variable B$ will now contain "THE QUI" and C$ will contain "ABCD"

5.4.7 RIGHT$ Function

The RIGHT$(X$,N) function returns a substring of characters from the string of characters specified by the argument. The substring returned consists of the N right-most characters of the argument string.

```
Example: 990 A$="THE QUICK BROWN"
         1000 T=7
         1010 B$=RIGHT$(A$,T)
         1020 C$=RIGHT$("ABCDEF",4)
```

The string variable B$ will now contain "K BROWN" and C$ will contain "CDEF".

5.4.8 MID$ Function

The MID$(X$,Y,Z) function returns a substring of characters from the string of characters specified by the first argument. The substring will consist of characters beginning at character position Y of the argument string, and continuing for a total of Z characters.

```
Example:  30 A$="THE QUICK BROWN"
          40 T=5
          50 B$=MID$(A$,T,6)
```

will return the string "QUICK " in B$.

5.5 TRANSCENDENTAL FUNCTIONS

Transcendental functions provide the user with an easy method of performing more complex calculations than provided by other BASIC operations. These functions are listed and explained below. The results obtained from these functions are only accurate to six significant digits.

FUNCTION	EXPLANATION
SIN(X)	Returns the SINE of X (X is in radians).
COS(X)	Returns the COSINE of X (X is in radians).
TAN(X)	Returns the TANGENT of X (X is in radians).
ATAN(X)	Returns ARC TANGENT of X (X is in radians).
LOG(X)	Returns the NATURAL LOGARITHM of X.
EXP(X)	Returns the base of the natural logarithm raised to the Xth power (inverse of LOG).
SQR(X)	Returns the SQUARE ROOT of X.

5.6 USER DEFINED FUNCTIONS

In addition to the functions previously described, BASIC also permits user defined functions or calling of machine language subroutines.

5.6.1 DEF Function

The DEF function permits the user to create a function which can be used within a program. To define this type of function, the statement starts with DEF, followed by the letters FN and one additional alphabetic letter, followed by a non-subscripted numeric variable enclosed in parentheses, and finally followed by the equal sign and any valid expression.

Format: DEF FNx(v)=expression
where x is a single letter A-Z and v is a variable name.

The user may then call upon this defined function by the three letter name (FNA through FNZ) just as he would any other function. A function MUST be defined before it is used and may NOT be defined more than once in any program.

A brief program is shown in the example below to illustrate the use of the DEF and a user-defined function.

```
Example: 100 DEF FNC(X)=2*3.1415926*X
         110 PRINT "ENTER RADIUS";
         120 INPUT R
         130 C=FNC(R)
         140 PRINT #2,"A CIRCLE WITH RADIUS OF ";R;
         150 PRINT #2,"HAS A CIRCUMFERENCE OF ";C
         160 STOP
```

5.6.2 USER Function

The USER function is used to call machine language subroutines to perform some operation or calculation not possible or desirable to write using BASIC. (If using the ROM-Resident BASIC Interpreter, refer to Appendix F for proper memory addresses.)

The form of the function is:

```
340 A=USER(B)
```

If no "user subroutine" is present, the above statement will act as a simple assignment (A=B). However, if a "user-subroutine" is written, the value of B may be passed to the subroutine, and one value as a result of the subroutine may be passed back to A.

To call this "user subroutine", its starting address must be stored in memory locations 67 and 68 (hexadecimal) and it must contain an "RTS" (op-code hexadecimal 39) at the subroutine's logical end.

The "user subroutine" can pass the arguments by getting the data at a location whose starting address is found in 5D and 5E (hexadecimal) of memory and storing the results there. The data is stored in a 7 byte area beginning at the address found in 5D and 5E. The first through fifth bytes contain the value in BCD 10's complement representation. The seventh byte contains the exponent in 2's complement representation, adjusted so the decimal point is to the left of the leftmost non-zero digit of the number. The sixth byte is zero.

Examples: 137.1 would be represented as
01 37 10 00 00 00 03

while -0.0012 would be represented as
98 80 00 00 00 00 FE

CHAPTER 6

DISK FILE I/O (MDOS ONLY)

6.1 INTRODUCTION

The MDOS version of BASIC allows the user to create and manage sequential disk data files. Disk data files are similar to data defined via the DATA statement. Like the DATA statement, the use of disk files allow the user to define data prior to executing a program and to read it with the INPUT statement. Unlike the DATA statement, data files may also be created with the PRINT statement.

Up to three data files may be accessed at one time. All data files must be in ASCII format (type 5). The statements used in disk file operations are explained in this chapter.

