

MLBASIC

(REVISION 1.00)

EXTENDED

BASIC

COMPILER

COPYRIGHT (C) 1984

by

WASATCHWARE
Salt Lake City, Utah

MLBASIC

Revision 1.0

EXTENDED

BASIC

COMPILER

WASATCHWARE



NOTICE

WASATCHWARE has prepared this manual for use by customers of the basic compiler "MLBASIC". The information herein is the property of WASATCHWARE and shall not be reproduced in the whole or in part without WASATCHWARE's prior written approval.

WASATCHWARE reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the material presented, including but not limited to typographical, arithmetic, or listing errors.

MLBASIC 1.0 User's manual

Revision History:
Original Release -October 1984

ML BASIC

(REVISION 1.0)

EXTENDED

BASIC

COMPILER

COPYRIGHT (C) 1984

by

WASATCHWARE
Salt Lake City, Utah

Royalty Information

The policy for distributing compiled programs using
MLBASIC runtime subroutines is as follows:

- You can distribute and sell any application program that you generate by compiling with MLBASIC without payment of royalties. A copyright notice reading "PORTIONS COPYRIGHTED BY WASATCHWARE, 1984" must appear on the medium.
- You cannot duplicate any other software in the MLBASIC compiler package except to backup your software. Other duplication of any of the software in the MLBASIC compiler package is illegal.

PREFACE

This manual provides a step-by-step introduction to WASATCHWARE's MLBASIC compiler. It is intended for users who are unfamiliar with the BASIC compiler. Users who are familiar with MLBASIC can use this manual as a reference for procedures and technical information.

This manual assumes that you know how to program in BASIC. Examples of how MLBASIC syntax differs from interpreter BASIC syntax are given.

Chapter 1 introduces you to WASATCHWARE's MLBASIC Basic compiler. Chapter 2 provides a description of the entire compilation process. Chapter 3 explains the many commands that MLBASIC has to offer. Chapter 4 explains all about MLBASIC variables, constants and expressions. Chapter 5 is devoted to the advanced programmer who needs to know technical information about MLBASIC. Chapter 6 goes through compiling several programs that provide uses for many of MLBASIC's commands. Chapter 7 contains explanations of the error messages produced while compiling programs and when running programs.

CONTENTS

| | PAGE |
|---|------|
| CHAPTER 1 INTRODUCTION | |
| 1.1 Overview of MLBASIC..... | 1 |
| 1.2 System requirements..... | 1 |
| 1.3 Diskett/Tape contents..... | 1 |
| 1.4 Compilation Vs. Interpretation..... | 2 |
| 1.5 Program Developement..... | 3 |
| CHAPTER 2 HOW TO COMPILE A PROGRAM | |
| 2.1 Explanation of MLBASIC options..... | 4 |
| 2.2 Compiling a program using default values..... | 5 |
| 2.3 Storage options..... | 7 |
| 2.4 Maping options..... | 8 |
| 2.5 Listing options..... | 9 |
| 2.6 Number Base Option..... | 10 |
| 2.7 Compilemode Options..... | 11 |
| 2.8 Example Compilation using specified values..... | 12 |
| CHAPTER 3 MLBASIC COMMANDS | |
| 3.1 I/O Commands | |
| 3.1.a CLOSE..... | 16 |
| 3.1.b CLOADM..... | 17 |
| 3.1.c CSAVEM..... | 18 |
| 3.1.d DIR..... | 19 |
| 3.1.e DRIVE..... | 20 |
| 3.1.f DSKI\$..... | 21 |
| 3.1.g DSKO\$..... | 22 |
| 3.1.h FIELD..... | 23 |
| 3.1.i FILES..... | 24 |
| 3.1.j GET..... | 25 |
| 3.1.k INPUT..... | 26 |
| 3.1.l KILL..... | 28 |
| 3.1.m LSET..... | 29 |
| 3.1.n OPEN..... | 30 |
| 3.1.o PRINT..... | 31 |
| 3.1.p PUT..... | 33 |
| 3.1.q RSET..... | 34 |
| 3.2 Program Control Commands | |
| 3.2.a CALL..... | 35 |
| 3.2.b END..... | 37 |
| 3.2.c EXEC..... | 38 |
| 3.2.d FOR-(STEP)-NEXT..... | 39 |
| 3.2.e GOSUB..... | 40 |
| 3.2.f GOTO..... | 41 |
| 3.2.g IF-THEN-(ELSE)..... | 42 |
| 3.2.h OFF ERROR..... | 43 |
| 3.2.i ON ERROR..... | 44 |
| 3.2.j ON-GO(TO,SUB)..... | 45 |
| 3.2.k RETURN..... | 46 |
| 3.2.l STOP..... | 47 |
| 3.2.m SUBROUTINE..... | 48 |

| | PAGE |
|--|------|
| 3.3 Math Functions | |
| 3.3.a ABS..... | 49 |
| 3.3.b ASC..... | 50 |
| 3.3.c ATN..... | 51 |
| 3.3.d COS..... | 52 |
| 3.3.e CVN..... | 53 |
| 3.3.f EOF..... | 54 |
| 3.3.g EXP..... | 55 |
| 3.3.h FIX..... | 56 |
| 3.3.i INSTR..... | 57 |
| 3.3.j INT..... | 58 |
| 3.3.k LEN..... | 59 |
| 3.3.l LOG..... | 60 |
| 3.3.m LOC..... | 61 |
| 3.3.n LOF..... | 62 |
| 3.3.o PEEK..... | 63 |
| 3.3.p POINT..... | 64 |
| 3.3.q PPOINT..... | 65 |
| 3.3.r RND..... | 66 |
| 3.3.s SGN..... | 67 |
| 3.3.t SIN..... | 68 |
| 3.3.u SQR..... | 69 |
| 3.3.v TAN..... | 70 |
| 3.3.w TIMER..... | 71 |
| 3.3.x VAL..... | 72 |
| 3.4 String Functions | |
| 3.4.a CHR\$..... | 73 |
| 3.4.b INKEY\$..... | 74 |
| 3.4.c LEFT\$..... | 75 |
| 3.4.d MIDS..... | 76 |
| 3.4.e MKNS..... | 77 |
| 3.4.f RIGHT\$..... | 78 |
| 3.4.g STR\$..... | 79 |
| 3.4.h STRING\$..... | 80 |
| 3.5 Graphics and Sound commands | |
| 3.5.a AUDIO..... | 81 |
| 3.5.b COLOR..... | 82 |
| 3.5.c CLS..... | 83 |
| 3.5.d CIRCLE..... | 84 |
| 3.5.e DRAW..... | 85 |
| 3.5.f LINE..... | 87 |
| 3.5.g PAINT..... | 88 |
| 3.5.h PCLEAR..... | 89 |
| 3.5.i PCLS..... | 90 |
| 3.5.j PLAY..... | 91 |
| 3.5.k PMODE..... | 92 |
| 3.5.l PRESET..... | 93 |
| 3.5.m PSET..... | 94 |
| 3.5.n RESET..... | 95 |
| 3.5.o SCREEN..... | 96 |
| 3.5.p SET..... | 97 |
| 3.5.q SOUND..... | 98 |

| | | |
|--|--------------|-----|
| 3.6 Other Commands (Handled by Interpreter) | | |
| 3.6.a | DATA..... | 99 |
| 3.6.b | DIM..... | 100 |
| 3.6.c | LLIST..... | 101 |
| 3.6.d | MOTOR..... | 102 |
| 3.6.e | POKE..... | 103 |
| 3.6.f | READ..... | 104 |
| 3.6.g | REM..... | 105 |
| 3.6.h | RESTORE..... | 106 |
| 3.6.i | RUN..... | 107 |
| 3.6.j | TAB..... | 108 |
| 3.6.k | VERIFY..... | 109 |
| 3.7 Special Commands | | |
| 3.7.a | DLD..... | 110 |
| 3.7.b | DST..... | 111 |
| 3.7.c | IBSHIFT..... | 112 |
| 3.7.d | LREG..... | 113 |
| 3.7.e | PCOPY..... | 114 |
| 3.7.f | PMODD..... | 115 |
| 3.7.g | PTV..... | 116 |
| 3.7.h | REAL..... | 117 |
| 3.7.i | SREG..... | 118 |
| 3.7.j | SWP..... | 119 |
| 3.7.k | VECTD..... | 120 |
| 3.7.l | VECTI..... | 121 |

CHAPTER 4 VARIABLES, CONSTANTS, OPERATORS and EXPRESSIONS

| | | |
|--------------------------------------|---|-----|
| 4.1 Constants | | |
| 4.1.a | Integer Constants..... | 122 |
| 4.1.b | String Constants..... | 122 |
| 4.1.c | Real Constants..... | 123 |
| 4.2 Variables | | |
| 4.2.a | Scalar Variable Names..... | 124 |
| 4.2.b | Integer Variables..... | 124 |
| 4.2.c | String Variables..... | 125 |
| 4.2.d | Real Variables..... | 126 |
| 4.2.e | Variable Type Conversions..... | 126 |
| 4.3 Variable Arrays | | |
| 4.3.a | Array Names..... | 127 |
| 4.3.b | Subscripts..... | 127 |
| 4.3.c | Memory requirements..... | 128 |
| 4.4 OPERATORS and EXPRESSIONS | | |
| 4.4.a | Arithmetic Operators and Expressions... | 129 |
| 4.4.b | Integer Arithmetic..... | 130 |
| 4.4.c | Logical Operators..... | 130 |
| 4.4.d | Relational Operators..... | 131 |
| 4.4.e | String Operators and Expressions..... | 131 |

| | PAGE |
|---|------|
| CHAPTER 5 TECHNICAL INFORMATION | |
| 5.1 Machine Language Interfacing..... | 132 |
| 5.2 Interfacing with Interpreter BASIC..... | 133 |
| 5.3 Interpreter Calls..... | 134 |
| 5.4 Subroutine Call description..... | 135 |
| CHAPTER 6 SAMPLE PROGRAMS | |
| 6.1 Program #1..... | 137 |
| 6.2 Program #2..... | 138 |
| 6.3 Program #3..... | 140 |
| 6.4 Program #4..... | 144 |
| CHAPTER 7 ERROR MESSAGES | |
| 7.1 Compiler Error Messages..... | 148 |
| 7.2 Runtime Error Messages..... | 150 |

1

MLBASIC 1.0 USER'S MANUAL

CHAPTER 1 INTRODUCTION

1.1 Overview of MLBASIC

MLBASIC (Version 1.0) is an enhanced Basic Compiler designed to allow as much compatibility with existing Interpreter Basic programs as would allow. MLBASIC is a full compiler that features all of the commands that are available with Standard, Extended and Disc BASIC. Furthermore, additional commands offered by MLBASIC make it possible to interface programs with assembly language and other Basic programs. The ability to call subroutines and pass arguments between the subprograms and the calling program makes it possible to write structured programs, only available with languages like FORTRAN and PASCAL.

MLBASIC allows users who are unfamiliar with machine language programs to create a machine language program from a Basic program with little or no effort. Default options that make compilation easy for the new users, can be replaced by specified values which allow the advanced user the freedom of how the program is to be compiled.

1.2 System Requirements

The following hardware is needed for MLBASIC to run:

1. 1 tape recorder (disk is optional)
2. 64k random access memory (RAM)
3. Microsoft Basic Interpreter

1.3 Diskette and Tape Contents

Disk Contents

1. "MLBASIC1.BAS" -Loader for cassette version of MLBASIC
2. "MLBASIC1.MAI" -Tape version of MLBASIC
3. "MLBASIC2.BAS" -Loader for disk version of MLBASIC
4. "MLBASIC2.MAI" -Disk version of MLBASIC
5. "PROGRAM1.BAS" -Example program #1
6. "PROGRAM2.BAS" -Example program #2
7. "PROGRAM3.BAS" -Example program #3
8. "PROGRAM4.BAS" -Example program #4

Tape Contents

1. "mlbasic1" -Loader for tape version of MLBASIC
2. "MLBASIC1" -Tape version of MLBASIC
3. "PROGRAM1" -Example program #1
4. "PROGRAM2" -Example program #2
5. "PROGRAM3" -Example program #3
6. "PROGRAM4" -Example program #4

MLBASIC 1.0 USER'S MANUAL

1.4 Compilation vs. Interpretation

A microprocessor can execute only its own machine instructions; it cannot execute Basic statements directly. Therefore, before the microprocessor can execute a program, the statements contained in the Basic program must be translated to the machine language of the microprocessor. Compilers and Interpreters are both programs that perform this translation. This section explains the difference between these two types of translation programs, and explains why and when you want to use the compiler.

Interpretation

An interpreter translates your BASIC program into machine language instructions line-by-line at runtime. To execute a Basic statement, the interpreter must analyze the statement, check for errors, translate the BASIC statement into machine language, and then execute those instructions. If the interpreter must execute a statement repeatedly (inside a FOR/NEXT loop, for example), it must repeat this translation process each time it executes the statement.

Basic programs are stored as a list of numbered lines, so each Basic program line is not available as an absolute memory address during interpretation. The interpreter must examine all the line numbers in the list, starting with the first, until it finds the line in a branch such as a GOTO or GOSUB statement.

Variables in a Basic program do not have absolute memory addresses either. When a Basic statement refers to a variable, the interpreter must search through a list of variables from the beginning until it finds the referenced variable.

Compilation

A compiler translates a source program and creates a new file, called an object file. The object file contains machine code that can be relocated or executed where it is. All translation takes place before runtime, which means no translation of your Basic source file occurs during the execution of your program. In addition, absolute memory addresses are associated with variables and with the lines referenced in GOTO and GOSUB statements, so that the computer does not have to search through a list of variables or line numbers during execution of your program.

The compiler also "optimizes" the program. This means that when the compiler executes a program, it does so in the fewest possible steps. This increases execution speed and decreases program size.

These factors combine to increase the execution speed of your program measurably. In most cases, execution of compiled Basic programs is 10 to 20 times faster than execution of the same program with the interpreter. If the program makes maximum use of integer variables, execution can be up to 100 times faster.

MLBASIC 1.0 USER'S MANUAL

1.5 Program Development

MLBASIC is compiler that accepts a BASIC program as source in the same format as the Interpreter accepts. This allows the user to develop programs using existing software that was designed for development of Interpreter BASIC programs (ie. Extended BASIC editor, fullscreen editors, etc).

The BASIC source, once written, should be saved to disk or tape. In most cases, the source program can be run using the Interpreter in order to debug the program for syntax or logic errors.

The final step in the program development process is to compile the source code using MLBASIC. The final product after compilation is an executable machine language program that is run by using the EXEC command.

MLBASIC 1.0 USER'S MANUAL

CHAPTER 2 HOW TO COMPILE A PROGRAM

2.1 Explanation of MLBASIC Options

MLBASIC is a highly versatile BASIC compiler which allows the user to select many of the parameters that control the compilation process. All the parameter options are set to default values initially so that users may easily compile a code and not have to worry about all the different options. These options allow the user to control compiler operations such as where the program is to be located in memory, what the storage medium for the source and object code is, how the computer is to accept and display numbers and how the compiler listings are handled.

The main categories of options available are the storage options, mapping options, listing option, compilemode option, and number base option. Sections 2.3 through 2.7 will cover each of these categories in detail. Section 2.8 will deal with an example compilation that uses some of these options.

MLBASIC 1.0 USER'S MANUAL

2.2 Compiling a Program Using Default Values

For ease of use, MLBASIC uses default values for all of the compilation options available. These default values cause the compiler to compile a program that is in memory and store the compiled code in the lowest address above the source code in memory. The only required input from the user is the character R. This simply tells MLBASIC to start compiling the program.

There are two versions of MLBASIC provided to the user. The first version is designed to use a minimal amount of memory; thus allowing ample room for the default "In Memory" compilation. The second version uses all available memory and is designed for compilation of large programs. The second version is designed to read the BASIC source program from disk and write the compiled program to disk; Therefore, users who do not have a disk drive can only use the first version.

How to Compile a Program1. Develop the program to be compiled.

- Using the available syntax , as described in Chapter 3, develop the BASIC source program that is to be compiled.

2. Save the BASIC program.

- CSAVE the BASIC program to tape or SAVE it to disk so that the program is safe if any compile errors occur.

3. Load MLBASIC into memory.

(A) Turn off the computer, and then turn it on (cold start).

(B) If you have cassette:

1. Rewind the tape to the beginning.
2. Type in the command CLOAD, and hit ENTER key.
3. Type in the word RUN, and hit ENTER key.

(C) If you have disk:

1. Insert MLBASIC diskette into Drive #0.
2. Type in the command RUN "MLBASIC1", and hit Enter key.

4. Decide on the storage option desired.

-If you want to compile a program "In Memory" (using the Mem option), CLOAD from tape or LOAD from Disk the BASIC program.

MLBASIC 1.0 USER'S MANUAL

5. Execute the compiler.

-To run the compiler, type in the command EXEC. The compiler will come back with a screen that lists all of the options and what the current default values are. The last line on the screen display will indicate to the user information on the required inputs or options being chosen.

6. Start compilation process.

-Enter any of the desired options. You may skip fields by hitting the Enter key.

(A) If you want to compile the program in memory, simply enter R, and compilation will begin.

(B) If you want to read the BASIC source from disk or cassette, position the cursor to the "BASIC Input" option line, hit C or D for cassette or disk, and enter the input filename. Hit R to start compiling. Note that the cursor is initially located next to the "Basic Input" option when the compiler is first executed.

(C) If you want to compile the object code to disk or cassette, position the cursor on the "M.L. Output" option line, hit C for cassette, or D for disk and enter the output filename. Hit R to begin compilation.

- MLBASIC compilation may be stopped by the user by pressing down, and keeping down, the Break key. Once the compiler has recognized the interrupt, it will wait for the user to hit another key before it continues. If the user hits the T key, the compiler will exit and display the message "ABORT COMPILATION". If the user hits any other key, compilation will resume. This feature is useful for pausing the compiler for examination of screen listings.

The above instructions include the general procedure for compilation. Variation from the outlined instructions are needed if such options like Manual compilemode or specified mapping addresses are used. Disk users, who need to compile large programs may use the second version of MLBASIC. The only difference for running the compiler in this case is that Step 3-C-2 should read: Type in the command RUN"MLBASIC2", and hit Enter key.

MLBASIC 1.0 USER'S MANUAL

2.3 Storage Options

There are six different combinations of how the compiler is to handle program source and object (compiled version) code. By default, both input and output by the compiler are performed "In Memory". The two main options for storage are:

(1) BASIC Input - This is where the compiler is to obtain the BASIC program that is to be compiled. The three available choices are to read the program from "M"-memory, "D"-disk or "C"-cassette. The letters in quotes are the characters that are used to tell the compiler which option to use. By default, the "M"-memory option is used where the program has previously been entered or loaded into memory. The disk or cassette options, if selected, will be followed by a query from the computer asking for the input filename. This filename is the name of the program that is on disk or cassette.

(2) M.L. Output - This is where the compiler is to put the final compiled program and necessary text and subroutine areas. The three available choices for outputting the compiled code are; "M"-memory, "D" -disk or "C" -cassette.

As in the BASIC Input option, the character in quotes is used to identify each choice. By default, the "M" -memory option is used, meaning that all compiled output is to be written to memory. This is the fastest way to compile a program, and therefore should be used, unless the compiled program is too large to fit in memory, or is to be saved on disk or tape. In the case where the object code is too large to be compiled in memory, the "D" -disk option must be used.

The cassette and disk options allow for storage of the compiled program on a non-volatile medium. Once an error free program has been compiled and saved to disk or tape, the executable program may be loaded into memory, and EXECuted with little effort. The disk option allows for unlimited size programs to be written, whereas the cassette option only allows programs that can fit in memory at compile time, to be compiled.

In summary, the various storage options allow flexibility in where the program is to be compiled. For large programs, the "D" Output and "D" or "C" Input options should be used. This allows for compilation of any size program (final program size may be up to 60k long!)

MLBASIC 1.0 USER'S MANUAL

2.4 Mapping Options

MLBASIC allows the user to select the locations of all four program segments that are produced when a BASIC program is compiled. These four locations are:

(1) Entry Point - This is the starting location for the main machine language program. The address in the EXEC command used to run the compiled program is called the Entry Point. By default, the Entry Point is the first location in the entire program.

(2) Text Table - The starting address of the area where all of the numeric string constants are stored, including text contained in PRINT and INPUT statements, is called the Text Table. The default value used for the Text Table beginning is the address immediately following the main program area.

(3) Subroutines - This is the starting address where the runtime machine language routines that contain all the necessary interfacing between the main program and the computer are located. This package of routines must accompany the final program for successful execution. By default, the Subroutines follow the Text Table in memory.

