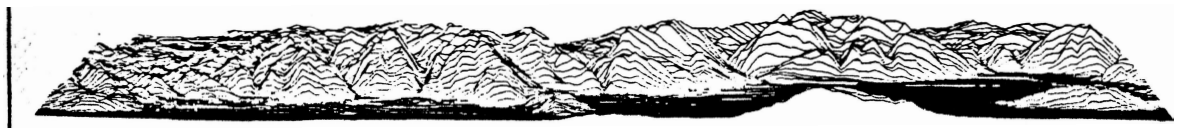
DRAFT

MLBASIC

REVISION 2.0

EXTENDED BASIC COMPILER

WASATCHWARE



MLBASIC

Revision 2.0

EXTENDED

BASIC

COMPILER

by WASATCHWARE

NOTICE

MLBASIC 2.0 User's manual

Revision History:
Original Release -July 1987

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, 1987" 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. Chapter 8 explains conversion techniques for compiling programs written for the Interpreter.

CONTENTS

PAGE

MLBASIC 2.0 USER'S MANUAL

CHAPTER 1 INTRODUCTION

1.1 Overview of MLBASIC

MLBASIC (Revision 2.0) 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 most of the commands that are available with Extended Disk 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. 128 K Color Computer 3
2. Radio Shack DOS

1.3 Disk Contents

File	Name	Description
1	#.BAS	MLBASIC loader program
2	COCO3LB2.BIN	MLBASIC loader subroutines
3	MLBASIC2.MAI	MLBASIC main program
4	COCO3LB3.BIN	MLBASIC runtime subroutines
5	PROGRAM1.BAS	Sample program #1 source
6	PROGRAM2.BAS	Sample program #2 source
7	PROGRAM1.BIN	Sample program #1 object
8	PROGRAM2.BIN	Sample program #2 object
9	R.BAS	Second 64k program loader
10	LOAD512.BAS	512k program loader

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.

1.5 Program Development

MLBASIC is designed to recognize a BASIC program as it exists in memory. In other words, MLBASIC reads the compressed commands, called tokens, as they exist in memory. Programs on disk are read in their standard format (SAVE,A not needed). 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. 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.

Programs that cannot run within the lower 32k of RAM are executed using the loader program called "R.BAS". This program loads the executable program into the second bank of 64k RAM. To prepare a program to be run in this mode, copies of the loader file "R.BAS" and the two subroutine files "COCO3LB2.BIN" and "COCO3LB3.BIN" must be made on the disk that is to contain the compiled program. To execute this routine, enter the command RUN "R and answer the question of the filename you want to load and run.

1.6 Memory use by MLBASIC

MLBASIC allows use of 64k of memory for program and variables, the 80 column high-resolution text screen, and the ROM routines all within the same program. If a machine language program exceeds the lower 32k of memory space, then the program is executed in a separate bank of 64k memory. 128k computers can use the second bank of 64k for programs to run, but the program that is running cannot have high-resolution graphics, since the high-resolution graphics area is also in this second bank of 64k.

Computers that have 512k of RAM can execute a compiled program so that banks other than those needed for high-resolution graphics are used. The loader program called "LOAD512" should be run instead of the loader program "R". Normally memory segments &H30 thru &H37 are used for storage for programs that can't fit in the lower 32k of the first bank of 64k RAM. The loader program "LOAD512" on the other hand uses segments &H00 thru &H06 to make up the second bank of memory (often called the TASK #1 bank). This frees up segments &H30 thru &H36 for use by the high-resolution graphics. The user may change the segments used by "LOAD512" by modifying lines 440 thru 500 to poke the segments that are desired.

CHAPTER 2 HOW TO COMPILE A PROGRAM

In the following chapter, procedures on how to compile programs using the many features offered by MLBASIC are given.

If you are new at using MLBASIC, try the following procedure to compile a simple program that shows how fast the compiler works:

1) Follow step 3 on page 6
2) Type in the short program:
```
1 FORI=1TO65000
2 NEXT:END
```
3) Enter EXEC to start the compiler
4) Hit CTRL to begin compilation
5) Wait 10 seconds
6) Enter EXEC to execute the machine program

Section 2.2 is the general procedure used to compile any program using MLBASIC.

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, number base option, and the default string length option. Sections 2.3 through 2.8 will cover each of these categories in detail.

SPEED TEST PROGRAM:

```
10 WIDTH 32 : CLS
20 FOR X = &H400 TO &H5FP
30 FOR Y = 0 to &HFF
40 POKE X,Y
50 NEXT Y,X
```

ML BASIC	MICROSOFT BASIC
9 Seconds	12 Minutes, 49 Seconds = 769 Seconds

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 CTRL key. This simply tells MLBASIC to start compiling the program.

How to Compile a Program

1. 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. If existing BASIC software is to be compiled, make the needed syntax changes that are identified in CHAPTER 8 (or those commands that give compiler errors when compiling a program the first time).

2. Save the BASIC program.

   - SAVE the BASIC program 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 (or enter POKE113,0 then hit the reset button).

     (B) Insert MLBASIC diskette into Drive #0.

     (C) Enter the command RUN "=", and hit the ENTER key.

4. Decide on the storage option desired.

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

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 or the default option listed.

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

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

(C) If you want to compile the object code to disk, position the cursor on the "MACHINE LANGUAGE OUTPUT" option line, hit D for disk and enter the output filename. Hit CTRL 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. The mapping options, as described in section 2.4, describe the capabilities of MLBASIC for experienced programmers who may want to interface compiled programs with other machine language routines.

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 SOURCE INPUT - This is where the compiler is to obtain the BASIC program that is to be compiled. The two available choices are to read the program from "M"-memory or "D"-disk. 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 option, 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.

(2) MACHINE LANGUAGE OUTPUT - This is where the compiler is to put the final compiled program and necessary text and subroutine areas. The two available choices for outputting the compiled code are; "M" -memory or "D" -disk.

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. In the case where the object code is too large to be compiled in memory (as indicated by the error message 'ERROR, M.L. OUTPUT EXCEEDS $7EFF'), the "D" -disk option must be used.

The disk option allows for storage of the compiled program on a non-volatile medium. Once an error free program has been compiled and saved to disk, the executable program may be loaded into memory, and EXECuted with little effort. The disk option allows for unlimited size programs to be written.

If the Disk option is used for either the input or output options, the disk in drive zero must not be write protected.

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

2.4 Mapping Options

MLBASIC automatically figures out where to locate the compiled program when using the default Automatic compilemode. If the Manual compilemode is selected, MLBASIC will allow the user to select the locations of all four program segments that are produced when a BASIC program is compiled. The locations are entered after the following four group headings:

(1) MAIN PROGRAM AREA - The number displayed 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) CHARACTER DATA AREA - The number entered is 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) SUBROUTINE LIBRARY AREA - The number entered here 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) VARIABLE STORAGE AREA - The value that needs entering 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.

If one selects the Disk output option, the Entry point of the program can be entered by the user. If no value is given, MLBASIC will compile the program such that the end of the program (not variable table) is in the last available memory location.

MLBASIC will compile programs in the lower 32k of the first bank of 64k (TASK #0) if the program can fit in it. The area above the lower 32k in this mode cannot be used because it contains the BASIC and Disk operating system. Programs that cannot fit within the lower 32k will be mapped to run in the second bank of 64k RAM (TASK #1). Large programs like these must be executed using the special loader program called "R.BAS".

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.

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. Variable tables for scalar and dimensioned variables showing variable locations in memory, type of variable, and its name.

E. Listing of compiler errors encountered during the first pass.

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 hexidecimal 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.

2.7 Default String Length

MLBASIC allocates a predefined number of characters for each string or string array element. The default string length used is 256 characters.

The user may change this value to any number from 1 to 32767. Strings that have a length greater than 256 characters cannot be manipulated using string functions because the string manipulation buffer is only 256 characters long.

The default string length is used by the compiler unless the %STRING compiler directive is used within the program code. The %STRING directive will override the default string length with the length supplied in the directive (see section 3.8.c for more information on %STRING).

2.8 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 or disk. 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. The Manual compilemode is useful if the programmer is interested in compiling two or more program that share the same subroutine library, or use some of the same variable area. If the manual mode is selected, the user is required to input the starting addresses for the Entry Point, Text Table, Subroutine Library, and Variable Table (see Section 2.4 for more information on these four locations).

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

3.1 I/O Commands

3.1.a CLOSE

Function
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

3.1.b CLOADM

Function
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

3.1.c CSAVEM

Function

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

3.1.d DIR

Function

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

3.1.e DRIVE

Function

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

3.1.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$ drivenumber,track,sector,string

drivenumber -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$

2.
```
100 %STRING=1:DIM A$(256):REM' I/O BUFFER =256 CHARACTERS

200 INT DR,TR,SE
300 DSKI$ DR,TR,SE,A$
400 FORI=0TO255:INT J:J=A$(I)
500 PRINT "BYTE ";I;"=";J:NEXT
```

-This is an alternative way read data into a buffer. In this example, each byte of data can be examined more easily.

3. DIMB$(18):FORI=1TO18:DSKI$0,17,I+1,B$(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 18.

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

3.1.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$0,0,1,ARRAY$
```
2.
```
10  DIM BUFFER$(18)
20  FORI=1TO18:DSKO$0,34,I+1,BUFFER$(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 1-18.

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.h FIELD

Function

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 exressions).

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.

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

B

3.1.i FILES

Function

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

3.1.j GET

Function

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 INPUT

Function

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 /,:; 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)
;	-Supress linefeed after input
,	-Linefeed after input

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,AS:%STRING=1:DIM B$(1000):INPUT#-1,$BS(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$, finally one character is input into the array element BS(100).

3. INPUT "ENTER A";A;

-This is the same as example #1, except a CR is not output to the screen after inputting variable, A.

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 MKN$ 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.

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, single charaters can be input from a device.

Differences from Interpreter

1. When data is input from the keyboard, the "ENTER" key must be hit after every entry.

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

B

3.1.1 KILL

Function
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,B=Standard)
B,ECB,DB

3.1.m LINEINPUT

Function

To input a record of bytes from a specified channel number.