6.2 OPEN Statement

The OPEN statement assigns a file number to an MDOS file name, and prepares the file for future data transfers. This file number is then used with the PRINT, INPUT, RESTORE, and CLOSE statements. A maximum of three data files may be open at one time. Each is opened by the OPEN statement as follows:

```
sn OPEN #n,filename[.suffix][:lun],m
```

where: sn is the BASIC statement number.

n is the file number (an integer from 3 through 9).

filename is the MDOS data file name.

suffix is the filename suffix (default is SA).

lun is the logical unit number (default is 0).

m is one of the following data transfer modes:

I is open input only, file name must exist.

O is open output only, file name must NOT exist.

U is open update, file may or may not exist.

Examples of the OPEN statement are:

```
140 OPEN #5,DATA45,I
150 OPEN #3,PFILE.DF:1,O
```

The file DATA45.SA on drive 0 must exist and will be opened for reading data from it only. File PFILE.DF on drive 1 will be created and opened for writing to it only. It must not have previously existed.

The U mode is most useful for appending data to the end of an existing data file. The program must first read to the end of old data, then additional data may be written to the file. If the last operation on a file open for update was a write, then the remainder of the file is automatically truncated upon closing the file. A data file may be deleted in BASIC by opening in the U mode and then closing without reading or writing the file. Therefore, caution must be observed when using the U mode.

The OPEN statement may be used in the direct mode as a command.

6.3 CLOSE Statement

The CLOSE statement releases the file number associated with an MDOS file name. If a file opened for output (O mode) is not properly closed, part or all of the last record (or last several records) may be lost. In addition, the file may contain extraneous data at the end of the file.

The format of the CLOSE statement is

```
sn CLOSE #n
```

where sn and n are the statement and file numbers.

When the execution of a program is abnormally terminated by an error or the BREAK key, the files are NOT released and NOT closed. Prior to running the program again, any open files must be closed.

The CLOSE statement may be used in the direct mode as a command.

A file opened for output and not written to, will, upon closing, be deleted from the MDOS diskette directory.

6.4 RESTORE Statement

When used in conjunction with a disk file, the RESTORE statement resets the file pointers to the beginning of a file, so that subsequent access will start at the beginning of the file.

The format of the RESTORE statement is

```
sn RESTORE #n
```

where sn and n are the statement and file numbers.

This statement is valid only for files opened in the input or update modes. The RESTORE statement may be used in the direct mode as a command.

6.5 INPUT Statement

Records from the disk file may be read via the INPUT statement.

```
sn INPUT #n, list
```

where sn and n are the statement and file numbers and list is a list of one or more variable names separated by commas.

This statement can only be used for a file open for the input or update modes and will read a record at a time (a record may consist of 1 or more fields). The INPUT statement may be used in the direct mode as a command.

6.6 PRINT Statement

Data records may be written to an MDOS file by the PRINT statment.

```
sn PRINT #n, list
```

where sn is the statement number, n is the file number, and list is a list of one or more variable names and/or constants, literals, or expressions.

Numeric data and string data will be written to the disk file in the same format as if they were being printed on a printer. Therefore, the list items on a PRINT statement may be separated by either a comma or semicolon, and in addition, the list may contain a literal string enclosed in quotes.

This statement is valid only for files opened in the output or update mode and will write one record to the disk.

6.7 EXAMPLES OF DISK FILE I/O

The following examples will serve to illustrate the use of the disk file I/O statements within a program.

6.7.1 Creation of a Data File

```
100 OPEN #3,DF1,0
110 FOR J=1 TO 20
120 K=J*3.1415926
130 PRINT #3,"J= ";J;"K= ";K
140 NEXT J
150 CLOSE #3
160 STOP
```

The above program will create a file named DF1.SA on drive 0 which will contain data in the form of:

```
J= 1 K= 3.1415926
J= 2 K= 6.2831852
...etc...
```

for values of J from 1 to 20.

6.7.2 Reading a Data File

```
100 OPEN #7,DF1,I
110 FOR A=1 TO 20
120 INPUT #7,D$,X,E$,Y
130 PRINT #2,D$;X;E$;Y
140 REM ** THE ABOVE LINE WILL PRINT ON LINE PRINTER
150 NEXT A
160 CLOSE #7
170 STOP
```

This will read the file previously created and print the contents on the line printer. Note that the variable list in the INPUT statement must correspond in type and order to the data in the file.

