2 3 0 1 0 2 M T U - B A S I C 1.5

REFERENCE MANUAL

OCTOBER, 1982 REV. A

TABLE OF CONTENTS

DISCLAIMER OF ALL WARRANTIES AND LIABILITY

No warranties, either express or implied, are made by Micro Technology Unlimited with respect to this manual or the software described herein, its quality, performance, merchantability, or fitness for any particular application. This product is sold "as is". The buyer assumes all risk as to quality and performance. Under no circumstances will MTU be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software, even if MTU has been advised of the possibility of such damages. Should the software prove defective following purchase, the buyer assumes the entire cost of all necessary servicing, repair or correction and any incidental, indirect, or consequential damages. Additional rights vary from state to state so some of the above exclusions and limitations may not apply to you.

Micro Technology Unlimited reserves the right to make changes to the product and specifications described in this manual at any time without notice.

1. INTRODUCTION

BASIC 1.5 is an upgraded version of MTU BASIC 1.0. The BASIC language provided by BASIC 1.5 is virtually identical to that in BASIC 1.0. Only two minor changes were made to the BASIC language itself. This means that the great majority of programs written to run under BASIC 1.0 will run without modification under BASIC 1.5. The reverse will also be true, provided the program doesn't exceed the memory capacity of BASIC 1.0.

BASIC 1.5 has two major enhancements. First, there is a much greater amount of memory available for program and data. BASIC 1.5 takes advantage of the special bank switching capabilities of the MTU-130 to separate program memory from data memory. Approximately 34100 bytes of memory in bank 0 are available for program, string characters, and libraries. In addition, there are 64767 bytes in bank 3 available for variables and arrays. In the case of string variables, a length and pointer are stored in the variable in bank 3, and the actual characters which make up the string are stored in bank 0. In general, the memory in bank 0 will be referred to as the Program Bank, and the memory in bank 3 as the Data Bank.

The second area of improvement is in execution speed. This is accomplished by using the 16 bit, 68000 microprocessor on the Datamover-256 board to perform various functions, if present. These functions include the floating point arithmetic, searching for simple variables, calculating the location of an array element from its subscripts, and string garbage collection. Also, the 68000 processor maintains a table of addresses of line numbers for the current BASIC program. This allows GOTO and GOSUB statements to find their destination much faster by using table-lookup. To allow Libraries to take advantage of the 68000, the LIB command has been upgraded so that it can load a block of code in another bank in addition to the block that it loads in bank 0. This allows a Library to include 68000 machine code along with the 6502 code.

There have been only two small changes made to the operation of the language. First, since program and data memory have been separated, the FRE() function has been modified appropriately. FRE(0) returns the amount of free space in the program memory. The FRE() function with a non-zero argument returns the amount of free space in the data memory. The second modification is the addition of a page zero location which specifies which bank PEEKs and POKEs will access. BASIC 1.5 will initialize this location to access bank 0. The Floating Point add, subtract, multiply, and divide math routines in this package are provided by:

DTACK GROUNDED
1415 E. MCFADDEN ST. F
SANTA ANA, CA 92705 USA

Even with these changes, the BASIC 1.5 language is 100% compatible with BASIC 1.0. BASIC 1.0 programs should run without modification on BASIC 1.5. Also, only libraries which access the data directly will require modification to run with BASIC 1.5. CIL is the only standard library which does this. IGL and VGL will work with BASIC 1.5 without modification.

Provided with BASIC 1.5 is a companion version of CIL.Z. This version is equivalent to the BASIC 1.0 version except that it has been modified to deal with separate program and data banks. It is still called the CIL Library so that programs which use the CIL won't have to be modified when running them under BASIC 1.5. The two CILs are not interchangeable, so you must make sure that you use the proper one for the version of BASIC you are using. The new CIL can determine if you are using BASIC 1.5, and give an error if you are not. However, the standard CIL that comes with BASIC 1.0 does not detect this. If you have a CIL.Z on a disk and you are not sure which BASIC it works with, link the CIL into BASIC 1.0. If it gives you an error, it works with BASIC 1.5. If it doesn't, then it works with

BASIC 1.0. In practice, your work disks should contain either BASIC 1.0 or BASIC 1.5, but not both. The CIL.Z on the work disk should be the one appropriate to the version of BASIC on the disk.