Format LINEINPUT /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. (SV)
```

Examples

1. LINEINPUT "ENTER STRING ";A$

2. LINEINPUT#1,A$,B$,C$

-This example gets three records of data and stores the in A$,B$ and C$ respectively.

Comments

1. LINEINPUT will input bytes of data from a device and store them into the specified string variable until an end of line byte (ASCII 13) is input. When this byte is input, a zero byte is stored at the end of the string variable to terminate the string data.

Differences from Interpreter

1. None.

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

B,ECB

3.1.n LSET

Function

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

3.1.o OPEN

Function

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

Format OPEN "mode" ,#buff,fname /,#ftype//,reclen/

```
buff     -Buffer number (IC,IV)
mode       -I=input,O=output,D=direct (random) access
fname   -Name of file to open (SE)
ftype   -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
reclen  -Length of direct access file (IE)
```

Examples

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

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

2. OPEN"D",#5,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"O",#1,"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)

B

3.1.p PRINT

Function

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

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

buffer	-Buffer to print data to (IV,IC) If @number is used instead of #buffer screen output is started at number (IV or IC between 0 and 511)
format	-PRINTUSING format allowed (SE)
pos	-column position to print next argument (IE)
arg	-Data to print (IE,RE,SE)
d	-separation character for arguments: ","=skip to next line (except with USING) ";"=do not skip a line or space (with USING, use only at end of arg. list)

Examples

1. PRINT"This is text"

-This prints a string constant to the screen.

2. PRINT#-1,USING"##.#-";-123.9,A,B,C;

- This prints a real constant and two variables 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.

```
 0 A=1.999:B=50000:C=1.9E+10
10 OPEN"O",#1,"TEST.1"
20 PRINT#1,A;B;C;:CLOSE
30 OPEN"I",#1,"TEST.1"
40 INPUT#1,A,B,C:CLOSE
50 PRINTA,B,C:END
```

- This simple program writes and then reads back three variables to disk.

4. PRINT#2,STR$(A);",";

- This example prints to device #2 the character equivalent of the variable, A with a delimitor, as the interpretor would in the command PRINT#2,A;.

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.

3. The semicolon is usually used after all arguments that are written to a file so they can easily be read back using INPUT.

4. If the character "$" is placed in front of the string variable name, only the first byte of that string will be output to the device.

5. 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 "," delimitor as a tab, where MLBASIC uses "," as a new line indicator.

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

B

3.1.q PUT

Function

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. If graphics commands are used while data is still in a cassette buffer, the data in that buffer will be lost (because graphic commands use the cassette buffer area for temporary variable storage).

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

3.1.r RSET

Function

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

3.2 Program Control Commands

3.2.a CALL

Function

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.

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,B=Standard)

B

3.2.b DEFUSR

Function

To define the entry location for a user machine language subroutine.

Format DEFUSRn=start

n -User function number (0-9)
start -Entry location of machine language routine (IE)

Examples

1. DEFUSR1=M+N

Comments

1. DEFUSR must be called before calling the function, USR.

Differences from Interpretor

1. None.

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

B

3.2.c END

Function

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

END

Examples

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

3.2.d EXEC

Function
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 hexidecimal $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.e 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, C)
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 counter variable must be of type INTEGER. If it is not, MLBASIC will convert that variable over to type INTEGER automatically.
2. The commands following the FOR statement are executed until the NEXT command is encountered.
3. 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.
4. 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.
5. 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

3.2.f GOSUB

Function

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 termintated 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

3.2.g GOTO

Function

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.h IF..THEN (ELSE)

Function

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

Format IF relation THEN st /ELSE st/

relation -A comparison, using any relational operator, between two expressions (IE,SE,RE)
st -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  A$=B$ 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 screeen.

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 THENlinenumer 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

3.2.i OFF ERROR

Function

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

3.2.j ON ERROR

Function

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

3.2.k 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

3.2.1 RETURN

Function

To return program control to the calling routine.

Format RETURN

Examples

```
      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,B=Standard)

B

3.2.m STOP

Function

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.n SUBROUTINE

Function

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 distinquished 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

3.2.o USR

Function

To call a user defined machine language subroutine within an integer expression.

Format m=USRn(arg)

m	-Variable that excepts the INTEGER value passed by the user function (IV)
n	-User function number (0-9)
arg	-Argument of user function (IE)

Examples

1.
```
100 FORI=1 TO 1000
200 B(I)=USR1(A(I)):NEXT
```
-In this example, the user function is filling the array, B with values that are a function of the array A.

Comments

1. Subroutines that are called by USR must end with a RTS or equivalent PULS PC.

2. The USR function first loads the [D] register with the integer argument. The machine language routine is then called via the JSR instruction. After returning from the routine, the integer value in the [D] register is transferred back as the result of the function.

Differences from Interpretor

1. None.

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

B

3.3 Math Functions

3.3.a ABS

Function
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

3.3.b ASC

Function

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

3.3.c ATN

Function
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

3.3.d COS

Function
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

3.3.e CVN

Function
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(A$)

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

Differences from Interpreter
1. None.

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

3.3.f EOF

Function

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

3.3.g EXP

Function
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.h FIX

Function

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

3.3.i HPOINT

Function

To return information on point x,y from the high-resolution graphics screen.

Format HPOINT(x,y)

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

Examples

1. A=HPOINT(R,J+3)

Comments

1. HPOINT returns a non-zero integer value if the point is set.

Differences from Interpretor

1.

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

B

3.3.j INSTR

Function

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 (SV)

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.k INT

Function

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(I/4.)=I/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

3.3.bb JOYSTK

Function

To return the horizontal or verticle coordinate of the joysticks.

FORMAT JOYSTK(j)

j -Joystick number (IE value 0 to 3)

Differences from Interpreter

1. None.

3.3.1 LEN

Function

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$=A$+STRING$(20-LEN(A$)," ")

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

3.3.m LOG

Function

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

3.3.n LOC

Function

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

3.3.o LOF

Function
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

3.3.p LPEEK

Function

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

Format LPEEK(expression)

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

Examples

1. 10 A=LPEEK(65536.*6)

In this example, the expression is equivalent to &H60000, but since MLBASIC sees the &H as an integer number, numbers greater than &HFFFF are truncated to 16 bits.

Comments

1. The argument passed to LPEEK is a virtual memory location. Since the ROM and lower 32k of RAM area for BASIC start at &H70000, the virtual address for regular memory location &H500 is virtual location &H70500.
2. LPEEK is the compliment to the LPOKE statement.

Differences from Interpreter

1. MLBASIC does not allow the &H type numbers as arguments for the function.

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

B,ECB

3.3.q PEEK

Function
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.r POINT

Function
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.

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

3.3.s PPOINT

Function
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.t RND

Function
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

3.3.u SGN

Function
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

3.3.v SIN

Function
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

3.3.w SQR

Function

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.x TAN

Function
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

3.3.y TIMER

Function
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

3.3.z VAL

Function

To return the numeric representation of a string.

Format VAL(string)

string -The numeric string (SV)

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

3.3.aa VARPTR

Function

To return the starting address in memory of a specified variable.

Format VARPTR(arg)

arg -Variable or variable array name (SV,IV,RV)

Examples

1. A=USR1(VARPTR(A$))

The VARPTR function is used to pass the location of variable A$ to a USR function.

Comments

1. VARPTR returns an integer number between 0 and &HFFFF.

Differences from Interpretor

1. None.

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

B,ECB

3.4 String Functions

3.4.a CHR$

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#-2,CHR$(18);

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

2. 100 PRINT#-2,CHR$(27)+CHR$(20);

The CHR$ function can be used to send escape codes to a printer as in this example.

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

3.4.b INKEY$

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

Format INKEY$

Examples
1.
```
100 A$=INKEY$:IFA$=""THEN100
101 PRINTA$;: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

3.4.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.d MID$

Function

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

Format MID$(string,position,length)/=midchars/

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)
midchars	-String to insert into final string (SE)

Example

1. 100 Z$=MID$(A$,10,LEN(A$)-5)

2. 200 MID$(A$,5,2)="XX"

This example sets the fifth and sixth characters of string A$, to the string "XX".

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.

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

B

3.4.e MKN$

Function

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

Format MKN$(number)

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

Examples

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

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

Comments

1. The MKN$ 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 MKN$ function. The Interpreter only allows the use with fielded strings.

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

B,ECB,DB

3.4.f RIGHT$

Function

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

3.4.g STR$

Function

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 STRING$

Function

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

3.5 Graphic and sound commands

3.5.a ATTR

Function

To set the display attributes of the high-resolution text screen.

Format ATTRforeground,background/,B//,U/

foreground	-Foreground color number (IE)
background	-Backgound color number (IE)
B	-Character blink ON
U	-Underline text

Examples

1. 100 ATTR0,0

2. PALETTE0,0:PALETTE8,63:ATTR0,0:WIDTH80:CLS1
This will give white letters on black background.

Comments

1. The PALETTE slot numbers for ATTR are as follows:

Color	Foreground slot	Background slot
0	8	0
1	9	1
2	10	2
3	11	3
4	12	4
5	13	5
6	14	6
7	15	7

Differences from Interpretor

1. None.

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

B,ECB

3.5.b AUDIO

Function
To turn on the sound from the cassette.

Format AUDIO ON:OFF

Examples
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.c COLOR

Function

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

3.5.d CLS

Function
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

Coments
1. If the color is omitted, the screen is cleared to the color green.
2. In the high resolution text mode, CLS clears the screen, changes the background color and displays the selected color.
3. The following numbers represent the allowable color codes:

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

Differences from Interpreter
1. None.

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

3.5.e CIRCLE

Function

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),10,1

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, hw must be included.
2. If the ending point is less than or equal to the starting point, a complete circle is draw.
3. The default values for the optional items are as follows:

hw	-The value 1
start	-The value 0
end	-The value 1

Differences from Interpreter

1. The color is required with MLBASIC.

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

B,ECB

3.5.f DRAW

Function

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,..)

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

3.5.g HCOLOR

Function

To specify the background and foreground colors of the high-resolution graphics screen.

Format HCOLORforeground,background

foreground -Color of foreground (IE value 0-15)
background -Color of background (IE value 0-15)

Examples

1. 100 HCOLOR 5,7

Comments

1. By default, the foreground color is slot #1, and the background color is slot #0. The slot that gives the needed color depends on the current HSCREEN mode. In the 16 color mode, colors 1 thru 15 correspond with slots 1 thru 15 (ie. slot#1=color#1, slot#2=color#2, etc.)

Differences from Interpreter

1. None.

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

B,ECB

3.5.h HCLS

Function

To clear the high-resolution graphics screen to a desired color.

Format HCLS /color/

color -Color of background screen (IE)

Examples

1. 100 HCLS
2. 100 HCLS 11

Coments

1. If the color is omitted, the screen is cleared to the current background color.

Differences from Interpreter

1. None.

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

B

3.5.i HCIRCLE

Function

To draw a circle to the high-resolution graphics screen.

Format HCIRCLE(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 HCIRCLE(90,90),10,1

This example draws a circle 10 units in radius, centered at (90,90)

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, 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:

hw	-The value 1
start	-The value 0
end	-The value 1

Differences from Interpreter

1. The color is required with MLBASIC.

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

B,ECB

3.5.j HDRAW

Function

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

Format HDRAWcommand string

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

Examples

1. 100 HDRAW"BM500,50,U90,R50,D30,L20"

Comments

1. See section 3.5.f comments on the allowable commands for HDRAW.

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$="D10;R10;U10;L10;"
101  HDRAW"BM100,50;XA$"
```

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

3.5.k HLINE

Function

To draw a line between two points.