(4) Variables - This is the starting address of the area where scalar and dimensioned variables are contained. This is an absolute address, and is only used during execution of the compiled program. During compilation, this area may be anywhere, but by default is located following the Subroutine library.

In summary, the mapping options allow the user to specify where the final compiled program is to be stored "In Memory". The first three areas in the compiled program are written to memory or the storage medium at compile time, whereas the fourth is not. If the "In Memory" option for storing the compiled sections is used (see 2.3 for more info), the user must be careful not to overwrite the compiler.

MLBASIC 1.0 USER'S MANUAL

2.5 Listing Options

MLBASIC allows three choices for listing the final compiled program. They are:

1. "S" - Screen option
2. "N" - No output option
3. "B" - Both screen and printer output option

The Screen option allows for users to view the compilation process line by line. Each line, as well as its location in the M.L. program, is displayed as it is compiled. This option allows the user to identify errors in the source code.

The No output option allows the user to see only the locations of the four program segments as the compiler works. The original screen that appears during initial execution of MLBASIC is kept for the duration of compilation. This option is most useful for identifying where the program is being written, and how long it is. By default, the No output option is used during compilation.

The Both option gives a screen listing identical to the "S" option, and in addition, produces a comprehensive printer listing of the compiled program. Included in the listing to the printer is:

- A. All of the beginning and ending locations of all four program sections.
- B. Locations of each BASIC line in the final compiled program.
- C. Listings of all BASIC source lines that are compiled.
- D. Vector locations of where all GOTO and GOSUB's branch to and where they were called from.
- E. Variable tables for scalar and dimensioned variables showing variable locations in memory, type of variable, and its name.

MLBASIC 1.0 USER'S MANUAL

2.6 Number Base Option

MLBASIC allows for the user to select the default number base to be used in Integer Variable inputs and printing. This number base is used only for integer variable I/O, and has nothing to do with how real variables are input or printed.

The default number base used by MLBASIC is base 10, decimal. If the user wants the compiled program to understand hexadecimal numbers for example, the user must specify 16 as the number base before compiling that program. In this case, any integer INPUT or PRINT statements within the compiled program will only understand base 16 numbers. It is important to realize that only the selected base is valid for integer I/O.

The allowable number bases to choose from are bases from 2 to 16. Base 2 for example gives binary output when an Integer is printed and only accepts binary when an integer is INPUT from the keyboard.

MLBASIC 1.0 USER'S MANUAL

2.7 Compilemode Options

MLBASIC allows the user to choose whether or not to let the compiler perform all of the compilation processing. If the user doesn't care about where the program is to reside in memory, then the Automatic compilemode should be chosen. Often the user may want to select all of the mapping options for compilation; in this case the Manual compilemode should be used. These two options for how the compiler is operated allows flexibility in the program development process. By default, the Automatic mode is used by MLBASIC.

The Automatic compilemode allows the user to quickly compile a source code into an EXECutable machine language program. In this mode, the most efficient mapping options are figured out. The automatic mode causes MLBASIC to perform a two pass compilation of the BASIC source code.

During the first pass, each program line is scanned for syntax errors. If any errors occur, the compiler will display the errors. If there were any errors during the first pass, compilation will stop. If there were no errors, compilation continues to the second pass.

During the second pass, all of the mapping parameters are figured out and compilation of the entire program proceeds. At this time, the source listing, if any, is output. At the end of the second pass, the Subroutine library is relocated to its proper location, whether it be in memory, on disk or tape. In addition, all of the GOTO and GOSUB vectors are stored in the machine language program at this time.

The Manual compilation mode is not usually used. It only performs one pass over the source code during compilation. This pass, similar to the second pass of the Automatic compilemode, checks for errors, outputs listings of compiled source and relocates the subroutines all at once. It may be handy sometimes to use this mode to output the program along with the error messages (in the Automatic compilemode, this is not possible). If the manual mode is selected, the user is required to input the starting addresses for the M.L. Program, Text Table, Subroutine Library, and Variable Table (see Section 2.4 for more information on these four locations).

MLBASIC 1.0 USER'S MANUAL

2.8 Example compilation using specified values

This example illustrates the uses of the Manual compilemode option and how one might use it. The purpose is to compile two programs in memory, using common variable and subroutines.

Program 1

```
MLBASIC -REVISION 1.0926 - COPYRIGHT 1984 WASATCHWARE
-----
ENTRY INTO M.L. PROGRAM=12000
VARIABLE TABLE STARTS AT 16827
TEXT TABLE STARTS AT 12239
SUBROUTINES START AT 12476
-----
12011 >-10 REM*****XXXXXXXXXXXXXX
12011 >-20 REM* MANUAL COMPILEMODE *
12011 >-30 REM* SAMPLE PROGRAM#1 *
12011 >-40 REM*****XXXXXXXXXXXXXX
12011 >-50 REM
12011 >-60 CLS:PRINT"THIS ROUTINE WILL CALL ANOTHER"
12035 >-70 PRINT"COMPILED PROGRAM USING THE"
12035 >-80 PRINT"SAME VARIABLES AND SUBROUTINES"
12039 >-90 PRINT"-JUST ONE EXAMPLE OF THE USE"
12086 >-100 PRINT"OF THE MANUAL COMPILEMODE"
12183 >-110 PRINT"","TO MAKE SURE WE SAY WHAT WE MEAN"
12120 >-120 INPUT"ENTER A NUMBER FOR VARIABLE A" "JA:PRINT
12141 >-130 PRINT"... CALLING OTHER PROGRAM ..."
12158 >-140 FORI=1TO1000:NEXT:REM' DELAY FOR DISPLAY
12161 >-150 AA=1:REM' CREATE A LAST VARIABLE ENTRY IN TABLE
12185 >-160 PTY(RA,B):REM' POINT TO VARIABLE, A
12196 >-170 B=B+1+11:REM' START OF NEXT PROGRAM
12220 >-180 EXECB
12226 >-190 END

END OF M.L. PROGRAM=12234
END OF TEXT TABLE=12475
END OF VARIABLE AREA=16892
```

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| A | 16839 | INTEGER | B | 16841 | INTEGER | C | 16843 | INTEGER |
| D | 16843 | INTEGER | E | 16847 | INTEGER | F | 16849 | INTEGER |
| G | 16851 | INTEGER | H | 16853 | INTEGER | I | 16855 | INTEGER |
| J | 16857 | INTEGER | K | 16859 | INTEGER | L | 16861 | INTEGER |
| M | 16863 | INTEGER | N | 16865 | INTEGER | O | 16867 | INTEGER |
| P | 16869 | INTEGER | Q | 16871 | INTEGER | R | 16873 | INTEGER |
| S | 16875 | INTEGER | T | 16877 | INTEGER | U | 16879 | INTEGER |
| V | 16881 | INTEGER | W | 16883 | INTEGER | X | 16885 | INTEGER |
| Y | 16887 | INTEGER | Z | 16889 | INTEGER | RA | 16891 | INTEGER |

(J ERRORS

MLBASIC 1.0 USER'S MANUAL

Program 2

MLBASIC -REVISION 1.092B - COPYRIGHT 1984 WASATCHWARE

ENTRY INTO M.L. PROGRAM=16892
 VARIABLE TABLE STARTS AT 16827
 TEXT TABLE STARTS AT 17112
 SUBROUTINES START AT 12476

```

16903 >-18 REM*****  

16903 >-28 REM* MANUAL COMPILEMODE *  

16903 >-30 REM* SAMPLE PROGRAM#2 *  

16903 >-40 REM*****  

16903 >-50 REM  

16903 >-60 CLS:PRINT"PROGRAM #2 HAS NOW BEEN EXECUTED"  

16927 >-70 PRINT" ","REMEMBER THE NUMBER YOU ENTERED?"  

16944 >-80 FORI=1TO60000:NEXT:REM' DELAY  

16967 >-90 PRINT"IT WAS ",A  

16991 >-100 PRINT" ", "... BS YOU SEE, THE VARIABLES"  

17008 >-110 PRINT"HAVE BEEN SHARED."  

17025 >-120 PRINT" ","WHAT DO YOU THINK OF THIS","PROGRAM NOW?"  

17042 >-130 FORJ=1TO3:FORI=1TO60000:NEXT:NEXT:REM' DELAY  

17091 >-140 CLS:RETURN:REM' RETURN TO CALLING ROUTINE  

17099 >-150 END
  
```

END OF M.L. PROGRAM=17107
 END OF TEXT TABLE=17276
 END OF VARIABLE AREA=16890

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| A | 16839 | INTEGER | B | 16841 | INTEGER | C | 16843 | INTEGER |
| D | 16845 | INTEGER | E | 16847 | INTEGER | F | 16849 | INTEGER |
| G | 16851 | INTEGER | H | 16853 | INTEGER | I | 16855 | INTEGER |
| J | 16857 | INTEGER | K | 16859 | INTEGER | L | 16861 | INTEGER |
| M | 16863 | INTEGER | N | 16865 | INTEGER | O | 16867 | INTEGER |
| F | 16869 | INTEGER | Q | 16871 | INTEGER | R | 16873 | INTEGER |
| S | 16875 | INTEGER | T | 16877 | INTEGER | U | 16879 | INTEGER |
| V | 16881 | INTEGER | W | 16883 | INTEGER | X | 16885 | INTEGER |
| Y | 16887 | INTEGER | Z | 16889 | INTEGER | | | |

0 ERRORS

MLBASIC 1.0 USER'S MANUAL

Procedure to Compile Programs 1 and 2

1. Load in the first version of MLBASIC, called "MLBASIC1", into memory (As described in 2.2 step 3).
2. Load in memory and compile Program #1, (which 1st needs to be entered and saved to tape or disk) using the "In Memory" option and all default values.
3. Write down the starting addresses for the subroutine and variable table. Also make a note where the variable table ends (this is displayed after compilation of program).
4. Load Program #2, (which 1st needs to be entered and saved to tape or disk) into memory, and execute MLBASIC by entering EXEC21700.
5. Compile Program #2 using the following procedure:
 - (A) Position cursor next to Entry Point Field. Type in the location of the end of the variable table of Program #1, determined from Step 2. Hit Enter to register that choice.
 - (B) Type in "R" and hit Enter to run the compiler. Write down where the Text Table starts (1st number of the two).
 - (C) After successful (error free) compilation, we are ready to make the final compilation. Execute MLBASIC by entering EXEC21700. Position the cursor next to the compilemode field and type in "M". This selects the Manual compilemode (Notice that a number next to the Entry Point Field appears when "M" is pressed. This is the lowest allowable address one may use for the ENTRY Point, if the "In Memory" and Manual options are used)
 - (D) Repeat Step 5-A.
 - (E) Enter the value for the Text Table, as determined in Step 5-B.
 - (F) Enter the value for the Subroutine and Variable tables as determined in Step 3.
 - (G) Enter "R", and hit the Enter key to run MLBASIC.

6. Save the final program to disk or tape using the Entry Point of Program #1, and the ending location for the Text Table of Program #2 (displayed after Step 4-G is completed) as the beginning and ending locations to save. To run the program, enter EXEC12000.

MLBASIC 1.0 USER'S MANUAL

CHAPTER 3 MLBASIC COMMANDS

In this chapter, each command allowed by MLBASIC will be fully described. An entire page is devoted to each command, thereby making it easy for the user to find a particular command in question.

Throughout this chapter, a general format accompanies the description of each command. When more than one specific arrangement is permitted, separate formats are shown. Within a general format, keywords, connectives, and special characters are shown in proper sequence. Unless otherwise stated, only the shown sequence can be used.

The general formats use the following convention:

- Each capitalized word or letter represents a required part of the instruction line. You must type in all the CAPITALIZED items that appear in the format line as CAPITALIZED words.
- Wherever an element is underlined, you must supply a legal BASIC representative of that element.
- Elements that are enclosed in slashes (/) are optional items.
- A colon (:) indicates a choice. When a colon appears in an instruction line, you can choose a parameter from either side of the colon.
- Items followed by ellipsis (...) may be repeated any number of times.
- You must use all punctuation marks in an instruction in the position they are shown in the format line. However, you should never include in an instruction any of the symbols including underlines, slashes and colons (although colons are used to separate individual commands that on on the same program line).
- Blank spaces are ignored by the compiler, but are necessary for separating variable names and commands.

The format item descriptions contain the allowable data parameter types as shown in parenthesis that follow the general description of that item. The following abbreviations are used to describe the allowable parameter types for each item:

| | |
|-----|--------------------------------------|
| IV | -Integer Variable |
| IC | -Integer Constant |
| SIV | -Scalar Integer Variable (no arrays) |
| RV | -Real Variable |
| RC | -Real Constant |
| SRV | -Scalar Real Variable (no arrays) |
| SV | -String Variable |
| SC | -String Constant |
| IE | -Integer Expression |
| RE | -Real Expression |
| SE | -String Expression |

MLBASIC 1.0 USER'S MANUAL

3.1 I/O Commands

3.1.a CLOSE3.1.aFunction

To close one or more files that were opened for I/O.

Format CLOSE /#channel/,...

channel -device number to be closed (IV, IC)

Examples

1. CLOSE

-This closes all open channels

2. CLOSE#1,#2,#-1

-This closes channels 1,2,-1

Comments

1. The CLOSE command should be used before program termination whenever any disk or cassette files are open. It is especially important to close files opened for output, since a close will output any remaining data left in the file buffer.

Differences from Interpreter

1. NONE

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.b3.1.b CLOADMFunction

To load a machine language program from cassette.

Format CLOADM /filename//,offset/

filename -Name of file (SC)

offset -Offset load value (IC)

Examples

1. CLOADM "MLTEST"

-Loads machine language file "MLTEST"

2. CLOADM "TEST1",1000

-Loads file "TEST1" with an offset of 1000 bytes

Comments

1. The filename may be omitted, in which case the next file found on the cassette will be loaded.

Differences from Interpreter

1. Only Constants are allowed as arguments.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.c3.1.c CSAVEMFunction

To save machine language programs or binary data to cassette

Format **CSAVEM filename,start,end,exec**

filename -Name of output file (SE)
start -Starting address in memory to save (IV,IC)
end -Address of last byte to save (IC,IV)
exec -Entry location for M.L. program (IC,IV)

Examples

1. CSAVEM "MLTEST",10000,12000,10500
-Save the machine language program "MLTEST" to tape
starting at 10000, thru 12000 and an exec address of 10500

Comments

1. Extended Basic is not required.

Differences from Interpreter

1. NONE

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.d3.1.d DIRFunction

To display a directory of the disk in the drive number you specify.

Format DIR /drivenumber/

drivenumber -Number of drive 0-3 (IC)

Examples

1. DIR
-Display directory of drive 0
2. DIR1
-Display directory of drive 1

Comments

1. If no drive number is given, the default drive directory is displayed.

Differences from Interpreter

1. Only Integer constants are allowed for drive number.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B, ECB, DB

MLBASIC 1.0 USER'S MANUAL

3.1.e3.1.e DRIVEFunction

Changes the drive default to a specified number between 0 and 3.

Format **DRIVE /drivenumber/**

drivenumber -Number of drive to select (IC)

Examples

1. DRIVE3

-This makes DRIVE3 the default drive

Comments

1. If DRIVE is not used, drive 0 is the default drive.

Differences from Interpreter

1. Only Integer Constants are allowed.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB, DB

MLBASIC 1.0 USER'S MANUAL

3.1.f3.1.f DSKI\$Function

Directly input a sector from a given track and drive into a string array that is dimensioned for at least 256 characters.

Format DSKI\$ drivenum,track,sector,string

| | |
|----------|-----------------------|
| drivenum | -Number of drive (IE) |
| track | -Track number (IE) |
| sector | -Sector number (IE) |
| string | -Name of string (SV) |

Examples

1. DSKI\$1,17,3,A\$
-Reads the directory track, sector 3 and stores it in array A\$ in elements A\$(0)-A\$(255)
2. D=1:DSKI\$ D,1,10-D,A\$(256)
-This example reads sector 9, track 1, from drive 1 into array A\$ in elements A\$(256)-A\$(511).
3. DIMB\$(256,12):FORI=0TO11:DSKI\$0,17,I+1,B\$(0,I):NEXT
-This reads the entire directory track into a buffer called B\$.

Comments

1. The track numbers may be a number from 0 to 34, the sector may be a number from 1 to 12.

Differences from Interpreter

1. The array that is to hold the input sector can hold all 256 bytes, whereas the Interpreter uses two arrays of 128 bytes each.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B, ECB, DB

MLBASIC 1.0 USER'S MANUAL

3.1.g3.1.g DSKO\$Function

Outputs a string buffer to a sector on a given track and drive.

Format DSKO\$ drivenumber,track,sector,string

drivenumber -Output drive (IE)
track -Track number (IE)
sector -Sector number (IE)
string -String array (SV)

Examples

1. 10 DSKO\$0,0,1,ARRAY\$
-This outputs the array elements A\$(0) thru A\$(255) to sector 1, track 0 of drive 0.
2. 10 DIM BUFFER\$(256,12)
20 FOR I=0 TO 11:DSKO\$0,34,I+1,BUFFER\$(0,I):NEXT
-This outputs buffer B\$ to the last track on drive 0.

Comments

1. As with DSKI\$, the allowable track numbers are 0-34 and the allowable sectors are 1-12.

Differences from Interpreter

1. Only one string is required to hold the 256 byte data that is to be written to disk with MLBASIC.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB, DB

3.1.hFIELDFunction

Organizes the space within a direct access buffer into fields. By assigning a name to each field, data can be written to the fields using the LSET and RSET commands, and later used as a string variable in PRINT statements, string expressions, etc.

Format FIELD #buffer,fieldsize /:AS fieldname ,...

buffer -Buffer to divide into fields (IC)

fieldsize -Size of field (IC)

fieldname -Name of field (up to 2 characters+ "\$")

Examples

1. FIELD#1, 100/A1\$,200 AS A2\$,50/ A3\$

-This forms 3 new fields in buffer 1 of length 100, 200 and 50 bytes each.

2. LSET A1\$="data"+A\$

-This example writes a string expression to the field, A1\$.

3. \$A\$=A1\$

-This example assigns the data stored in field, A1\$, to the string variable, A\$.

Comments

1. The name of the field may be used as a string variable is normally used (eg. PRINT, string expressions).

2. Data may only be written to a field using the LSET and RSET commands. In other words, field names may not appear on the left side of a string equation, or with the command INPUT.

3. If more than one FIELD command that use the same buffer numbers are in a program, make sure no FIELD commands having different buffer numbers appear in the middle of the FIELD commands.

4. The maximum size of any field must be less than 256 bytes (1-255 allowed).

Differences from Interpreter

1. The Interpreter only allows "AS" to be used to separate field parameters. MLBASIC offers "/" as another allowable delimiter so that Non-Disk users can use FIELD for random access cassette I/O.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.13.1.1 FILESFunction

To tell the computer how many buffers to reserve in lower memory, and the total number of bytes to reserve for the buffers.

Format FILES buffers, buffersize

buffers -Total number of buffers to reserve (IE)
buffersize -Total number of bytes to reserve (IE)

Examples

1. FILES4,1300

-This reserves four buffers and a total space of 1300 bytes for all disk buffers.

Comments

1. On startup of the computer, there are 2 buffers and a total of 256 bytes for the buffers assigned before any FILES command is given.

2. Care must be taken when using this command. Memory available for buffers must be large enough to accommodate the buffersize.

3. If a buffer is currently open when the FILES statement is executed, the data in the buffer is lost as all buffer tables are re-initialized.

4. The graphic pages conflict with the disk buffers, so make sure that the first graphic page used is above the disk buffers.

Differences from Interpreter

1. NONE

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB, DB

MLBASIC 1.0 USER'S MANUAL

3.1.j GET3.1.jFunction

Gets the record number specified and stores it in the specified buffer.

Format GET #buffer,recordnumber

| | |
|--------------|----------------------|
| buffer | -Buffer number (IE) |
| recordnumber | -Record to read (IE) |

Examples

1. GET#1,I-1

- This reads in record number (I-1) into buffer #1.

Comments

1. This command does not support the graphics option for GET.

2. Non-Disk users may use this command for random access cassette Inputting of individual cassette blocks.

Differences from Interpreter

1. Graphics mode not supported in MLBASIC.
2. Cassette option not allowed with Interpreter.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B

3.1.k INPUT3.1.kFunction

This command inputs data from the specified channel number and stores the data in the variable specified in the argument list.