2. BASIC 1.5 STARTUP PROCEDURE

The full syntax of the startup command is as follows:

BASIC [[⟨top of memory⟩] [, [⟨terminal width⟩] [, ⟨top of data⟩]]]

where

top of memory = the first high memory location not to be used by BASIC expressed as a decimal number. Defaults to 48460 (BE00 hex).

terminal width = the number of printing characters that may be output before an automatic carriage return occurs. Defaults to 192.

top of data = the first high memory location in bank 3 not to be used for data. Defaults to the top of bank 3.

If the top of memory parameter is specified, both the current top of memory and the default top of memory will be set to the specified location. If not specified, both default to 48460 (BE00 hex). The difference between these two tops of memory is that the current top of memory will be moved down as library programs are linked into BASIC 1.5. The current top of memory will move back up to the default top of memory when the libraries are freed. At no time will BASIC 1.5 use any memory at or above the default top of memory, not even for library programs. This is the same as in BASIC 1.0.

The terminal width parameter controls how many printing characters may be output before an automatic carriage return occurs. Based on the terminal width, the last legal tab stop is determined. Since the Console device under CODOS does its own automatic carriage return, it isn't necessary to specify 80 as the terminal width parameter. The terminal width defaults to 192, which is the same length as the default CODOS input buffer. This parameter works the same as in BASIC 1.0.

If the top of data parameter is specified, then the top of memory in the data bank (bank 3) is set to the specified location. If not specified, it defaults to the top of bank 3 (FFFF:3 hex). BASIC 1.5 will not use any memory at or above this location in bank 3. This in no way affects the amount of memory available in the program bank.

During initialization, BASIC 1.5 will perform some extra steps not needed by BASIC 1.0. First, BASIC 1.5 will check to see how much RAM is present in bank 3. If no RAM is found BASIC 1.5 will indicate that it is unable to run and will return to CODOS. BASIC 1.5 will then determine how much contiguous memory is available in bank 3, up to the top of data. If no top of data was specified in the startup command, BASIC 1.5 will use as much memory as available in bank 3.

Once bank 3 has been properly "sized", BASIC 1.5 will check to see if a DATAMOVER 68000 is present in the system. If it is, then BASIC 1.5 will attempt to startup the 68000 processor. If this is also successful, BASIC 1.5 will modify itself to use the 68000 for the functions specified in the introduction. If either of the steps are unsuccessful, BASIC 1.5 will leave itself unpatched, thus performing all functions on the 6502. If BASIC 1.5 is able to make use of the 68000 processor, then it will add the line "68000 FLOATING POINT PACKAGE IS OPERATIONAL" to the startup message.

If BASIC 1.5 is using the 68000 processor, then the BYE command will halt the 68000 when leaving BASIC 1.5. If a warm entry back to BASIC 1.5 is performed (i.e. executing a CODOS "GO 0" command), then BASIC 1.5 will expect to be able to restart the 68000. If it is unsuccessful, then BASIC 1.5 will give a "68000 INIT ERROR" followed by the "READY." prompt. You should immediately execute a BYE command to return to CODOS, then cold restart BASIC 1.5.