6.8 End of File Detection

MDOS BASIC does not have end of file (EOF) detection without causing an error, so the user must devise a method of detection of the end of file. This is often accomplished by using an EOF record containing data in one or more fields which will not occur for valid records. For example, if the first field of each record contained an employee number, and the maximum digits possible is four, then the use of a numeric value of 99999 might be a choice to detect the EOF.

6.9 Updating a Data File

Updating takes the form of deleting a record, adding a record, or changing a record. In either case, the technique is the same. The following are the steps a programmer may take to update an MDOS BASIC data file:

1. Open the old data file for input (I).
2. Open a new scratch data file for output (O).
3. Input a record from the old data file.
4. If this record is to be deleted, go back to step 3.
5. If this record is to be altered, change any data.
6. Write this record to the scratch file.
7. If a new record is to be inserted before the next old record, create it and write it to the scratch file.
8. Proceed through the complete old data file.
9. When done with the old file, close the old data file and the scratch file.
10. Open the old data file in U mode.
11. Close the old data file. (This deletes it.)
12. Open a new file with old data file name (O).
13. Open the scratch file (I).
14. Copy, record by record from scratch to data files.
15. Close both files.
16. Open scratch file in U mode, then close to delete it.

6.10 Deleting a Data File

As can be seen from the previous section, a data file may be quite easily deleted with a BASIC program by opening in the U mode and then closing the file. However, if the user desires protection from this happening accidentally, delete protection should be placed on the data file using the MDOS NAME command. A delete protected file cannot be deleted by a BASIC program.

6.11 Data File Data Representation

The data in an MDOS BASIC created data file is stored in ASCII format with space compression and is compatible with the MDOS editor and other MDOS commands such as LIST. The BASIC created data file may be edited, and conversely a file intended for data input to a BASIC program may be created by the editor or other means.

APPENDIX A

DECIMAL ↔ HEXADECIMAL CONVERSION CHARTS

From hex: locate each hex digit in its corresponding column position and note the decimal equivalents. Add these to obtain the decimal value.

From decimal: (1) locate the largest decimal value in the table that will fit into the decimal number to be converted, and (2) note its hex equivalent and hex column position. (3) Find the decimal remainder. Repeat the process on this and subsequent remainders.

HEXADECIMAL COLUMNS											
6		5		4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
7 6 5 4		3 2 1 0		7 6 5 4		3 2 1 0		7 6 5 4		3 2 1 0	
BYTE				BYTE				BYTE			

POWERS OF 2

2^n	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

$2^0 = 16^0$
$2^4 = 16^1$
$2^8 = 16^2$
$2^{12} = 16^3$
$2^{16} = 16^4$
$2^{20} = 16^5$
$2^{24} = 16^6$
$2^{28} = 16^7$
$2^{32} = 16^8$
$2^{36} = 16^9$
$2^{40} = 16^{10}$
$2^{44} = 16^{11}$
$2^{48} = 16^{12}$
$2^{52} = 16^{13}$
$2^{56} = 16^{14}$
$2^{60} = 16^{15}$

POWERS OF 16

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

APPENDIX B

ASCII TO DECIMAL CONVERSION

LSD	MOST SIGNIFICANT DIGITS 0	1	2	3	4	5	6	7	8	9	10	11	12
0	NUL	LF	DC4	RS	(	2	<	F	P	Z	d	n	x
1	SOH	VT	NAK	US	)	3	=	G	Q	[	e	o	y
2	STX	FF	SYN	SP	*	4	>	H	R	\	f	p	z
3	ETX	CR	ETB	!	+	5	?	I	S	]	g	q	{
4	EOT	SO	CAN	"	,	6	@	J	T	^	h	r	|
5	ENQ	SI	EM	#	-	7	A	K	U	_	i	s	}
6	ACK	DLE	SUB	$	.	8	B	L	V	`	j	t	~
7	BEL	DC1	ESC	%	/	9	C	M	W	a	k	u	DEL
8	BS	DC2	FS	&	0	:	D	N	X	b	l	v	
9	HT	DC3	GS	'	1	;	E	O	Y	c	m	w	

Examples: A is decimal 65
BEL is 7
x is 120

APPENDIX C

ENHANCING PROGRAM OPERATION

When preparing a program in BASIC, execution time can be reduced by observing the rules listed below:

1. The use of spaces should be limited since they consume both memory allocation and interpreter time.

2. Since BASIC must search for subroutines and functions, the user should arrange the program in such a manner that the most used subroutine or function is located at the start of the program, followed by the second most used, etc. If this is done, the time required for BASIC to search will be reduced.