Format INPUT /string; //#buffer,/arg,...

| | |
|--------|--|
| string | -Message to appear before keyboard input (SC) |
| buffer | -Device number (IE) |
| | -1 =cassette |
| | 0 =keyboard (not needed) |
| | 1-15 =Disk files |
| arg | -Name of variable or array where input data is stored. (IV,RV,SV) |

Examples

1. INPUT "ENTER A NUMBER ";A

-This prints "ENTER A NUMBER" to the screen and awaits an input from the keyboard. When you enter the number and hit RETURN, the number is stored in the variable named A.

2. INPUT#-1,A,\$A\$,B\$(100)

-This inputs data from the cassette in the following order: a number is first input into the variable A, then an entire string is read (characters terminated by a zero byte) into the array A\$ starting with element A\$(0), finally one character is input into the array element B\$(100).

Comments

1. The format that is accepted as input from cassette and disk files is binary format by default. This is the most efficient way to store data, and since this is how it is stored in memory, no conversion of data types is necessary. This means that CVN and MKNS are not not needed for efficient I/O.

2. Data that has been written to the file previously using an ASCII format must be read in as a string and converted to a real or integer number using VAL.

MLBASIC 1.0 USER'S MANUAL

3. String variables may be input element by element by specifying a string element in the argument list. By placing the special character "\$" in front of the string variable name, full strings can be input from a device. The terminating character for the string is a zero byte.

Differences from Interpreter

1. When data is input from the keyboard, the "ENTER" key must be hit after every entry.
2. A return to new line is not registered on the screen after a number is entered from the keyboard. This allows for more flexible formats when inputting from the screen. For example, the Interpreter command INPUTA is equivalent to the MLBASIC command INPUTA:PRINT.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.13.1.1 KILLFunction

To delete a file from the disk permanently.

Format **KILL filename**

filename -Name of file to kill (SE)

Examples

1. KILL "FILE1"+":1"

- Delete FILE1 from drive 1's directory

Comments

1. The kill command closes all open files before deleting a file.

Differences from Interpreter

1. NONE

Roms Needed (ECB=Extended, DB=Disk, S=Standard)

B, ECB, DB

MLBASIC 1.0 USER'S MANUAL

3.1.m3.1.m LSETFunction

To left justify a string into a previously specified field within a random access buffer.

Format LSET fieldname=string

fieldname -Name of field in buffer (2 characters+ "\$")
string -String to be stored into field (SE)

Examples

1. LSET A1\$="The number is "+STR\$(A)

-This stores a string expression into field A1\$

Comments

1. If the string expression is larger than the field, the string is truncated to fit the field, and where the last byte in the field is a zero.

2. If the string is shorter than the field, blanks (ASCII 32) are filled in to the right of the string with a zero byte in the last position in the field.

3. In all cases, a zero is used to terminate the field that is being written to. This means that a zero should be accounted for in the allocation of the buffer. Each field in that buffer will have a zero as its last character.

4. Data that is written to fields can be used as a string in string expressions. The zero byte that terminates the field is needed to terminate the field string when used in an expression.

Differences from Interpreter

1. The format for terminating the field with a zero is different than the Interpreter.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.n OPEN3.1.nFunction

To open a file for input, output or direct access. The device can be either cassette or disk.

Format OPEN #buffer, "mode", filename /, #filetype //, recordlen /

| | |
|-----------|--|
| buffer | -Buffer number (IC, IV) |
| mode | -I=input, O=output, D=direct (random) access |
| filename | -Name of file to open (SE) |
| filetype | -Type of file as follows: |
| | \$000 Basic program |
| | \$0FF Basic program in ASCII |
| | \$100 Binary data |
| | \$1FF ASCII data |
| | \$200 Machine language program |
| | \$300 Text stored in binary |
| | \$3FF Text stored in ASCII |
| recordlen | -Length of direct access file (IE) |

Examples

1. OPEN#1,"I","TESTFILE:1"

- This opens buffer #1 for input from file "TESTFILE" on drive #1.

2. OPEN#5,"D",A\$+"DAT",#\$200,100

- This opens buffer #5 to file A\$ plus the extension ".DAT" for random access I/O. The file type is specified as text stored in binary. The record length is 100 bytes.

3. OPEN#1,"O","FILE",#&H200

- This opens channel #1 to an output file named "FILE". The type of the file is a machine language program.

Comments

1. The default record length for random (Direct) access files is 256 bytes.

2. The random access option can be used for cassette I/O provided the proper steps are made to make sure that the recorder is on record when you PUT a record, and that the recorder is on play when you GET a record.

3. Although the OPEN#-1,"D" option is not allowed, the cassette file may be opened for direct access using the following mode:

"I" -Open for input if file exists

"O" -Open for output if file is to be created

4. The maximum length of a cassette record is 255 bytes, as opposed to an unlimited size with disk files.

Differences from Interpreter

1. Interpreter does not support direct access cassette I/O.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

MLBASIC 1.0 USER'S MANUAL

3.1.03.1.0 PRINTFunction

To output data to a buffer, the printer or the screen.

Format PRINT /#buffer,//USINGformat; //TAB(pos)/ arg delim arg...

| | |
|---------------|--|
| <u>buffer</u> | -Buffer to print data to (IV,IC) |
| | If @ <u>number</u> is used instead of # <u>buffer</u> |
| | screen output is started at <u>number</u> |
| | (IV or IC between 0 and 511) |
| <u>format</u> | -PRINTUSING format allowed (SE) |
| <u>pos</u> | -column position to print next argument (IE) |
| <u>arg</u> | -Data to print (IE,RE,SE) |
| <u>delim</u> | -separation character for arguments: ","=skip to next line ";"=do not skip a line or space |

Examples

1. PRINT"This is text"
-This prints a string constant to the screen.
2. PRINT#-1,USING"##.#-";-123.9
- This prints a real constant to the cassette file using a specified format in the form of ASCII characters.
3. PRINT#2,TAB(5-I)"DATA=";TAN(A/SIN(1+1.9*COS(A)))
-This prints to disk a string and real expression starting at column position 5-I.

Comments

1. MLBASIC prints all real numbers to cassette and disk in their binary format, unless the USING format is used in the command. Likewise, the INPUT command will read in the data in the binary format, thereby making PRINT and INPUT compatible ways of storing and later recalling numeric data.
2. Strings that are written to disk or cassette are terminated with a zero byte as a means of separating the items in the file.

MLBASIC 1.0 USER'S MANUAL

3. The following characters may be used as field specifiers in the PRINTUSING format string:

The position of digits as they are to be printed. The number of #s establishes the numeric field. Unused digits are left as blanks (ASCII 32) to the left and zeros (ASCII 48) to the right of the decimal point.

. The position of the decimal point is marked by a "." in the numeric field.

, The comma, when placed anywhere between the first digit and the decimal point in the field, will display a comma to the left of every third digit that lies to the left of the decimal point.

** Two asterisks at the beginning of the numeric field indicates that all unused positions to the left of the decimal point will be filled with asterisks "**".

\$ By placing a dollar sign in front of the format, a dollar sign will appear in front of the output number.

\$\$ Two dollar signs placed at the beginning of the format will make the dollar sign appear one space to the left of the largest digit.

**\$ If these characters are used at the beginning of the format string, then the vacant positions to the left of the number will be filled with asterisks and a dollar sign one space to the left of the largest digit.

+ The plus sign will appear before positive numbers and a negative sign before all negative numbers if the "+" appears in front of the format string.

- If the minus sign appears at the end of the format string, a negative sign will appear after all negative numbers and a space after all positive numbers.

↑↑↑↑ If four "Up arrows" appear at the end of the format string, the number will be printed out in standard exponential form.

! An exclamation mark alone in the format string will cause the first string character to be printed by itself.

% % To specify a string field of more than one character, where the number of spaces that lie between the %s is equal to the length of the field.

Differences from Interpreter

1. The Interpreter uses the "," delimiter as a tab, where MLBASIC uses "," as a new line indicator.

2. MLBASIC requires a "\$" to precede a string expression that is to be printed. If the "\$" is not placed in front of a string variable when printed, only that element will be printed to the device.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.p PUT3.1.pFunction

To assign the data in the desired buffer a record number and store information on disk or cassette.

Format PUT #buffer,recordnum

· buffer -Output device number (IE)
· recordnum -Number of record on file
· (or record length of cassette) (IE)

Examples

1. PUT#1,1
-assigns current data in buffer #1, the record number 1.
2. PUT#-1,100
-writes the first 100 bytes in buffer #-1 to a cassette record (or block in this case).

Comments

1. The PUT command has been allowed to use the cassette for direct access input/output. The record number in this case must be accounted for in the applications program that uses the PUT command. The recorder must be positioned to the next block to be written (or overwritten), and the recorder must be on RECORD. A way of positioning the cassette to the proper block in a cassette file that is being written is to (1) make sure the cassette is in the PLAY mode by using prompts in the program, (2) to use GET#-1,R-1 ,where R is the record that is to be written. The internal software for the GET command will prompt the user to rewind the cassette tape to the beginning of the file, and then the correct block number will be searched for in the file.

2. The cassette option, PUT#-1, must include the length of the record, or errors will occur when writing to tape. Maximum cassette record lengths are 255 bytes.

Differences from Interpreter

1. The graphics options for the PUT command are not supported with MLBASIC.
2. The cassette option for PUT is not supported by the Interpreter.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.1.q RSET3.1.qFunction

To right justify a data string within a given field and buffer.

Format **RSET fieldname=string**

fieldname -Field Identifier name (2 characters+ "\$")
string -Data string to be put in field (SE)

Examples

1. RSET A1\$=VARIABLE\$(0)+STRING\$(10,"*")+STR\$(A-100)
-This example right justifies a complex string expression within the previously declared field named A1\$

Comments

1. If the string expression is larger than the field, the string is truncated to fit the field, and the last byte in the field is a zero.
2. If the string is shorter than the field, blanks (ASCII 32) are filled in to the left of the string with a zero byte in the last position in the field.
3. In all cases, a zero is used to terminate the field that is being written to. This means that a zero should be accounted for in the allocation of the buffer. Each field in that buffer will have a zero as its last character.
4. Data that is written to fields can be used as a string in string expressions. The zero byte that terminates the field is needed to terminate the field string when used in an expression.

Differences from Interpreter

1. The format for terminating the field with a zero is different than the Interpreter.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2 Program Control Commands

3.2.a CALL3.2.aFunction

The CALL statement is used to execute a subroutine by referencing its name and a list of parameters. These parameters are shared between the subroutine (subprogram) and the calling program.

Format CALL subroutine(arg,...)

subroutine -Name of subroutine (7 characters max.)
arg -Parameter to be passed to subprogram
 Value is shared with subroutine.
 (IV, IC, RV, RC, SC, SV)

Examples

1. 10 CALL EXAMPLE(A,9.9,B(1,10))
20 REM' PROGRAM
30 REM' CONTINUES

1000 SUBROUTINE EXAMPLE(I,J,K(0,0))
1001 REAL J:DIMK(20,20)
1002 I=INT(J/SIN(K(0,0)))
1003 RETURN

-This example shows the way one may call a subroutine. In this example, the subroutine EXAMPLE is called with the three parameters A, 9.9 and B(1,10) being passed in the argument list. The subroutine identifies the data that is in the caller's variable, A, as the variable I, J as the number 9.9 and K(0,0) as B(1,10). The result of the subroutine call puts the value of INT(9.9/SIN(B(1,10))) in the main program's variable A. Note that the variable I and J in the calling program is unaffected by the call.

MLBASIC 1.0 USER'S MANUAL

Comments

1. The arguments that are passed in the argument list of the CALL statement are pointers that are referenced by the subroutine program. The value or array of values that is pointed to in the argument list is contained in the calling program's storage area. This means that the calling program can share its variables with the subprogram that is being called.
2. The subroutines, also called subprograms, return values to the calling program unit only through actual-dummy argument correspondence. In other words, the first variable in the SUBROUTINE statement's list is set equal to the constant or variable that is first on the list in the CALL statement, and so on for all the arguments in the list.
3. If an array is an argument on the list in a CALL statement, the first element that is referenced by the subroutine is the element that appears on the CALL statement list.
4. If the SUBROUTINE is to return a value to the calling program, the argument in the list of the CALL statement must be a variable.

Differences from Interpreter

1. Interpreter does not support CALL.

Roms Needed (ECB=Extended, DB=Disk, S=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2.b3.2.b ENDFunction

The END command is used to indicate where compilation is to terminate. When the program is run, and an END is encountered, program termination will occur.

Format ENDExamples

1. 1000 PRINT"Exit":END

-When the program gets to line 1000, the message "Exit" will appear on the screen and the program will terminate.

Comments

1. The END is compiled the same as the STOP statement. Normal termination within the program should be done using STOP.

2. The END is the last statement of the program to be compiled. In other words, the END statement is used to tell the compiler that it has reached the end of the source to be compiled.

Differences from Interpreter

1. Interpreter allows the END to be anywhere in the source, while MLBASIC only permits the command at the end of the source.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2.c3.2.c EXECFunction

To execute a machine language program.

Format EXEC address

address -Entry location of machine
language program (IE)

Examples

1. EXEC10000
 - Execute the machine language program beginning at address 10000
2. POKE65502, 1:EXEC\$A1C1:POKE65503, 1
 - In this example, the 64k RAM mode is first turned off, then the address to poll the keyboard in ROM is called (starting at hexadecimal \$A1C1). When the machine language program finishes (with an RTS for those M.L. programmers), the map type is returned from the 32k ROM enabled to the 64k RAM enabled map type.

Comments

1. The EXEC command is used to execute an absolute address in memory. This means that a machine language program must exist at the address that is to be executed, or else unpredictable results will occur.
2. As many levels of calls using EXEC may be performed, as long as the memory permits.
3. The EXEC command, when compiled, is a useful way to execute a machine language program, located in the upper 32k of RAM, while running under Interpreter BASIC.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.2.d3.2.d FOR..NEXT (STEP)Function

To allow a series of instructions to be performed in a loop for a given number of times.

Format . . . `FOR counter=start TO finish /STEPstep/...NEXT/counter/`

counter -Index variable used to count thru
a given loop (IV)

start -Initial value counter assumes
when entering loop (IV,IC)

finish -Final value counter assumes in loop (IV,IC)
step -Increment to be added to counter (IV,IC)

Examples

1. `10 FORX=1 TO 10:NEXT`

-In this example, the counter variable, X, is incremented by 1 from 1 to 10.

2. `10 FORA(I)=J TO B(10,10)STEP-C(I,J)`

-In this example, the counter variable, A(I), is decremented by the amount contained in the integer array element C(I,J). Furthermore, the initial value is the integer variable J and the final value, (which in this case is less than the initial value) is B(10,10).

Comments

1. The commands following the FOR statement are executed until the NEXT command is encountered.

2. The counter is incremented by a specified amount when the NEXT command is executed. At this point, after incrementing, the counter variable is compared to the final value. If the counter is now out of the range of the initial and final values, program control will continue to the command following the NEXT command.

3. If the STEP is not specified, the increment is assumed to be 1. If the step is negative, the final value must be less than the initial value.

4. FOR..NEXT loops may be nested, that is, you can place a FOR..NEXT loop inside another FOR..NEXT loop. Nested loops must have a unique counter for each loop. The NEXT command for the inside loop must appear before the NEXT command for the outer loop. Up to 20 nested loops are allowed.

Differences from Interpreter

1. Interpreter allows for expressions for the counter, initial and final values.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2.c GOSUB3.2.cFunction

To branch to and return from a subroutine beginning at a specified line number.

Format GOSUBlinenumber

linenumber -The first line in the subroutine. (0-65535)

Examples

1. 1 GOSUB1000:STOP

.

.

.

1000 PRINT"Entering Subroutine 1000":RETURN

-In the above example, line 1000 is called from line 1 and then execution is terminated by the STOP.

2. 10 ON 1+J/100+I GOSUB1000,2000,3000,4000

-In this example, the line number used in the GOSUB is computed in the expression 1+J/100+I.

Comments

1. You can call a subroutine any number of times in a program. Subroutines may be nested within another subroutine.

2. A RETURN statement in a subroutine causes a branch to the command following the most recent GOSUB statement.

3. A subroutine may contain as many RETURNS as logical flow requires.

4. If linenumber contains a nonexecutable command (eg. REM, DIM, REAL), then execution proceeds at the first executable statement encountered after "linenumber".

Differences from Interpreter

1. NONE

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2.f3.2.f GOTOFunction

To perform an unconditional branch from the current position in the program to a designated line number.

Format GOTOlinenumber

linenumber -Line number in BASIC source
(integer between 0 and 65535)

Examples

1. 10 GOTO1000

-In the above example, program control is transferred to the statements on line 1000.

Comments

1. If linenumber contains a nonexecutable command (eg. REM,DIM,REAL), then execution proceeds at the first executable statement encountered after "linenumber".

Differences from Interpreter

1. NONE

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.2.g IF..THEN (ELSE)3.2.gFunction

To make a decision regarding program flow based on the result returned by an expression.

Format **IF relation THEN statement(s) /ELSE statements(s)/**

relation -A comparison, using any relational operator, between two expressions (IE,SE,RE)
 statement(s) -Commands or statements (except IF..THEN)

Examples

1. 10 IF A=100 THENGOTO30

20 REM' skipped if A=100

30 REM' continue program

-This example shows a simple IF THEN statement. A shortcut for THENGOTO is just THEN, therefore line 10 may read- IF A=100 THEN30.

2. 10 IF SAS=B\$ THENPRINT\$A\$;"=";B\$ ELSEPRINT\$A\$;" ";B\$
 -In this example, two string variables are compared, and the result is to print the relation of the two strings on the screen. Note the "\$" must precede the 1st string expression if strings are to be compared (otherwise the two binary numbers A\$(0) and B\$(0) will be compared!).

3. 10 IF A+10.9/SIN(9*R)<P+R/TAN(U) THENGOSUB1000:GOTO10
 ELSEGOTO1000

-In this example, two real expressions are compared.

Comments

1. If the relation is true (its value is not zero), the THEN clause is executed. Execution continues until an ELSE is reached or the end of the BASIC compiled line is reached, in which case the program continues on the next BASIC compiled line.

2. If the relation is false, the THEN clause is ignored and the ELSE clause (if present) is executed. Execution continues until the end of the compiled BASIC line is reached.

3. The combination of commands THENGOTOlinenumber may be abbreviated as THENlinenumber for simplicity, as long as an ELSE does not follow.

Differences from Interpreter

1. The Interpreter allows nested IF..THEN statements.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2.h3.2.h OFF ERRORFunction

To disable any previously defined error handling routine.

Format OFF ERROR

Examples

1. 10 ON ERROR GOTO100
20 INPUT"Enter a number";A
30 OFF ERROR:REM' disable error vector

100 PRINT"INPUT ERROR, TRY AGAIN"
101 GOTO20:REM' RETRY IF ERROR OCCURS

-In the above program, OFF ERROR is used to turn off the ON ERROR that was defined on line 10.

Comments

1. OFF ERROR causes control to bypass any error called during execution of the statements that follow this command, until another ON ERROR statement is executed.
2. If the program does not contain any ON ERROR commands, the OFF ERROR is assumed and therefore does not have to be included in the program.

Differences from Interpreter

1. The Interpreter does not handle OFF ERROR.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B

MLBASIC 1.0 USER'S MANUAL

3.2.13.2.1 ON ERRORFunction

To enable control to pass to a line or a subroutine when an error condition occurs during execution of the compiled program.

Format ON ERROR GOTO:GOSUB linenumber

linenumber -line number where control is passed
(Integer value 0-65535)

Examples

1. 10 ON ERROR GOTO1000
20 INPUT "Enter input filename ";\$A\$
30 OPEN "I",#3,A\$
40 OFF ERROR

1000 PRINT"FILE NOT FOUND":GOTO20

In this example, if the file that is to be opened for input is not found on the disk, an error occurs, in which case the computer asks for the filename again.

Comments

1. ON ERROR GOSUB calls must call a routine that contains a RETURN, otherwise program execution may be altered if an error occurs.
2. If the program does not contain any ON ERROR commands, the OFF ERROR is assumed and therefore does not have to be included in the program.

Differences from Interpreter

1. The Interpreter does not handle ON ERROR.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2.13.2.1 ON GO(TO,SUB)Function

To branch to one of several specified line numbers, depending on the value returned when an expression is evaluated.

Format ON expression GOTO:GOSUBLinenumber,...

expression -Value which determines what
 the destination line is (positive IE)
linenumber -Line number in BASIC source
 (integer between 0 and 65535)

Examples

1. 10 ON TT-INT(SIN(U-1)) GOSUB100,200,300
20 .
30 .

In this example, subroutines 100,200 and 300 are called if the expression has the respective values of 1,2 or 3.