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

x1	-X coordinate of starting point (IE)
y1	-Y coordinate of starting point (IE)
x2	-X coordinate of ending point (IE)
y2	-Y coordinate of ending point (IE)
action	-How to draw the line. Allowable are: PSET - Sets line to foreground color PRESET - Sets line to background color
option	-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 HLINE(1,1)-(11,11),PSET
In this example, a line is drawn from (1,1) to (11,11).
2. 100 HLINE(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 639 in the X-direction and from 0 to 191 in the Y-direction when in the highest resolution mode (HSCREEN4).

Differences from Interpreter

1. MLBASIC requires that the starting point be defined.

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

B,ECB

3.5.1 HPAINT

Function

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

Format HPAINT(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 HPAINT(90,60),4,4

Comments

1. The color used in the HPAINT command must be allowable under the current high-resolution HSCREEN mode.

Differences from Interpreter

1. None.

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

B,ECB

3.5.m HPRINT

Function

To print a character string on the high-resolution screen.

Format HPRINT (x,y),message

x -X coordinate of first character to print (IE)
y -Y coordinate of first character to print (IE)
message -String to print (SE)

Examples

1. HPRINT(20,20),"Your score is"+STR$(SC)
2. HPRINT(15,10),"The Answer is :"+A$

Comments

1. The character size that is printed to the screen depends on the current HSCREEN mode. HSCREEN 3 or 4 modes allow 80 columns and 24 rows of characters. HSCREEN 1 or 2 modes allow 40 columns and 24 rows of text.

Differences from Interpretor

1. MLBASIC only allows the message to be printed in the form of a single string. STR$ and other functions can be used to convert numbers to strings for printing.

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

B,ECB

3.5.n HRESET

Function

To reset a point to the background color on the high-resolution graphics screen.

Format HRESET(x coord,y coord)

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

Examples

1. 100 HRESET(10,10)

Comments

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

Differences from Interpreter

1. None.

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

B,ECB

3.5.o HSCREEN

Function

To define a high-resolution graphics screen mode.

Format

HSCREEN mode

mode -High resolution screen mode (IE value 0-4)

Examples

1. 100 HSCREEN 4

Comments

1. The HSCREEN modes 0 thru 4 have the following settings:

mode	X grid points low res.	Y grid points low res.	Colors
0	320	192	4
1	320	192	16
2	320	192	2
3	640	192	4
4	640	192	

2. HSCREEN also clears the screen of the requested high-resolution mode.

Differences from Interpreter

1. None.

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

B,ECB

3.5.p HSET

Function

To set a point to a specified color on the high-resolution graphics screen.

Format HSET(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 HSET(20,20,2)

Comments

1. If the color code is omitted, the current foreground color is used.

Differences from Interpreter

1. None.

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

B,ECB

3.5.q LINE

Function

To draw a line between two points.

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

x1	-X coordinate of starting point (IE)
y1	-Y coordinate of starting point (IE)
x2	-X coordinate of ending point (IE)
y2	-Y coordinate of ending point (IE)
action	-How to draw the line. Allowable are: PSET - Sets line to foreground color PRESET - Sets line to background color
option	-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

3.5.r LOCATE

Function

To locate the cursor on the high-resolution text screen.

Format LOCATE x,y

x -Column number starting with 0 (IE)
y -Row number starting with 0 (IE)

Examples

1. 100 LOCATE 0,22:PRINT"ERROR":LOCATE0,0

Comments

1. The column number may be between 0 and 39 for WIDTH40 mode, and between 0 and 79 for WIDTH80 mode. The row can be between 0 and 23 for both widths.

Differences from Interpretor

1. None.

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

B,ECB

3.5.s PALETTE

Function

To set the palette slots used to display colors.

Format PALETTE slot,color:RGB:CMP

slot	-Palette register (IE value 0-63)
color	-Color Code (IE value 0-15)
RGB	-For RGB color monitors
CMP	-For composit monitors

Examples

1. 100 PALETTE 0,0:PALETTE8,63
2. 100 PALETTE RGB

Comments

1. If RGB or CMP are used, the slot and color must not be used.

Differences from Interpretor

1. None.

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

B,ECB

3.5.t PAINT

Function

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

3.5.u PCLEAR

Function

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 or 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

3.5.v PCLS

Function

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

3.5.w PLAY

Function

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;ABAO2;FEDCBA"

Comments

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

Command	Function
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$="CDEFG;O3;ABAO2;FEDCBA"
101  PLAY"T4;V5;XA$"
```

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

3.5.x PMODE

Function

To select the desired low-resolution 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

3.5.y PRESET

Function
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

3.5.z PSET

Function

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,B=Standard)

B,ECB

3.5.aa RESET

Function

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

3.5.bb SCREEN

Function

To define the low-resolution 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

3.5.cc SET

Function
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

3.5.dd SOUND

Function
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

3.5.ee WIDTH

Function

To set the number of columns in the text screen and to select low- or high-resolution graphic modes.

Format WIDTH mode

mode -Column width (32,40 or 80)

Examples

1. WIDTH32
2. WIDTH 80

Comments

1. WIDTH changes the screen display to the specified mode and clears the screen in that mode.
2. Be careful not to call WIDTH40 or WIDTH80 while in WIDTH32 mode and after issuing CLEAR commands that set the top of BASIC between &H2000 and &H3FFF. This is because, BASIC needs to occupy this region with the high-resolution text screen.

Differences from Interpretor

1. None.

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

B,ECB

3.6 Other Commands

3.6.a DATA

Function

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 character format
% - 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 used with strings, a terminating zero is not stored. In this case, a READVAR(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

3.6.b DIM

Function
To reserve space in memory for a variable array.

Format DIM arrayname,...

arrayname -Name of array followed by number of elements 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. The command REAL is used to dimension real arrays when using the %INT directive.
3. If the %INT directive is used in a program, all non string arrays declared using DIM will be of type INTEGER, otherwise the array will be of type REAL.

Differences from Interpreter
1. Only single letter array names are recognized in MLBASIC.
2. A maximum of 2 dimensions are allowed by MLBASIC.

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

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

3.6.d LPOKE

Function
To store one byte of data to virtual memory.

Format LPOKE virtual,data

virtual -Virtual memory location (RE)
data -Byte to store (IE)

Examples
1. LPOKE 460000.,123

Comments
1. The LPOKE command is the compliment to the LPEEK function.
2. See LPEEK for more information on virtual addresses.

Differences from Interpretor
1. MLBASIC does not allow numbers beginning with &H to exceed &HFFFF, therefore numbers like &H10000 must be converted to a real constant (like 65536.).

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

3.6.e MOTOR

Function
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

3.6.f POKE

Function
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

3.6.g READ

Function

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.
$=read one character into array
%=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
103 READA$
```

Comments

1. If the "S" mode is 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 "S" and "%" options.
2. 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