3. Whenever possible, use non-subscripted variables. Subscripted variables take considerable processing time.

4. Using the arithmetic functions listed in section 5.5 (SIN, COS, TAN, etc.) is time consuming. Therefore, use these functions only when necessary. Exponentiation uses both the LOG and EXP functions, making exponentiation quite slow. To square a number, use A*A instead of A©2.

5. BASIC searches the symbol table for a referenced variable. Variables are inserted into the table as they are referenced. Therefore, time can be saved if a frequently used variable is used early in the program so that this variable is located near the start of the symbol table.

6. Numeric constants are converted each time they are encountered in the program. Therefore, if a constant is used often, assign it to a variable and use the variable instead.

APPENDIX D

MINIMIZING MEMORY REQUIREMENTS

When writing a BASIC program, the size of memory required for the program can be reduced by observing the rules listed below:

1. REM (Remark) statements take space in memory, so use them sparingly. One byte is required for each character in the remark.

2. Memory requirements for variables:

   Each non-subscripted numeric variable requires 8 bytes.

   Each non-subscripted string variable requires 20 bytes.

   Each numeric array requires 6 bytes plus 6 bytes per element.

   Each string array requires 6 bytes plus 18 bytes per element.

3. An implicitly dimensioned variable creates a 10 element array (or 10 by 10 if two dimensions are used). Therefore, if you do not intend to use 10 elements, save memory by explicitly dimensioning the variable.

4. Each line in the program requires 2 bytes for the statement number, 1 byte for the keyword (first word in a statement), and 1 byte for every character following the keyword including spaces, and 1 byte for the end of line terminator. By using as few spaces as possible, memory can be reduced.

5. BASIC requires the use of the first 256 bytes of memory for scratchpad use.

APPENDIX E

ERROR MESSAGES

Various errors encountered in a BASIC program during execution will result in BASIC printing an error message. Each error message number is defined below and is displayed in the following format:

ERROR #03 IN LINE 0110

Line 0000 means that the error was at the "command" level and not caused by a specific program statement.

ERROR #	MEANING
01	Variable exceeds maximum (255) or is negative when used with TAB, POKE, CHR$, or the ON statement.
02	INPUT statement error.
03	Illegal character or variable.
04	No ending quote mark.
05	DIM error.
06	Illegal arithmetic or improper relational operator.
07	Statement number not found.
08	Divide by zero attempted.
09	Excessive subroutine nesting (8 max.).
10	RETURN used without prior GOSUB statement.
11	Illegal variable.
12	Unrecognizable statement.
13	Parentheses error.
14	Memory full.
15	Subscript error.
16	Excessive FOR - NEXT loops (maximum of 8).
17	NEXT without FOR.
18	FOR - NEXT nesting error.

19	READ statement error.
20	ON statement error.
21	Input overflow. Input line exceeds 72 characters.
22	DEF function syntax error.
23	Function syntax error or function not defined.
24	STR$ usage error or mixing of numeric and string variables.
25	String buffer overflow or substring too long.
26	I/O operation error.
27	VAL function error.
28	Invalid or duplicate file number.
29	Maximum of 3 data files already open.
30	Invalid file name.
31	Invalid mode type (not I, O, or U).
32	Data file cannot be opened, closed, or accessed.
33	MDOS file is not in ASCII format.
34	Disk data record format error.
35	LOG function error.
36	BASIC program cannot be altered and continued from a STOP.

APPENDIX F

ROM-RESIDENT BASIC INTERPRETER

INTRODUCTION

BASIC is also supplied as a 7K-byte firmware package to be used in conjunction with the MINIbug II and MICRObug monitors. Source statements, as they are entered via the console, are stored in the system workspace buffer. A Random-Access-Memory must be available at this address, the capacity of which will determine the maximum size of the BASIC source program and the amount of elementary variables which can be accommodated in the user's installation. 8K bytes of R/W storage provide a comfortable RAM capacity. The size of the workspace buffer may be expanded up to 16K bytes.

The operator communicates with the BASIC Interpreter via a console which is linked to the appropriate monitor - MINIbug/MICRObug. The console may include the following peripherals: keyboard, printer, reader/punch unit, cassette recorder, CRT.

The ROM-Resident BASIC Interpreter is basically identical to that of the EXORciser-Resident BASIC Interpreter. This appendix describes the differences. The term MINIbug, as used in this appendix, can be considered to refer to the MINIbug II and MICRObug monitors.