Comments

1. In the ON...GOSUB statement, each line number in the list must be the first line number of the subroutine.
2. If the expression has a value of zero or a value greater than the number of linenumbers in the list, execution will continue to the next statement.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard).

B

MLBASIC 1.0 USER'S MANUAL

3.2.k

3.2.k RETURN

Function

To return program control to the calling routine.

Format RETURNExamples

```
10 GOSUB1000  
20 .
```

```
1000 REM' subroutine entry
```

```
1000 .
```

```
1002 RETURN
```

In the above example, the subroutine 1000 was called, and when a RETURN in line 1002 is executed, program control goes to line 20.

Comments

1. The RETURN command must be used at the end of a subroutine that is called using GOSUB.
2. The RETURN command must be used to return control to the calling program when used with CALL and SUBROUTINE.

Differences from Interpreter

1. None

Roms Needed (ECB=Extended, DB=Disk, S=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.2.13.2.1 STOPFunction

To terminate program execution, and to resume control to the command level.

Format STOP

Examples

1. 10 STOP

Comments

1. The STOP should be used for program termination within the main body of the program.
2. Execution of the STOP command is the same as the END command.
3. When the STOP is executed, the 64k RAM mode is changed back to the 32k RAM-32k ROM mode, and control is returned to the interpreter.

Differences from Interpreter

1. The STOP does not allow re-entry into the machine language program using CONT, whereas the Interpreter allows this.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.2.m SUBROUTINE3.2.mFunction

To allow reference to a set of statements or commands by a single name and a list of parameters.

Format SUBROUTINE name(arg,...)

| | |
|------|--|
| name | -Name of Subprogram (up to 7 characters) |
| arg | -Variable to be passed to calling program or used as constant in subprogram (IV,RV,SV) |

Examples

1.

```

10 REM' test of how to call a subroutine
20 INPUT"enter a number ";A:PRINT
30 CALL TESTONE(A)
40 STOP
100 SUBROUTINE TESTONE(B)
101 PRINT"NUMBER=";B
102 RETURN
200 END

```

In this example, the subroutine TESTONE is called and the number that was input on line 20 is printed.

Comments

1. The subroutines, also called subprograms, return values to the calling program unit only through actual-dummy argument correspondence. In other words, the first variable in the SUBROUTINE statements list is set equal to the constant or variable that is first on the list in the CALL statement, and so on for all of the arguments on the list.

2. If the SUBROUTINE is to return a value to the calling program, the argument in the list of the CALL statement must be a variable.

3. Within the subroutine, name may only appear in the SUBROUTINE statement immediately following the word SUBROUTINE. Subroutine names are uniquely distinguished by their first seven characters.

4. The subroutine list must contain the same number of arguments as is contained in the CALL statement's list.

5. The arguments that are passed in the argument list of the CALL statement are pointers that are referenced by the subroutine program.

6. If an array is an argument on the list in a CALL statement, the first element that is referenced by the subroutine is the element that appears on the CALL statement list.

Differences from Interpreter

1. The Interpreter does not handle the SUBROUTINE statement.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B

MLBASIC 1.0 USER'S MANUAL

3.3 Math Functions

3.3.a ABS3.3.aFunction

To return the absolute value of the expression given as the argument.

Format ABS(expression)

expression -The value that gets passed to the function. (RE)

Examples

1.. 10 A=ABS(100-SIN(10-I)*2)

Comments

1. Negative expressions are made positive and the magnitude is unchanged. Positive numbers are unchanged.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.3.b ASC3.3.bFunction

To return the ASCII value of the string expression given as the argument.

Format ASC(expression)

expression -The value that gets passed to the function. (one letter SE)

Examples

1. 10 A=ASC(A\$)

In this example, the value of byte A\$(0) is returned to A.

Comments

1. This command is not necessary in MLBASIC; but is used only for compatibility with the interpreter. Example 1 could just as well be written as 10 A=A\$ and the result would be the same.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

LBASIC 1.0 USER'S MANUAL

3.3.c3.3.c ATNFunction

To return the arc tangent of the expression given as the argument.

Format ATN(expression)

expression -The value that gets passed to
the function. (IE,RE)

Examples

1. 10 DEGREES=ATN(Y/X)

Comments

1. The result is in the range of -pi/2 to pi/2.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

MLBASIC 1.0 USER'S MANUAL

3.3.d COS3.3.dFunction

To return the cosine of the expression given as the argument.

Format COS(expression)

expression -The value that gets passed to
the function. (IE,RE in Radians)

Examples

1. 10 X=R*COS(THETA)

This is the conversion from polar coordinates to the rectangular coordinate -X.

Comments

1. The value returned is a real value from -1 to 1.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.3.c3.3.c CVNFunction

Converts a binary coded string into a real number.

Format CVN(expression)

expression - The string containing the 5 byte
binary representation of a real number
(SE at least 5 bytes long)

Examples

1. 10 X=CVN(AS)

Comments

1. The string that gets passed to the CVN routine usually has been previously encoded using the MKNS\$ string function.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB, DB

3.3.f3.3.f EOFFunction

To return the end of file status of the expression given as the argument.

Format EOF(expression)

expression -The buffer that is being checked for the end of file
(IE values -2,0,1,2,...15)

Examples

1. 10 IF EOF(-1)=-1 THENCLOSE:STOP

In this example, if the end of file is reached on the cassette file, all files are closed and program execution stops.

Comments

1. The EOF function must have as its argument a buffer number, whose buffer was previously opened for input (OPEN "I" type).

2. If the end of file has been reached after an INPUT, the EOF call will return a -1, otherwise it returns a zero.

Differences from Interpreter

1. MLBASIC treats the values returned from an EOF call as an integer value.

2. The Interpreter allows the EOF call to be a true or false (LOGICAL) value.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.3.g3.3.g EXPFunction

To return the natural exponent of the expression given as the argument.

Format EXP(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10 A=EXP(4.988+I)

Comments

1. If the expresion is too large, an overflow error will occur when called.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

3.3.h3.3.h FIXFunction

To return the truncated (integer) value of the expression given as the argument.

Format FIX(expression)

expression -The value that gets passed to the function. (RE)

Examples

1. 10 WHOLENUMBER=FIX(A)

Comments

1. The difference between FIX and INT is that FIX does not return the next lower number for a negative expression.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

MLBASIC 1.0 USER'S MANUAL

3.3.13.3.1 INSTRFunction

To return the location in a string of another string.

Format INSTR(start,search,target)

start -Beginning character to start search (IE)
search -String in which the search is made (SV)
target -The string that is being searched for (SE)

Examples

1. 10 POSITION=INSTR(1,A\$,"Target")

Comments

1. If the start is greater than the length of the search string, a zero is returned.
2. If the string to be searched for is not found, INSTR will return a zero.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.3.j3.3.j INTFunction

To return the next highest integer value of the expression given as the argument.

Format INT(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10 IF INT(1/4.)=1/4. THENPRINT

In this example, a new line is printed to the screen when I=0,4,8 and so on. A real expression is formed when the variable, I, is divided by the real constant, "4." (otherwise, the expression I/4 will be integer if I is integer).

Comments

1. The INT function will truncate the decimal part of a number.

2. To round a number to the nearest whole integer, one must add 0.5 to the real expression. For example, the statement AB=INT(I+.5) rounds the variable, I, to the nearest whole number.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.3.k3.3.k LENFunction

To return the length of a string.

Format LEN(expression)

expression -The string value that gets passed to
the function. (SE)

Examples

1. 10 \$A\$=A\$+STRINGS(20-LEN(A\$),"")

In this example, the string variable, A\$, is made to be 20
characters long with the help of the LEN function.

Comments

1. If the string is of zero length, (the first element in
the string is a zero), LEN returns a 0.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.3.13.3.1 LOGFunction

To return the natural logarithm of the expression given as the argument.

Format LOG(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10A=LOG(1.987)

Comments

1. The LOG is the power to which the number e, 2.718271828, must be raised to result in the given argument.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.3.m3.3.m LOCFunction

To return the next record number of the specified buffer.

Format LOC(expression)

expression -The buffer number (IE)

Examples

1. 10 A=LOC(BUFFER)

Comments

1. The LOC function may only be used with files that have been opened for direct access ("D" option in OPEN).
2. The location of the next record is set to 1 if no records have been read (using GET).
3. The current record that exists in the buffer is equal to the LOF of that buffer minus one.

Differences from Interpreter

1. MLBASIC allows LOC(-1), while the Interpreter does not.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB, DB

MLBASIC 1.0 USER'S MANUAL

3.3.n LOF3.3.nFunction

To return the last record of a specified buffer.

Format LOF(expression)

expression -The buffer number (IE)

Examples

1. 10 IF LOC(1)-1=LOF(1) THENCLOSE:RETURN

In this example, if the location of the current record is the last record, execution terminates.

Comments

1. The buffer must have been opened using the direct access ("D") mode.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B, ECB, DB

MLBASIC 1.0 USER'S MANUAL

3.3.0 PEEK3.3.0Function

To return the one byte value of the specified memory location.

Format PEEK(expression)

expression -The memory location that gets passed to the function. (IE)

Examples

1. 10 A=PEEK(25)*256+PEEK(26)

Comments

1. The argument must be an allowable memory location (0-65535).

2. PEEK is the compliment to the POKE statement.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.3.p POINT3.3.pFunction

To return the value of the specified graphics cell.

Format POINT(x coord,y coord)

x coord -X coordinate in current graphics page (IE)
y coord -Y coordinate in current graphics page (IE)

Examples

1. 10 IF POINT(10,5)=0 THENPRINT"OFF" ELSEPRINT"ON"

Comments

1. The value returned is equal to -1 if the character mode is on.
2. If the graphics mode is on, the value returned is the current color which is any allowable non negative integer.

Differences from Interpreter

1. None.

Boms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.3.q PPOINT3.3.qFunction

To return the color of the specified graphics cell.

Format PPOINT(x coord,y coord)

x coord -X coordinate in current graphics page (IE)
y coord -Y coordinate in current graphics page (IE)

Examples

1. 10 C=PPOINT(X,Y)

Comments

1. The X and Y coordinates must be within the allowable range of the current graphics mode, otherwise misleading values will be returned.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

3.3.r3.3.r RNDFunction

To return a pseudo-random number between one and the expression given as the argument.

Format RND(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10 A=RND(100)

Comments

1. The argument in RND must be greater than one.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.3.s3.3.s SGNFunction

To return the sign of the expression given as the argument.

Format SGN(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10 IF SGN(A)<0 THENPRINT"NEGATIVE"
ELSEPRINT"NON-NEGATIVE"

Comments

1. The value returned is as follows:
 - 1 if expression > 0
 - 0 if expression=0
 - 1 if expression < 0

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.3.t3.3.t SINFunction

To return the sine of the expression given as the argument.

Format SIN(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10 A=SIN(THETA-3.14159)

Comments

1. The argument must be in radians.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.3.u3.3.u SQRFunction

To return the square root of the expression given as the argument.

Format SQR(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10 DISTANCE=SQR(X*X+Y*Y)

The square root function is used to find the distance between two points..

Comments

1. The argument must be greater than or equal to zero. Negative arguments result in a function call error.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

3.3.v3.3.v TANFunction

To return the tangent of the expression given as the argument.

Format TAN(expression)

expression -The value that gets passed to the function. (IE,RE)

Examples

1. 10 OPPOSITE=ADJACENT*TAN(THETA)

The tangent function can be used to find the length of an unknown side, given one side and the angle between the two sides.

Comments

1. The argument must be in radians
2. The tangent function is undefined at $\pi/2$ and $-\pi/2$. An overflow error will occur if the argument is sufficiently close to these points.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.3.w3.3.w TIMERFunction

To return a timer value from the microprocessor clock.

Format TIMER

Alternate Format TIMER=initvalue

 initvalue -Value that the clock is
 initialized to (IE)

Examples

1. 10 A=TIMER
2. 10 TIMER=0

Comments

1. The cassette and printing operations stop the counter.
2. The counter is incremented about every 1/60th of a second.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.3.x VAL3.3.xFunction

To return the numeric representation of a string.

Format **VAL(string)**

string -The numeric string (SE)

Examples

1. 100 NUMBER=VAL("1234.999")

-In this example, the variable, NU, is loaded with the number 1234.999. If NU was an integer (ie. it was not declared with REAL), the variable will be loaded with the number 1234.

Comments

1. The value returned is a real number.
2. The string expression must be a legal numeric string, otherwise an error will occur.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.4 String Functions

3.4.a CHR\$

3.4.a

Function

To return the character for the given argument.

Format **CHR\$(expression)**

expression -Any integer number
between 0 and 255 (IE)

Examples

1. 100 PRINT#-1,CHR\$(18);

The CHR\$ function is being used to select the graphics mode
for the line printer.

Comments

1. The CHR\$ function returns a single byte that contains
the quantity specified in the argument.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.4.b**3.4.b INKEY\$**Function

To return the character code for one scan of the keyboard.

Format INKEY\$

Examples

1. 100 \$A\$=INKEY\$:IFA\$=OTHEN100
101 PRINT\$A\$;:RETURN

In this example, the keyboard is scanned using the INKEY\$ function. The routine continues to scan the keyboard until a key is typed in. When the key is typed, the character is output to the screen and program control returns with the ASCII character code in element #0 of the string array, A\$.

Comments

1. If no key is typed when the INKEY\$ routine scans the keyboard, the routine will return a zero.
2. No characters are echoed with the INKEY\$ routine.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.4.c3.4.c LEFT\$Function

To return a specified amount of the left side of a specified string.

Format LEFT\$(string,length)

string -The string from which the final string is formed (SV)
length -The length of returned string (IE)

Examples

1. 100 \$A\$=LEFT\$(A\$,LEN(A\$)-1)

In this example, all but the last character in the string variable, A\$ is returned.

Comments

1. The maximum length of the string expression is 255 bytes.
2. If the length of the final string is greater than the string argument, the resulting string will only be as long as the initial string argument.

Differences from Interpreter

1. Only a string variable is allowed as the string argument in MLBASIC.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.4.d3.4.d MID\$Function

To return a specified amount of the middle of a specified string.

Format MID\$(string,position,length)

| | |
|----------|---|
| string | -The string from which the final string is formed (SV) |
| position | -The location in original string where new string starts (IE) |
| length | -The length of returned string (IE) |

Example 100 \$Z\$=MID\$(A\$,10,LEN(A\$)-5)

Comments

1. The maximum length of the string expression is 255 bytes.
2. If the length of the final string is greater than the string argument, the resulting string will only be as long as the initial string argument.

Differences from Interpreter

1. Only a string variable is allowed as the string argument in MLBASIC.
 2. The length must be included with MLBASIC.
 3. MLBASIC does not support the command MID\$(oldstr,position,length)=string. The command can be duplicated, since MLBASIC allows changing elements within a string. The following command in MLBASIC would perform the MID\$= function, as noted above:
- \$oldstr(position)=string+RIGHT\$(oldstr,LEN(oldstr)-length)

For Example, the Interpreter command:

10 MID\$(A\$,12,3)="NEW"

Can be performed by the following MLBASIC command:

10 \$A\$(12)="NEW"+RIGHT\$(A\$,LEN(A\$)-3)

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

MLBASIC 1.0 USER'S MANUAL

3.4.e MKNS3.4.eFunction

To return a 5-byte coded string that represents a real number in binary form.

Format MKNS(number)

number -The real value that gets coded into the 5 byte string (RE)

Examples

1. 100 \$A\$=MKNS(99.99999+I)
2. 100 LSET A1\$=MKNS(A)+MKNS(12345.)

The MKNS function is most useful in forming data within a direct access buffer field.

Comments

1. The MKNS function may form real numbers on mass storage devices such that the INPUT command may read the data back into a real variable without having to call the CVN function.

Differences from Interpreter

1. MLBASIC allows more general use of the MKNS function. The Interpreter only allows the use with fielded strings.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB, DB

3.4.f RIGHTS3.4.fFunction

To return a specified amount of the right side of a specified string.

Format **RIGHT\$(string,length)**

string -The string from which the final string
 is formed (SV)
length -The length of returned string (IE)

Examples

1. 100 \$A\$=RIGHT\$(B\$,100)

Comments

1. The maximum length of the string expression is 255 bytes.
2. If the length of the final string is greater than the string argument, the resulting string will only be as long as the initial string argument.

Differences from Interpreter

1. Only a string variable is allowed as the string argument in MLBASIC.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.4.g STR\$3.4.gFunction

To return the ASCII string of a given real number.

Format **STR\$(number)**

number -The number that gets converted to
an ASCII string (IE,RE)

Examples

1. 100 \$A\$=STR\$(100+A)

Comments

1. The first character in the string returned is the sign character. If the number is negative, this character is a "-", otherwise it is a space.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.4.h STRINGS3.4.hFunction

To return a string containing a specified number of a specified character.

Format STRING\$(length,character)

length -The number of times the character is to be repeated in the string (IE)

character -The one byte character code (IC,IV,SC)

Examples

1. 10 \$A\$=A\$+STRING\$(20-LEN(A\$)," ")

In the above example, the string variable, A\$, is padded with spaces on the end such that the string is always 20 bytes long.

Comments

1. The maximum length of the string is 255 bytes.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5 Graphics and Sound Commands

3.5.a AUDIO3.5.aFunction

To turn on the sound from the cassette.

Format AUDIO ON:OFFExamples

1. 100 AUDIO ON

Comments

1. The audio is normally turned off, so AUDIO OFF is not needed.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.5.b3.5.b COLORFunction

To specify the background and foreground colors of the graphics screen.

Format COLORforeground,background

foreground -Color of foreground
(IE as allowed in current PMODE)
background -Color of background
(IE as allowed in current PMODE)

Examples

1. 100 COLOR 5,7

Comments

1. If COLOR is not used, the computer sets the foreground to the highest color code allowed and the background to the lowest allowable color code.

2. The following numbers represent the allowable color codes:

- 0 - Black
- 1 - Green
- 2 - Yellow
- 3 - Blue
- 4 - Red
- 5 - Buff
- 6 - Cyan
- 7 - Magenta
- 8 - Orange

Differences from Interpreter

- 1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.c CLS3.5.cFunction

To clear the text screen to a desired color.

Format CLS /color/

color -Color of foreground screen (IE)

Examples

1. 100 CLS
2. 100 CLS6

Comments

1. If the color is omitted, the screen is cleared to the color green.

2. The following numbers represent the allowable color codes:

- 0 - Black
- 1 - Green
- 2 - Yellow
- 3 - Blue
- 4 - Red
- 5 - Buff
- 6 - Cyan
- 7 - Magenta
- 8 - Orange

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.5.d3.5.d CIRCLEFunction

To draw a circle to the graphics screen.

Format

CIRCLE(x,y),radius/,color//,hw/,start,end/

x

-X coordinate of circle's center (IE)

y

-Y coordinate of circle's center (IE)

radius

-The circle's radius. One unit of radius is equal to one point on the screen (IE)

color

-The color code of the circle

hw

-The height/width ratio (RE from 0 to 256)

start

-The starting point on circle where circle is made (RE from 0 to 1)

end

-The point in the arc where the circle is terminated (IE from 0 to 1)

Examples

1. 100 CIRCLE(50,50),10

This example draws a circle 10 units in radius, centered at (50,50)

2. 100 CIRCLE(50,50),10,.1,.1,.1,.2

This example draws an arc centered at (50,50), with a radius of ten units, from .1 to .2 in the color green.

Comments

1. Items that appear in the list before an optional item that is selected must be included. For example, if start and end are used, color and hw must be included.

2. If the ending point is less than or equal to the starting point, a complete circle is drawn.

3. The default values for the optional items are as follows:

color -Current foreground color
hw -The value 1
start -The value 0
end -The value 1

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
 B, ECB

3.5.e DRAW3.5.eFunction

To draw in the graphics mode according to a given sequence of pre-established commands.

Format DRAWcommand string

command string -The string that contains the shape to draw (SE)

Examples

1. 100 DRAW"BM100,50,U10,R10,D10,L10"

In this example, a box, 10 units per side, is drawn.

Comments

1. The following commands are allowable:

Motion Commands

M - Draw to X,Y coordinate equal to the origin plus a specified X,Y offset.

U - Move up a specified number of units

D - Move down a specified number of units

L - Move left a specified number of units

R - Move right a specified number of units

E - Move up then right a specified number of units

F - Move up then left a specified number of units

G - Move down and left a specified number of units

H - Move down and right a specified number of units

X - Execute a BASIC defined substring

Modes

C - Color code to use

0 - Black

1 - Green

2 - Yellow

3 - Blue

4 - Red

5 - Buff

6 - Cyan

7 - Magenta

8 - Orange

A - Angle (0=0 degrees, 1=90, 2=180, 3=270)

S - Scale factor in 1/4 increments

(1=1/4 scale, 2=1/2, 3=3/4, 4=full, 5=5/4, ...)