3. USE OF BASIC 1.5 WITH PROGRAMMOVER BOARD

As shipped, BASIC 1.5 is optimized for use with the DATAMOVER board and its 68000 processor. If you are going to use BASIC 1.5 with the PROGRAMMOVER (Z-80) board, a few changes are required in order to guarantee proper startup on the PROGRAMMOVER. The first step is to address your PROGRAMMOVER board to occupy bank 3 in the 6502 address space. Next you should make sure that the control register for the PROGRAMMOVER board is addressed at $BFB7. (These settings are the standard configuration when PROGRAMMOVERS leave MTU.) Once this is done, you are ready to modify BASIC 1.5 for proper operation on the PROGRAMMOVER board. To make the needed modifications, you should perform the following sequence of commands on a work disk containing a copy of BASIC 1.5 (DO NOT do this procedure on your Distribution BASIC 1.5 disk!):

```
SET BFB7 04
GET BASIC                (note: be sure this is your BASIC 1.5)
SET 7000 04
GETLOC BASIC

BASIC.C =0A00   0A00   402F
                03A1   03F5
                0700   077D      printed by GETLOC command
                BFBE   BFBF
                1000:2 1BDD
                0200:3 02F8

SAVE MBASIC =0A00 0A00 402F 03A1 03F5 0700 077D 7000 =BFB7 7000 0200:3 02F8
```

Before executing the SAVE command, you should note if there are any differences between that shown above for the GETLOC command, and that which is printed on your display. If any of the numbers are different, you should check to see if that number is used in the SAVE command. If it is, you should modify the SAVE command to use the number shown on your display. Once you have saved the modified BASIC, try executing the command:

```
MBASIC
```

to see if it will run correctly. It should print the number of bytes free in the program and data bank, then display the "READY." prompt. Assuming this works okay, you may delete the old BASIC.C and rename MBASIC.C to just BASIC.C.

These modifications make sure that the PROGRAMMOVER board is properly reset, and discards the block of 68000 code which can't be used without the DATAMOVER board.

4. SYSTEMS WITH BOTH DATAMOVER AND PROGRAMMOVER BOARDS

In systems with both the DATAMOVER and PROGRAMMOVER boards, the natural choice is to use the DATAMOVER with BASIC 1.5. The distribution BASIC 1.5 will work in this situation provided the PROGRAMMOVER is disabled prior to starting up BASIC. If the PROGRAMMOVER board is not already disabled, you may disable it by pressing the RESET key, or executing the following command:

```
SET BFB7 0
```

At this point you may startup BASIC 1.5 as described in Section 2 of this manual.

5. ADDITIONAL ERROR MESSAGES

The following are additional error messages which are present in BASIC 1.5.

OUT OF PROGRAM MEMORY

BASIC 1.5 has run out of memory in the program bank. This means that the program is too large or the program is using too many string characters. (Remember that for strings, a length and pointer are stored in the data bank, and the actual characters of the string are stored in the program bank.)

OUT OF DATA MEMORY

BASIC 1.5 has run out of memory in the data bank. This will most likely be due to arrays which are too large.

OUT OF STACK MEMORY

BASIC 1.5 has run out of stack space on the 6502 processor. This can occur if there are too many nested FOR..NEXT loops or GOSUBs. Most likely, a FOR..NEXT loop or GOSUB has been left unterminated. An unterminated FOR..NEXT or GOSUB will gradually use up a little more of the 6502 stack each time it is executed, until it finally results in an OUT OF STACK MEMORY ERROR.

LINE NUMBER TABLE FULL

When the 68000 is used, BASIC 1.5 will maintain a table of line numbers, and where each line begins in memory. The size of this table allows for 2046 lines. The line numbers may be 0 to 65000, but a maximum total of 2046 lines are permitted. If the number of lines in your BASIC program exceed 2046, a LINE NUMBER TABLE FULL error will occur. You can reduce the number of lines in your program by putting multiple statements on a line. It is unlikely that you will ever encounter this error since you will more likely run out of program memory before reaching 2046 lines.

68000 INIT ERROR

BASIC 1.5 was unable to restart the 68000 processor upon warm entry (i.e. GO 0) back into BASIC. This can only happen if BASIC 1.5 was orginally successful in starting the 68000 and then patched itself to use it. If this error occurs, the only recourse is to return to CODOS with a "BYE" command and do a cold restart of BASIC 1.5 (i.e. execute the command "BASIC" again). If you execute something other than "BYE" after the error message, BASIC 1.5 will likely hang. To get out of this, you should press the RESET key to return to CODOS.