INVOKING THE BASIC INTERPRETER

The BASIC Interpreter may be invoked whenever MINIbug performs its main control loop; at that time, an asterisk is displayed on the console. Typing the execute command "G 4000" causes the interpreter to be entered. The interpreter then indicates its readiness by displaying the following message:

```
ROM BASIC 1.31
READY
#
```

Prior to doing this, the workspace buffer has been cleared and sized automatically; therefore, any BASIC source program which may have been entered previously would be lost. This is due to the fact that BASIC was entered via its cold-start entry point - 4000 (assuming the ROM-set is installed in a Micromodule 4 or Editor/Assembler/BASIC Module).

A second entry point (hex address 4006) provides a means to gain access to the interpreter without altering the workspace buffer which may store a BASIC source program.

This warm-start entry point will normally be reached whenever the operator desires to re-enter the Interpreter from MINIbug. This may occur when MINIbug is accessed from Interpreter as a result of the BASIC "PATCH" command, or as a result of a hardware reset condition (the RESTART pushbutton is activated).

The BASIC "NEW" command provides a third way to re-initialize the Interpreter operations. The effect of the NEW command, typed while already under control of BASIC, is the same as when entering the Interpreter cold-start address - i.e., the workspace buffer is automatically cleared.

RETURNING TO MINIbug/MICRObug

Should the operator desire to re-enter the MINIbug monitor while control is under the Interpreter, he may either:

1. push the RESTART button, or
2. type in the BASIC "PATCH" command.

In either case, care must be exercised when re-entering the Interpreter, as described in the above paragraph.

SAVING THE SOURCE PROGRAM

After the BASIC source program has been successfully tested, it may be saved on paper tape or cassette. The BASIC "SAVE" command is used to that end. "SAVE" issues the necessary commands to control a paper tape or digital cassette unit.

When an audio record is used (M68ADS equipped with the M68CIM Cassette Interface Module), make sure that the baud rate is set to 300. Rewind the tape. Type in "SAVE", followed by a carriage return. As soon as a series of squares is appearing on the ADS screen, push the RECORD button of the recorder. The program will be recorded. When the # sign is displayed on completion of the record operation, push the recorder STOP button.

SENDING OUTPUT TO A LINE PRINTER

By specifying a port address of #2 in a PRINT or LIST command, output will be directed to a line printer if the following steps are implemented:

1. Write the software driver corresponding to its hardware (PIA) interface.

2. Locate this driver at base address $6000.

3. Provide the hardware interface (PIA).

4. Use a 2708 type PROM programmer, such as MEX68PP3, to change locations $5BEF/F0 in the last ROM (BA7) from the default console address to the line printer driver base address, $6000.

5. Output is sent to the driver character by character via the A accumulator.

6. The B accumulator and X-Reg must not be altered by the user-written driver.

7. It is the user's responsibility to initialize the hardware interface (PIA'S).

The Interpreter does not make any assumptions about the hardware/software interface. A sample driver program is shown, which is also compatible with the ROM Memory Assembler/Editor, version 1.1. Refer to Table F-1 for other ROM BASIC constant and jump vector locations.

TABLE F-1. ROM BASIC Constant and Jump Vector Locations

LOCATION(S)	CONTENTS
	CONSTANT TABLE
$5BE7/8	Start of workspace buffer address
5BE9/A	Monitor top of stack address
5BEB/C	Console input, no echo subroutine address
5BED/E	Console output subroutine address
5BEF/F0	User-supplied line printer driver address (console address, as supplied)
5BF1/2	ACIA Control/Status register address
5BF3/4	ACIA Data Register address
5BF5/6	Address of ACIA Control register setup byte
	JUMP TABLE (DO NOT MODIFY 1ST BYTE)
5BF7/9	Jump to output 2 hex characters subroutine
5BFA/C	Jump to output 4 hex characters + blank subroutine
5BFD/F	Jump to monitor entry point.

CONTROL X COMMAND

Depressing the X key while the CTRL key is held down causes the current line buffer to be cleared and the system to move the line printer to the beginning of a new line. The word DELETED is not printed as in the EXORciser-resident BASIC version. See paragraph 4.4.1.

CONTROL C COMMAND

Depressing and holding down the CTRL key on the terminal keyboard and then depressing the C key will cause the BASIC program to halt the current operation and to respond with the word READY. The # prompt is not printed as it is for the equivalent command - BREAK key - used with the EXORciser-resident BASIC. See paragraph 4.3.4. After the word READY has been printed, the BASIC program will then accept further commands. The CTRL C command is often used to stop a LIST command before completion, or to halt the execution of a program.

