AA–P601B–TV

Rainbow™ 100

MBASIC™-86

User's Guide

Developed by Microsoft Corporation

digital equipment corporation

First Printing, November 1982
Revised, June 1983

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

CP/M® is a registered trademark of Digital Research Inc. CP/M®–80 and CP/M®–86 are trademarks of Digital Research Inc.

MBASIC and Multiplan are trademarks of Microsoft Corporation.

The following are trademarks of Digital Equipment Corporation:

digital™

DEC	MASSBUS	UNIBUS
DECmate	PDP	VAX
DECsystem–10	P/OS	VMS
DECSYSTEM–20	Professional	VT
DECUS	Rainbow	Work Processor
DECwriter	RSTS	
DIBOL	RSX	

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

Printed in U.S.A.

Contents

Figures

Screens

Preface

Welcome to MBASIC-86

MBASIC-86 runs on DIGITAL's Rainbow 100 computer with the CP/M-86/80 operating system. It meets the requirements for the ANSI subset standard for BASIC, and supports many unique features rarely found in other BASIC languages. In addition, MBASIC-86 has sophisticated string handling and structured programming features that are especially suited for applications development.

To get started, we have prepared a set of easy-to-use documentation. In the back of these volumes is a card that welcomes your comments. Please let us hear from you.

Intended Reader

This guide is intended for the first-time user of MBASIC-86. The purposes of this guide are to show you how to get started with MBASIC-86 on the Rainbow 100 computer and to present an overview of MBASIC-86.

This guide assumes you know how to program in BASIC, but you may not be familiar with MBASIC-86. If you are not familiar with programming fundamentals, refer to the textbooks listed at the end of this Preface.

Before reading this guide, you should have read the *Rainbow 100 Getting Started* and *Rainbow 100 User's Guide.*

Guide Organization

MBASIC-86 documentation consists of two manuals: the *MBASIC-86 User's Guide,* which you are now reading, and the *MBASIC-86 Reference Manual.*

The *MBASIC-86 User's Guide* is designed to help you get started with MBASIC-86 and is organized as follows:

Chapter 1 tells you to make backup copies of the MBASIC-86 master diskette. It then shows you how to run the test program *MTEST,* to start MBASIC-86, and to use MBASIC-86 commands.

Chapter 2 discusses converting programs from other BASIC languages.

Chapter 3 discusses diskette I/O and contains programming examples.

Chapter 4 discusses assembly language subroutines.

Appendix A lists DIGITAL's International Help Line telephone numbers.

Appendix B shows you how to combine the CP/M-86/80 operating system with MBASIC-86 onto a single diskette.

Conventions Used

This guide uses the following conventions:

- In examples between you and the computer, what the computer displays on the screen is shown in black. The characters you type from the keyboard are shown in color.
- Input you enter when you run an MBASIC-86 program is shown in color.
- Be sure to type all spaces and punctuation marks exactly as they are printed.

- All command, statement, and function names are printed in uppercase characters.
- When you see <Return>, press the Return key on the keyboard.
- When you see <Ctrl/C>, hold the control key (Ctrl key on the keyboard) while you press the C key. Be sure to hold both keys down at the same time.
- Square brackets ([ ]) indicate optional user input.
- Braces ({ }) indicate that you have the choice between two or more entries. You must select at least one unless the entries are optional.
- Information followed by an ellipsis (. . .) can be repeated any number of times (up to the length of the line).

Learning More About MBASIC-86

This guide and the *MBASIC-86 Reference Manual* provide complete instructions for using MBASIC-86. However, no teaching material for MBASIC-86 programming has been supplied in these manuals. The *MBASIC-86 Reference Manual* is strictly a syntax and semantics reference for MBASIC-86.

If you are new to MBASIC-86 and need help in learning to program, there are a large number of texts available. The following books can be helpful.

1. Bent, Robert J. and George C. Sethares. *BASIC An Introduction to Computer Programming.* Brooks/Cole, 1982.
2. Dwyer, Thomas A. and Margot Critchfield. *BASIC and the Personal Computer.* Addison-Wesley, 1982.
3. Heiserman, David L. *Programming in BASIC for Personal Computers.* Prentice-Hall, 1981.
4. Hirsch, Seymour. *BASIC Programming Self-Taught.* Reston Publishing, 1980.
5. Kemeny, John G. and Thomas E. Kurtz. *BASIC Programming.* John Wiley, 1980.

1

Getting Started with MBASIC-86

Using MBASIC-86

Copying the MBASIC-86 Diskette

Before running MBASIC-86, protect your MBASIC-86 master diskette by making a copy of it. To do this refer to the COPY command discussion in the *Rainbow 100 User's Guide.*

MBASIC-86 Diskette Files

Your MBASIC-86 diskette contains these files:

MBASIC.CMD
MTEST.BAS

The file MBASIC.CMD is the program that interprets your MBASIC-86 commands, statements, and functions.

The file MTEST.BAS is the program that tests the operation of MBASIC-86.

Testing MBASIC-86

The first time you use MBASIC-86, run the test program MTEST. MTEST checks to see that MBASIC-86 is operating properly. If any error messages are displayed during the test, try running MTEST again. If the problems persist, contact your vendor or call DIGITAL's Help Line. See Appendix A for Help Line numbers.

To run MTEST complete the following procedure:

1. Insert the CP/M-86/80 working diskette into drive A. Close the door.
2. Insert the MBASIC-86 working diskette into drive B. Close the door.
3. Type:

   ```
   A
   ```

   The computer displays an introductory message followed by:

   ```
   A>
   ```

4. Type:

   ```
   B:<Return>
   ```

   The computer displays:

   ```
   B>
   ```

5. Type:

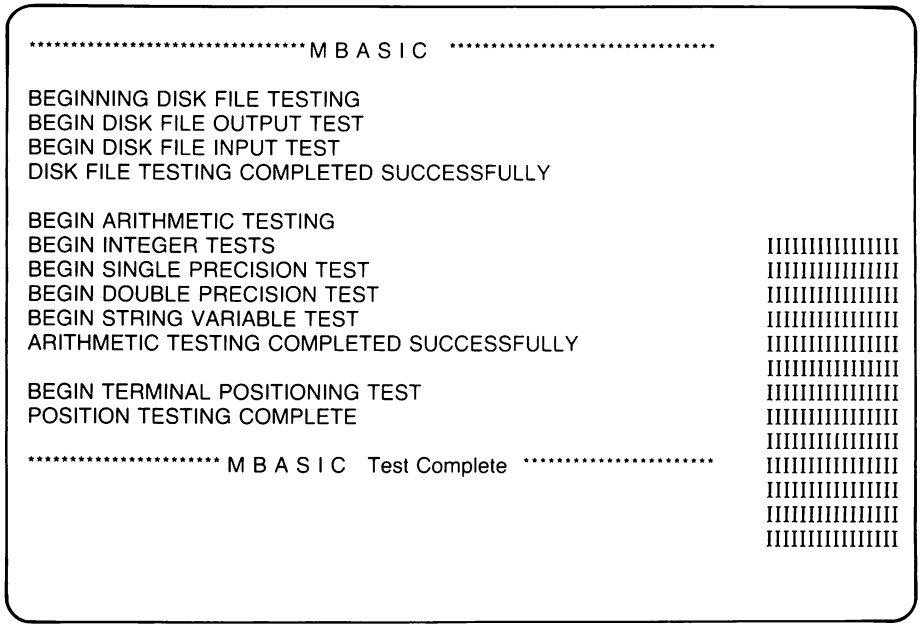
   ```
   MBASIC MTEST <Return>
   ```

MBASIC-86 flashes a message and then displays the messages shown in Screen 1. The test is completed in about two minutes and control is automatically returned to the CP/M-86/80 operating system.