3.6.h REM

Function

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

3.6.i RESTORE

Function

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

Format RESTORE

Examples

1.
```
100 DATA1,2,3,4
101 RESTORE:REM' initialize data
102 FORI=1TO4: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.

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

3.6.j RUN

Function
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 progam.

Differences from Interpreter
1. None.

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

3.6.k TAB

Function

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

3.6.1 TROFF

Function
To turn off the line tracing routine.

Format TROFF

Examples
1. 100 TRON
   .
   .
   1000 TROFF:REM' END DEBUGGING HERE

Comments
1. The TROFF is the default value used by MLBASIC.

Differences from Interpretor
1. None.

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

3.6.m TRON

Function

To turn on the line execution tracing routine.

Format TRON

Examples

1.
```
100 TRON
 .
 .
1000 TROFF:REM' END DEBUGGING HERE
```

Comments

1. The TRON command will cause the display of the current line number just before execution of that line. It is therefore useful in pin-pointing errors in a program after it has been compiled.
2. TRON does not display commands that follow the ":" command separator.

Differences from Interpretor

1. None.

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

B

3.6.n VERIFY

Function
To select the verification option for disk output.

Format VERIFY ON:OFF

Examples
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

Function

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

Format DLD(memory,name)

memory -Location in memory of first byte of the two byte integer (IV,IC from 0 to 65535)

name -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

3.7.b DST

Function
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.c IBSHFT

Function

To shift a 16 bit integer by a specified number of bits 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.d INT

Function

To declare INTEGER type variables and variable arrays.

Format INTname,..

name -Name of variable

Examples

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

In this example, the scalar variables A and A1 are declared as INTEGER variables. In addition, the array A is declared as INTEGER and is dimensioned for a 10x10 array. The array B is declared INTEGER also, and is dimensioned as having 1000 elements.

2. 100 INT A1,A(10,10),B(1000)

This example produces the same result as in the first example. The scalar variable, A, is not included because the array A was declared INTEGER.

Comments

1. The INT command is required to declare a variable if that variable is used for an index to an array.

2. If an array is declared using INT, the corresponding scalar variable with the same name is forced to be 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.e LREG

Function

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.f PCOPY

Function

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

Format PCOPYstart,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 B

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

The alternative way to do an Interpreter BASIC PCOPY would be to use the interpreter call symbol "[" before the command. For example, 100 [PCOPY1 to 5.

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

B

3.7.g PTV

Function

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

3.7.h REAL

Function

To declare real type variables and variable arrays.

Format REALname,..

name -Name of variable or array

Examples

1.
```
100 %INT
101  REAL A1,A(10,10),B(1000)
```

In this example, the scalar variable A1 is declared as a real variable. In addition, the array A is declared as real and is dimensioned for a 10x10 array (note that the scalar variable, A, will be of type INTEGER). The array B is declared real also, and is dimensioned as having 1000 elements.

Comments

1. The REAL command is used to declare a variable as a real variable if the %INT directive is used to globally declare all variables as 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.
4. If the %INT directive is used, scalar variables that have the same one letter name as the array, when declared using REAL, will be of type INTEGER and cannot be type real.

Differences from Interpreter

1. The Interpreter does not allow use of this command.
2. A maximum of two dimensions are allowed by MLBASIC.

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

B

3.7.i SREG

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.j VECTD

Function
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

3.7.k VECTI

Function

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.

2. 100 VECTI($A00A):REM' SAMPLE ALL FOUR JOYSTICKS
   101 A1=PEEK($15A):A2=PEEK($15B)
   102 B1=PEEK($15C):B2=PEEK($15D)

This example is how the JOYSTK function can be duplicated. It is equivalent to the commands:

A1=JOYSTK(0):A2=JOYSTK(1)
B1=JOYSTK(2):B2=JOYSTK(3)

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

3.8 Compiler Directives

3.8.a %INT

Function

To change the default variable type within a program to INTEGER.

Format %INT

Examples

1.
```
100   %INT
101  DIM A(100):REAL B(1000),C
```

Comments

1. %INT is a compiler directive that is used to change the way a program is compiled.
2. The %INT directive causes all single letter scalar variables to be type INTEGER.
3. %INT should be used in the beginning of a program so as to avoid conflicts with any previously dimensioned arrays.
4. When the %INT directive is used, variables may be declared type real by using the REAL command.

3.8.b %REAL

Function

To change the default variable type within a program back to REAL.

Format %REAL

Examples

1.
```
 10 %INT
     .
     .
     .
100 CALL TEST(A)
     .
     .
     .
9000 SUBROUTINE TEST(B)
9001 %REAL:REM' Need alot of real variables
9002 INT B
```

This example shows that the %REAL directive is needed to force variables within SUBROUTINE sub-programs back to type real, since the %INT directive was used in the calling program.

Comments

1. %REAL is a compiler directive that is used to change the way a program is compiled.
2. The %REAL directive is the default used by MLBASIC, so it is not needed unless %INT is used.
3. %REAL will re-map the single letter scalar variables (A-Z), to the next available locations in the variable table area.

3.8.c %STRING

Function

To change the default string length within a program.

Format %STRING=length

length -Maximum length of all strings (IE)

Examples

1.
```
10 %STRING=5
20 INPUT A$
30 PRINT A$
40 %STRING=10
50 INPUT B$
60 PRINT B$
70 GOTO10
```

This example shows ways to use %STRING. Try inputting a string for A$ that is larger than 5 characters, when you print A$, only the first five characters are saved, the rest of the characters have been over-written by the B$ string.

Comments

1. The %STRING directive must be used with caution; be sure that any string that is to be used is used with the same default string length throughout the program.
2. If %STRING is not used, the default string length is used.
3. Strings can have lengths greater than 256 characters, but cannot be manipulated using string functions or the "+" operator because the string manipulation buffer is limited to 255 characters.

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.e 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"

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

Valid	Invalid
-100.10	-100
1.99 E+10	199E+12
1.0 E-110	1 E-10
-99.6E+10	-99

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 (variables that have not been dimensioned) and variable arrays (variables that have been dimensioned). There are also three types of variables; Real, Integer, and String.

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".

String variable names, as with all array names, can only be one character long (ie. A$,B$,C$,...Z$). Any character that follows the first character in a string name is ignored.

4.2.b Integer Variables

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

The default type for variables is real. The INT command or %INT directive is used to define INTEGER type variables.

4.2.c String Variables

String variables are variables that can store a sequence of characters. Like other variable types, the string variable can be changed any number of times throughout the execution of a program.

String variable names can only be one character long. This means that there are 26 possible string variables that can be used in a program unit.

String variables occupy a predefined amount of storage, as defined when the program is compiled. The maximum allowable length of a string, as with the Interpreter, is 255 characters. This is the default space allocated by MLBASIC, but can be changed by using the %STRING compiler directive or by entering a value in the start-up menu of MLBASIC.

4.2.d Real Variables

Real variables are variables that contain floating point numbers. 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.

4.3 Variable Arrays

MLBASIC allows up to two dimensions for any variable array. 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 defined to have 256 (0 thru 255) elements.

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, and of type INTEGER. If the index is not a counter variable of a FOR-NEXT loop, then the variable must be declared as INTEGER in the beginning of the program using the INT command (see section 3.7.d).

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(1,2) is arranged like:

Address	Element	Relative Location W.R.T. 1st Element
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).

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 Operator	Operation	Sample Expression
↑	Exponentiation	X↑Y
*	Multiplication	X*10.1
/	Division	X/Y
AND	Logical AND	X ANDSF000
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:

Algebraic Form	MLBASIC Form
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  1110000011111000
     ----------------
     0100000001100000
```

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

2. Example OR

```
     0101010101111100
OR   1111000011110000
     ----------------
     1111010111111100
```

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

3. Example NOT

```
     0101010101010101
NOT  0101101010011000
     ----------------
     0000111111001101
```

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

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 concatinate the elements within an expression into one final string. Maximum lengths of the final concatinated string is limited to about 255 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
```

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 %STRING=1:DIM A$(20):REM' Word to Tokenize
103 INPUT"Enter BASIC Word ";A$
104 A=LEN(A$):REM' Number of Bytes to Poke
105 FORI=0TOA:POKEI+500,A$(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
```

5.3 Interpreter Calls

MLBASIC allows a compiled program to execute a BASIC command via the special character "[" (or shift down arrow key).

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-NEXT 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
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:C$="S"+STR$(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
```

The following example shows how to use HGET and HPUT (low-resolution graphics GET/PUT can be performed in the same way), using the Interpreter Call method.