USER FUNCTION

The ROM BASIC "user function" operates the same as the RAM version except the addresses are different, as shown below:

USER SUBROUTINE	RAM BASIC	ROM BASIC
Starting Address	$0067/8	$0049/A
Pointer to data arguments	$005D/E	$003F/40

See paragraph 5.6.2 for more information.

INSTALLING NEW 7K ROM BASIC VERSION 1.31

a. Replace the old B1 ROM (51AW1565X22) with the new B1 ROM (51AW1565X45).

b. For MINIBUG BASIC, replace the old B7 ROM (51AW1565X28) with the new B7 ROM (51AW1565X46).

c. For MICROBUG BASIC, replace the old B7 ROM (51AW1565X29) with the new B7 ROM (51AW1565X47).

d. The other ROM's, B2 (51AW1565X23) through B6 (51AW1565X27), remain the same.

TESTING ROM BASIC PROGRAM INTEGRITY

A checksum byte (the two's complement of the sum of all bytes in the ROM's) is included in 7K ROM BASIC, version 1.31. Therefore, the total of all bytes, excluding the constant and jump vector locations as shown in Table F-1 in the 7K program, is zero for both the MINIbug and MICRObug versions. Inclusion of this checksum byte (location $5BE6) makes it easy to test the integrity of the program through the use of the simple routine provided below, which sums all bytes in the A accumulator. The routine is position independent and may be entered into any