Making a System/Application Diskette

Appendix B shows you how to combine the CP/M-86/80 operating system and MBASIC-86 onto one diskette which is called the system/application diskette.

The procedure is optional.



MR-S-2674-83

Screen 1. MTEST Display

Starting MBASIC-86

Use one of the following procedures to start MBASIC-86.

Using Working Diskettes. To run MBASIC-86 complete the following procedure:

1. Start or reset the Rainbow 100 computer.
2. Insert the CP/M-86/80 working diskette into drive A. Close the door.
3. Insert the MBASIC-86 working diskette into drive B. Close the door.
4. Type:

   ```
   A
   ```

   The computer displays:

   ```
   A>
   ```

5. Type:

   ```
   B:<Return>
   ```

   The computer displays:

   ```
   B>
   ```

6. Type:

   ```
   MBASIC<Return>
   ```

MBASIC-86 displays its start-up message. See Screen 2.

> **NOTE**
> The text displayed in Screen 2 may vary slightly from that displayed on your Rainbow 100 screen.

Using a System/Application Diskette. If you are using a system/application diskette, complete the following procedure.

1. Start or reset the Rainbow 100 computer.
2. Insert the system/application diskette into drive A. Close the door.
3. Insert a blank, formatted diskette into drive B. Close the door.
4. Type:

   ```
   A
   ```

   The computer displays:

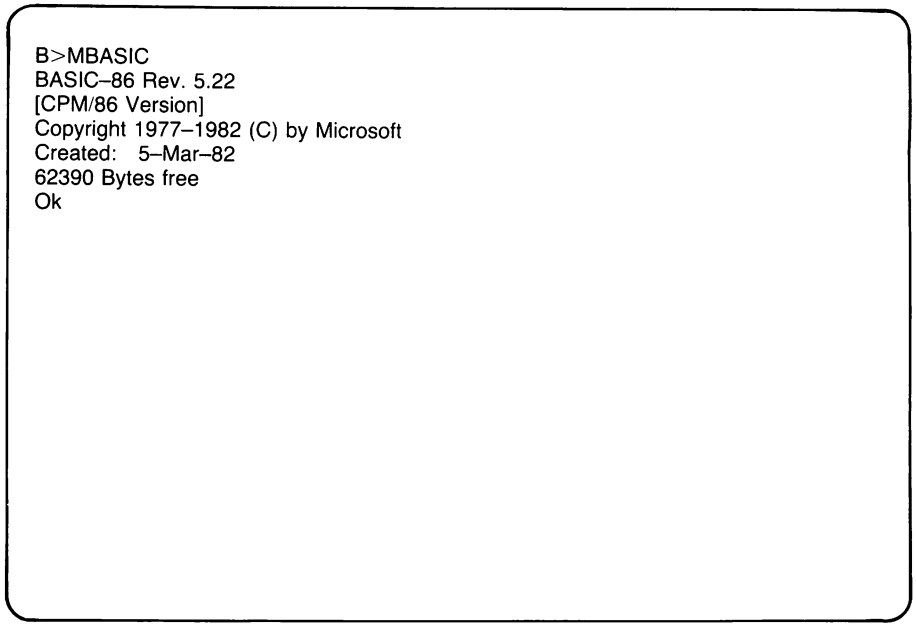
   ```
   A>
   ```

5. Type:

   ```
   MBASIC<Return>
   ```

   MBASIC-86 displays its start-up message. See Screen 2.

> **NOTE**
> The text displayed in Screen 2 may vary slightly from that displayed on your Rainbow 100 screen.



MR-S-2675-83

Screen 2. MBASIC-86 Start-up Display

Leaving MBASIC-86

To return to the CP/M-86/80 operating system, use the SYSTEM command. SYSTEM closes all files and then returns to the CP/M-86/80 operating system. If you type Ctrl/C, you are returned to MBASIC-86, not to the CP/M-86/80 operating system.

Entering MBASIC-86 Statements

Type the MBASIC-86 program statement and press the Return key. To continue a statement to the next line (but not assigning a new line number), press the line feed (LF) key.

IMPORTANT
Some versions of BASIC allow you to leave out spaces delimiting commands. MBASIC-86 does not allow this. For example, the statement that follows is illegal in terms of MBASIC-86 syntax:

```
10 FORT=1TO90:PRINTTAB(12):IFT<50PRINT"<50";:NEXTT
```

In MBASIC-86, the statement must be phrased in the following way:

```
10 FOR T=1 TO 90:PRINT TAB(12):IF T<50 THEN PRINT "<50";: NEXT T
```

In addition, when you type a GOTO or GOSUB, the line number must be preceded by a space, as in the statement:

```
20 GOTO 500
```

Using the MBASIC Command

The format of the command line is:

A>MBASIC ["filename"][/F:number of files]
[/M:highest memory location][/S:maximum record size]

If filename is present, MBASIC-86 proceeds as if a RUN filename command were typed. A default type of .BAS is used if no extension is supplied and the filename is less than nine characters long. This allows MBASIC-86 programs to be executed in batch mode using the SUBMIT facility of the CP/M-86/80 operating system. Such programs should include a SYSTEM statement to return to the CP/M-86/80 operating system when they have finished, allowing the next program in the batch stream to execute.

If /F:number of files is present, it sets the number of diskette data files that may be open at any one time during the execution of a MBASIC-86 program. Each file data block allocated in this fashion requires 166 bytes of memory. If the /F option is omitted, the number of files defaults to three.

The /M:highest memory location option sets the highest memory location that will be used by MBASIC-86. In some cases it is desirable to set the amount of memory well below the CP/M-86/80's operating system BDOS to reserve space for assembly language subroutines. In all cases, the highest memory location should be below the start of BDOS (whose address is contained in locations 6 and 7). If the /M:option is omitted, all memory up to the start of BDOS is used.

/S:maximum record size can be added at the end of the command line to set the maximum record size for use with random files. The default record size is 128 bytes.

NOTE
Number of files, highest memory location and maximum record size are numbers that may be either decimal, octal (preceded by &O) or hexadecimal (preceded by &H).