MLBASIC 1.0 USER'S MANUAL

Options

- N - Do not update cursor origin
- B - Do not draw, just move

Differences from Interpreter

1. The substring execute command must execute a string defined in the Interpreter mode. The "[" special character is used in the following example:

```
100  [A$="D10;R10;U10;L10;"  
101  DRAW"BM100,50;XA$"
```

In this example, the substring defined has no effect on the string variable A\$, if used elsewhere in the program.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.f3.5.f LINEFunction

To draw a line between two points.

Format **LINE(x₁,y₁)-(x₂,y₂),action/option/**

| | |
|----------------------|---|
| <u>x₁</u> | -X coordinate of starting point (IE) |
| <u>y₁</u> | -Y coordinate of starting point (IE) |
| <u>x₂</u> | -X coordinate of ending point (IE) |
| <u>y₂</u> | -Y coordinate of ending point (IE) |
| <u>action</u> | -How to draw the line. Allowable are: PSET - Sets line to foreground color PRESET - Sets line to background color |
| <u>option</u> | -Box option: B - Draw a box using points as the corners of the box BF - Draw a box, and fill it in |

Examples

1. 100 LINE(1,1)-(11,11),PSET

In this example, a line is drawn from (1,1) to (11,11).

2. 100 LINE(1,1)-(11,11),PSET,BF

In this example, a box is filled in between (1,1) and (11,11).

Comments

1. The allowable limits on the X and Y coordinates are from 0 to 255 in the X-direction and from 0 to 191 in the Y-direction.

Differences from Interpreter

1. MLBASIC requires that the starting point be defined.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.g PAINT3.5.gFunction

To paint the screen between a pre-established border, in a specified color.

Format PAINT(x coord,y coord),color,border

| | |
|---------|---|
| x coord | -X coordinate where painting begins (IE) |
| y coord | -Y coordinate where painting begins (IE) |
| color | -Color code to paint with (IE) |
| border | -Color code of border where painting is to stop (IE) |

Examples

1. 100 PAINT(10,10),4,4

Comments

1. The color used in the PAINT command must be allowable under the current PMODE and color set.
2. When the color specified is higher than the allowable color, the color has the color set number subtracted from it. For example, if there were four available colors and the color code 5 was used, the actual color painted will be the code 1 (=5-4).

Differences from Interpreter

1. None.

ROMs Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.h3.5.h PCLEARFunction

To reserve space in memory for a graphic page.

Format PCLEARpage

page -Total number of 1.5K graphics pages (IE)

Examples

1. 10 PCLEAR16

In this example, 16 graphics pages are being reserved in memory. This allows 4 high resolution screens to exist in memory at the same time.

Comments

1. The PCLEAR command clears memory in order to make room for the graphic pages.

2. The PCLEAR command, if used improperly, will crash the program, or give runtime error warnings.

3. The maximum allowable number of pages that can be cleared depend on the amount of memory available in the lower 32k of memory.

4. Graphic pages are not allowed to exist in the upper 32k of RAM (32768-65535)

Differences from Interpreter

1. MLBASIC allows more than 8 graphic pages to be cleared.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.1 PCLS3.5.1Function

To clear the graphics screen.

Format PCLS/color/

color -Color code to clear screen in (IE)

Examples

1. 100 PCLS3

Comments

1. If the color is omitted, the current background color is used.
2. The PCLS command is used to clear the graphics screen in the same way as CLS is used to clear the text screen.
3. The following numbers represent the allowable color codes:

| | |
|---|-----------|
| 0 | - Black |
| 1 | - Green |
| 2 | - Yellow |
| 3 | - Blue |
| 4 | - Red |
| 5 | - Buff |
| 6 | - Cyan |
| 7 | - Magenta |
| 8 | - Orange |

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.j3.5.j PLAYFunction

To play music according to a pre-established sequence of commands.

Format PLAYstring

string - Sequence of commands that define the musical "score". (SE)

Examples

1. 100 PLAY A\$+B\$+"CDEFG;O3;ABA02;FEDCBA"

Comments

1. The following commands are allowed in the PLAY statement string:

| <u>Command</u> | <u>Function</u> |
|----------------|---|
| Note | - The Note to be played consisting of: A number from 1 to 12 or The letters A to G (plus #=sharp, -=flat) |
| O | - Allows selection of other octaves (1-5) |
| L | - Allows choosing of the note length where the number that follows has the length in 1/L time. For example: L1=whole, L2=half, L4=quarter, L16=one sixteenth (allowable lengths are 1 to 255) |
| T | - The tempo to be selected (1 to 255). The tempo T2 is used by default |
| V | - The volume may be selected (1 to 31). The volume V15 is used by default |
| P | - The pause-length (1 to 255) where the duration is 1/P. For example: P1=full, P4=quarter, P8=eighth, P2P4=3/2, etc |
| X | - Execute a substring defined in BASIC |

Differences from Interpreter

1. The substring execute command must execute a string defined in the Interpreter mode. The "[" special character is used in the following example:

```
100 [A$="CDEFG;O3;ABA02;FEDCBA"
101 PLAY"T4;V5;XA$"
```

In this example, the substring defined has no affect on the string variable A\$, if used elsewhere in the program.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.k PMODE3.5.kFunction

To select the desired graphics mode and page.

Format PMODEmode,page

mode -Graphics mode to select (IE value 0 to 4)
page -Starting graphics page (IE value 1 to 8)

Examples

1. 100 PMODE4,1

Comments

1. If the PMODE is not used in a graphics program, the default is PMODE2,1.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

NLBASIC 1.0 USER'S MANUAL

3.5.1 PRESET3.5.1Function

To reset a point to the background color.

Format **PRESET(x coord,y coord)**

x coord -X coordinate of point (IE)

y coord -Y coordinate of point (IE)

Examples

1. 100 PRESET(10,10)

Comments

1. The PRESET command does not need a color for the argument since the color used is always the current background color.

2. The RESET command differs from the PRESET command in that the first is for low-resolution graphics, and the latter is for all-resolution graphics.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.m3.5.m PSETFunction

To set a point to a specified color.

Format PSET(x coord,y coord/,color/)

x coord -X coordinate of point (IE)
y coord -Y coordinate of point (IE)
color -Color code of point to set (IE)

Examples

1. 100 PSET(20,20,2)

Comments

1. If the color code is omitted, the current foreground color is used.
2. The following numbers represent the allowable color codes:

| | |
|---|-----------|
| 0 | - Black |
| 1 | - Green |
| 2 | - Yellow |
| 3 | - Blue |
| 4 | - Red |
| 5 | - Buff |
| 6 | - Cyan |
| 7 | - Magenta |
| 8 | - Orange |

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, S=Standard)
B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.n RESET3.5.nFunction

To reset a point to the background color.

Format **RESET(x coord,y coord)**

 x coord -X coordinate of point (IE)
 y coord -Y coordinate of point (IE)

Examples

1. 100 RESET(10,10)

Comments

1. The RESET command does not need a color for the argument since the color used is always the current background color.
2. The RESET command differs from the PRESET command in that the first is for low-resolution graphics, and the latter is for all-resolution graphics.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.5.03.5.0 SCREENFunction

To define the screen display and color set.

Format SCREENtype, set

| | |
|------|---|
| type | -Type of screen 0=text, 1=graphics (IE) |
| set | -Color set to use |
| | 0=Green, Yellow, Blue, Red -4 Color Mode |
| | 0=Black, Green -2 Color Mode |
| | 1=Buff, Cyan, Orange, Magenta -4 Color Mode |
| | 1=Black, Buff -2 Color Mode |

Examples

1. 100 SCREEN1,1

Comments

1. If the color set is greater than one, the value one is used.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B, ECB

MLBASIC 1.0 USER'S MANUAL

3.5.p SET3.5.pFunction

To set a point to a specified color.

Format SET(x coord,y coord/,color/)

x coord -X coordinate of point (IE)
y coord -Y coordinate of point (IE)
color -Color code of point to set (IE)

Examples

1. 100 SET(20,20,2)

Comments

1. If the color code is omitted, the current foreground color is used.
2. The following numbers represent the allowable color codes:

0 - Black
1 - Green
2 - Yellow
3 - Blue
4 - Red
5 - Buff
6 - Cyan
7 - Magenta
8 - Orange

3. SET only allows low-resolution graphic mode.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)
B

MLBASIC 1.0 USER'S MANUAL

3.5.q SOUND3.5.qFunction

To sound a specific tone for a specific duration.

Format SOUNDtone,duration

tone -Tone of sound (IE from 1 to 255)

duration -Length of note (IE from 1 to 255)

Examples

1. 100 SOUND100,100

Comments

1. The duration of one unit is about 6/100ths of a second. This means that the range of durations is from 6/100ths of a second to 15.3 seconds.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.6 Other Commands

3.6.a DATA3.6.aFunction

To store string and numeric constants for use with the READ statement.

Format DATA/mode,/data,...

| | |
|------|---|
| mode | -Mode of storing data constants \$ - Store data in a string format (terminate string bytes with 0) % - Store number as one byte integer (two byte integers are default) |
| data | -String or numeric data (SC,IC,RC) |

Examples

1. 100 DATA\$"THIS IS A STRING"

In this example, a string constant is stored, which can later be read using a command like READ\$A\$.

2. 100 DATA%160,99,56,200,109,107,23,123,88

101 DATA%190,193,198,99,87,57

102 GOSUB100:REM' Execute a M.L. ROUTINE

In this example, the data lines 100-101 contain machine language instructions. Each item in the data list occupies only one byte in memory. It is possible to store machine language routines in data statements, and execute them using GOSUB or GOTO.

Comments

1. If the "\$" mode is not used with strings, a terminating zero is not stored. In this case, a READ VAR\$(I) type command might be used to read the string data one character at a time.

2. All DATA statements must be grouped together. In other words, the DATA statements must not have any other commands like PRINT, INPUT, etc, between them. The location of the group of DATA statements can be anywhere in the program.

3. A RESTORE must be used to initialize the data pointer to the beginning of the data list.

Differences from Interpreter

1. A RESTORE must be used to initialize the data pointer to the beginning of the data list in MLBASIC.

2. Data statements must be grouped.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.6.b DIM3.6.bFunction

To reserve space in memory for a variable array.

Format **DIM arrayname,...**

arrayname - Name of array followed by number
of bytes to reserve for each dimension

Examples

1. 100 DIM A(100,10),B\$(10,10),C\$(100)

In this example, a 100 by 10 integer array is defined.
Also, a 10 by 10 and a 100 element string array are declared.

Comments

1. Only single character variable names are recognized, although any length name is acceptable.
2. Dimensioned integer arrays may not have a dimensioned real array with the same name.
3. The command REAL is used to dimension real arrays.

Differences from Interpreter

1. Only single letter array names are recognized in MLBASIC.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.6.c**3.6.c LLIST**Function

To list a sequence of BASIC lines to the printer.

Format LLISTrange

range -Value or range of values (IC)

Examples

1. 100 LLIST0-65000

In this example, all possible lines will be listed to printer if line 100 is executed (in the compiled program).

Comments

1. Only BASIC program lines are printed. Compiled programs are not listed.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MILBASIC 1.0 USER'S MANUAL

3.6.d MOTOR3.6.dFunction

To control the cassette motor.

Format MOTOR ON:OFF

Examples

1. 100 MOTOR ON

Comments

1. The cassette motor is OFF by default.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.6.e3.6.e POKEFunction

To store a byte in memory.

Format POKE memory,byte

memory -Location in memory to store byte (IE)
byte -Value from 0 to 255 (IE)

Examples

1. POKE25,6:POKE26,1

The POKE command is often used to control Interpreter functions. In this example, the start of the BASIC program in memory is POKEd into memory.

Comments

1. The POKE is complemented by the command PEEK.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.6.f READ3.6.fFunction

To read a numeric or string value from a DATA list and to assign it to a variable.

Format READ/mode/name,...

mode -Type of data to be read.
 \$=string data with zero byte terminator
 %=one byte binary data
name -Name of variable to read data into (RV, IV, SV)

Examples

1. 100 RESTORE
- 101 READ\$A\$
- 102 READ%B

Comments

1. If the mode is not used with a string variable, the READ will return one byte to the specified string element.

Differences from Interpreter

1. The Interpreter does not support the "\$" and "%" options.
2. MLBASIC requires "\$" be used in front of the string variable name, if the entire string is to be read into the array.
3. With MLBASIC, a RESTORE must be used before the first READ statement, so that the DATA list is initialized to the first item.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B

MLBASIC 1.0 USER'S MANUAL

3.6.g3.6.g REMFunction

To display a message within a program.

Format REMremarks

remarks -Any nonzero byte.

Examples

1. 1000 GOSUB20000:REM' CALL ROUTINE TO SORT

The REM is often used to indicate to the programmer what is going on in the program itself.

Comments

1. The REM statement must be the last statement in a BASIC line.
2. The REM statement does not occupy any space in the final compiled program. MLBASIC simply skips over these commands, and does not have to translate them.
3. If the line containing a REM has no executable instructions (ie. PRINT, INPUT, etc), then program control passes to the first executable command after the REM statement.
4. REM statements can be branched into from a GOSUB or GOTO call. Execution will begin with the first executable command that follows the REM statement.

Differences from Interpreter

1. MLBASIC only allows REM to appear at the end of a line.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.6.h RESTORE3.6.hFunction

To initialize the data pointer to the first item in DATA list.

Format RESTORE

Examples

1. 100 DATA 1,2,3,4
- 101 RESTORE:REM' initialize data
- 102 FOR I=1 TO 4:READA(I):NEXT

Comments

1. The RESTORE must be used before any READ statement is executed.
2. After a RESTORE is executed, the next READ statement will begin reading data from the first item in the first DATA statement that appears in the program.
3. The RESTORE must appear after the DATA statement. In other words, the RESTORE may not appear in the program before a DATA statement.

Differences from Interpreter

1. MLBASIC requires the RESTORE to be used before any READ command is executed.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B

MLBASIC 1.0 USER'S MANUAL

3.6.1 RUN3.6.1Function

To execute a BASIC program.

Format **RUN/linenumber/**

linenumber -Number of entry into BASIC program (IC)

Examples

1. 100 RUN1000

Comments

1. This command is used to run a BASIC program from within a compiled machine language program.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.6.j TAB3.6.jFunction

To position output in a PRINT statement to a specified column.

Format TAB(position)

position -Position of tab (IE)

Examples

1. 100 PRINT#-2,"TOTAL=";TAB(30)TOTAL

Comments

1. If the current column position is less than the tab position, spaces (ASCII #32) are output to the device, until the tab position is reached.
2. If the current column position is greater than the tab position, backspaces are output until the tab position equals the current column position. Note that printers that do not support backspacing (ASCII #8), cannot have a TAB less than the current print location.

Differences from Interpreter

1. With MLBASIC, the TAB will output backspaces if the current column position is greater than the tab.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.6.k VERIFY3.6.kFunction

To select the verification option for disk output.

Format VERIFY ON:OFFExamples

1. 100 VERIFY ON

Comments

1. If the VERIFY ON command is used, all disk output will be verified with memory contents.
2. By default, the VERIFY option is not "ON", therefore a VERIFY OFF is not necessary in a program.

Differences from Interpreter

1. None.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B, ECB, DB

3.7 Special Commands

3.7.a DLD

3.7.a

Function

To load a 16 bit integer value from memory into a variable.

Format DLD(memory,name)

| | |
|---------------|--|
| <u>memory</u> | -Location in memory of first byte of the two byte integer (IV,IC from 0 to 65535) |
| <u>name</u> | -Name of variable which stores the 16 bit integer (IV) |

Examples

1. DLD(25,BSTART)

The DLD command is used to find the starting location of the BASIC program in memory and store the result in an integer variable called BS.

Comments

1. The DLD command is the 16 bit equivalent to the PEEK command.
2. DLD is not allowed inside an expression as PEEK is allowed.
3. DLD is the complement to the command DST.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.7.b DST3.7.bFunction

To store a 16 bit integer into two bytes of memory.

Format DST(memory,value)

memory -Location in memory of first byte
of the two byte integer (IV,IC from 0 to 65535)
value -16 bit integer that is stored (IC,IV)

Examples

1. 100 DST(40000,1000)

Comments

1. The DST command is the 16 bit equivalent to the POKE command (which only stores an 8 bit value).

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended,DB=Disk,B=Standard)

B

3.7.c3.7.c IBSHFTFunction

To shift a 16 bit integer by a specified number of bytes either to the right or to the left.

Format **IBSHFT(name,shift,direction)**

name -Variable that is to be shifted (IV)
shift -This is the number of bits the
 integer is shifted by (IC, IV from 1 to 16)
direction -This determines whether to shift
 left or right. (IC,IV)
 If the direction is:
 0 => shift to the left
 greater than 0 => shift to the right

Examples

1. 100 IBSHFT(A1,5,1)

In this example, the integer variable, A1, is shifted to the right 5 bits. This is equivalent to the command $A1=A1/32$, but is much faster.

2. 100 IBSHFT(A,8,0)

In this example, the integer variable, A, is shifted to the left by 8 bits. This is equivalent to the command $A=A*256$, but is much faster.

Comments

1. The IBSHFT command is very useful for graphics routines that perform alot of bit manipulation.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)
B

3.7.d3.7.d LREGFunction

To load a specified hardware register with an integer value.

Format LREG(register,value)

register -Name of hardware register. Allowable names are:

"X" -Index Register, X
"Y" -Index Register, Y
"U" -User Stack pointer
"S" -Hardware stack pointer
"D" -Data register
"PC"-Program counter
"CC"-Control register
"DP"-Direct page Register

value -Integer to be stored in register (IC,IV)

Examples

1. 100 LREG("S",INITVALUE)

In this example, the stack is being reset to a value contained in the integer variable, IN.

Comments

1. The LREG command is most useful for setting up calls to machine language routines that require initial values for the hardware registers.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.7.e3.7.e PCOPYFunction

To copy a specified amount of memory to another location in memory.

Format **PCOPY start,destination,end**

| | |
|--------------------|--|
| start | -Beginning location of data to move (IC,IV) |
| destination | -First location in memory where data is moved to (IC,IV) |
| end | -Ending location of data to move (IC,IV) |

Examples

1. 100 PCOPYA,1537,B

Comments

1. This command is the fastest way to transfer a section of memory from one location to another.

Differences from Interpreter

1. The Interpreter does not allow use of this command.
2. The PCOPY command used in the Interpreter allows for only a specified "page" of memory to be copied from one location to another. The way to convert an Interpreter PCOPY into the MLBASIC form is as follows:

To convert PCOPY A TO B

- (A) Let $A1=A*1536$
- (B) Let $A2=A1+1535$
- (C) Let $B1=B*1536$
- (D) The command is ready to form
PCOPY A1,B1,A2

Example-Interpreter form
100 PCOPY A TO BMLBASIC form

100 A1=A*1536:B1=B*1536:A2=A1+1535
101 PCOPYA1,B1,A2

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.7.f PMODD3.7.fFunction

To determine one of three graphics modes starting at a specified 512 byte boundary.

Format PMODDmode,page

mode -Graphics mode to select. (IC,IV)
The allowable modes are as follows:
2 - Text mode
3 - Graphics 6-C (128x192 4Color)
4 - Graphics 6-R (256x192 2Color)

Page -Starting 512 byte page in screen (IC,IV)

Examples

1. 100 INPUT"Enter starting page ";PAGE:PMODE4,PAGE
101 GOTO101

In this example, the user is asked to select a desired high resolution graphics page.

2. 10 PMODE4,10

In this example, the high resolution, 2 color, graphics mode is selected, and the screen is located starting at location 5120.

Comments

1. The PMODD command allows any graphics page to be displayed on the screen (Including pages above 32k!).
2. The PMODD2,2 command is the normal command used to select the standard text screen.
3. The selection of a different graphics mode than that used in a PMODE command previously has no effect on the mode used in the PMODE command.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.7.g PTV3.7.gFunction

To load a specified integer variable with the pointer to a specified variable.

Format PTV(variable,pointer)

variable -Variable or array element name (IV,RV,SV)
pointer -Variable where pointer is stored (IV)

Examples

1. PTV(A\$,START)

In this example the integer variable, ST, is loaded with the pointer to string array element, A\$(0).

2. PTV(A(10,10),A)

In this example the integer variable, A, is loaded with the pointer to A(10,10).