available RAM through the system console, using the memory examine and change function provided by the system monitor. Use the monitor go-to location and execute function to start the test. Since, on completion, execution of the SWI instruction causes the MPU registers to be displayed, inspection of the A accumulator gives the result: pass = Ø, fail ≠ Ø. The same technique can also be used to monitor the integrity of programs stored in user EPROM's during program development.

```
                                                Comments Column

$0000 CE 6400   START   LDX  #$4000
$0003 4F                CLRA
$0004 AB 00     LOOP    ADDA X          SUM ROM BYTES $4000-$5BE6
$0006 08                INX
$0007 8C 8000           CPX  #$5BE7
$000A 26 F8             BNE  LOOP
$000C 3F                SWI             ** BREAKPOINT A REG = TOTAL **
```

```
00001                       NAM    LPREX
00002                       TTL    *** SAMPLE USER LPR PROG. FOR MAE68
00003                       OPT    CREF

00005                 * 10 MAY 1979:
00006                 *
00007                 * LINE PRINTER DRIVER FOR CENTRONICS TYPE
00008                 * INTERFACE THROUGH A PIA WITH OUTPUT
00009                 * CHARACTER ON A SIDE, INPUT STATUS ON B SIDE
00010                 *
00011                 * FOR 2MHZ OR LESS MPU CYCLE TIME
00012                 * THIS IS THE SIMILAR TO THE RESIDENT MDOS
00013                 * CONTROLER ROM LINE PRINTER ROUTINES.
00014                 *       VER. 1.2  19 FEB 79
00015                 * COPYRIGHT 1978, 1979 BY MOTOROLA INC
00016                 *


00018                 * LINE PRINTER PIA ADDRESSES *
00019                 * -------------------------
00020                 *
00021                 *   CAUTION:  THE PIA ADDRESSES MUST BE PROPERLY
00022                 *   -------   SELECTED FOR COMPATIBILITY WITH
00023                 *             EACH SYSTEM'S MEMORY MAP TO AVOID
00024                 *             REDUNDANT AND/OR NON-VALID MEMORY
00025                 *             SECTIONS!
00026                 *
00027          EC10   A DATA   EQU    $EC10    *** EXBUG COMPATIBLE ***
00028          EC11   A CNTRL1 EQU    $EC11    .   ---------------
00029          EC12   A STAT   EQU    $EC12
00030          EC13   A CNTRL2 EQU    $EC13

00032          4000   A ROMBAS EQU    $4000    START OF 7K ROM BASIC


00034                 ************************
00035                 * LPR CALLING VECTORS *
00036                 ************************
00037                 *
00038A 6000                    ORG    $6000
00039A 6000 7E 602A   A LPROUT JMP    LISTW    SEND CHAR IN AR TO LPR
00040                 *                          (WAIT IF ERROR)
00041A 6003 7E 6017   A LPRINT JMP    LPINIT   INIT. LPR (PIA'S)
00042                 * NOTE: THE ABOVE TWO VECTORS ARE REQUIRED 'JUMP'
00043                 *       TYPE VECTORS FOR MAE68 VER. 1.1 & UP!
00044                 *-----------------------------------------------
00045                 *
00046                 * OPTIONAL, BUT USEFUL VECTORS FOLLOW:
00047A 6006 7E 602F   A LPRO   JMP    LIST     SEND CHAR IN AR TO LPR
00048                 *                          (EXIT IF ERROR)
00049A 6009 BD 6003   A BASIC  JSR    LPRINT   START ROM (7-K) BASIC
00050                 *                          (INITIALIZES LPR!)
00051A 600C 7E 4000   A BASIC2 JMP    ROMBAS   START ROM BASIC W/O LPR IN
00052                 *
00053                 * OTHER VECTORS AS DESIRED CAN GO HERE!
```

```
00055                    * ACTUAL PRINTER ROUTINES HERE *


00057                    * STROBE PRINTER
00058           600F   A LIST5  EQU    *
00059A 600F 8D 02 6013          BSR    LIST7
00060A 6011 86 3C      A        LDAA   #$3C
00061A 6013 B7 EC11    A LIST7  STAA   CNTRL1
00062A 6016 39                  RTS


00064                    * SUBROUTINE TO INITIALIZE LPR PIA'S
00065           6017   A LPINIT EQU    *
00066A 6017 7F EC11    A        CLR    CNTRL1    OPEN DATA DIRECTION REG'S
00067A 601A 7F EC13    A        CLR    CNTRL2    .  (NO HDWR POWER UP RESET
00068A 601D CE FF3C    A        LDX    #$FF3C    A= DATA OUTPUT & CTRL WORD
00069A 6020 FF EC10    A        STX    DATA
00070A 6023 CE 003C    A        LDX    #$003C    B= STATUS INPUT & CTRL WOR
00071A 6026 FF EC12    A        STX    STAT      .  BIT 0= ON-LINE
00072A 6029 39                  RTS              .  BIT 1= PAPER EMPTY


00074                    * SUBROUTINE TO PRINT CHARACTER FROM A ACC
00075                    *   WAITS IF PRINTER ERROR
00076A 602A 8D 03 602F LISTW    BSR    LIST      SEND TO PRINTER
00077A 602C 25 FC 602A          BCS    LISTW     WAIT IF ERROR
00078A 602E 39                  RTS


00080                    * SUBROUTINE TO PRINT CHARACTER FROM A ACC
00081                    * AND CHECK FOR PRINTER ERROR
00082                    *   IF ERROR, CARRY IS SET ON RETURN (C=1)
00083           602F   A LIST   EQU    *
00084A 602F B7 EC10    A        STAA   DATA      SEND DATA
00085A 6032 8D 14 6048          BSR    EXIT      CLEAR ACKNOWLEDGE
00086A 6034 86 34      A        LDAA   #$34
00087A 6036 8D D7 600F          BSR    LIST5     SEND STROBE
00088A 6038 B6 EC12    A LIST3  LDAA   STAT      CHECK STATUS
00089A 603B 84 03      A        ANDA   #3        BIT 0=SELECT, BIT 1=PAPER
00090A 603D 4A                  DECA             .  (AR SHOULD HAVE BE =$01
00091A 603E 26 07 6047          BNE    LERROR    NO PAPER OR NOT SELECTED
00092A 6040 7D EC11    A        TST    CNTRL1    ACKNOWLEDGE? (CLEAR CARRY)
00093A 6043 2A F3 6038          BPL    LIST3     NO
00094A 6045 20 01 6048          BRA    EXIT      YES-ALL OK
00095                    *
00096                    * HERE FOR LPR ERROR EXIT
00097A 6047 0D           LERROR SEC              SET ERROR INDICATION
00098A 6048 B6 EC10    A EXIT   LDAA   DATA      RESTORE A
00099A 604B 39                  RTS
```

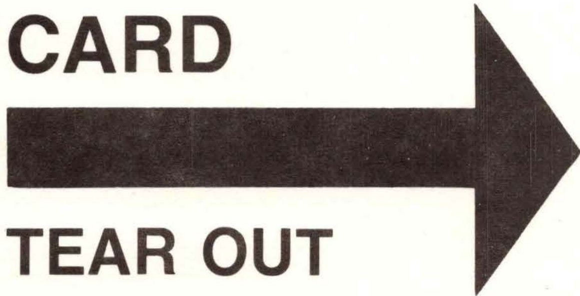
```
00101                          END
TOTAL ERRORS 00000--00000


   6009 BASIC  00049*
   600C BASIC2 00051*
   EC11 CNTRL1 00028*00061 00066 00092
   EC13 CNTRL2 00030*00067
   EC10 DATA   00027*00069 00084 00098
   6048 EXIT   00085 00094 00098*
   6047 LERROR 00091 00097*
   602F LIST   00047 00076 00083*
   6038 LIST3  00088*00093
   600F LIST5  00058*00087
   6013 LIST7  00059 00061*
   602A LISTW  00039 00076*00077
   6017 LPINIT 00041 00065*
   6003 LPRINT 00041*00049
   6006 LPRO   00047*
   6000 LPROUT 00039*
   4000 ROMBAS 00032*00051
   EC12 STAT   00029*00071 00088
```

APPENDIX G

BASIC INTERPRETER SUMMARY

The detachable reference card provided below summarizes the loading procedures, instructions, and error messages for the BASIC Interpreter program.