6. EXPECTED SPEED IMPROVEMENT USING BASIC 1.5

If BASIC 1.5 is run without making use of the 68000, BASIC programs will run slightly slower than with BASIC 1.0. This is because BASIC 1.5 has to do bank switching, which is slower than not doing bank switching. Except in FOR..NEXT loops which loop thousands of times, the speed difference will be unnoticeable. If BASIC 1.5 is able to use the 68000, then a certain amount of speed improvement can be expected.

The amount of speed improvement depends upon the program, but can range from unnoticeable to significant (i.e. 10 times faster). Most programs will probably show a speed improvement of less than 3 times as fast. The programs which will show the most improvement will be those which use the transcendental functions extensively, or programs which use numeric arrays very heavily.

7. THE FRE FUNCTION (8.5)

PURPOSE: To return the number of unused bytes available in either the program memory or the data memory.

SYNTAX: FRE(⟨bank⟩)

ARGUMENTS:

⟨bank⟩ = a numeric expression that selects which bank. Zero selects program bank. Non-zero selects data bank.

DISCUSSION:

The FRE function returns the number of unused bytes in either the program memory or the data memory. The argument selects which bank. If the argument is zero, then the FRE function returns the number of bytes available in the program bank (bank 0). If the argument is non-zero, then the FRE function returns the number of bytes available in the data bank (bank 3).

EXAMPLES:

```
10 PRINT FRE(0)
```

will print the number of bytes free in the program bank.

```
20 PRINT FRE(1)
```

will print the number of bytes free in the data bank.

8. THE POKE STATEMENT (6.15)

PURPOSE: To store a value at a specified memory location.

SYNTAX: POKE ⟨address⟩ , ⟨byte⟩

ARGUMENTS:

⟨address⟩ = a numeric expression which evaluates to the decimal address of the desired memory location.

⟨byte⟩ = a numeric expression which evaluates to the byte you wish to store. It must be a number from 0 to 255.

DISCUSSION:

The POKE statement is used to store a byte value at a specified memory location. If the address argument is not within the range 0 to 65535, or the byte argument is not within the range 0 to 255, an ILLEGAL QUANTITY ERROR will be given. Addresses in the range 0 to 255 (i.e. page zero) will always access bank 0. All other addresses will access the bank specified by the page zero location 146 (92 hex). The byte at 146 is initialized to 0 whenever cold or warm starting BASIC 1.5.

EXAMPLES:

```
10 POKE 146,1:POKE 49152,255
```

will set the byte at 146 to 1, then set the byte at 49152 ($C000) in bank 1 to 255. This should cause a short horizontal line to appear in the upper left corner of the display.

```
20 POKE 146,3:POKE 64000,0
```

will set the byte at 146 to 3, then set the byte at 64000 in bank 3 to 0.

NOTES:

1. Since all page zero addresses are forced to bank 0, you can't POKE into the first 256 bytes of the other banks.

9. THE PEEK FUNCTION (8.9)

PURPOSE: To return the value of the byte at a specified memory location.

SYNTAX: PEEK(⟨address⟩)

ARGUMENTS:

⟨address⟩ = a numeric expression which evaluates to the decimal address of the desired memory location.

DISCUSSION:

The PEEK function is used to obtain the value of a byte at a specified memory location. If the address argument is not within the range 0 to 65535 an ILLEGAL QUANTITY ERROR will be given. Addresses in the range 0 to 255 (i.e. page zero) will always access bank 0. All other addresses will access the bank specified by the page zero location 146 (92 hex). The byte at 146 is initialized to 0 whenever cold or warm starting BASIC 1.5.

EXAMPLES:

```
10 PRINT PEEK(146)
```

will print the value of the byte at location 146 in bank 0. This will display what bank will be accessed by PEEKs and POKEs for addresses outside page zero.

```
10 POKE 146,1:A = PEEK(49152)
```

will set the byte at 146 to 1, then set the variable A to the value of the byte at location 49152 in bank 1.

```
20 POKE 146,3:X = PEEK(768)
```

will set the byte at 146 to 3, and set the variable X to the value of the byte at location 768 in bank 3.