Examples:

`A>MBASIC PAYROLL.BAS`	Use all memory and three files, load and execute PAYROLL.BAS.
`A>MBASIC INVENT/F:6`	Use all memory and six files, load and execute INVENT.BAS.
`A>MBASIC /M:32768`	Use first 32K bytes of memory and three files.
`A>MBASIC DATACK/F:2/M:&H9000`	Use first 36K bytes of memory, two files, and execute DATACK.BAS.

Diskette Files

Diskette filenames follow the normal CP/M-86/80 operating system naming conventions. All filenames can include the drive name as the first two characters to specify a diskette drive; otherwise the currently selected drive is assumed. A default type of .BAS is used on LOAD, SAVE, MERGE and RUN "filename" commands if no "." appears in the filename and the filename is less than nine characters long.

MBASIC-86 supports large random files. The maximum logical record number is 32767.

Displaying File Names

To display the names of files stored on the current diskette, use the FILES command.

Examples:

```
B>FILES                    Display all file names.
B>FILES "*.BAS"            Display all file with the type .BAS.
```

Closing Files

Before you remove the diskette, use the RESET command to close all diskette files and write the directory information to the diskette.

2

Converting Programs to MBASIC-86

If you have programs written in a BASIC other than MBASIC-86, some minor adjustments may be necessary before running them with MBASIC-86. Here are some specific things to look for when converting BASIC programs.

String Dimensions

Delete all statements that are used to declare the length of strings. A statement such as DIM A$(I,J), which dimensions a string array for J elements of length I, should be converted to the MBASIC-86 statement DIM A$(J).

Some BASIC languages use a comma or ampersand for string concatenation. Each of these must be changed to a plus sign, which is the operator for MBASIC-86 string concatenation.

In MBASIC-86, the MID$, RIGHT$, and LEFT$ functions are used to take substrings of strings. Forms such as A$(I) to access the Ith character in A$, or

A$(I,J) to take a substring of A$ from position I to position J, must be changed as follows:

Other BASIC	*MBASIC-86*
`X$=A$(I)`	`X$=MID$(A$,I,1)`
`X$=A$(I,J)`	`X$=MID$(A$,I,J-I+1)`

If the substring reference is on the left side of an assignment and X$ is used to replace characters in A$, convert as follows:

Other BASIC	*MBASIC-86*
`A$(I)=X$`	`MID$(A$,I,1)=X$`
`A$(I,J) = X$`	`MID$(A$,I,J-I+1)=X$`

Multiple Assignments

Some BASIC languages allow statements of the form:

```
10 LET B=C=0
```

to set B and C equal to zero. MBASIC-86 interprets the second equal sign as a logical operator and sets B equal to −1 if C equaled 0. Convert this statement to two assignment statements:

```
10 C=0:B=0
```

Multiple Statements

Some BASIC languages use a backslash (\) to separate multiple statements on a line. In MBASIC-86 all statements on a line are separated by a colon (:). You can not use multiple statements with an OPEN statement that opens a file.

MAT Functions

Programs using the MAT functions available in some BASIC languages must be rewritten using FOR...NEXT loops to execute properly.

3

MBASIC-86 Diskette I/O

This chapter discusses diskette I/O (Input/Output) procedures for the beginning MBASIC-86 user. Wherever a filename is required in a diskette command or statement, use a name that conforms to the CP/M-86/80 operating system's requirements for filenames. Use the form filename.typ where filename is nine or less alphanumeric characters and .typ is a file type consisting of three or less alphanumeric characters.

Program File Commands

Here is a review of the commands and statements used in program file manipulation.

SAVE "filename"[,A]	Writes to the diskette the program that is currently stored in memory. The A option writes the program as a series of ASCII characters. Otherwise, MBASIC-86 uses a compressed binary format.

LOAD "filename"[,R]	Loads the program from the diskette into memory. The R option runs the program immediately. LOAD always deletes the current contents of memory and closes all files before LOADing. If R is included, however, open data files are kept open. Thus programs can be chained or loaded in sections and access the same data files. LOAD "filename",R and RUN "filename",R are equivalent.
RUN "filename"[,R]	RUN "filename" loads the program from the diskette into memory and runs it. RUN deletes the current contents of memory and closes all files before loading the program. If the R option is included, however, all open data files are kept open. RUN "filename",R and LOAD "filename",R are equivalent commands.
MERGE "filename"	Loads the program from the diskette into memory but does not delete the current contents of memory. The program line numbers on the diskette are merged with the line numbers in memory. If two lines have the same number, only the line from the diskette program is saved. After a MERGE command, the "merged" program is stored in memory, and MBASIC-86 returns to command level.
KILL "filename"	Deletes the file from the diskette. "filename" can be a program file, or a sequential or random access data file.
NAME "old filename" AS "new filename"	To change the name of a diskette file, use the NAME command. NAME can be used with program files, random files, or sequential files.

Protected File

If you wish to save a program in an encoded binary format, use the Protect option with the SAVE command. For example:

```
SAVE "MYPROG",P
```

A program saved this way cannot be listed or edited. You can also save an unprotected copy of the program for listing and editing purposes.

Diskette Data Files – Sequential and Random I/O

There are two types of diskette data files that can be created and accessed by a MBASIC-86 program: sequential files and random access files.

Sequential Files

Sequential files are easier to create than random files but are limited in flexibility and speed when it comes to accessing the data. The data that is written to a sequential file is a series of ASCII characters stored, one item after another (sequentially), in the order it is sent and is read back in the same way.

The statements and functions that are used with sequential files are:

OPEN	PRINT#	INPUT#	WRITE
CLOSE	PRINT# USING	LINE INPUT#	EOF
LOC			

The following program steps are reqred to create a sequential file and access the data in the file:

1. OPEN the file in "O" mode. OPEN "O",#1,"DATA"

2.	Write data to the file using the PRINT# statement. (WRITE# can be used instead.)	PRINT#1,A$;B$;C$
3.	To access the data in the file, you must CLOSE the file and reOPEN it in "I" mode.	CLOSE #1 OPEN "I",#1,"DATA"
4.	Use the INPUT# statement to read data from the sequential file into the program.	INPUT#1,X$,Y$,Z$

Program 1 is a short program that creates a sequential file, "DATA", from information you type at the terminal.

Program 1. Create a Sequential Data File

```
10 OPEN "O",#1,"DATA"
20 INPUT "NAME";N$
25 IF N$="DONE" THEN END
30 INPUT "DEPARTMENT";D$
40 INPUT "DATE HIRED";H$
50 PRINT#1,N$;",";D$;",";H$
60 PRINT:GOTO 20
RUN
NAME? MICKEY MOUSE
DEPARTMENT? AUDIO/VISUAL AIDS
DATE HIRED? 01/12/72

NAME? SHERLOCK HOLMES
DEPARTMENT? RESEARCH
DATE HIRED? 12/03/65

NAME? EBENEEZER SCROOGE
DEPARTMENT? ACCOUNTING
DATE HIRED? 04/27/78

NAME? SUPER MANN
DEPARTMENT? MAINTENANCE
DATE HIRED? 08/16/78
```

```
NAME? ETC.
```

Now look at Program 2. It accesses the file "DATA" that was created in Program 1 and displays the name of everyone hired in 1978.