```
100 [HBUFF 1,2000
200 HSCREEN4
300 HLINE (10,0)-(20,10),PSET,B
400 [HGET (10,0)-(20,10),1
500 HCLS
600 FORI=1 TO 150 STEP5
700 POKE 1000,I
800 [I=PEEK(1000):REM' PASS MLBASIC I TO INTERPRETER I
900 [HPUT(50+I,10+I)-(60+I,20+I),1,PSET
1000 NEXT
1010 GOTO 1010
```

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.

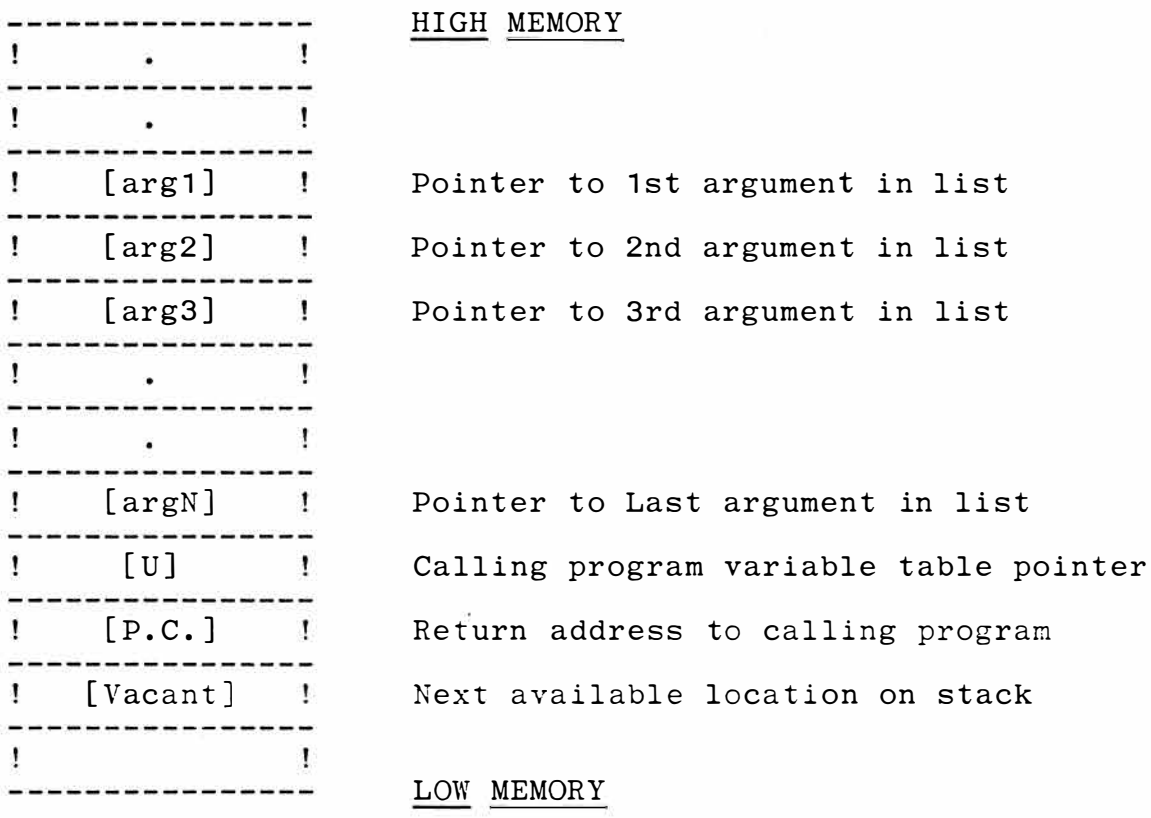
(4) The SUBROUTINE is executed. Program control returns to the calling routine when a RETURN is executed in the subprogram.

(5) The variable table pointer is loaded with its original value (obtained from the stack).

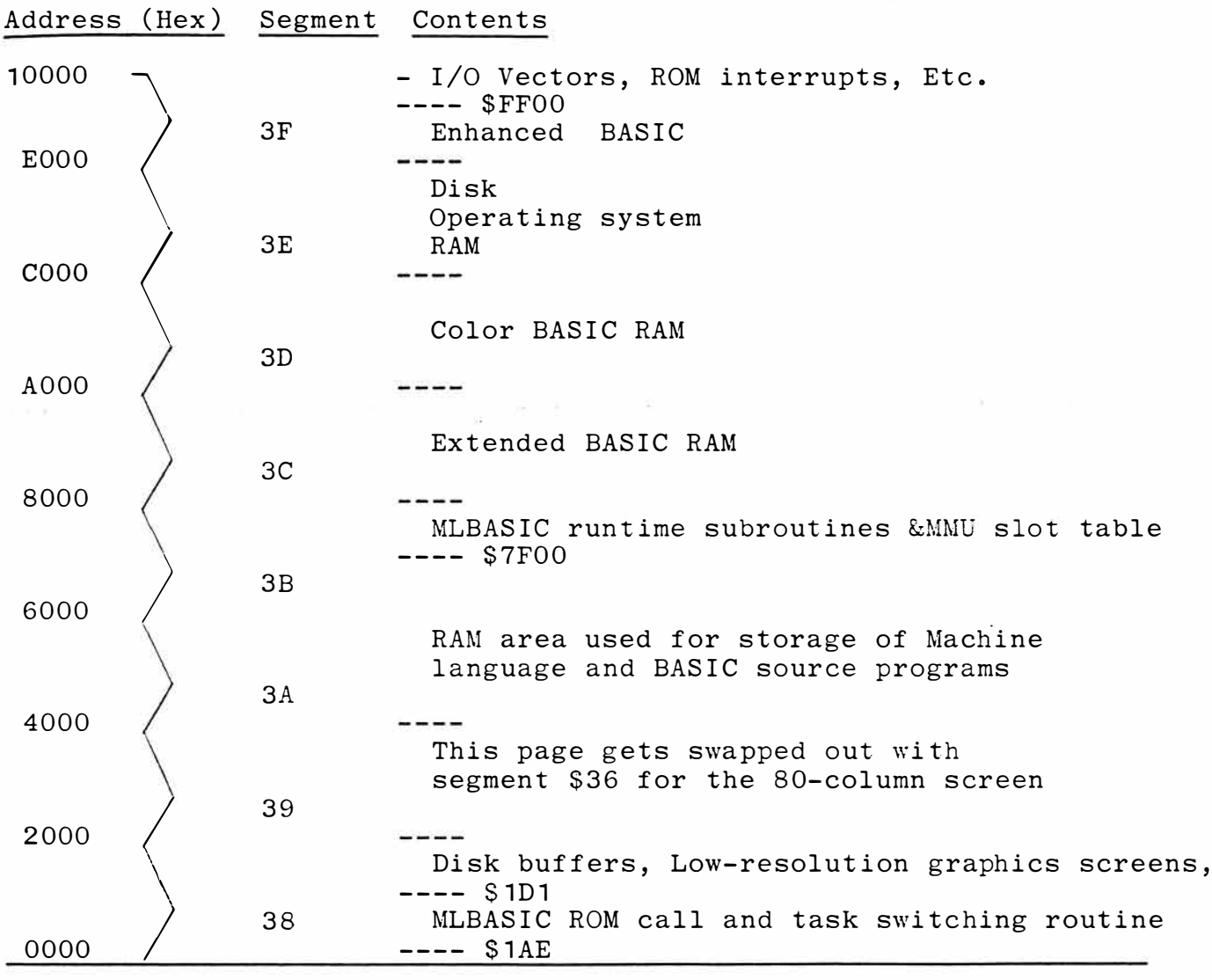
(6) The stack is reset to its original position before the CALL statement was executed. This in affect moves the stack beyond the argument list variable/constant pointers which were saved in the first step of the process.

(7) Program control resumes with the next executable statement after the CALL statement.

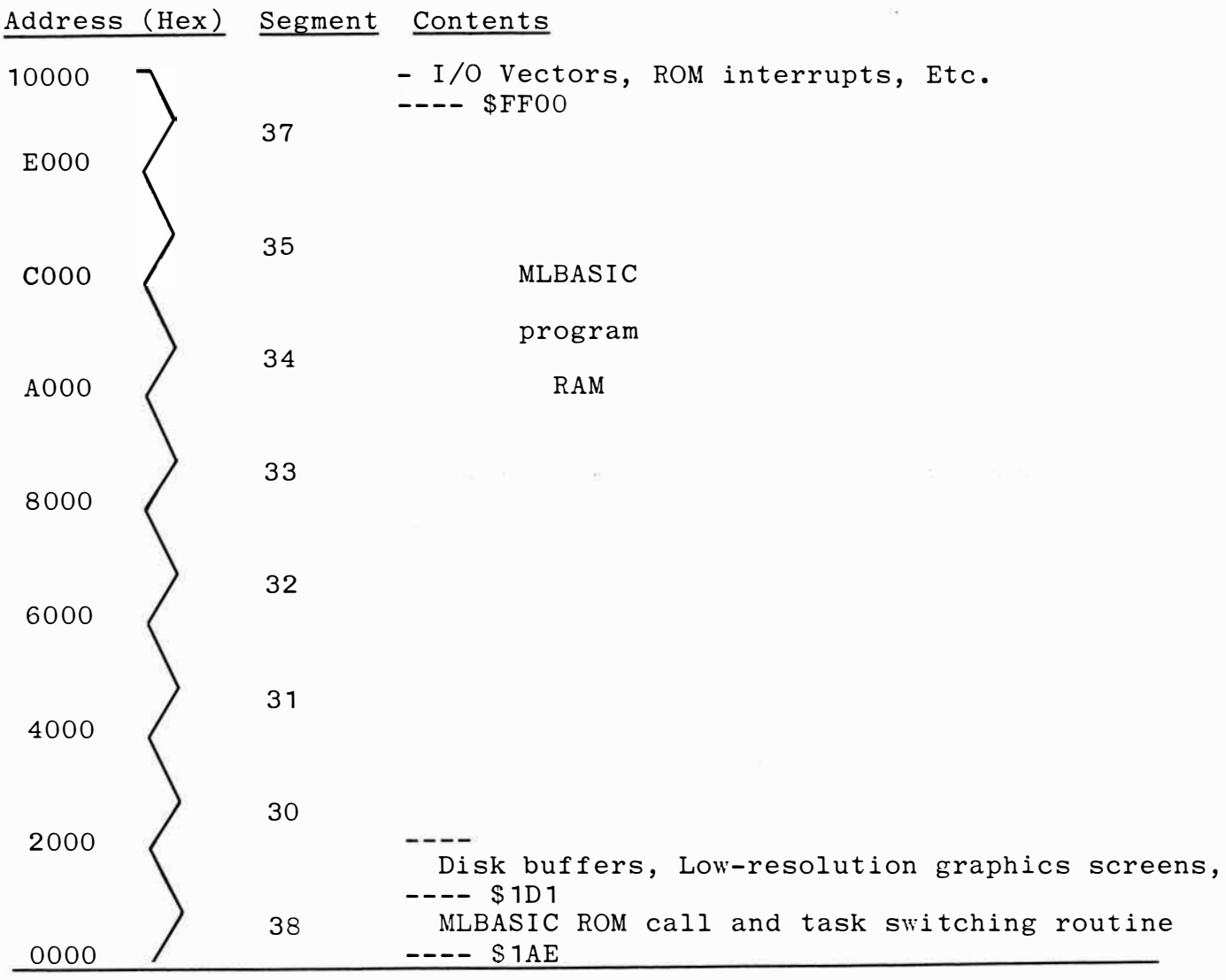
The following diagram illustrates how the "S" stack looks after Step 3 of the process (this is how it looks immediately before the SUBROUTINE is executed).



5.5 MLBASIC 2.0 Memory map



Task #0 Set of MMU slots (RAM mode)



Task #1 Set of MMU slots (RAM mode)

CHAPTER 6 SAMPLE PROGRAMS

Program #1

This program is used to delete remarks in a program. It can be useful for programs that exceed memory in BASIC.

```
MLBASIC Revision 2.0  -  COPYRIGHT (C) 1987  by, WASATCHWARE
--------------------------------------------------------------------------------
INPUT =MEMORY
OUTPUT=MEMORY

10    REM'**************************
20    REM'*  DELETE REMARKS IN A  *
30    REM'*     BASIC PROGRAM     *
40    REM'*   STORED IN MEMORY    *
50    REM'*  -(C) WASATCHWARE -   *
60    REM'*          1987         *
70    REM'**************************
80    %INT
90    %STRING=1
100   REM'
110   CLS:PRINT" THIS PROGRAM DELETES REMARKS"," FROM THE PROGRAM EXISTING IN"," 
 MEMORY."
120   DIMA$(300):REM'LINE BUFFER
130   DLD(25,A):DLD(27,B):B=B-2:DLD(A,C)
140   D=A:REM' SET START OF NEW LINE TO OLD
150   E=A+4:F=C-1:J=0
160   FORI=E TO F:REM' LOOP ON CHARACTERS IN LINE
170   G=PEEK(I):IFG=130THEN190
180   $A$(J)=G:J=J+1:GOTO210:REM' FILL OUTPUT BUFFER,NEXT CHARACTER
190   IFI>E THEN220
200   $A$(0)=G:J=1:GOTO230
210   NEXTI:GOTO240
220   J=J-1:REM' GET RID OF COLON
230   G=0:$A$(J)=G:J=J+1:REM' END OF LINE MARKER
240   REM' STORE NEW LINE OVER OLD,J=LENGTH,D=START
250   L=A+2:DLD(L,L):REM' L CONTAINS LINE #
260   E=D+4:F=E+J:F=F-1:H=0:FORI=E TO F:G=A$(H):H=H+1:POKEI,G:NEXT
270   F=F+1:DST(D,F):REM' STORE NEXT LINE LOCATION
280   H=D+2:DST(H,L):REM' STORE CURRENT LINE NUMBER
290   D=F:REM' SET NEW LINE LOCATION
300   A=C:DLD(A,C):REM' A=START OF NEXT OLD LINE,C=NEXT+1
310   IFA<B THEN150
320   REM' STORE FINALE POINTERS
330   DST(D,0):D=D+2:DST(27,D)
340   VECTI(114):REM' PERFORM A WARM START TO INITIALIZE VARIABLE POINTERS

34 PROGRAM LINES
5 GOTO/GOSUBS


                    ***** LINE NUMBER LOCATION MAP *****

LINE   LOCN    LINE   LOCN    LINE   LOCN    LINE   LOCN    LINE   LOCN

10     19008   20     19008   30     19008   40     19008   50     19008
60     19008   70     19008   80     19008   90     19008   100    19015
110    19015   120    19039   130    19039   140    19079   150    19086
160    19128   170    19134   180    19155   190    19197   200    19208
210    19227   220    19250   230    19268   240    19312   250    19312
260    19336   270    19466   280    19496   290    19523   300    19531
310    19543   320    19552   330    19552