Comments

1. The PTV may not be used in an expression like
A=PTV(A,A).

2. The PTV command is equivalent to the VARPTR command.

Differences from Interpreter

1. The interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.7.h3.7.h REALFunction

To declare real type variables and variable arrays.

Format REALname...

name -Name if variable or
array maximum element number(s)

Examples

1. 100 REAL A,A1,A(10,10),B(1000)

In this example, the scalar variables A and A1 are declared as real variables. In addition, the array A is declared as real and is dimensioned for a 10x10 array. The array B is declared real also, and is dimensioned as having 100 elements.

Comments

1. The REAL command must be used to declare a variable as a real variable. Otherwise the variable will be of type integer.

2. After a variable has been declared as a real, that variable will be compiled in the following lines as a real variable. If the variable was used in lines that came before the line containing the REAL declaration, the variable is treated as an integer.

3. Compiler printouts will indicate whether a variable has been declared a real or not.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard).

B

3.7.1 SREG

3.7.1

Function

To load a specified variable with the contents of a specified hardware register.

Format SREG(register,name)

register -Name of hardware register. Allowable names are:

"X" -Index Register, X
"Y" -Index Register, Y
"U" -User Stack pointer
"S" -Hardware stack pointer
"D" -Data register
"PC"-Program counter
"CC"-Control register
"DP"-Direct page Register

name -Variable where register is stored (IV)

Examples

1. 100 SREG("PC",START)

In this example, the current location of the machine language program counter is stored in variable, ST.

Comments

1. The SREG command is most useful for program debugging. Other uses for the SREG command would be to recover data from the "D" register after a ROM call using the VECTI and VECTD commands.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard) B

3.7.j3.7.j SWPFunction

To swap the contents of an Interpreter variable or array with the contents of an MLBASIC variable or array.

Format **SWP(name1, name2)**

name1 -Name of MLBASIC variable (RV, SV).
name2 -Name of Interpreter variable to swap

Examples

1. 100 SWP(A,SWAP)

In this example, the MLBASIC and Interpreter variables, A and SW respectively, have their values exchanged.

2. 100 SWP(A\$,A\$)

In this example, the MLBASIC string array A\$ is swapped with the Interpreter string A\$.

Comments

1. The SWP does not accept MLBASIC variables if they are of type integer.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

3.7.k VECTD3.7.kFunction

To execute a machine language routine address located in ROM.

Format **VECTD(address)**

address -Address in ROM to be executed (IC,IV)

Examples

1. 100 VECTD(41175)

In this example, the location that prints the Interpreter revision number is executed.

Comments

1. This command is designed to switch from the all RAM map type (which enables you to use all 64k of memory), to the 32k-RAM/32k-ROM map type. After execution of the ROM routine, the map is switched back to re-enable all 64k of RAM.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

MLBASIC 1.0 USER'S MANUAL

3.7.13.7.1 VECTIFunction

To execute a machine language routine contained in ROM using indirect addressing.

Format **VECTI(address)**

address - Location in ROM that contains the 16 bit address that is to be executed (IC,IV)

Examples

1. 100 VECTI(\$A004):VECTI(\$A006)

In this example, the routines that turns the cassette on and reads a block from the cassette are executed.

Comments

1. The Indirect addressing allows the user to execute a machine language routine in ROM that is pointed to in a table contained in ROM.

Differences from Interpreter

1. The Interpreter does not allow use of this command.

Roms Needed (ECB=Extended, DB=Disk, B=Standard)

B

CHAPTER 4 VARIABLES, CONSTANTS, OPERATORS and EXPRESSIONS**4.1 Constants**

MLBASIC allows for 3 different types of constants; INTEGER, STRING and REAL. All constants are fixed values that are stored in the text area of the machine language program during compilation. Constants therefore cannot be changed when the program is run.

4.1.a Integer Constants

An integer constant contains an optional sign (+ or -) followed by decimal or hexadecimal digits. If hexadecimal digits follow, the "\$" or "&H" letters must precede the digits. No decimal points or commas are allowed. The Value an unsigned integer may have ranges from 0 to 65535. Numbers larger than 32,767 are treated as negative "two's complement" values when used with Real variables or constants in an expression. Arithmetic statements that do not contain real values use integers as positive numbers only (see Section 4.2.c for more info on conversions). Certain commands allow for Integer Constants to be expressed as one or two characters in quotes.

Examples of Integer Constants

| Valid | Invalid |
|--------|---------|
| 12345 | 12,345 |
| -100 | -100000 |
| 65000 | -65000 |
| \$FF01 | FF01 |
| &HA10B | A10B |
| "Ap" | Ap |
| "A" | A |

4.1.b String Constants

A string constant is a sequence of up to 255 characters enclosed in quotation marks. String constants may contain any character except a zero (ie. any value between 1 and 255). Strings are terminated by a logical zero byte when stored in memory by the compiler.

Examples of String Constants

1. "This is a string"
2. "\$25,000.01"
3. "\$,& and any character can be in strings"

MLBASIC 1.0 USER'S MANUAL

4.1.c. Real Constants

A real constant contains an optional sign (+ or -) followed by decimal digits which must contain, be preceded by, or followed by a decimal point. A real constant may be in exponential format, where the number is followed by an "E", followed by a + or - and decimal digits that describe the exponent.

In all cases, the decimal point is mandatory. If the decimal point is omitted, integer conversion will occur, resulting in possible overflow or underflow errors. Real constants are stored in the text area in their actual 5 byte binary format.

Examples of Real Constants

| <u>Valid</u> | <u>Invalid</u> |
|--------------|----------------|
| -100.10 | -100 |
| 1.99 E+10 | 199E+12 |
| 1.0 E-10 | 1 E-10 |
| -99.6E+10 | -99 |

MLBASIC 1.0 USER'S MANUAL

4.2 Variables

Variables are names that represent values used in BASIC programs. Variables can represent either a numeric value or a string expression. Allowable names of variables are unlimited, except for reserved Basic words that are used to identify BASIC commands and statements (ie. PRINT, GET, etc). There are two main groups of variables; scalar variables and variable arrays. There are also three types of variables; Real, Integer, and String. String variables must be variable arrays, while Real and Integer variables can be either scalar variables or arrays.

4.2.a Scalar Variable Names

MLBASIC allows a unique variable using the first 2 characters in the variable name. In other words, any letters that follow the first two letters in a variable name are ignored by the compiler. For example, the 3 variables; "A123", "A1VAR", and "A12" are all equivalent to "A1".

4.2.b Integer Variables

Integer variables follow the same guidelines as constants; values may be between zero and 65535 (&FFFF).

All variables, except string variables, are defaulted to the integer type, unless previously declared using the REAL command.

MLBASIC 1.0 USER'S MANUAL

4.2.c String Variables

String variables are single dimensioned arrays of characters terminated by a zero byte. The size of the string is determined by the allocated space. Unlike the Interpreter, MLBASIC requires that all strings be dimensioned before they are used. For example, to allow for string variable, A\$, to be 1000 bytes long; the command DIM A\$(1000) must precede use of the string. In the above example, only one string is being dimensioned.

String variables can be one or two dimensional, where a 2 dimensional string could be thought of as a multiple number of single dimensioned string arrays.

Strings may have their individual elements changed by treating each element as an Integer variable whose value must be between 0 and 255. For example, A\$(10)=0 places a terminator byte in position #11 (zero counts as #1), thereby creating a string of length 10. String elements may appear in an arithmetic expression, where that string element is treated as an integer whose value ranges from 0 to 255. For example, the equation: A=10*Z\$(10) is a valid equation with MLBASIC.

String variables may have their entire contents changed (in all elements) by using the special character \$ before the variable. For example, \$A\$(10)="String" will assign the string, "String", to elements #10-15 and will put a zero byte in element #16.

Single dimension strings may be abbreviated to a form similar to the way the Interpreter accepts variables, if the first element in the string is #0. For example, the expression \$A\$(0)="String" may be abbreviated to \$A\$="String". When the string variable is equated to another string expression, the special character \$ that precedes a string expression, should not be used. For example, \$A\$=B\$ is the proper method for equating B\$ to A\$. This rule applies to many of the commands in MLBASIC, so consult Chapter 3 for proper use.

Strings may be used where the 1st character is not the 1st element in the string. This is similar to the MID\$ = command available only with Extended Basic. For example, \$A\$(100)="String" will form a string starting at the 101st element of A\$.

4.2.d Real Variables

Real variables must first be declared using the REAL command. Allowable values are in the range of $+/- 1.0E+38$. The Binary Format is the same as the Interpreter, (ie. 5 Bytes with a one byte exponent). This means that computation using the Interpreter should give the same results as MLBASIC compiled REAL expression computation.

4.2.e Variable Conversions

Whenever necessary, MLBASIC converts a numeric constant or variable from one type to another.

The following rules apply to conversion of variable types in an arithmetic expression:

- (1) Expressions that involve both Real and Integer type variables, constants or functions, will be considered of type Real.
- (2) Expressions that involve only Real variables, constants or functions, will be of type Real.
- (3) Expressions that only contain Integer type variables, constants or functions, will be of type Integer.
- (4) Functions or commands that require a specific type expression will convert that expression to the required type, if it is not so already.
- (5) Integer expressions are converted to real expressions as "Two's Complement" integers. This means that an integer whose value is greater than 32767 will be converted into a negative real number.
- (6) Real expressions that are outside the range of $+/- 32767$ cannot be converted to type integer. If conversion is performed, a runtime error #5 will occur.

MLBASIC 1.0 USER'S MANUAL

4.3 Variable Arrays

MLBASIC allows up to two dimensions for any variable array. In the case of strings, variable arrays are required in order to store the series of characters that form the string. In all cases, arrays must be declared using the DIM or REAL commands, before that array is used in an expression.

4.3.a Array Names

The allowable names for arrays are the same as with scalars, except only the first letter is used to identify the name. This means that there are only 26 unique variable array names to choose from. All in all, there are 52 arrays available for use, since MLBASIC does allow for 26 String arrays and 26 Real/Integer arrays.

When real arrays are declared the first letter of that name must not appear in any other Integer variable array name.

Example

DIM A(100),A\$(255):REAL A10(10)

Real variable array, A10(100), would have the effect of overriding the first dimensioned variable A() with the REAL array A10(). The string array A\$ is unchanged by the REAL declaration.

4.3.b Array Subscripts

MLBASIC does not support expressions as the subscript. The only allowable parameters are Integer Variables and Integer Constants. For example, the command A(10+IV)=B is not allowed, but instead should be set up like: A=10+IV:A(A)=B. Furthermore, the index variable must only be one character in length.

MLBASIC 1.0 USER'S MANUAL

4.3.c Memory Requirements

Arrays are allocated space in RAM at compilation time, as opposed to allocation when a program is "run" under the Interpreter. This makes array addressing very fast, since the program knows exactly where the array is when it is "accessed" by the program.

String arrays are allocated at compilation time, just as the Integer and Real arrays are. This speeds up the time needed to manipulate and access strings greatly, as opposed to the Interpreter which may spend large amounts of time just allocating strings and collecting the "garbage" that builds up rather quickly. The slow speed of string manipulations under Interpretive Basic is because the strings have to be allocated dynamically at run time.

Two dimensional arrays are arranged in the order of 1st dimension elements next to each other, repeated for each of the 2nd dimension elements.

For example, the array A(2,3) is arranged like:

| <u>Address</u> | <u>Element</u> | <u>Relative Location W.R.T. 1st Element</u> |
|----------------|----------------|---|
| LOW MEM | A(0,0) | 0 |
| . | A(1,0) | 1 |
| . | A(0,1) | 2 |
| . | A(1,1) | 3 |
| . | A(0,2) | 4 |
| HIGH MEM | A(1,2) | 5 |

Note that A(0,0) is the first element, not A(1,1).

MLBASIC 1.0 USER'S MANUAL

4.4 Operators and Expressions

Expressions can be arithmetic, and/or logical. They consist of a combination of constants, variables, array elements and operators.

4.4.a. Arithmetic Expressions and Operators

Arithmetic expressions can be comprised of both logical and arithmetic operators. This allows for extremely flexible manipulation of variables, constants, and other expressions. Arithmetic expressions can have two types of data; Integer and Real. The type of computation involved in the expression depends on the function, variable and constant types used in the expression.

When the operators appear in an arithmetic/logical expression, computation is performed from left to right in the following order:

- (1) Multiplication, Division, Exponentiation, NOT, OR, AND
- (2) Addition, Subtraction

The differences from Interpreter Basic is that exponentiation is at the same priority level as multiplication and division. To change the order of priority, use parenthesis. Expressions within the innermost parenthesis are calculated first. Inside the parenthesis, the usual order of computation is used.

Operators allowed

| <u>Operator</u> | <u>Operation</u> | <u>Sample Expression</u> |
|-----------------|------------------|--------------------------|
| \uparrow | Exponentiation | X \uparrow Y |
| * | Multiplication | X*10.1 |
| / | Division | X/Y |
| AND | Logical AND | X AND\$FOOO |
| OR | Logical OR | X OR128 |
| NOT | Exclusive OR | X NOT Y |
| - | Subtraction | X-10 |
| + | Addition | X+10 |

The following are some sample algebraic expressions and the MLBASIC counterpart:

| <u>Algebraic Form</u> | <u>MLBASIC Form</u> |
|-----------------------|----------------------|
| $X+2Y(S-12.9/Y)$ | $X+2*Y*(S-12.9/Y)$ |
| $X-Y/Z$ | $X-Y/Z$ |
| $12A+13B-CLOGI$ | $12*A+13*B-C*LOG(I)$ |

4.4.b Integer Arithmetic

MLBASIC will compile an expression using integer arithmetic if all constants, functions, and variables in the expression are of the Integer type. Integer operators are; +,-,/,* ,AND,OR and NOT. Integer arithmetic allows for extremely fast calculations where the final result is an integer value. Where fractions or large numbers are not a concern, Integer arithmetic should be used.

4.4.c Logical Operators

MLBASIC allows arithmetic expressions to contain logical operators along with the normal arithmetic operators. Logical operators perform a full 16Bit operation on the operands. The allowable operators are: AND, OR, and NOT. The following examples illustrate the effect of logical operators between two 16Bit Integers.

1. Example AND

| |
|---------------------|
| 0100101101100001 |
| AND 111000001111000 |
| ----- |
| 010000001100000 |

-Final result has bits set only if both bits above are set

2. Example OR

| |
|--------------------|
| 0101010101111100 |
| OR 111100001111000 |
| ----- |
| 1111010111111100 |

-Final result has bits set if one of the bits above are set

3. Example NOT

| |
|----------------------|
| 0101010101010101 |
| NOT 0101101010011000 |
| ----- |
| 000011111001101 |

-Final result has bits set if only one of the above bits are set

MLBASIC 1.0 USER'S MANUAL

4.4.d Relational Operators

Relational operators are used to compare two values. Relational operators are only allowed in IF..THEN..ELSE commands and are not allowed in arithmetic expressions. Available operators are: >, <, <>, =>, =<, and =.

4.4.e String Operators and Expressions

A string expression is an expression made up of string constants, string variables, and string functions. The plus sign "+" is used to concatenate the elements within an expression into one final string. Maximum lengths of the final concatenated string is limited to about 300 bytes.

You can compare strings using the same relational operators that are used with numbers. String comparisons, performed with the IF-THEN-ELSE command, are made by taking one character at a time from each string and comparing the ASCII codes. If all the ASCII codes are the same, the strings are equal. If the codes differ, the lower code number precedes the higher. Comparisons between unequal length strings will always make the shorter string less than the larger one.

CHAPTER 5 TECHNICAL INFORMATION

5.1 Machine Language Interfacing

MLBASIC allows programmers to interface their own assembly language programs with the compiled program. This enables more flexibility in how the program is to operate.

The special commands LREG and SREG allow for exchange of data between the 6809 registers and variables used in the program compiled with MLBASIC.

The method that an externally assembled machine language program can be interfaced is as follows:

- (1) Store the assembled program in DATA statements using the "%" mode.
- (2) Load the hardware registers with any initial values using LREG (if required with the assembly program).
- (3) Call the assembly program that is contained in the DATA statements using GOSUB linenumber, where linenumber is the first line containing the machine language program data.
- (4) After the call to the machine language program, any data that is to be obtained from the hardware registers may be stored in an integer variable using the SREG command.

The following example uses the previously described method to call a machine language routine that will poll the keyboard until the keyboard is pressed and then store the value in the integer variable, A.

```
100 REM' Test of DATA calls
101 REM
102 DATA%1:REM 1st item to be skipped
103 DATA%183,255,222,28,175,173,159,160
104 DATA%0,39,250,26,80,183,255,223,57
120 GOSUB103:REM' Call Keyboard Poll Routine
121 SREG("D",A):REM' Load variable A with [D] register
122 A=A/256:REM' A now has ascii value of key
123 PRINT"Number =";A
124 END
```

MLBASIC 1.0 USER'S MANUAL

5.2 Interfacing MLBASIC with the Interpreter

MLBASIC allows programmers to use machine language routines contained in the ROM of their machine. These routines may be executed from within a compiled program with the use of the VECTD and VECTI commands.

The VECTD and VECTI commands allow for calls to routines contained in ROM by performing a map switch between the all RAM and the half RAM - half ROM map types. In addition, the X,Y, and U register are saved on the stack and are returned unchanged after completion of the ROM routine.

The following example illustrates the use of the tokenization routine found in the ROM of the computer.

```
100 REM' Tokenize a string
101 REM' Final Token is stored in A
102 DIM A$(20):REM' Word to Tokenize
103 INPUT"Enter BASIC Word ";$A$
104 A=LEN(A$):REM' Number of Bytes to Poke
105 FOR I=0 TO A:POKE I+500,A$(I):NEXT:REM' Store String
106 DST(166,500):REM' Point to string
107 VECTD(47137):REM' Call ROM Tokenization Routine
108 DLD(732,A):REM' Load A with 2 byte token
109 PRINT" ", "Token=";A
110 END
```

MLBASIC 1.0 USER'S MANUAL

5.3 Interpreter Calls

MLBASIC allows a compiled program to execute a BASIC command via the special character "[".

By placing the "[" character in front of a command, the compiler will use the Interpreter call routine to execute the command, at run time, under the Interpreter. With the exception of the INPUT, GOTO, GOSUB, and FOR commands, all of the BASIC commands normally used in an Interpreter BASIC program can be executed using the Interpreter Call.

The Interpreter usually is not needed because most of the available BASIC commands may be compiled using MLBASIC. One example of why an Interpreter call might be needed is to define a string for use in the DRAW and PLAY commands.

The DRAW and PLAY commands have a sub-command, called "X", that executes a substring of commands. This substring of commands must be contained in an Interpreter defined string variable. To accomplish this, the string that is executed must be defined with the aid of an Interpreter Call.