Program 2. Accessing a Sequential File

```
10 OPEN "I",#1,"DATA"
20 INPUT#1,N$,D$,H$
30 IF RIGHT$(H$,2) = "78" THEN PRINT N$
40 GOTO 20
RUN
EBENEEZER SCROOGE
SUPER MANN
Input past end in 20
Ok
```

Program 2 reads, sequentially, every item in the file. When all the data has been read, line 20 causes an "Input past end" error. To avoid getting this error, insert line 15 which uses the EOF function to test for end-of-file:

```
15 IF EOF(1) THEN END
```

and change line 40 to GOTO 15.

A program that creates a sequential file can also write formatted data to the disk with the PRINT# USING statement. For example, the statement

```
PRINT#A1,USING"####.##,";A,B,C,D
```

can be used to write numeric data to diskette without explicit delimiters. The comma at the end of the format string serves to separate the items in the diskette file.

The LOC function, when used with a sequential file, returns the number of sectors that have been written to or read from the file since it was OPENed. A sector is a 128-byte block of data.

Adding Data To A Sequential File. If you have a sequential file stored on a diskette and later want to add more data to the end of it, you cannot simply open the file in "O" mode and start writing data. As soon as you open a sequential file in "O" mode, you destroy its current contents. The following procedure can be used to add data to an existing file called "NAMES".

1. OPEN "NAMES" in "I" mode.
2. OPEN a second file called "COPY" in "O" mode.
3. Read in the data in "NAMES" and write it to "COPY".
4. CLOSE "NAMES" and KILL it.
5. Write the new information to "COPY".
6. Rename "COPY" as "NAMES" and CLOSE.
7. Now there is a file on disk called "NAMES" that includes all the previous data plus the new data you just added.

Program 3 illustrates this technique. It can be used to create or add onto a file called NAMES. This program also illustrates the use of LINE INPUT# to read strings with embedded commas from the disk file. Remember, LINE INPUT# reads in characters from the disk until it sees a carriage return (it does not stop at quotes or commas) or until it reads 255 characters.

Program 3. Adding Data to a Sequential File

```
10 ON ERROR GOTO 2000
20 OPEN "I",#1, "NAMES"
30 REM IF FILE EXISTS< WRITE IT TO "COPY"
40 OPEN "O",#2,"COPY"
50 IF EOF(1) THEN 90
60 LINE INPUT#1,A$
70 PRINT#2,A$
80 GOTO 50
90 CLOSE #1
100 KILL "NAMES"
110 REM ADD NEW ENTRIES TO FILE
120 INPUT "NAME";N$
130 IF N$="" THEN 200 'CARRIAGE RETURN EXISTS INPUT LOOP
140 LINE INPUT "ADDRESS? ";A$
150 LINE INPUT "BIRTHDAY ";B$
```

```
160 PRINT#2,N$
170 PRINT#2,A$
180 PRINT#2,B$
190 PRINT:GOTO 120
200 CLOSE
205 REM CHANGE FILENAME BACK TO "NAMES"
210 NAME "COPY" AS "NAMES"
2000 IF ERR=53 AND ERL=20 THEN OPEN "O",#2,"COPY":RESUME 120
2010 ON ERROR GOTO 0
```

The error trapping routine in line 2000 traps a "File does not exist" error in line 20. If this happens, the statements that copy the file are skipped, and "COPY" is created as if it were a new file.

Random Files

Creating and accessing random files requires more program steps than sequential files, but there are advantages to using random files. One advantage is that random files require less room on the diskette, because MBASIC-86 stores them in a packed binary format. A sequential file is stored as a series of ASCII characters.

The biggest advantage to random files is that data can be accessed randomly, that is, anywhere on the diskette—it is not necessary to read through all the information, as with sequential files. This is possible because the information is stored and accessed in distinct units called records and each record is numbered.

The statements and functions that are used with random files are:

OPEN	FIELD	LSET/RSET	GET
PUT	CLOSE	LOC	
MKI$	CVI	MKS$	CVS
MKD$	CVD		

Creating A Random File. The following program steps are required to create a random file.

1. OPEN the file for random access ("R" mode). This example specifies a record length of 32 bytes. If the record length is omitted, the default is 128 bytes.	`OPEN "R",#1,"FILE",32`
2. Use the FIELD statement to allocate space in the random buffer for the variables that are written to the random file.	`FIELD #1, 20 AS N$,` `4 AS A$, 8 AS P$`

3. Use LSET to move the data into the random buffer. Numeric values must be made into strings when placed in the buffer. To do this, use the "make" functions: MKI$ to make an integer value into a string, MKS$ for a single precision value, and MKD$ for a double precision value.	`LSET N$ = X$`
	`LSET A$ = MKS$(AMT)`
	`LSET P$ = TEL$`

4. Write the data from the buffer to the diskette using the PUT statement.	`PUT #1,CODE%`

Program 4 takes information that is typed at the keyboard and writes it to a random file. Each time the PUT statement is executed, a record is written to the file. The two-digit code that is input in line 30 becomes the record number.

> **NOTE**
> Do not use a FIELDed string variable in an INPUT or LET statement. This causes the pointer for that variable to point into string space instead of the random file buffer.

Program 4. Create a Random File

```
10 OPEN "R",#1,"FILE",32
20 FIELD #1,20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "2-DIGIT CODE";CODE%
40 INPUT "NAME";X$
50 INPUT "AMOUNT";AMT
60 INPUT "PHONE";TEL$:PRINT
70 LSET N$=X$
80 LSET A$=MKS$(AMT)
90 LSET P$=TEL$
100 PUT #1,CODE%
110 GOTO 30
```

Accessing A Random File. The following program steps are required to access a random file:

1. OPEN the file in "R" mode.	OPEN "R",#1,"FILE",32
2. Use the FIELD statement to allocate space in the random buffer for the variables that will be read from the file.	FIELD #1,20 AS N$, 4 AS A$, 8 AS P$
3. Use the GET statement to move the desired record into the random buffer.	GET #1,CODE%
4. The data in the buffer can now be accessed by the program. Numeric values must be converted back to numbers using the "convert" functions: CVI for integers, CVS for single precision values, and CVD for double precision values.	PRINT N$ PRINT CVS(A$)

Program 5 accesses the random file "FILE" that was created in Program 4. By inputting the two-digit code at the keyboard, the information associated with that code is read from the file and displayed on the screen.