                    ***** VARIABLE TABLE *****
1. SCALAR VARIABLES

NAME  LOCATION  TYPE      NAME  LOCATION  TYPE      NAME  LOCATION  TYPE

A     24936     INTEGER   B     24938     INTEGER   C     25000     INTEGER
D     25002     INTEGER   E     25004     INTEGER   F     25006     INTEGER
G     25008     INTEGER   H     25010     INTEGER   I     25012     INTEGER
J     25014     INTEGER   K     25016     INTEGER   L     25018     INTEGER
M     25020     INTEGER   N     25022     INTEGER   O     25024     INTEGER
P     25026     INTEGER   Q     25028     INTEGER   R     25030     INTEGER
S     25032     INTEGER   T     25034     INTEGER   U     25036     INTEGER
V     25038     INTEGER   W     25040     INTEGER   X     25042     INTEGER
Y     25044     INTEGER   Z     25046     INTEGER

2. DIMENSIONED VARIABLES

NAME  LOCATION  TYPE      1st DIMENSION

A$    25050     STRING    301

MAIN PROGRAM AREA          -19000  TO 19609
CHARACTER DATA AREA        -19613  TO 19682
SUBROUTINE LIBRARY AREA    -19683  TO 24983
VARIABLE STORAGE AREA      -24984  TO 25350

0  ERRORS
```

Program #2

This program is a basic text editor with wordprocessing capabilities. The program can be used to become familiar with the part of MLBASIC that is not available with regular Interpreter BASIC.

To compile the program:

(1) Load in MLBASIC
(2) Enter CLEAR 200,19301
(3) LOAD "PROGRAM2"
(4) Enter EXEC
(5) Hit CTRL , and wait for it to be compiled
(6) Save program to disk: SAVEM"TEXT",19500,32375,19500

To load the program and execute it, you must use the 64k task #1 loader program called "R.BAS". To run the program, the menus will help you thru most of the functions. In edit mode, the "H" key lists the options. Some of the more complicated functions are:

"U" -Bank a section of text (up to 2500 bytes in this version)
You first position the cursor to where you want in the text and then hit "U" to start banking. Move cursor along to the end of the text you want to bank, then hit "U" again.

"V" -Insert the banked buffer (using "U"). It may be inserted more than once.

Other things to know about the editor section are that the following keys perform the function of moving the cursor position around the text as:

, -Go to top of Text.
. -Same as ","
ENTER -Go to next page

Hopefully this will get one started into a better understanding of the optins available with MLBASIC.

```
MLBASIC Revision 2.0  -  COPYRIGHT (C) 1987  by WASATCHWARE
--------------------------------------------------------------------------------
INPUT =MEMORY
OUTPUT=MEMORY

10    REM' ******************************
11    REM' **    WORDPRO 2.0 -DISK     **
12    REM' **        06/01/87          **
13    REM' ******************************
14    REM
15    %INT
16    %STRING=1
31    DATA9999
50    DATA$"                        ANYNAMEHERE"
51    DATA$"                        1234 Road Lane "
52    DATA$"                        Whoknowswhere, U.S. "
53    DATA$"                            98765      "
54    DATA$"Thank You,"
55    DATA$"John Doe  "
56    DATA$"Manager, ANYNAMEHERE"
70    DIM F$(20),G$(20):REM' FILENAMES
86    DIM A(10),E$(80),D$(2500),B$(80),C$(10),A$(20000)
87    A(2)=0:S=0:B=0:A(3)=VARPTR(A$):REM'SET BUFFER START
88    A(4)=0:REM'INITIALIZE BANKING BUFFER
100   CLS:PRINT"    WORDPRO (VERSION 1.0)","     (C) 1987 WASATCHWARE","ENTER O
PTION #","1.- INPUT","2.- DIR","3.- PRINT","4.- EDIT"
101   PRINT"5.- TEXT SAVE/LOAD","6.- EXIT"
120   PRINT:INPUTZ
121   IFZ=1THEN1000
122   IFZ=2THEN2000
123   IFZ=3THEN3000
124   IFZ=4THEN4000
125   IFZ=5THEN7000
126   IFZ<>6THEN100
127   STOP
1000  A=B:GOTO6000:REM'SELECT INPUT MENU
1001  GOSUB11100:GOTO100
2000  DIR:GOTO100
3000  GOSUB12100:CLS:PRINT"READY PRINTER & ENTER:","LEFT,RIGHT MARGIN ";:INPUTD:
PRINT",";:INPUTE:PRINT" ","TOP,BOTTOM OF PAGE "
3001  INPUTF:PRINT",";:INPUTG:PRINT:T=1
3002  PRINT"ENTER 2ND MARGIN ";:INPUTA(1)
3003  PRINT" ","ENTER FORM LENGTH ";:INPUTL
3004  IFS=1THEN3010
3005  PRINT" ","IS IT A LETTER (1=YES)?";:INPUTS
3009  PRINT" ","ENTER STARTING PAGE ";:INPUTP
3010  PRINT" ","ENTER THE TITLE"
3011  INPUT B$:T=LEN(B$)
3020  Q=1:R=1:F=F-1:K=0:T=T-1
3021  GOSUB5200:IFZ=3THEN100
3022  REM'BEGIN LOOP TO PRINTOUT TEXT
3023  FORI=A TO B:Z=A$(I)
3024  GOSUB11000:REM'SCAN FOR INTERRUPT
3028  IFQ>=0 THEN3032
3030  GOSUB5100
3032  IFZ=13THEN3070
3034  IFZ=64THEN3080
3036  IFZ=32THEN3060
3038  IFZ=9THEN3090
3039  IFZ=19THEN3100
3040  IFZ=12THEN3200
3050  K=K+1:E$(K)=Z:GOTO3500
3060  IFK=0THEN3068
3061  Q=Q+K:IFQ>E THEN3066
3062  GOSUB5400
3064  PRINT#-2," ";:Q=Q+1:GOTO3500
3066  PRINT#-2:R=R+1:Q=1:GOSUB5300:Q=Q+K:GOTO3062
3068  PRINT#-2," ";:Q=Q+1:GOTO3500
3070  IFK<>0THEN3074
3072  PRINT#-2:R=R+1:Q=1:A(2)=0:GOSUB5300:GOTO3500
3074  Q=Q+K:IFQ>E THEN3076
3075  GOSUB5400:GOTO3072
3076  PRINT#-2:R=R+1:Q=1:GOSUB5300:Q=Q+K:GOTO3075
3080  A(2)=0:FORJ=R TO L:PRINT#-2:NEXT:P=P+1:R=1
3082  Q=1:GOSUB5200:IFZ=3THEN100
3083  GOTO3500
3090  IFK=0THEN3092
3091  Q=Q+K:GOSUB5400
3092  PRINT#-2,"     ";:Q=Q+5:GOTO3500
3100  A(2)=1:REM'SET FLAG TO INDENT TO 2ND MARGIN
3101  GOSUB5600:GOTO3500
3200  Z=25:GOTO3050
3500  NEXT
4000  A=0:CLS:PRINT"E>";
4001  GOSUB5001
4002  IFZ=94THEN4050
4003  IFZ=10THEN4060
4004  IFZ=3THEN100
4005  IFZ=9THEN4085
4006  IFZ=8THEN4080
4007  IFZ="I"THEN4090
4008  IFZ="D"THEN4100
4009  IFZ="C"THEN4200
4010  IFZ="B"THEN4250
4011  IFZ=13THEN4300
4012  IFZ="A"THEN4310
4013  IFZ="G"THEN4350
4014  IFZ="S"THEN4380
4015  IFZ="R"THEN4400
4016  IFZ="H"THEN4420
4017  IFZ="U"THEN4500
4018  IFZ="V"THEN4520
4019  IFZ=","THEN4000
4020  IFZ="."THEN4000
4021  IF Z=" " THEN4085
4022  IF Z=103 THEN4110
4023  IF Z=4 THEN4120
4024  IF Z="]" THENFORX=1 TO 10:GOSUB4065:NEXT:GOTO4001
4040  GOTO4001
4050  IFA<100THEN4000
4051  CLS:PRINT"^^";:A=A-100:FORI=1TO100
4052  GOSUB4065:NEXT:A=A-100:GOTO4001
4060  Z=A$(A):IFZ=13THEN4063
4061  GOSUB4065:IFA>=B THEN4001
4062  GOTO4060
4063  PRINT:A=A+1:GOTO4001
4065  Z=A$(A):GOSUB10000
4079  A=A+1:RETURN
4080  PPINT$C$(I);:A=A-1:GOTO4001
4085  GOSUB4065:GOTO4001
4090  GOSUB11900
4095  GOSUB11700:GOTO4001
4100  INPUTZ
4101  GOSUB11600:GOTO4001
4110  REM  F1 JUMP TO LOCATION IN BUFFER
4111  PRINT" ","POINTER=";A:INPUT"ENTER ADDRESS ";A
4112  GOTO 4001
4120  REM' F2 SET BUFFER LENGTH
4121  CLS:PRINT"BUFFER=";B:INPUT"ENTER BUFFER SIZE (0-> LEAVE AS IS)";Z
4122  IF Z>0 THEN B=Z
```