The following example shows how a DRAW command, using the "X" sub-command, uses the Interpreter Call method to define a string variable.

```
100 REM' Example DRAW using BASIC SUB-Command string
101 DIMC$(10):REM' SCALE STRING
102 PCLEAR4:PMODE3,1:SCREEN1,0:PCLS
103 [A$="BL16;R16;D16;L16;U16"
104 [B$="BL4;BU4;R24;D24;L24;U24"
105 FORS=1TO20:SC$="S"+STR$(S)
106 DRAW "C3;BM128,85;"+C$+"XA$;XB$;"
107 DRAW "C1;BM128,85;"+C$+"XA$;XB$;"
108 NEXT
109 FORS=20TO1STEP-1:$C$="S"+STR$(S)
110 DRAW "C3;BM128,85;"+C$+"XA$;XB$;"
111 DRAW "C1;BM128,85;"+C$+"XA$;XB$;"
112 NEXT
120 GOTO104
130 END
```

5.4 Subroutine Call Description

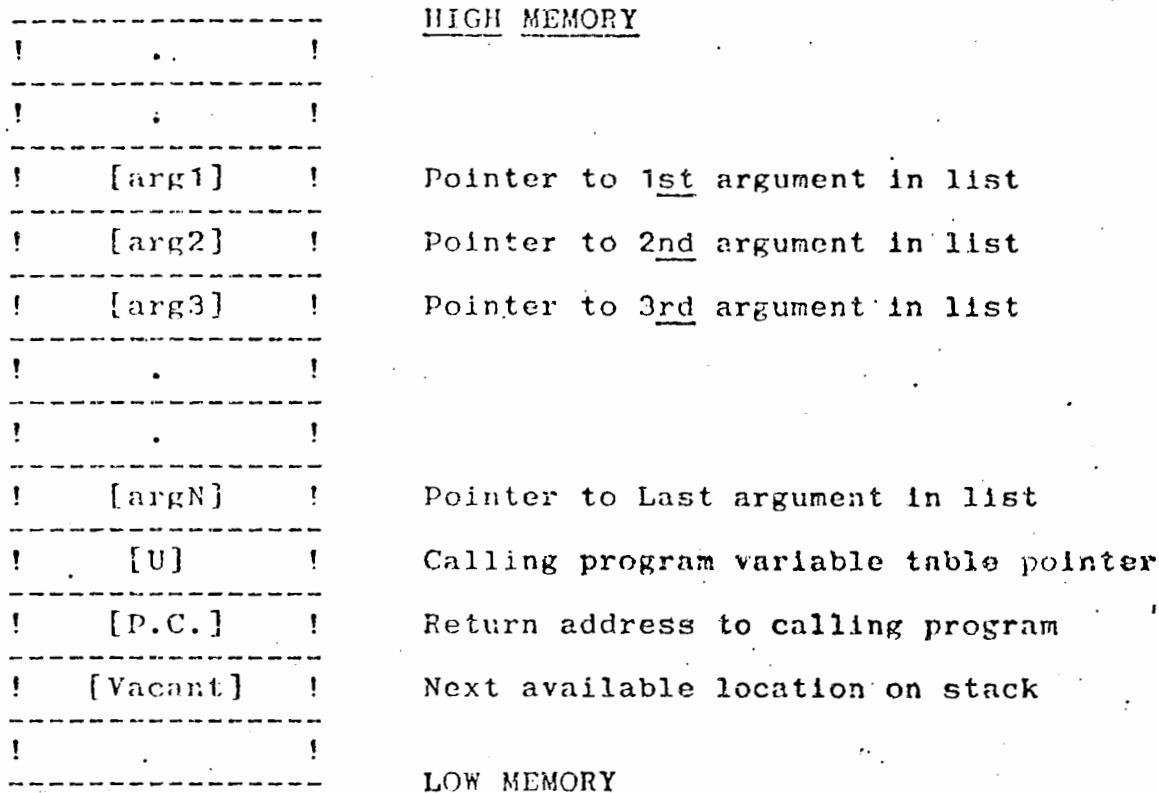
In this section a description of how the CALL and SUBROUTINE statements operate internally is given. The way a CALL statement passes parameters and program control to the SUBROUTINE is discussed for information purposes.

The following sequence describes what happens when a CALL statement is executed.

- (1) The pointers to each variable or constant in the CALL statement argument list are pushed onto the "S" stack. Each pointer occupies 2 bytes of memory on the stack. The first argument in the list is the first pointer on the stack, the second argument is the second pointer and so on.
- (2) The variable table pointer ("U" register) is saved on the stack.
- (3) The jump is made to the SUBROUTINE with the return address being the last item saved on the stack.
- (4) The SUBROUTINE is executed. Program control returns to the calling routine when a RETURN is executed in the subprogram.
- (5) The variable table pointer is loaded with its original value (obtained from the stack).
- (6) The stack is reset to its original position before the CALL statement was executed. This in affect moves the stack beyond the argument list variable/constant pointers which were saved in the first step of the process.
- (7) Program control resumes with the next executable statement after the CALL statement.

MLBASIC 1.0 USER'S MANUAL

The following diagram illustrates how the "S" stack looks after Step 3 of the process (this is how it looks immediately before the SUBROUTINE is executed).



MLBASIC 1.0 USER'S MANUAL

CHAPTER 6 SAMPLE PROGRAMS

6.1 Program #1

This program is used to save the BASIC program that exists in memory to tape. This format for saving the program to tape is needed for cassette users of MLBASIC, who wish to read the BASIC program from tape. Standard format cassette files are not compatible for compilation with MLBASIC.

Program Listing

 MLBASIC -REVISION 1.0928 - COPYRIGHT 1984 WASATCHWARE

ENTRY INTO M.L. PROGRAM=12000
 VARIABLE TABLE STARTS AT 16691
 TEXT TABLE STARTS AT 12263
 SUBROUTINES START AT 12340

```

12011 >-10 REM' ROUTINE TO PERFORM A
12011 >-20 REM' BASIC PROGRAM TRANSFER
12011 >-30 REM' TO TAPE. MLBASIC FORMAT
12011 >-40 REM' TOKENIZED DATA FILES
12011 >-50 REM'
12011 >-60 REM' THIS PROGRAM IS TO BE
12011 >-70 REM' COMPILED BEFORE IT IS
12011 >-80 REM' EXECUTED.
12011 >-90 REM'
12011 >-100 CLS:PRINT" MLBASIC PROGRAM DUMP FORMAT", " -TAPE UTILITY-"
12035 >-110 DIM A$(30)
12035 >-120 PRINT" ", "ENTER OUTPUT FILENAME":INPUT$A$:PRINT
12067 >-130 OPEN"Q", #1, A$, #0
12092 >-140 DLD(25,A):DLD(27,B):A=A-3:REM' PROGRAM AREA
12126 >-150 IFB=A/512THENB=B+100 GOTO150:REM' 2 BLOCKS MINIMUM FOR MLBASIC
12172 >-160 FORI=A TO B:PRINTB$6,I:
12196 >-170 A$=PEEK(1):PRINTH-1,A$/:NEXT
12243 >-180 CLOSE:END

END OF M.L. PROGRAM=12258
END OF TEXT TABLE=12339
END OF VARIABLE AREA=16784
  
```

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| A | 16703 | INTEGER | B | 16705 | INTEGER | C | 16707 | INTEGER |
| D | 16709 | INTEGER | E | 16711 | INTEGER | F | 16713 | INTEGER |
| G | 16715 | INTEGER | H | 16717 | INTEGER | I | 16719 | INTEGER |
| J | 16721 | INTEGER | K | 16723 | INTEGER | L | 16725 | INTEGER |
| M | 16727 | INTEGER | N | 16729 | INTEGER | O | 16731 | INTEGER |
| P | 16733 | INTEGER | Q | 16735 | INTEGER | R | 16737 | INTEGER |
| S | 16739 | INTEGER | T | 16741 | INTEGER | U | 16743 | INTEGER |
| V | 16745 | INTEGER | W | 16747 | INTEGER | X | 16749 | INTEGER |
| Y | 16751 | INTEGER | Z | 16753 | INTEGER | | | |

2. DIMENSIONED VARIABLES

| NAME | LOCATION | TYPE | 1st DIMENSION |
|------|----------|------|---------------|
|------|----------|------|---------------|

| | | | |
|----|-------|--------|----|
| AS | 16755 | STRING | 30 |
|----|-------|--------|----|

8 ERRORS.

6.2 Program #2

This program is useful for deleting remarks in a program. It works on the program while it is in memory.

Program Listing

```

MLBASIC -REVISION 1.0928 - COPYRIGHT 1984 WWSATCHWARE
-----
ENTRY INTO M.L. PROGPRM=12888
VARIABLE TABLE STARTS AT 17813
TEXT TABLE STARTS AT 12594
SUBROUTINES START AT 12664

12811 >-18 REM'*****  

12811 >-20 REM' * DELETE REMARKS IN A *
12811 >-30 REM'* BASIC PROGRAM *
12811 >-40 REM'* -(C) WWSATCHWARE -
12811 >-50 REM'* 1984 *
12811 >-50 REM'*****  

12811 >-70 REM'  

12811 >-80 CLS:PRINT" THIS PROGRAM DELETES REMARKS"," FROM THE PROGRAM EXISTING  

IN"," MEMORY."  

12833 >-90 DIMAS(308):REM'LINE BUFFER  

12833 >-100 DLD(25,A):DLD(27,B):B=B-2:DLD(A,C)  

12875 >-110 D=R:REM' SET START OF NEW LINE TO OLD  

12894 >-120 E=R+4:F=C-1:J=0  

12126 >-130 FORJ=E TO F:REM' LOOP ON CHARACTERS IN LINE  

12132 >-140 G=PEEK(I):IFG>130THEN168  

12133 >-150 R$(J)=G:J=J+1:GOTO180:REM' FILL OUTPUT BUFFER,NEXT CHARACTER  

12196 >-160 IFI>E THEN190  

12201 >-170 R$(D)=G:J=1:GOTO200  

12226 >-180 NEXTI:GOTO210  

12243 >-190 J=J-1:REM' GET RID OF COLON  

12261 >-200 G=B:R$(J)=G:J=J+1:REM' END OF LINE MARKER  

12300 >-210 REM' STORE NEW LINE OVER OLD,J=LENGTH,D=START  

12308 >-220 L=R+2:DLD(L,L):REM' L CONTAINS LINE #  

12324 >-230 E=D+4:F=E+J:F=F-1:H=E:FORI=E TO F:G=R(H):H=H+1:POKEI,G:NEXT  

12451 >-240 F=F+1:DST(D,F):REM' STORE NEXT LINE LOCATION  

12491 >-250 H=D+2:DST(H,L):REM' STORE CURRENT LINE NUMBER  

12502 >-260 D=F:REM' SET NEW LINE LOCATION  

12516 >-270 A=C:DLD(A,C):REM' A=START OF NEXT OLD LINE,C=NEXT+1  

12530 >-280 IFAKB THEN128  

12533 >-290 REM' STORE FINALE POINTERS  

12539 >-300 DST(D,B):D=D+2:DST(27,D)  

12581 >-310 END

```

MLBASIC 1.0 USER'S MANUAL

END OF M.L. PROGRAM=12589
 END OF TEXT TABLE=12663
 END OF VARIABLE AREA=17378

31 PROGRAM LINES
 0 GOTO/GOSUBS

***** VECTOR TABLE [FROM-TO] *****
 E12151 -12190 JE12188 -12220 JE12199 -12243 JE12218 -12261 J
 E12241 -12300 J

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| R | 17027 | INTEGER | B | 17029 | INTEGER | C | 17031 | INTEGER |
| D | 17033 | INTEGER | E | 17035 | INTEGER | F | 17037 | INTEGER |
| G | 17039 | INTEGER | H | 17041 | INTEGER | I | 17043 | INTEGER |
| J | 17045 | INTEGER | K | 17047 | INTEGER | L | 17049 | INTEGER |
| M | 17051 | INTEGER | N | 17053 | INTEGER | O | 17055 | INTEGER |
| P | 17057 | INTEGER | O | 17059 | INTEGER | R | 17061 | INTEGER |
| S | 17063 | INTEGER | T | 17065 | INTEGER | U | 17067 | INTEGER |
| V | 17069 | INTEGER | W | 17071 | INTEGER | X | 17073 | INTEGER |
| Y | 17075 | INTEGER | Z | 17077 | INTEGER | | | |

2. DIMENSIONED VARIABLES

| NAME | LOCATION | TYPE | 1st DIMENSION |
|------|----------|--------|---------------|
| AS | 17079 | STRING | 300 |

0 ERRORS

MLBASIC 1.0 USER'S MANUAL

6.3 Program #3

This program demonstrates the use of subroutines. In addition, the routine to output upper and lower case characters to a high resolution screen is used in a sample driver program.

MLBASIC -REVISION 1.0928 - COPYRIGHT 1984 HABSWARE

ENTRY INTO M.L. PROGRAM=12000
VARIABLE TABLE STARTS AT 19194
TEXT TABLE STARTS AT 14791
SUBROUTINES START AT 14843

```
12011 >-10 REM' TEST DRIVER FOR 42X24 TEXT SUBROUTINE
12011 >-20 REM'
12011 >-30 INPUT"ENTER STARTING 512BYTE PAGE":R
12025 >-40 B=R*512:C=R+6143:REM' FORM GRAPHICS BOUNDARY
12035 >-50 CLS0:PRINT#264,"READY TO ENTER TEXT";
12036 >-60 CALL CHAR(1,B,C):PMODD4,R:REM' CLEAR SCREEN AND INITIALIZE
12128 >-70 A=INKEY$():IF A=0THEN70
12145 >-80 CALL CHAR(A,1,B,C):REM' OUTPUT CHARACTER TO SCREEN
12173 >-90 GOTO70:REM' REPEAT
12176 >-28000 SUBROUTINE ENTRY
```

END OF M.L. PROGRAM=12178
END OF TEXT TABLE=14842
END OF VARIABLE AREA=19237

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| A | 19205 | INTEGER | B | 19208 | INTEGER | C | 19210 | INTEGER |
| D | 19212 | INTEGER | E | 19214 | INTEGER | F | 19216 | INTEGER |
| G | 19218 | INTEGER | H | 19220 | INTEGER | I | 19222 | INTEGER |
| J | 19224 | INTEGER | K | 19226 | INTEGER | L | 19228 | INTEGER |
| M | 19230 | INTEGER | N | 19232 | INTEGER | O | 19234 | INTEGER |
| P | 19236 | INTEGER | O | 19238 | INTEGER | R | 19240 | INTEGER |
| S | 19242 | INTEGER | T | 19244 | INTEGER | U | 19246 | INTEGER |
| V | 19248 | INTEGER | W | 19250 | INTEGER | X | 19252 | INTEGER |
| Y | 19254 | INTEGER | Z | 19256 | INTEGER | | | |

0 ERRORS

```
SUBROUTINE CHAR(A,ZZ,S,E)
12182 >-20001 REM' PRINT CHARACTER IN 42X24 FORMAT
12182 >-20002 REM' A=ASCII CHARACTER
12182 >-20003 REM' ZZ=INITIALIZE OR NOT (0=INIT,1=NOT)
12182 >-20004 REM' S=START OF GRAPHICS PAGE(512 BYTE BOUNDARY)
12182 >-20005 REM' E-END OF GRAPHICS AREA
12182 >-20006 DIM R(123,6),1(5)
12182 >-20007 IF ZZ=1 THEN 20106
12196 >-20008 B=0:C=0:I(1)=P:REM' SET POINTERS TO TOP
12219 >-20009 DATA3,0,0,0,0,0
12236 >-20010 DATA32,32,32,32,32,0,32
12250 >-20011 DATA72,72,72,0,0,0
12264 >-20012 DATA20,0,248,0,248,0,0,0
12278 >-20013 DATA32,128,128,112,0,248,64
12292 >-20014 DATA200,200,16,32,64,132,132
12306 >-20015 DATA16,128,128,96,128,128,16
12320 >-20016 DATA32,32,32,0,0,0,0
12334 >-20017 DATA16,32,64,64,32,16
12348 >-20018 DATA64,32,16,16,16,32,64
12362 >-20019 DATA8,136,0,0,248,0,0,136,0
12376 >-20020 DATA8,32,32,248,32,32,0
12390 >-20021 DATA0,0,0,48,48,16,32
```

MLBASIC 1.0 USER'S MANUAL

12484 >-200222 DATA0,0,0,248,0,0,0
12485 >-20023 DATA0,0,0,0,48,48
12492 >-20024 DATA8,8,16,32,64,128,128
12496 >-20025 DATA48,72,72,72,72,72,48
12498 >-20026 DATA32,96,32,32,32,32,112
12499 >-20027 DRTR112,136,8,48,64,128,248
12500 >-20028 DRTR112,136,8,48,8,136,112
12502 >-20029 DATA16,48,80,144,248,16,16
12516 >-20030 DATA248,128,248,8,8,136,112
12530 >-20031 DRTR112,128,128,248,136,136,112
12544 >-20032 DATA248,8,16,32,64,128,128
12558 >-20033 DRTR112,136,136,112,136,136,112
12572 >-20034 DRTR112,136,136,128,8,8,112
12586 >-20035 DATA8,32,32,8,32,32,8
12600 >-20036 DATA8,48,48,8,48,16,32
12614 >-20037 DRTR8,16,32,64,32,16,8
12628 >-20038 DPTR0,0,248,0,248,0,0
12642 >-20039 DRTR128,64,32,16,32,64,128
12656 >-20040 DATA112,136,8,16,32,0,32
12670 >-20041 DRTR112,136,8,104,154,136,112
12684 >-20042 DATA32,80,136,136,248,136,136
12698 >-20043 DRTR248,72,72,112,72,72,248
12712 >-20044 DRTR112,136,128,128,128,136,112
12726 >-20045 DRTR248,72,72,72,72,72,248
12740 >-20046 DRTR248,128,128,248,128,128,248
12754 >-20047 DRTR248,128,128,248,128,128,128
12768 >-20048 DATA128,128,128,152,136,136,120
12782 >-20049 DATA136,136,136,248,136,136,136
12796 >-20050 DRTR112,32,32,32,32,32,112
12810 >-20051 DATA8,8,8,8,8,136,112
12824 >-20052 DATA136,144,168,192,168,144,136
12838 >-20053 DRTR128,128,128,128,128,128,248
12852 >-20054 DRTR136,216,168,168,136,136,136
12866 >-20055 DATA136,200,168,152,136,136,136
12880 >-20056 DRTR248,136,136,136,136,248
12894 >-20057 DRTR248,136,136,248,128,128,128
12908 >-20058 DATA112,136,136,136,168,144,104
12922 >-20059 DRTR248,136,136,248,168,144,136
12936 >-20060 DRTR112,136,64,32,16,136,112
12950 >-20061 DRTR248,32,32,32,32,32,32
12964 >-20062 DRTR136,136,136,136,136,136,112
12978 >-20063 DATA136,136,136,80,80,32,32
12992 >-20064 DRTR136,136,136,168,168,216,136
13006 >-20065 DATA136,136,80,32,80,136,136
13020 >-20066 DATA136,136,80,32,32,32,32
13034 >-20067 DRTR248,8,16,32,64,128,248
13048 >-20068 DRTR112,64,64,64,64,64,112
13062 >-20069 DATA128,128,64,32,16,8,8
13076 >-20070 DRTR112,16,16,16,16,16,112
13090 >-20071 DATA0,0,0,0,0,0,0
13104 >-20072 DATA0,0,0,0,0,0,0
13118 >-20073 DATA0,0,0,0,0,0,0
13132 >-20074 DATA0,0,24,8,128,136,128
13146 >-20075 DATA128,128,128,240,136,136,176
13160 >-20076 DATA0,0,112,136,128,136,112
13174 >-20077 DATA8,8,128,136,136,128
13188 >-20078 DATA0,0,112,136,248,128,112
13202 >-20079 DATA48,32,32,248,32,32,96
13216 >-20080 DATA0,0,128,136,128,8,48
13230 >-20081 DATA128,128,176,200,136,136,136
13244 >-20082 DATA0,32,8,32,32,32,112
13258 >-20083 DATA0,16,8,16,16,144,96
13272 >-20084 DRTR128,128,144,168,192,168,144
13286 >-20085 DPTR4,64,64,64,64,64,96
13290 >-20086 DATA0,0,136,216,168,136,136
13314 >-20087 DATA0,0,176,204,136,136,136
13328 >-20088 DATA0,0,112,136,136,136,112