Program 5. Access a Random File

```
10 OPEN "R",#1,"FILE",32
20 FIELD #1, 20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "2 DIGIT CODE";CODE%
40 GET #1, CODE%
50 PRINT N$
60 PRINT USING "$$$$$.$$";CVS(A$)
70 PRINT P$:PRINT
80 GOTO 30
```

With random files, the LOC function returns the record number just read or written from a GET or PUT statement. For example, the statement:

```
IF LOC(1)>50 THEN END
```

ends the program execution if the current record number in file #1 is greater than 50.

Program 6 is an inventory program that illustrates random file access. In this program, the record number is used as the part number, and it is assumed the inventory contains no more than 100 different part numbers. Lines 900–960 initialize the data file by writing CHR#(255) as the first character of each record. This is used later (line 270 and line 500) to determine whether an entry already exists for that part number.

Line 130–220 display the different inventory functions that the program performs. When you type in the desired function number, line 230 branches to the appropriate subroutine.

Remember to press the line feed (LF) key to continue long lines. For example, when you type line 225, press the line feed (LF) key after typing THEN PRINT.

Program 6. Inventory

```
120 OPEN "R",1,"INVEN.DAT",39
125 FIELD#1,1 AS F$,30 AS D$, 2 AS Q$,2 AS R$,4 AS P$
130 PRINT:PRINT "FUNCTION:":PRINT
```

```
135 PRINT 1,"INITIALIZE FILE"
140 PRINT 2,"CREATE A NEW ENTRY"
150 PRINT 3,"DISPLAY INVENTORY FOR ONE PART"
160 PRINT 4,"ADD TO STOCK"
170 PRINT 5,"SUBTRACT FROM STOCK"
180 PRINT 6,"DISPLAY ALL ITEMS BELOW REORDER LEVEL"
220 PRINT:PRINT:INPUT"FUNCTION";FUNCTION
225 IF (FUNCTION<1)OR(FUNCTION>6) THEN PRINT
          "BAD FUNCTION NUMBER":GOTO 130
230 ON FUNCTION GOSUB 900,250,390,480,560,680
240 GOTO 220
250 REM BUILD ENTRY
260 GOSUB 840
270 IF ASC(F$)<>255 THEN INPUT"OVERWRITE";A$:
          IF A$<>"Y" THEN RETURN
280 LSET F$=CHR$(0)
290 INPUT "DESCRIPTION";DESC$
300 LSET D$=DESC$
310 INPUT "QUANTITY IN STOCK";Q%
320 LSET Q$=MKI$(Q%)
330 INPUT "REORDER LEVEL";R%
340 LSET R$=MKI$(R%)
350 INPUT "UNIT PRICE";P
360 LSET P$=MKS$(P)
370 PUT#1,PART%
380 RETURN
390 REM DISPLAY ENTRY
400 GOSUB 840
410 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
420 PRINT USING "PART NUMBER ###";PART%
430 PRINT D$
440 PRINT USING "QUANTITY ON HAND #####";CVI(Q$)
450 PRINT USING "REORDER LEVEL #####";CVI(R$)
460 PRINT USING "UNIT PRICE $$###.##";CVS(P$)
470 RETURN
480 REM ADD TO STOCK
490 GOSUB 840
500 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
510 PRINT D$:INPUT "QUANTITY TO ADD ";A%
520 Q%=CVI(Q$)+A%
530 LSET Q$=MKI$(Q%)
540 PUT#1,PART%
```

```
550 RETURN
560 REM REMOVE FROM STOCK
570 GOSUB 840
580 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
590 PRINT D$
600 INPUT "QUANTITY TO SUBTRACT";S%
610 Q%=CVI(Q$)
620 IF (Q%-S%)<0 THEN PRINT "ONLY";Q%;" IN STOCK":GOTO 600
630 Q%=Q%-S%
640 IF Q%=<CVI(R$) THEN PRINT "QUANTITY NOW";Q%;
          " REORDER LEVEL";CVI(R$)
650 LSET Q$=MKI$(Q%)
660 PUT#1,PART%
670 RETURN
680 REM DISPLAY ITEMS BELOW REORDER LEVEL
690 FOR I=1 TO 100
710 GET #1,I
720 IF (CVI(Q$)<CVI(R$))AND(ASC(F$)<>255)
          THEN PRINT D$; "QUANTITY"; CVI(Q$);
          TAB(50);"REORDER LEVEL";CVI(R$)
730 NEXT I
740 RETURN
840 INPUT "PART NUMBER";PART%
850 IF(PART%<1)OR(PART%>100) THEN PRINT "BAD PART NUMBER":
          GO TO 840 ELSE GET #1, PART%: RETURN
890 END
900 REM INITIALIZE FILE
910 INPUT "ARE YOU SURE";B$:IF B$<>"Y" THEN RETURN
920 LSET F$=CHR$(255)
930 FOR I=1 TO 100
940 PUT#1,I
950 NEXT I
960 RETURN
```

4

MBASIC-86 Assembly Language Subroutines

MBASIC-86 uses the USR function and the CALL statement to interface with assembly language subroutines.

The USR function allows assembly language subroutines to be called in the same way MBASIC-86 intrinsic functions are called.

Memory Allocation

Important

Memory space must be set aside for an assembly language subroutine before it can be loaded. Enter the highest memory location minus the amount of memory needed for the assembly language subroutine(s) with the /M: switch. Refer to Chapter 1 for a description of the M option of the MBASIC command.

In addition to the MBASIC-86 interpreter code area, MBASIC-86 uses up to 64K bytes of memory beginning at its data segment (DS).

If, when an assembly language subroutine is called, more stack space is needed, MBASIC-86's stack can be saved and a new stack set up for use by the assem-

bly language subroutine. MBASIC-86's stack must be restored, however, before returning from the subroutine.

The assembly language subroutine can be loaded into memory by means of the CP/M-86/80 operating system or the MBASIC-86 POKE statement.

Using the Call Statement

The CALL statement is the recommended way of interfacing 8086 machine language programs with MBASIC-86. You should not use the old style user call (x = USR(n)).

Format: CALL variable name [(argument list)]

Where: variable name contains the address that is the starting point in memory of the subroutine being CALLed.

argument list contains the variables or constants, separated by commas, that are to be passed to the routine.

Invoking the CALL statement causes the following to occur:

1. For each parameter in the argument list, the two byte offset of the parameter's location within the Data Segment [DS] is pushed onto the stack.
2. MBASIC-86's return address Code Segment [CS], and offset [IP] are pushed onto the stack.
3. Control is transferred to the user's routine via a long call to the segment address given in the last DEF SEG statement, and offset given in variable name.

These actions are illustrated in Figures 1 and 2.

The user's routine now has control. Parameters can be referenced by moving the Stack Pointer [SP] to the Base Pointer [BP] and adding a positive offset to [BP].