NOTES:

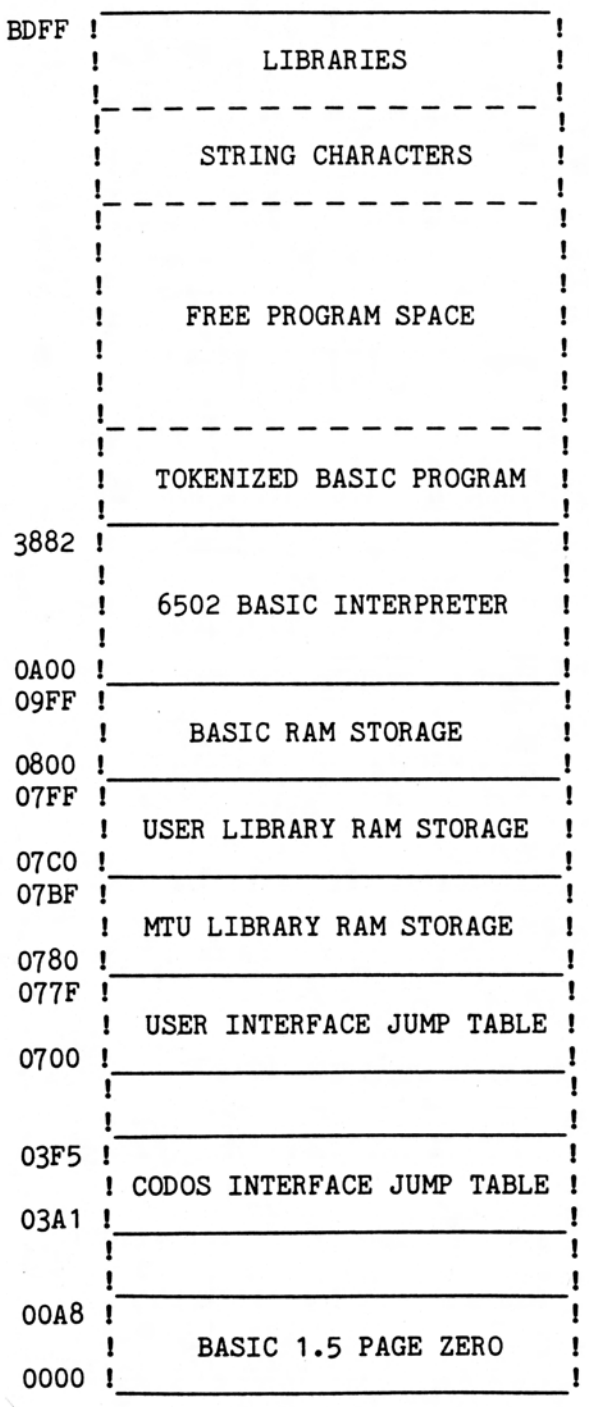
1. Since all page zero addresses are forced to bank 0, you can't PEEK the first 256 bytes of the other banks.

APPENDIX A

BASIC 1.5 MEMORY MAP

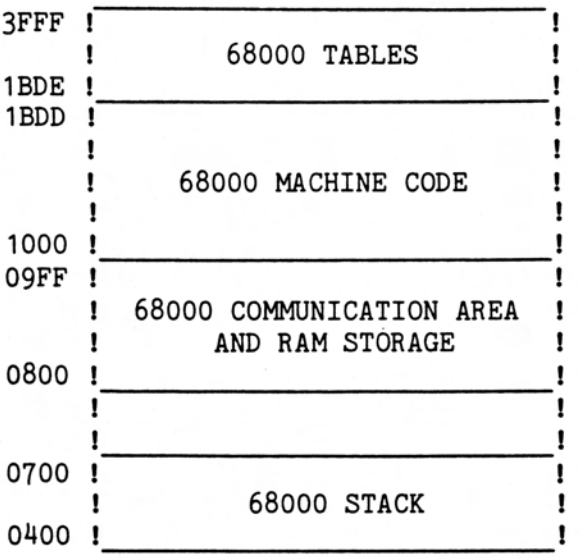
The following is the default memory map for BASIC 1.5.