BASIC INTERPRETER REFERENCE CARD



TEAR OUT FOR HANDY POCKET REFERENCE

LOADING PROCEDURES (User Responses Underlined)

CASSETTE/PAPER TAPE

```
*E LOAD
SGL/CONT S
BASIC8
*E 100; G
M6800 BASIC 1.XX
COPYRIGHT (C) — 19XX
READY
#
```

DISKETTE (MDOS)

```
*E MDOS
=BASIC INFILE (or
 BASIC INFILE,OUFILE)
MDOS BASIC 2.XX
COPYRIGHT (C) — 19XX
READY
#
```

SUMMARY OF INSTRUCTIONS

STATEMENTS

INPUT/OUTPUT
- INPUT
- DATA
- READ
- RESTORE
- PRINT
- OPEN
- CLOSE

DECLARATION
- DIM

CONTROL
- FOR
- NEXT
- STOP
- END
- GOTO
- GOSUB
- RETURN
- ON
- IF
- THEN
- USER

ASSIGNMENT
- LET (OPTIONAL)
- Mathematical Oper.
- ^
- - (Unary)
- *
- /
- +
- −

REMARKS
- REM

COMMANDS

SYSTEM
- LOAD
- APPEND
- SAVE
- EXIT
- POKE
- LIST
- TRACE ON
- TRACE OFF

PROGRAM EXECUTION
- RUN
- CONT
- NEW

EDITING
- Control C
- Control X
- Control O
- PATCH
- Delete key
- Break key

SIZING
- LINE
- DIGITS

INTRINSIC FUNCTIONS

CONTROL
- TAB
- POS

DATA
- RND(X)
- INT(X)
- ABS(X)
- SGN(X)
- LEN(X$)
- ASC(X$)
- CHR$(X)
- VAL(X$)
- STR$(X)
- LEFT$(X$,N)
- RIGHT$(X$,N)
- MID$(X$,X,Y)
- PEEK(X)

MATHEMATICAL
- SIN(X)
- COS(X)
- TAN(X)
- ATAN(X)
- LOG(X)
- EXP(X)
- SQR(X)

USER DEFINED
- DEF

BASIC INTERPRETER

ERROR MESSAGE

ERROR #	MEANING
01	Oversize variable (over 255) or negative used with TAB, POKE, CHR$ or ON.
02	INPUT Statement Error.
03	Illegal character or variable.
04	No ending quotation mark.
05	DIM error.
06	Illegal arithmetic or improper relational operator.
07	Statement number not found.
08	Divide-by-zero attempted.
09	Excessive subroutine nesting. (8 max.).
10	RETURN without GOSUB.
11	Illegal variable.
12	Unrecognizable statement.
13	Parentheses error.
14	Memory full.
15	Subscript error.
16	Excessive FOR-NEXT loops (8 max.).
17	NEXT "X" without FOR loop defining "X".
18	FOR/NEXT nesting error.
19	READ statement error.
20	ON statement error.
21	Input overflow — line exceeds 72 characters.
22	Syntax error in DEF function.
23	Syntax error in function or function not defined.
24	STR$ usage error or mixing of numeric and string variables.
25	String buffer overflow or substring too long.
26	I/O operation error.
27	Error in VAL function usage.
28	Invalid or duplicate I/O file number.
29	Maximum of 3 data files already opened.
30	Invalid file name.
31	Invalid data transfer mode type (not I, O, or U).
32	Data file cannot be opened, closed, or accessed.
33	MDOS file is not in ASCII format.
34	Disk data record format error.
35	Log function error.
36	Continued program was altered.

SUGGESTION/PROBLEM REPORT

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Microsystems
P.O. Box 20912
Attention: Publications Manager
Mail Drop M374
Phoenix, Az. 85036

Comments

Product: ______________________ Manual: ______________________

Please Print

Name	Title	
Company	Division	
Street	Mail Drop	Phone Number
City	State	Zip

Hardware Support: (800) 528-1908
Software Support: (602) 831-4108