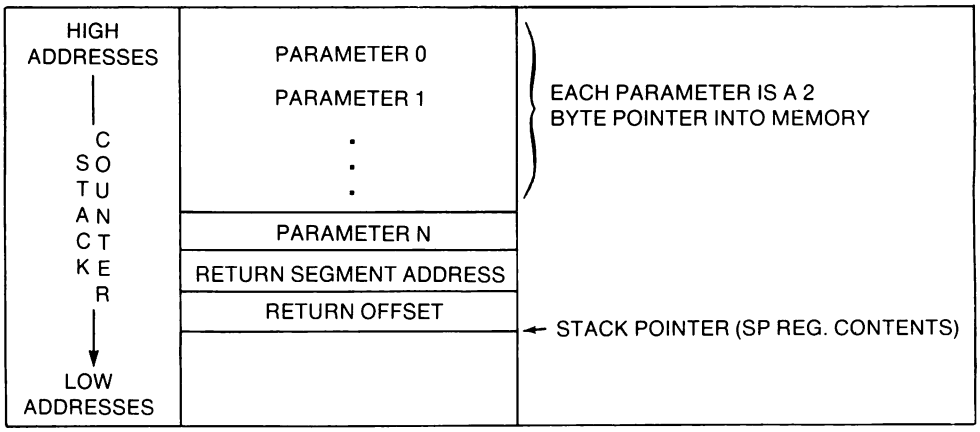


Figure 1. Stack Layout When Call Statement is Activated

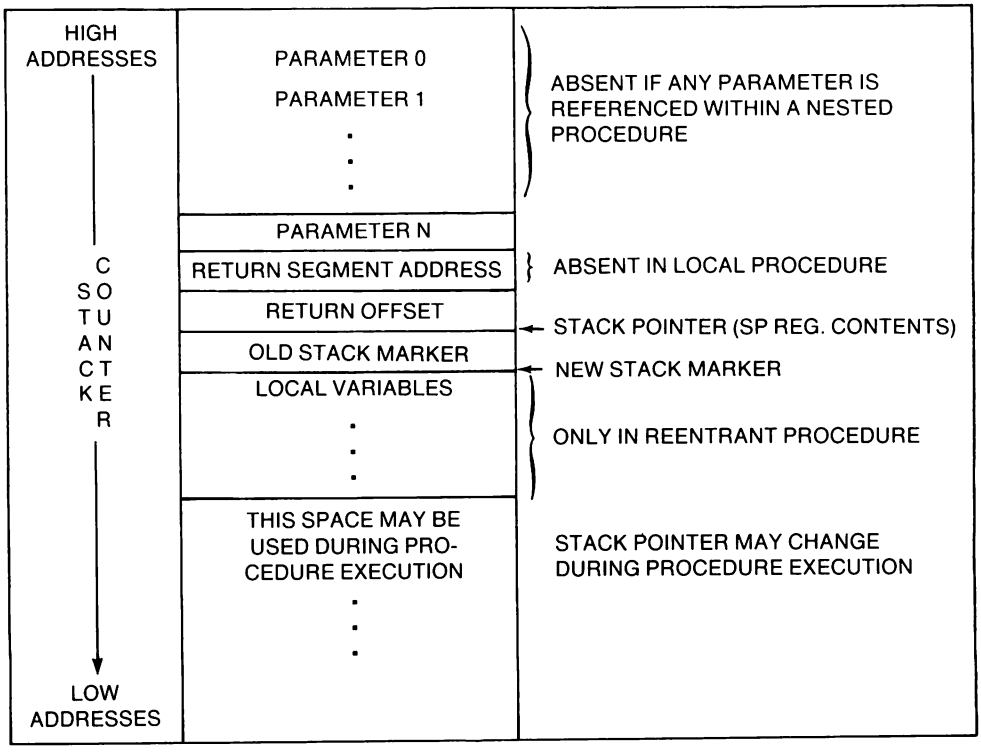


Figure 2. Stack Layout During Execution of a Call Statement

Observe the following rules when coding a subroutine:

1. The CALLed routine can destroy the AX, BX, CX, DX, SI, DI, and BP registers.
2. The CALLed program MUST know the number and length of the parameters passed. References to parameters are positive offsets added to [BP] (assuming the called routine moved the current stack pointer into BP; i.e., MOV BP,SP). That is, the location of p1 is at 8[BP], p2 is at 6[BP], p3 is at 4[BP],...etc.
3. The CALLed routine must do a RET n (where n is two times the number of parameters in the argument list) to adjust the stack to the start of the calling sequence.
4. Values are returned to MBASIC-86 by including in the argument list the variable name that receives the result.
5. If the argument is a string, the parameter's offset points to 3 bytes called the "string descriptor." Byte 0 of the string descriptor contains the length of the string (0 to 255). Bytes 1 and 2, respectively, are the lower and upper 8 bits of the string starting address in string space.

   NOTE
   If the argument is a string literal in the program, the string descriptor points to program text. Be careful not to alter or destroy your program this way. To avoid unpredictable results, add + "" to the string literal in the program. Example:

   ```
   20 A$ = "BASIC"+""
   ```

   This forces the string literal to be copied into string space. Now the string can be modified without affecting the program.

6. Strings can be altered by user routines, but the length *MUST NOT* be changed. MBASIC-86 cannot correctly manipulate strings if their lengths are modified by external routines.

Example:

```
100 DEF SEG=&H8000
110 FOO=0
120 CALL FOO(A,B$,C)
          .
          .
          .
```

Line 100 sets the segment to 8000 Hex. The value of FOO is added into the address as the low word after the DEF SEG value is left shifted 8 bits. Here, FOO is set to zero, so that the call to FOO executes the subroutine at location 8000H.

The following sequence of assembly language demonstrates access of the parameters passed and storing a return result in the variable 'C'.

```
MOV     BP,SP           ;Get current Stack posn in BP.
MOV     BX,6[BP]        ;Get address of B$ descriptor.
MOV     CL,[BX]         ;Get length of B$ in CL.
MOV     DX,1[BX]        ;Get addr of B$ text in DX.
              .
              .
              .
MOV     SI,8[BP]        ;Get address of 'A' in SI.
MOV     DI,4[BP]        ;Get pointer to 'C' in DI
MOVS    WORD            ;Store variable 'A' in 'C'.
RET     6               ;Restore Stack, return.
```

NOTE

The called program must know the variable type for numeric parameters passed. In the above example, the instruction

```
MOVS WORD
```

copies only 2 bytes. This works if variables A and C are integers. If variables A and C are single precision, 4 bytes must be copied and 8 bytes if double precision.

Using USR Function Calls

The format of the USR function call is:

x = USR[digit](argument)

where: digit is from 0 to 9. digit specifies which USR routine is being called. See the section of the *MBASIC-86 Reference Manual* that discusses the DEF USR statement. If digit is omitted, USR0 is assumed.

argument is any numeric or string expression.

x is the variable receiving the result of the function call. Its type (numeric or string) must be consistent with the argument passed, or can be set to Integer by calling MAKINT in the user's routine before returning to MBASIC-86.