```
 4124  GOTO4001
 4200  GOSUB5200:$A$(A)=Z:A=A+1:GOTO4001
 4250  PRINT" ","BUFFER= ";B;"BYTES":GOTO4001
 4300  Z=A$(A):IFZ=64THEN4303
 4301  GOSUB4065:IFA>=B THEN4001
 4302  GOTO4300
 4303  GOSUB4065:GOTO4001
 4310  PRINT" ","LINE POINTER=";A;"BYTES":GOTO4001
 4350  REM'GRAPHICS ROUTINE
 4351  PRINT" ","G>";:Z=A$(A)
 4352  PRINT$A$(A);"(";Z;")";:INPUTZ
 4353  IFZ=0THEN4355
 4354  $A$(A)=Z:GOTO4001
 4355  PRINT".... NO CHANGE MADE":GOTO4001
 4380  GOSUB11800
 4383  GOSUB11500
 4390  A=A-C:PRINT" ","FOUND IT":GOTO4001
 4400  CLS:PRINT"INPUT STRING TO BE REPLACED"
 4401  GOSUB11800
 4402  PRINT" ","& THE REPLACEMENT STRING?"
 4403  X=C:GOSUB11900:D=C:C=X
 4404  GOSUB11500:A=A-C:IFA>B THEN4000
 4405  GOSUB5001:IFZ=3THEN4000
 4406  IFZ=83THEN4408
 4407  A=A+C:GOTO4404
 4408  Z=C:GOSUB11600:Z=8:GOSUB10000:REM'DELETE STRING, PRINT BACKSPACE
 4409  X=C:C=D:GOSUB11700:C=X:GOTO4404
 4420  CLS:PRINT"..... EDIT COMMAND INDEX .....","A -DISPLAY CURSOR LOCATION","B
 -DISPLAY BUFFER SIZE","C -CHANGE CURRENT CHARACTER"
 4422  PRINT"D -DELETE N CHARACTERS","I -INSERT TEXT","G -DISPLAY/ENTER NEW VALUE
","R -REPLACE STRING","S -SEARCH FOR STRING"
 4424  PRINT"U -BANK A SECTION OF TEXT","V -UNBANK (INSERT BUFFER)","BREAK -EXIT
EDIT MODE"
 4425  PRINT". -GO TO START OF BUFFER","F1 -JUMP TO A LOCATION IN BUFFER","F2 -CH
ANGE BUFFER SIZE":GOTO4001
 4500  REM'INSERT DATA INTO D$ BUFFER
 4502  IFA(4)<>0THEN4510
 4504  CLS:PRINT"BEGIN BANKING TEXT. MOVE CURSOR","TO END OF AREA, THEN HIT 'U'"
 4506  A(4)=A:GOTO4310
 4510  Z=A-A(4):C=0:IFZ>2499THEN4518
 4512  FORX=A(4) TO A:C=C+1:Y=A$(X):$D$(C)=Y:NEXT
 4514  A=A-Z:GOSUB11600
 4516  A(5)=Z:A(4)=0:PRINT" ",A(5);"BYTES ARE NOW BANKED":GOTO4001
 4518  PRINT" ","BUFFER OVERFLOW":Z=2499:GOTO4512
 4520  C=A(5):PRINT" ",C;"BYTES INSERTED":GOTO4095
 5000  GOSUB5001:PRINT$C$(1);:RETURN
 5001  C$(1)=INKEY$:Z=C$(1):IFZ=0THEN5001
 5002  RETURN
 5010  PRINT" ","HIT ANY KEY TO CONTINUE":GOTO5001
 5100  FORJ=1TO D:PRINT#-2," ";Q=Q+1:NEXT
 5102  IFA(2)=0THEN5104
 5103  GOSUB5600
 5104  RETURN
 5200  IFR>=F THEN5205
 5201  IFS<>1THEN5203
 5202  PRINT" ","READY NEXT PAGE":GOSUB5000:GOTO5205
 5203  PRINT#-2,"                                        ";P," ":R=R+2
 5204  GOSUB5500:FORJ=R TO F:PRINT#-2:R=R+1:NEXT
 5205  RETURN
 5300  IFR<=G THEN5302
 5301  FORJ=R TO L:PRINT#-2:NEXT:R=1:P=P+1:GOSUB5200:Q=1
 5302  GOSUB5100:RETURN
 5400  FORJ=1TO K:PRINT#-2,$E$(J);:NEXT:K=0:RETURN
 5500  GOSUB5100:PRINT#-2,B$
 5501  Q=1:R=R+1:RETURN
 5600  FORJ=1TO A(1):PRINT#-2," ";:NEXT:Q=Q+A(1):RETURN
 6000  CLS:PRINT"ENTER INPUT OPTION:","1.- LETTER","2.- TEXT":INPUTZ:PRINT
 6002  IFZ=1THEN6006
 6003  REM' INPUT LETTER HEADING INTO TEXT BUFFER, A$
 6004  CLS:PRINT"I>";:GOTO1001
 6006  RESTORE
 6008  READI:IFI<>9999THEN6008
 6010  S=1
 6012  FORA=0TO30:READ$A$(A):NEXT
 6014  W=13:$A$(31)=W
 6016  FORA=32TO66:READ$A$(A):NEXT
 6018  $A$(67)=W
 6020  FORA=68TO107:READA$(A):NEXT
 6022  $A$(108)=W
 6024  FORA=109TO143:READ$A$(A):NEXT
 6026  A=144:GOSUB11400:GOSUB11400:GOSUB11400
 6028  GOSUB11100
 6030  Y=35:GOSUB11200:Y=10:GOSUB11300:GOSUB11400
 6031  GOSUB11400:GOSUB11400:GOSUB11400
 6032  Y=35:GOSUB11200:Y=10:GOSUB11300:GOSUB11400
 6034  Y=35:GOSUB11200:Y=20:GOSUB11300:GOSUB11400:B=A:GOTO100
 7000  REM' I/O SECTION
 7002  CLS:PRINT"    STORAGE  SECTION"," ","ENTER OPTION #"
 7004  PRINT"0. -EXIT MENU","1. -LOAD TEXT FROM DISK","2. -SAVE TEXT TO DISK","3.
 -APPEND TEXT FROM DISK","4. -SAVE SEGMENT OF TEXT TO DISK"
 7005  PRINT"5. -VIEW DISK DIRECTORY"
 7006  INPUTZ:U=1
 7008  IFZ=1THEN7100
 7010  IFZ=2THEN7200
 7012  IFZ=3THEN7300
 7014  IFZ=4THEN7600
 7015  IFZ=5THEN7700
 7016  IF Z=0 THEN100
 7018  GOTO7002
 7100  REM 'LOAD TEXT
 7102  GOSUB7400:REM' INIT FILE
 7110  B=0:A=0
 7112  REM
 7114  INPUT#U,$A$(A):Z=A$(A):IF Z<>13THENA=A+1:GOTO7114
 7115  A=A+1:IF EOF(U)=0THEN7114
 7116  B=A:CLOSE:GOTO100
 7200  REM' SAVE FILE
 7202  GOSUB7500
 7204  A=0:REM' INIT TO START OF BUFFER
 7206  Y=B-1:REM' END OF TEXT TO OUTPUT
 7208  REM
 7210  FORX=A TO Y:PRINT#U,$A$(X);:NEXT:REM' OUTPUT BUFFER
 7211  X=X-1:Z=A$(X):IF Z<>13 THENPRINT#U," ":REM' EOL IS REQUIRED
 7212  CLOSE
 7218  GOTO100
 7300  REM' APPEND
 7301  GOSUB7400:A=B:GOTO7112
 7400  INPUT"ENTER INPUT FILENAME ";F$
 7402  OPEN"I",#U,F$
 7408  RETURN
 7500  INPUT"ENTER OUTPUT FILENAME ";G$
 7502  OPEN"O",#U,G$
 7505  RETURN
 7600  REM' OUTPUT A SEGMENT
 7601  GOSUB7500:PRINT"ENTER START,END OF TEXT":INPUTA,Y:GOTO7208
 7700  DIR:PRINT"NUMBER OF FREE GRANULES=";:PRINT FREE(0)
 7701  GOTO 7006
10000  GOSUB11000:IFZ<>9THEN10002
10001  PRINT"     ";:RETURN
10002  IFZ<>64THEN10004
10003  CLS:RETURN
10004  IFZ<>13THEN10006
```

```
10005 V=144:GOTO10009
10006 IFZ<>12THEN10008
10007 V=140:GOTO10009
10008 Y=Z
10009 $C$(1)=V:PRINT$C$(1);
11000 REM'BREAK ROUTINE
11001 C$(2)=INKEY$:W=C$(2)
11002 IFW=19THEN11004
11003 RETURN
11004 REM'POLL FOR ANY KEY TO CONTINUE
11005 W=Z:GOSUB5001:Z=W:RETURN
11100 GOSUB5001:REM'GET INPUT CHAR
11101 IFZ<>8THEN11103
11102 PRINT$C$(1);:A=A-1:GOTO11100
11103 IFZ<>3THEN11105
11104 B=A:RETURN
11105 IFZ=8THEN11100
11106 IFZ=94THEN11100
11107 IFZ=10THEN11100
11110 $A$(A)=Z:GOSUB4065:GOTO11100
11200 W=32:FORI=1TO Y:$A$(A)=W:A=A+1:NEXT:RETURN
11300 FORI=1TO Y:READ$A$(A):A=A+1:NEXT:RETURN
11400 W=13:$A$(A)=W:A=A+1:RETURN
11500 Y=E$(1):Z=A$(A):GOSUB10000:IFA>=B THEN11905
11501 A=A+1:IFZ<>Y THEN11500
11502 IF C>1 THEN FORI=2TO C :ELSERETURN
11503 Z=A$(A):Y=E$(I)
11504 IFY<>Z THEN11507
11505 A=A+1:NEXT:RETURN
11506 GOTO11500
11507 A=A-I:A=A+2:GOTO11500:REM'RESET & TRY AGAIN
11600 FORI=A TO B:Y=I+Z:X=A$(Y)
11601 $A$(I)=X:NEXT:B=B-Z:RETURN
11700 FORI=B TO A STEP-1:Z=I+C:Y=A$(I)
11701 $A$(Z)=Y:NEXT
11702 FORI=1TO C:Z=D$(I):$A$(A)=Z:A=A+1:NEXT
11703 B=B+C:RETURN
11800 C=0:REM'INPUT E$ BUFFER
11801 GOSUB5000:IFZ=3THEN11905
11802 IFZ<>8THEN11804
11803 C=C-1:GOTO11801
11804 C=C+1:$E$(C)=Z:GOTO11801
11900 C=0:REM'INPUT D$ BUFFER
11901 GOSUB5001:GOSUB10000:IFZ=3THEN11905
11902 IFZ<>8THEN11904
11903 C=C-1:GOTO11901
11904 C=C+1:$D$(C)=Z:GOTO11901
11905 RETURN
12000 PRINT" ","READY CASSETTE?":GOSUB5001:RETURN
12100 CLS:PRINT"ENTER BAUD RATE","1 -600 BAUD","2 -1200 BAUD"
12102 INPUTZ:IFZ=2THEN12106
12104 POKE150,87:RETURN
12106 POKE150,41:RETURN
12200 Z=PEEK(129):IFZ=0THEN12204
12202 PRINT"I/O ERROR #";Z
12204 RETURN
12300 GOSUB12200:IFZ=0THEN12304
12302 PRINT"SIZE=";I;"LOC=";J:J=J-I:J=J+255
12304 RETURN
30000 END