BANK 0

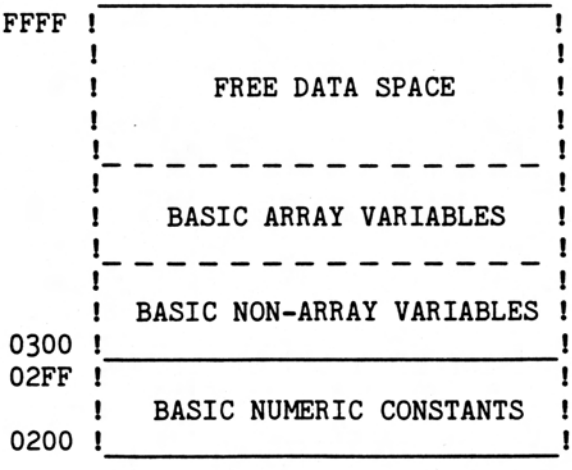


BANK 2

(If 68000 is present)



BANK 3



APPENDIX B

ADDITIONAL NOTES FOR WRITING LIBRARIES FOR BASIC 1.5

Since program and data space have been separated, the page zero pointers which identify where various things are found had to be modified. The following describes the complete set of page zero pointers which identify the beginning and end of the BASIC program, string characters, simple variables, and array variables.

LOCATION	NAME	DESCRIPTION
$2F-30	PRGTXT	This is a pointer to the beginning of the BASIC program in memory. This pointer refers to bank 0.
$35-36	PRGEND	This is a pointer to the first location after the end of the BASIC program. This also represents the bottom limit to which strings can go. This pointer refers to bank 0.
$37-38	STRBTM	This is a pointer to the current bottom of string storage space. The next string to be created will be placed just under this location. This pointer refers to bank 0.
$3B-3C	MEMTOP	This is a pointer to BASIC's top of program memory. This defines the top of string storage space. Loading Libraries will move this pointer down as necessary. This pointer refers to bank 0.
$31-32	SMPVAR	This is a pointer to the beginning of simple (i.e. non-array) variable storage space. Non-array variables are found beginning at this location. This pointer refers to bank 3.
$33-34	ARYVAR	This is a pointer to the beginning of array variable storage space. This also represents the end of simple variable storage space. This pointer refers to bank 3.
$06-07	ARYTOP	This is a pointer to the top of array storage space. This pointer refers to bank 3.
$08-09	DBKTOP	This is a pointer to the limit of memory in bank 3 which BASIC 1.5 may use. This pointer refers to bank 3.

The LIB command in BASIC 1.0 requires that a library program consist of a single block of object code that loads into bank 0. A library file may contain multiple copies of this program. The LIB command will load the first version in the file which doesn't conflict with libraries already loaded. The LIB command in BASIC 1.5 has been upgraded to allow a library program to have an additional block of object code which loads into a bank other than bank 0. The part of the library program which loads into bank 0 must still meet the same requirements as before, with one exception. The first byte in the Library Header should be 128+'M' instead of 128+'L' (see page 4 of Writing A BASIC Library Manual). This indicates an extra block of code is to be loaded into another bank. BASIC 1.5 can still load Libraries with 128+'L' in the header. In that case, only the bank 0 block will be loaded.

When the LIB command encounters one of these "extra block" libraries, it will scan the file for a bank 0 block which will load without conflict, ignoring any blocks which load in other banks. Once it has loaded the bank 0 block, the LIB command will search the remainder of the file, loading the first non-bank 0 block it finds. This means a non-bank 0 block may follow each bank 0 block, or one non-bank 0 block may be placed at the end of the file. If the LIB command fails to find a non-bank 0 block, it will give a LOAD ERROR.