A DEF SEG statement *MUST* be executed prior to a USR call to assure that the Code Segment points to the subroutine being called. The segment given in the DEF SEG statement determines the starting segment of the subroutine.

For each USR function used, a corresponding DEF USR statement must be executed to define the USR call offset. The address given in the DEF USR statement determines the starting address of the subroutine.

When the USR function call is made, register [AL] contains a value that specifies the type of argument that is given. The value in [AL] can be one of the following:

2 Two-byte integer (two's complement)

3 String

4 Single precision floating point number

8 Double precision floating point number

If the argument is a number, the [BX] register pair points to the Floating Point Accumulator (FAC) where the argument is stored and:

FAC	Is the exponent minus 128, and the binary point is to the left of the most significant bit of the mantissa.
FAC-1	Contains the highest 7 bits of mantissa with leading 1 suppressed (implied). Bit 7 is the sign of the number (0 = positive, 1 = negative).

If the argument is an integer:

FAC-2	Contains the upper 8 bits of the argument.
FAC-3	Contains the lower 8 bits of the argument.

If the argument is a single precision floating point number:

FAC-2	Contains the middle 8 bits of mantissa.
FAC-3	Contains the lowest 8 bits of mantissa.

If the argument is a double precision floating point number:

FAC-7—FAC-4	Contain four more bytes of mantissa (FAC-7 contains the lowest 8 bits).

If the argument is a string, the [DX] register pair points to 3 bytes called the "string descriptor." Byte 0 of the string descriptor contains the length of the string (0 to 255). Bytes 1 and 2, respectively, are the lower and upper 8 bits of the string starting address in MBASIC-86's Data Segment.

NOTE

If the argument is a string literal in the program, the string descriptor will point to program text. Be careful not to alter or destroy your program this way. See the CALL statement above. Usually, the value returned by a USR function is the same type (integer, string, single precision or double precision) as the argument that was passed to it.

Example:

```
110 DEF USR0=&H8000 'Assumes user give/M:32767'
120 X=5 'Note that X is single precision'
130 Y = USR0 (X)
140 PRINT Y
```

Load the following assembly language routine to multiply the argument passed by 2 and return an integer result.

Always be sure that your programs are defined by a PROC FAR statement.

```
DOUBLE          SEGMENT

                ASSUME   CS:DOUBLE

FRCICNTOFFSET   EQU      103H
MAKINTOFFSET    EQU      107H

FRCINT          LABEL    DWORD
                DW       FRCINTOFFSET
FRCSEG          DW       ?

MAKINT          LABEL    DWORD
                DW       MAKINTOFFSET
MAKSEG          DW       ?

USRPRG          PROC     FAR
                POP      SI
                POP      AX          ;RECOVER MBASIC-86'S CS
                PUSH     AX
                PUSH     SI
                MOV      FRCSEG,AX   ;Set segment for long indirect CALL
                MOV      MAKSEG,AX
                CALL     FRCINT      ;Force arg in FAC to int in [BX]
                ADD      BX,BX       ; [BX] - [BX] * 2
                CALL     MAKINT      ;Put Result back in FAC
                RET                  ;Long return to BASIC
USRPRG          ENDP

DOUBLE          ENDS
```

When FRCINT or MAKINT is called and when the subroutine terminates with a return, ES, DS, and SS must have the same value they had when the subroutine was entered. These registers point to MBASIC-86's Data Segment.

FRCINT is at 103 hex and MAKINT is at 107 hex.

Appendices

A

Getting Help

Help Line Phone Numbers

Country	Phone Number
U.S.A.	(800) DEC-8000
Canada	(800) 267-5251
United Kingdom	(0256) 59 200
Belgium	(02)-24 26 790
West Germany	(089)-95 91 66 44
Italy	(02)-617 53 81 or 617 53 82
Japan	(0424) 64-3302
Denmark	(04)-30 10 05
Spain	(1)-73 34 307
Finland	(90)-42 33 32
Holland	(1820)-31 100
Switzerland	(01)-810 51 21
Sweden	(08)-98 88 35
Norway	(02)-25 64 22
France	(1)-687 31 52
Austria	(222)-67 76 41 extension 444
Australia Sydney	(02) 412-5555
All other areas	(008) 226377

B

Making a System/Application Diskette

If you are using MBASIC–86 routinely, you should copy the CP/M–86/80 operating system and MBASIC–86 to a *single diskette.* By doing this, you can use one diskette to start the operating system and also to run MBASIC–86. Therefore, a second diskette drive is available to hold a diskette used to store program and data files.

The diskette you copy the CP/M–86/80 operating system and MBASIC–86 onto is called a *system/application* diskette because it contains the operating *system* and the *application* program. This appendix tells you how to make a system/application diskette.

Follow the steps in the next two sections of this appendix to make a system/application diskette. In the section titled:

1. "Copying the Operating System Files," you copy the operating system onto a blank diskette.
2. "Copying the MBASIC–86 Files," you copy MBASIC–86 files onto the diskette you just copied the operating system.

Copying the Operating System Files

1. Display the Rainbow 100 Main System Menu according to one of the following procedures:

   - If the Rainbow 100 computer is turned off — Make sure that no diskettes are in the drives. Turn on the Rainbow 100 computer with the drive doors opened or closed. The Main System Menu should be displayed on the screen.

   - If the Rainbow 100 computer is turned on — Reset the Rainbow 100 computer by pressing:

     ```
     <Set-Up>
     ```

     and typing:

     ```
     <Ctrl/Set-Up>
     ```

     The Main System Menu should be displayed on the screen.

2. Remove the CP/M–86/80 working diskette from its protective envelope.
3. Open the drive A door and insert the CP/M-86/80 working diskette. Close the drive A door. If the diskette has a write-protect tab, use another diskette.
4. Start the CP/M–86/80 operating system by pressing:

   ```
   A
   ```

   The computer displays:

   ```
   A>
   ```

5. Remove a blank diskette from the diskette box in the Rainbow 100 CP/M–86/80 Operating System Kit. The part number, BL-N402A-BK, should be printed on the blank diskette's label. You copy the operating system to this diskette. It becomes the system/application diskette.
6. Remove the blank diskette from its protective envelope.

7. Open the drive B door and insert the blank diskette. Close the drive B door.

   NOTE
   If you did not turn on or reset the computer just before inserting the diskettes, type:

   ```
   <Ctrl/C>
   ```

   after the A> to tell the operating system that you have inserted new diskettes into the drives.

8. The following procedure, which copies the operating system to the blank diskette using the SUBMIT program, should complete in about about two minutes. You type only one instruction. In this instruction, A is the source drive and B is the destination drive. The remainder of the instructions are typed by the computer.