MLBASIC 1.0 USER'S MANUAL

13342 >-20089 DATA0,0,240,136,240,128,129
 13356 >-20090 DATA0,0,112,136,136,120,0
 13370 >-20091 DATA0,0,132,224,120,128,128
 13384 >-20092 DATA0,0,112,128,112,0,112
 13398 >-20093 DATA0,32,240,32,32,32,48
 13412 >-20094 DATA0,0,136,136,136,152,134
 13426 >-20095 DATA0,0,136,136,80,80,32
 13440 >-20096 DATA0,0,136,136,168,216,136
 13454 >-20097 DATA0,0,136,88,32,88,136
 13468 >-20098 DATA0,0,136,136,240,0,56
 13482 >-20099 DATA0,0,240,16,32,64,240
 13496 >-20100 FOR I=5 TO E:POKEI,255:NEXT
 13517 >-20101 RESTORE
 13546 >-20102 FOR I=32 TO 122
 13532 >-20103 FOR J=8 TO 6
 13558 >-20104 READ B(I,J)
 13534 >-20105 NEXT:IF I>1 THEN 20109
 13555 >-20106 IF I>12 THEN 20109
 13576 >-20107 IF I<1 THEN I<1>=0:ELSE I<1>=1
 13715 >-20108 RETURN
 13716 >-20109 IF A>64 THEN 20112
 13707 >-20110 FOR I=S TO E:POKEI,255:NEXT
 13776 >-20111 B=B\10:RETURN
 13793 >-20112 REM' OUTPUT CHARACTER
 13793 >-20113 X=C*256:IF A>13 THEN 20115
 13631 >-20114 GOTO 20113
 13834 >-20115 IF A>8 THEN 20124
 13648 >-20116 N=B\GOSUB 20130
 13859 >-20117 B=B+1:IF B>42 THEN 20123
 13853 >-20118 B=0:C=C+1:IF C>24 THEN 20123
 13914 >-20119 X=S+5120:Y=S+6143:Z=S+1024
 13977 >-20120 PCOPY Z,S,Y
 13998 >-20121 FOR I=X TO Y:POKEI,255:NEXT
 14033 >-20122 C=C\4
 14042 >-20123 RETURN
 14050 >-20124 REM' BACKSPACE ROUTINE
 14058 >-20125 IF B=0 THEN 20127
 14060 >-20126 B=B-1:A=32:N=1:GOSUB 20130:RETURN
 14093 >-20127 B=41:IF C=0 THEN 20129
 14116 >-20128 C=C-1:A=32:N=1:GOSUB 20130:RETURN
 14155 >-20129 B=0:RETURN
 14163 >-20130 Z=B\6:Z=Z\B:U=Z:Z=Z+S
 14221 >-20131 Z=Z+X
 14239 >-20132 X=U\8:Y=B\6:Y=Y-K:Y=B-Y
 14307 >-20133 FOR I=8 TO 7
 14313 >-20134 D=I\32:D=0+Z
 14346 >-20135 P=PEEK(D):D=D+1:W=PEEK(D):F=P*256
 14405 >-20136 F=F+W:REM' HRS VALUE TO BE MASKED
 14423 >-20137 F=0-F:REM' INVERT WORD
 14441 >-20138 IF N=1 THEN 20146
 14452 >-20139 C=R(C,I)>1:IF C<32 THEN G=255
 14508 >-20140 IF I<1><1>1 THEN 20142
 14527 >-20141 IF I=7 THEN G=255:REM' UNDERLINE
 14532 >-20142 1BSHFT(G,Y,B)
 14573 >-20143 F=F OR G
 14593 >-20144 F=0-F:REM' UN-INVERT F
 14611 >-20145 P=F/256:POKE0,P:P=P*256
 14659 >-20146 P=F-P:POKE0,P
 14689 >-20147 NEXT:RETURN
 14710 >-20148 REM' SKIP OUT OF 2000 INTO HERE
 14710 >-20149 REM' FOR CLEAR ROUTINE
 14710 >-20150 C=240:1BSHFT(C,Y,B)
 14738 >-20151 G=B-C:F=F AND G
 14774 >-20152 GOTO 20144
 14777 >-20153 RETURN
 14779 >-20154 END

MLBASIC 1.0 USER'S MANUAL

END OF M.L. PROGRAM=14786
 END OF TEXT TABLE=14842
 END OF VARIABLE AREA=21301

154 PROGRAM LINES
 15 GOTO/GOSUBS

***** VECTOR TABLE [FROM-TO] *****
 E12154 -13655 DC13674 -13716 DC13735 -13793 DC13829 -13834]
 E13632 -13883 DC13846 -14050 DC13857 -14163 DC13881 -14049]
 E13912 -14049 DC14058 -14099 DC14096 -14163 DC14114 -14155]
 E14152 -14163 DC14458 -14710 DC14525 -14552]

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| A | S+ 18 | INTEGER | B | 19272 | INTEGER | C | 19274 | INTEGER |
| D | 19276 | INTEGER | E | S+ 4 | INTEGER | F | 19280 | INTEGER |
| G | 19282 | INTEGER | H | 19284 | INTEGER | I | 19286 | INTEGER |
| J | 19288 | INTEGER | K | 19290 | INTEGER | L | 19292 | INTEGER |
| M | 19294 | INTEGER | N | 19296 | INTEGER | O | 19298 | INTEGER |
| P | 19300 | INTEGER | Q | 19302 | INTEGER | R | 19304 | INTEGER |
| S | S+ 6 | INTEGER | T | 19306 | INTEGER | U | 19310 | INTEGER |
| V | 19312 | INTEGER | W | 19314 | INTEGER | X | 19316 | INTEGER |
| Y | 19318 | INTEGER | Z | 19320 | INTEGER | ZZ | S+ 8 | INTEGER |

2. DIMENSIONED VARIABLES

| NAME | LOCATION | TYPE | 1st DIMENSION |
|------|----------|---------|---------------|
| A | 19324 | INTEGER | 123 |
| I | 21292 | INTEGER | 5 |

8 ERRORS

. SUBROUTINES DRILLED
 NAME FROM -> TO
 CHAR 12111 -> 12176
 CHAR 12166 -> 12176

MLBASIC 1.0 USER'S MANUAL

6.4 Program #4

This program demonstrates the use of the VECTD command. The routines that tokenize and untokenize the BASIC commands are executed in this example.

Program Listing

```

MLBASIC -REVISION 1.03AB - COPYRIGHT 1984 WRATCHWARE
-----
ENTRY INTO M.L. PROGRAM=2EE0
VARIABLE TABLE STARTS AT 4288
TEXT TABLE STARTS AT 3194
SUBROUTINES START AT 328C

2EEB >-R REM*****  

2EEB >-14 REM* VECT COMMAND DEMO *  

2EEB >-1E REM* TOKENIZE/UNTOKENIZE *  

2EEB >-28 REM* ROUTINE *  

2EEB >-32 REM* *  

2EEB >-3C REM* -LEARN USE OF VECTD *  

2EEB >-45 REM*****  

2EEB >-50 REM  

2FEB >-5A DIM R$(100)REM' COMMAND BSTRING  

2FEB >-64 CLS:PRINT" TOKENIZE-UNTOKENIZE TEST"  

2F03 >-6E PRINT"ENTER OPTION:","1. -TOKENIZE A COMMAND","2. -UNTOKENIZE 2BYTE WD  

RD","3. -EXIT"  

2F14 >-78 INPUTA:PRINT  

2F22 >-82 ON A GOTO200,300,400  

2F40 >-88 CLS:PRINT" TOKENIZE A COMMAND"," "  

2F50 >-D2 INPUT" ENTER COMMAND ",A$:PRINT  

2F6E >-DC CALL TOKEN(A,A$):REM' TOKENIZE STRING  

2F81 >-E6 PRINT"TOKEN=":R  

2F99 >-F0 INPUT"ANY MORE? (Y/N) ",A$:PRINT  

2FAF >-FA IF $LEFT$(A$,1)="Y"THEN GOTO210 :ELSE GOTO100  

2FD5 >-120 CLS:PRINT" UNTOKENIZE A COMMAND"," "  

2FED >-136 INPUT" ENTER 2BYTE TOKEN ",A:PRINT  

3002 >-140 CALL UNTOKEN(A,A$):REM' GET COMMAND  

3015 >-14A PRINT"COMMAND=",A$  

302E >-154 INPUT"ANY MORE? (Y/N) ",A$:PRINT  

3044 >-15E IF $LEFT$(A$,1)="Y"THEN GOTO310 :ELSE GOTO100  

306A >-190 STOP  

3073 >-2710 SUBROUTINE ENTRY

END OF M.L. PROGRAM=3073
END OF TEXT TABLE=3288
END OF VARIABLE AREA=442E

18 PROGRAM LINES
3 GOTO/GOSUBS

```

MLBASIC 1.0 USER'S MANUAL

***** VECTOR TABLE EFROM-TOD *****
 E31FB -2F4B DC31FA -2FD5 DC31FC -306A J

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| R | 17303 | INTEGER | B | 17305 | INTEGER | C | 17307 | INTEGER |
| D | 17309 | INTEGER | E | 17311 | INTEGER | F | 17313 | INTEGER |
| G | 17315 | INTEGER | H | 17317 | INTEGER | I | 17319 | INTEGER |
| U | 17321 | INTEGER | K | 17323 | INTEGER | L | 17325 | INTEGER |
| M | 17327 | INTEGER | N | 17329 | INTEGER | O | 17331 | INTEGER |
| P | 17333 | INTEGER | Q | 17335 | INTEGER | R | 17337 | INTEGER |
| S | 17339 | INTEGER | T | 17341 | INTEGER | U | 17343 | INTEGER |
| V | 17345 | INTEGER | W | 17347 | INTEGER | X | 17349 | INTEGER |
| Y | 17351 | INTEGER | Z | 17353 | INTEGER | | | |

2. DIMENSIONED VARIABLES

| | | | | | | | | |
|---|-------|---------|---|-------|---------|---|-------|---------|
| S | 17063 | INTEGER | T | 17065 | INTEGER | U | 17067 | INTEGER |
| V | 17069 | INTEGER | W | 17071 | INTEGER | X | 17073 | INTEGER |
| Y | 17075 | INTEGER | Z | 17077 | INTEGER | | | |

2. DIMENSIONED VARIABLES

NAME LOCATION TYPE 1st DIMENSION

R\$ 17079 STRING 300

0 ERRORS

```
-----  

      SUBROUTINE TOKEN(R,R$(B))  

3079 >-271A DIM R$(20)  

3079 >-2724 A=LEN(R$)  

308F >-272E FOR I=PTOR:POKE I+500,R$(I):NEXT:REM' STORE STRING  

308C >-2738 DST(166,500):REM' POINT TO STRING  

30DA >-2742 VECTD(47137):REM' CALL ROM ROUTINE TO TOKENIZE A STRING.  

30E6 >-274C DLD(732,A):REM' GET TWO BYTE TOKEN  

30F3 >-2756 RETURN  

30F4 >-3A98 SUBROUTINE ENTRY
```

END OF M.L. PROGRAM=30F6

END OF TEXT TABLE=328B

END OF VARIABLE AREA=446E

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| R | 5+ 6 | INTEGER | B | 17469 | INTEGER | C | 17471 | INTEGER |
| D | 17473 | INTEGER | E | 17475 | INTEGER | F | 17477 | INTEGER |
| G | 17479 | INTEGER | H | 17481 | INTEGER | I | 17483 | INTEGER |
| J | 17485 | INTEGER | K | 17487 | INTEGER | L | 17489 | INTEGER |
| M | 17491 | INTEGER | N | 17493 | INTEGER | O | 17495 | INTEGER |
| P | 17497 | INTEGER | Q | 17499 | INTEGER | R | 17501 | INTEGER |
| S | 17503 | INTEGER | T | 17505 | INTEGER | U | 17507 | INTEGER |
| V | 17509 | INTEGER | W | 17511 | INTEGER | X | 17513 | INTEGER |
| Y | 17515 | INTEGER | Z | 17517 | INTEGER | | | |

2. DIMENSIONED VARIABLES

NAME LOCATION TYPE 1st DIMENSION

R\$ 1 STRING 14

0 ERRORS

MLBASIC 1.0 USER'S MANUAL

```

SUBROUTINE UNTOKEN(R$,RS)
30FA >-3AA2 DIM RS(255)
30FA >-3AAC REM' ROUTINE TO UNTOKENIZE BASIC COMMANDS
30FA >-3AE6 REM' INPUT- 2 BYTE TOKEN IN R$
30FA >-3AC0 REM' OUTPUT- COMMAND IN RS
30FA >-3ACA FOR I=733 TO 750:POKE I,R$NEXT:REM' CLEAR BUFFER
311D >-3AD4 DST($00,A):DST($02,B):REM' STORE TOKEN
3138 >-3ADE LREG("X",496):REM' POINT TO TOKEN-4 BYTES
313D >-3AEB VECTD(47042):REM' EXECUTE ROM UNTOKENIZE ROUTINE
3149 >-3AF2 FOR I=0 TO 15:REM'LOOP ON CHARACTERS IN COMMAND
314F >-3AFC RS(I)=PEEK(I+733):NEXT
3166 >-3B05 RETURN:REM' ALL DONE
3187 >-EA60 END

```

END OF M.L. PROGRAM#31BF

END OF TEXT TABLE#3288

END OF VARIABLE AREA#44RE

***** VARIABLE TABLE *****

1. SCALAR VARIABLES

| NAME | LOCATION | TYPE | NAME | LOCATION | TYPE | NAME | LOCATION | TYPE |
|------|----------|---------|------|----------|---------|------|----------|---------|
| A | B4 S | INTEGER | B | 17533 | INTEGER | C | 17535 | INTEGER |
| D | 17537 | INTEGER | E | 17539 | INTEGER | F | 17541 | INTEGER |
| G | 17543 | INTEGER | H | 17545 | INTEGER | I | 17547 | INTEGER |
| J | 17549 | INTEGER | K | 17551 | INTEGER | L | 17553 | INTEGER |
| M | 17555 | INTEGER | N | 17557 | INTEGER | O | 17559 | INTEGER |
| P | 17561 | INTEGER | Q | 17563 | INTEGER | R | 17565 | INTEGER |
| S | 17567 | INTEGER | T | 17569 | INTEGER | U | 17571 | INTEGER |
| V | 17573 | INTEGER | W | 17575 | INTEGER | X | 17577 | INTEGER |
| Y | 17579 | INTEGER | Z | 17581 | INTEGER | | | |

2. DIMENSIONED VARIABLES

| NAME | LOCATION | TYPE | 1st DIMENSION |
|------|----------|------|---------------|
|------|----------|------|---------------|

| | | | |
|----|---|--------|----|
| RS | 1 | STRING | FF |
|----|---|--------|----|

3. ERRORS

SUBROUTINES CALLED
 NAME FROM -> TO
 KEN 2F7A -> 3073
 UNTOKEN 300E -> 30F4

MLBASIC 1.0 USER'S MANUAL

CHAPTER 7 ERROR MESSAGES

The error handling portion of MLBASIC allows for easy detection of program errors. There are two types of errors that can occur, they are: (1) errors that occur during compilation of the program and (2) errors that occur during execution of the compiled program.

During compilation, COMPILER ERRORS are the result of syntax errors in the BASIC program that is being compiled. The BASIC program that does not conform to required specifications of MLBASIC will not be compiled correctly. The compiler will make a note of any compiler error and continue to the next command. This means that all errors may be detected by the compiler at one time. Too many errors may cause MLBASIC to misread the final END statement, resulting in compilation of non existing lines (If this occurs, abort compilation by pressing down the BREAK key for about 5 seconds, then enter T).

The RUNTIME ERRORS are errors that occur because a command is executed improperly. One example of this might be division by zero. Runtime errors occur only during execution of a compiled BASIC program. Diagnostic messages will be output indicating what the error was and what the values of certain hardware registers were when the error occurred.

7.1 Compiler Error Messages

In the following section, all of the COMPILER ERROR numbers will be described so that one can determine what the possible causes for the error are. In some cases the compiler error number will not indicate the actual problem in the command that was being compiled.

The compiler error message consists of the following three values:

- (1) Compiler error number -This is a number that identifies what type of error occurred.
- (2) Line number -This indicates the line number of the BASIC program that contains the error.
- (3) Character number -This indicates what character in the command being compiled caused the error. If an error occurred in a line that has more than one command, make sure to start counting the characters from the start of the command and not from the start of the line.

Sample error message to screen

ERROR# (1) LINE (2) CHR (3)

Sample error message to printer

.....BASIC COMPILER ERROR # (1) AT LINE # (2) - CHARACTER # (3)

**Note- Remember, when an error occurs during compilation, you can abort further compilation of the source by pressing down the BREAK key for a few seconds and then hitting the T key. The message "ABORT COMPIRATION" will indicate that the compiler was properly aborted.

MLBASIC 1.0 USER'S MANUAL

Compiler Error Messages

| Number | Meaning |
|--------|---|
| 1 | Improper command terminator. Either a ":" or an end of line (zero byte) is expected. Syntax error in the previous command. |
| 2 | Error in GOTO or GOSUB statement. The word "SUB" or "TO" is missing. |
| 3 | Error in IF..THEN routine. Illegal logical operator. |
| 4 | Error in FOR..NEXT command. Missing "=". |
| 5 | Error if FOR..NEXT command. Missing "TO". |
| 6 | Error in DATA statement. Illegal data type. |
| 7 | Error in equation evaluation routine. Missing "=". Possible error in spelling of command. Command may need Extended or Disk BASIC ROM(s). |
| 8 | Error in numeric expression. Missing ")". The numeric expression may be too complex to compile. Must break expression up into compilable parts. |
| 9 | Illegal integer operator. Allowable are "+", "-", "/", "*", "OR", "AND" and "NOT" |
| 10 | Illegal real operator. Allowable are "+", "-", "/", "*" and " |
| 11 | Unknown command error. |
| 12 | Unknown function. Function not supported by compiler. "\$" is needed before string expression. |
| 13 | Illegal integer constant. Allowed are decimal and HEX " \$" or "&H" must precede hex number. |
| 14 | Unknown string function. String function not supported by MLBASIC. |
| 15 | Undimensioned REAL or INTEGER variable array. Must use DIM or REAL to declare variable arrays. |
| 16 | Undimensioned STRING variable array. <u>All</u> strings must be dimensioned before they are used. |
| 17 | Undefined FIELD. The FIELD must be defined <u>before</u> RSET or LSET use that field. |
| 18 | Missing DATA statements. RESTORE must <u>follow</u> DATA statements. |

MLBASIC 1.0 USER'S MANUAL

(.2 Runtime Error Messages

Runtime errors occur during execution of the compiled program. These errors occur because of many reasons. The most common errors are arithmetic and Input/Output (I/O) errors.

Runtime errors are printed on the screen if it is currently open. If the printer was being used last, the output will go to the printer instead of the screen.

All runtime errors can be handled by software to resolve certain problems that may arise when operating a program. The ON ERROR command is used to perform error trapping when any runtime error occurs (see section 3.2.1).

The following message is output when a runtime error occurs:

RUNTIME ERROR # (1)
(D) (X) (Y) (U) (CC) (DP)

Index to items in parenthesis

- (1) -This is the runtime error number
- (D) -The contents of hardware register "D" (in HEX)
- (X) -The contents of "X" hardware index register (in HEX)
- (Y) -The contents of "Y" hardware index register (in HEX)
- (U) -The user stack register (in HEX)
- (CC) -The control code register (in HEX)
- (DP) -The direct page register (in HEX)

MLBASIC 1.0 USER'S MANUAL

Runtime Error Messages

| Number | Meaning |
|--------|---|
| 1 | "NF" -There is a NEXT without a FOR. |
| 2 | "SN" -There was an error in compilation. |
| 3 | "RG" -Return without a GOSUB call. Interpreter only. |
| 4 | "OD" -No more data in data list. Interpreter only. |
| 5 | "FC" -There was an illegal value passed to a function. |
| 6 | "OV" -Real number overflow. Value outside of the allowable range of +/- 1.7E+38. |
| 7 | "OM" -There is not enough memory to execute current command. Interpreter only. |
| 8 | "UL" -Undefined line referenced by a GOTO or GOSUB command. Interpreter only. |
| 9 | "BS" -The subscript of the variable array is out of range. Interpreter only. |
| 10 | "DD" -The array has already been dimensioned. Interpreter only. |
| 11 | "/0" -The calculation involved a division by zero. |
| 12 | "ID" -INPUT from keyboard used in an <u>Interpreter call</u> (See 5.4). Interpreter only. |
| 13 | "TM" -Type of argument conflict with function. Interpreter only. |
| 14 | "OS" -Not enough string space to perform Interpreter call. Interpreter only. |
| 15 | "LS" -The string exceeds the maximum length of 255 bytes. Interpreter only. |
| 16 | "ST" -The string formula is too complex to evaluate. Break the string into shorter parts. Interpreter only. |
| 17 | "CN" -CONT command not allowed in compiled program. |
| 18 | "FD" -Wrong file mode. Interpreter only. |
| 19 | "AO" -The buffer is already open. CLOSE buffer before trying to open this channel. |

MLBASIC 1.0 USER'S MANUAL

| Number | Meaning |
|--------|--|
| 20 | "DN" -Device number not allowed. Use only allowable device numbers. |
| 21 | "IO" -Input error in reading data from device. |
| 22 | "FM" -Bad mode for input/output of data. Use mode selected in the OPEN statement for buffer. |
| 23 | "NO" -The buffer has not been opened. Use OPEN to open a file. |
| 24 | "IE" -Input of data past the last item in the file. |
| 25 | "DS" -There is a direct statement in the file (ie. [INPUT]). Interpreter only. |
| 26 | NOT USED |
| 27 | "NE" -The file opened for input was not found. |
| 28 | "BR" -The record is not within the allowable range. |
| 29 | "DF" -The disk is full. Use another diskette. |
| 30 | "OB" -There is not enough buffer space. Reserve more space using FILES. |
| 31 | "WP" -The disk is write-protected. |
| 32 | "FN" -The filename is unacceptable. Interpreter only. |
| 33 | "FS" -The disk directory has been incorrectly written to. Try to recover as many files from disk as possible. Disk needs to be re-formatted. |
| 34 | "AE" -The file already exists. KILL file, or RENAME it to another name. |
| 35 | "FO" -Field overflow error. Interpreter only. |
| 36 | "SE" -The string used in LSET or RSET has not been fielded. Interpreter only. |
| 37 | "VF" -Error in verification of data written to disk. |
| 38 | "ER" -Too much data in I/O on Direct access file. |