310 PROGRAM LINES
135 GOTO/GOSUBS
```

***** LINE NUMBER LOCATION MAP *****

LINE	LOCN	LINE	LOCN	LINE	LOCN	LINE	LOCN	LINE	LOCN
10	19508	11	19508	12	19508	13	19508	14	19508
15	19508	16	19508	31	19515	50	19520	51	19551
52	19586	53	19626	54	19661	55	19671	56	19681
70	19701	86	19701	87	19701	88	19736	100	19744
101	19768	120	19785	121	19805	122	19816	123	19827
124	19838	125	19849	126	19860	127	19871	1000	19875
1001	19884	2000	19990	3000	19900	3001	19967	3002	20020
3003	20045	3004	20070	3005	20081	3009	20106	3010	20131
3011	20148	3020	20176	3021	20236	3022	20250	3023	20250
3024	20270	3028	20273	3030	20284	3032	20287	3034	20298
3036	20309	3038	20320	3039	20331	3040	20342	3050	20353
3060	20395	3061	20406	3062	20432	3064	20435	3066	20470
3068	20529	3070	20564	3072	20575	3074	20624	3075	20650
3076	20656	3080	20715	3082	20784	3083	20806	3090	20809
3091	20820	3092	20841	3100	20876	3101	20884	3200	20890
3500	20901	4000	20920	4001	20946	4002	20949	4003	20960
4004	20971	4005	20982	4006	20993	4007	21004	4008	21015
4009	21025	4010	21037	4011	21048	4012	21059	4013	21070
4014	21081	4015	21092	4016	21103	4017	21114	4018	21125
4019	21136	4020	21147	4021	21158	4022	21169	4023	21180
4024	21191	4040	21240	4050	21243	4051	21253	4052	21294
4060	21336	4061	21351	4062	21373	4063	21376	4065	21402
4079	21422	4080	21439	4095	21472	4090	21478	4095	21481
4100	21487	4101	21500	4110	21506	4111	21506	4112	21552
4120	21555	4121	21555	4122	21609	4124	21633	4200	21636
4250	21678	4300	21715	4301	21740	4302	21752	4303	21755
4310	21761	4350	21798	4351	21798	4352	21827	4353	21899
4354	21900	4355	21923	4380	21943	4383	21946	4390	21949
4400	21984	4401	22008	4402	22011	4403	22028	4404	22052
4405	22077	4406	22091	4407	22102	4408	22120	4409	22141
4420	22168	4422	22192	4424	22209	4425	22226	4500	22246
4502	22246	4504	22257	4506	22281	4510	22291	4512	22326
4514	22402	4516	22421	4518	22480	4520	22508	5000	22557
5001	22575	5002	22607	5010	22608	5100	22628	5102	22696
5103	22637	5104	22700	5200	22701	5201	22712	5202	22723
5203	22746	5204	22806	5205	22862	5300	22863	5301	22874
5302	22946	5400	22950	5500	23009	5501	23030	5600	23057
6000	23116	6002	23160	6003	23171	6004	23171	6006	23193
6008	23202	6010	23228	6012	23236	6014	23286	6016	23300
6018	23250	6020	23359	6022	23405	6024	23414	6026	23464
6028	23480	6030	23483	6031	23508	6032	23517	6034	23542
7000	23576	7002	23576	7004	23600	7005	23617	7006	23634
7008	23655	7010	23666	7012	23677	7014	23698	7015	23699
7016	23710	7018	23721	7100	23724	7102	23724	7110	23727
7112	23741	7114	23741	7115	23819	7116	23860	7200	23876
7202	23876	7204	23879	7206	23886	7208	23903	7210	23903
7211	23953	7212	24020	7218	24027	7300	24030	7301	24030
7400	24042	7402	24065	7408	24086	7500	24087	7502	24110
7505	24131	7500	24132	7501	24132	7700	24178	7701	24204
10000	24207	10001	24221	10002	24234	10003	24245	10004	24253
10005	24264	10006	24275	10007	24286	10008	24297	10009	24305
11000	24328	11001	24328	11002	24352	11003	24363	11004	24364
11005	24364	11100	24384	11101	24387	11102	24398	11103	24431
11104	24442	11105	24449	11106	24460	11107	24471	11110	24482
11200	24508	11300	24529	11400	24649	11500	24691	11501	24730
11502	24757	11503	24783	11504	24818	11505	24829	11506	24865
11507	24868	11600	24901	11601	24936	11700	24993	11701	25030
11702	25070	11703	25143	11800	25159	11801	25166	11802	25180
11803	25191	11804	25210	11900	25249	11901	25256	11902	25273
11903	25284	11904	25303	11905	25342	12000	25343	12100	25364
12102	25389	12104	25413	12106	25425	12200	25438	12202	25459
12204	25484	12300	25485	12302	25499	12304	25575		

```
                    ***** VARIABLE TABLE *****
1. SCALAR VARIABLES

NAME  LOCATION  TYPE      NAME  LOCATION  TYPE      NAME  LOCATION  TYPE

/\      32388   INTEGER   B       32390   INTEGER   C       32392   INTEGER
D       32394   INTEGER   E       32396   INTEGER   F       32398   INTEGER
G       32400   INTEGER   H       32402   INTEGER   I       32404   INTEGER
J       32406   INTEGER   K       32408   INTEGER   L       32410   INTEGER
M       32412   INTEGER   N       32414   INTEGER   O       32416   INTEGER
P       32418   INTEGER   Q       32420   INTEGER   R       32422   INTEGER
S       32424   INTEGER   T       32426   INTEGER   U       32428   INTEGER
V       32430   INTEGER   W       32432   INTEGER   X       32434   INTEGER
Y       32436   INTEGER   Z       32438   INTEGER

2. DIMENSIONED VARIABLES

NAME  LOCATION  TYPE      1st DIMENSION

A       32484   INTEGER   11
A$      35180   STRING    20001
B$      35088   STRING    81
C$      35169   STRING    11
D$      32587   STRING    2501
E$      32506   STRING    81
F$      32442   STRING    21
G$      32463   STRING    21

MAIN PROGRAM AREA          -19500  TO 25579
CHARACTER DATA AREA        -25583  TO 27074
SUBROUTINE LIBRARY AREA    -27075  TO 32375
VARIABLE STORAGE AREA      -32376  TO 35180

0  ERRORS
```

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 COMPILATION" will indicate that the compiler was properly aborted.

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.
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.
17	Undefined FIELD. The FIELD must be defined before RSET or LSET use that field.
18	Missing DATA statements.
19	RESERVED
20	Illegal compiler directive
21	FOR with no NEXT
22	NEXT without FOR
23	Multiple IF..THEN statement on same line. Break up line into single IF..THEN statements

7.2 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.i).

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)

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 Interpreter call (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.

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 incorrecly written to. Try to recover as many files from disk as possible. Disk needs to be re-formattted.
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.
39	"HR" -High-resolution graphics error.
40	"HP" -High-resolution print error.

CHAPTER 8 PROGRAM CONVERSION TIPS

8.1 Example Conversions

There are certain differences between programs written for ordinary Interpreter BASIC and a similar program written for MLBASIC. In this section, there are examples showing the changes needed so that a command written for Interpreter BASIC will perform the same when compiled by MLBASIC.

1. LOADM Command

Interpreter Form:
```
LOADM"FILENAME"
```

MLBASIC Form:
```
OPEN"I",#1,"FILENAME"
INT A,B
INPUT#1,$A$,A,B:B=B-A+3
FORI=0 TO B:INPUT#1,$A$
POKEI+A,A$:NEXT
INPUT#1,A:DST(157,A):CLOSE#1
```

2. IF-THEN Command

```
IF A=B OR B=C THEN1
```
```
IF A=B
IF B=C THEN1
```

```
IF A=B THEN100ELSE200
```
```
IFA=B THENGOTO100ELSEGOTO200
```

```
IF A=B AND B=C THEN1
```
```
IF (A-B)OR(B-C)=0 THEN1
```

```
IF A<>B AND B<>C THEN1
```
```
IF (A-B)OR(B-C)<>0 THEN1
```

```
IF EOF(1) THEN1
```
```
IF EOF(1)<>0THEN1
```

```
IF A=BTHEN IFC=BTHEN100
```
```
52 IFA<>B THEN54
53 IF C=BTHEN100
54 ....
```

3. PCOPY Command

```
PCOPYA TO B
```
```
A1=A*1536:A2=A1+1535
A3=B*1536:PCOPY A1,A3,A2
```

4. FOR Loops

```
FORI=100TO-100STEP-1
```
```
FORJ=200TO0STEP-1:I=J-100.
```

5. Array Indexes

```
A(IC)=A(I+1)
```
```
INT Y,Z:Y=IC:Z=I+1:A(Y)=A(Z)
```

6. REAL Constants

```
A=&H65000
```
```
A=6.*65536.+&H5000
```

7. Graphics GET/PUT

```
DIM V(20,20)
GET(I,J)-(K,L),V
```
```
[DIM V(20,20)
POKE1000,I:POKE1001,J:POKE1002,K:POKE1003,L
[I=PEEK(1000):[J=PEEK(1001)
[K=PEEK(1002):[L=PEEK(1003):[GET(I,J)-(K,L),V
```

```
PCLS
PUT(I+M,J+M)-(K+M,L+M),V
```
```
PCLS
POKE1000,M:[M=PEEK(1000)
[PUT(I+M,J+M)-(K+M,L+M),V
```

8.2 Conversion of ASCII files

Disk or cassette files that have been created from programs written by the Interpreter can be read using MLBASIC in most cases. MLBASIC and normal Basic use different item separators. MLBASIC uses a zero byte to separate strings, while the Interpreter uses commas or the CHR$(13) character to separate items in the file. If a number was written to a file with the Interpreter (for example PRINT#1,123.4), the number is stored as a string. In order to read that number back using a MLBASIC compiled program, the number must be read as a string, and then converted into a real number (ie. INPUT#1,A$:A=CVN(A$)).

The following program can be used to convert an ASCII disk file, that has been created with an Interpreted Basic program, into the proper format so it can be read back by a MLBASIC compiled program.

```
1 OPEN"I",#1,"FILENAME"
2 OPEN"O",#2,"FILENAME"+".ML"
3 %STRING=1:DIM A$(2):INT A
4 INPUT#1,$A$:A=A$
5 IF A=","THEN$A$=0
6 IF A=13 THEN$A$=0
7 IF EOF(1)<>0 THEN CLOSE:STOP
8 PRINT#2,$A$;:GOTO4
9 END
```