   NOTE
   If any error messages are displayed at any time during the following procedure, refer to the error messages section in the *Rainbow 100 User's Guide.*

   After the A>, type:

   ```
   A>SUBMIT SYSCOPY A B<Return>
   ```

   As the operating system is being copied, the small lights beside each drive turn on and off and the drives make clicking and whirring sounds. When the copying procedure is completed, the operating system displays A>. Screen 3 shows the entire dialog.

```
A>SUBMIT SYSCOPY A B

A>LDCOPY A: B:
LDCOPY VERS 1.5


A>PIP B:=A:*.SYS[ROV]

COPYING -
CPM.SYS
Z80CCP.SYS
Z80.SYS
PRMTVPVT.SYS

A>PIP B:=A:MAINT.CMD[OV]

A>PIP B:=A:PIP.CMD[OV]

A>PIP B:=A:COPY.COM[OV]

A>PIP B:=A:SUBMIT.CMD[OV]

A>PIP B:=A:STAT.CMD[OV]

A>
```

MR-S-2330-82

Screen 3. SYSCOPY Dialog

9. Check that all the files were copied. After the A>, type:

   ```
   A>DIR B:<Return>
   ```

   The file names in Screen 4 should be displayed on the screen followed by:

   ```
   A>
   ```

```
A>DIR B:
B: MAINT      CMD : PIP        CMD : COPY       COM : SUBMIT     CMD
B: STAT       CMD

SYSTEM FILE(S) EXIST
A>
```

MR-S-2332-82

Screen 4. Directory File Names on System/Application Diskette

10. Then, after the A>, type:

    A>DIRS B:<Return>

    The file names in Screen 5 should be displayed on the screen followed by:

    A>

```
A>DIRS B:
B: CPM        SYS : Z80CCP    SYS : Z80        SYS : PRMTVPVT SYS

NON-SYSTEM FILE(S) EXIST
A>
```

MR-S-2334-82

Screen 5. System File Names on System/Application Diskette

You have now successfully copied the operating system files to the diskette in drive B. Go to the next section to copy the MBASIC-86 files onto this diskette.

Copying the MBASIC–86 Files

1. Open the drive A door and remove the CP/M–86/80 working diskette. Return it to its protective envelope and store it in a safe place.
2. Open the drive B door and remove the system/application diskette. This was the blank diskette that you have just finished copying the operating system files to.
3. Insert the system/application diskette (the diskette you just removed from drive B) into drive A. Close the drive A door.
4. Insert the MBASIC–86 working diskette into drive B. Close the drive B door.

5. To tell the operating system that you have changed diskettes, type:

```
<Ctrl/C>
```

When you type Ctrl/C, you hear clicking sounds from the drive and the small light beside drive A turns on briefly. The operating system displays the following on your screen:

```
A>^C

A>
```

6. Copy MBASIC–86 onto the diskette already containing the operating system (in drive A). To do this, you use PIP, a program that copies files from one diskette to another diskette.

> **NOTE**
> If any error messages are displayed at any time during the following procedure, refer to the error messages section in the *Rainbow 100 User's Guide.*

Type:

```
A>PIP A:=B:*.*[OV]<Return>
```

where:

A:	is the location you are copying to (the destination drive).
B:	is the location you are copying from (the source drive).
.	is a symbol indicating all files.
[OV]	are added instructions for PIP.

As MBASIC–86 files are being copied, the small lights beside each drive turn on and off and the drives make clicking and whirring sounds. PIP displays:

```
COPYING -
```

followed by a list of all the file names as they are copied. The file names are the same names as those on the source diskette. When all the files are copied, the operating system displays:

```
A>
```

7. Open the drive B door and remove the MBASIC–86 working diskette. Return it to its protective envelope and store it in a safe place.
8. If desired, you can now insert a data diskette into drive B to use with the application program. You can use this diskette to store information. Close the drive B door.
9. To tell the operating system that you have changed diskettes, type:

```
<Ctrl/C>
```

When you type Ctrl/C, you hear clicking sounds from the drive and the small light beside drive A turns on briefly. The operating system displays the following on your screen:

```
A>^C

A>
```

You have now successfully created a system/application diskette which contains the operating system files and MBASIC–86. With this diskette, you can now start the operating system and use MBASIC–86.

Described above is the preferred method of creating a system/application diskette. Refer to the discussion of the SUBMIT program in the *Rainbow 100 User's Guide* if you want to customize this procedure to copy other files.

Index

G

I

K

L

M

N

O

P

R

S

T

U

W

HOW TO ORDER ADDITIONAL DOCUMENTATION

If you want to order additional documentation by phone:

And you live in:	Call:	Between the hours of:
New Hampshire, Alaska or Hawaii	603-884-6660	8:30 AM and 6:00 PM Eastern Time
Continental USA or Puerto Rico	1-800-258-1710	8:30 AM and 6:00 PM Eastern Time
Canada (Ottawa-Hull)	613-234-7726	8:00 AM and 5:00 PM Eastern Time
Canada (British Columbia)	1-800-267-6146	8:00 AM and 5:00 PM Eastern Time
Canada (all other)	112-800-267-6146	8:00 AM and 5:00 PM Eastern Time

If you want to order additional documentation by direct mail:

And you live in:	Write to:
USA or Puerto Rico	DIGITAL EQUIPMENT CORPORATION Attn: Accessories and Supplies Centers P.O. Box CS2008 Nashua, NH 03061 NOTE: Prepaid orders from Puerto Rico must be placed with the local DIGITAL subsidiary (Phone 809-754-7575)
Canada	DIGITAL EQUIPMENT OF CANADA LTD. 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: A&SG Business Manager
Other than USA, Puerto Rico or Canada	DIGITAL EQUIPMENT CORPORATION Accessories and Supplies Center A&SG Business Manager c/o Digital's local subsidiary or approved distributor

Digital Equipment Corporation
Personal Computer Software Products
Software License Transfer Agreement

I understand and agree to abide by the Software License Agreement that accompanies the software product.

I have obtained the Software Product from the following Transferor:

_____
(Name of Transferor)

_____
(Address)

The Software Product(s) I am receiving is: (List Product(s) and include Serial Number, if any):

_____ _____

_____ _____

_____ _____

_____ _____

The Software Product(s) will be used on CPU Serial Number: _____

(Transferee: name of person/organization receiving the Software Product from Transferor.)

By: _____
(Signature of Transferee)

_____
(Print Name)

Company: _____

Address: _____

_____

Date: _____

Do Not Tear – Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
CONTINENTAL BOULEVARD
MERRIMACK NH 03054
ATTN: PERSONAL COMPUTER SOFTWARE

Do Not Tear – Fold Here and Tape

Cut Along Dotted Line

Rainbow™ 100
MBASIC™–86 User's Guide
AA–P601B–TV

READER'S COMMENTS

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

☐ First-time computer user
☐ Experienced computer user
☐ Application package user
☐ Programmer
☐ Other (please specify)

Name

Date

Organization

Street

City

State

Zip Code
or Country

Do Not Tear – Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MRO1-2/L12
MARLBOROUGH, MA 01752

Do Not Tear – Fold Here and Tape

Cut Along Dotted Line