FOR INTERNAL USE ONLY

HEWLETT-PACKARD
SERIES 80 TO SERIES 200 BASIC
TRANSLATION GUIDE

5957-4373 SEP 83

HEWLETT PACKARD

DESKTOP COMPUTER DIVISION · 3404 E. Harmony Road, Ft. Collins, Colorado 80525, Telephone 303 226-3800

September 21, 1983

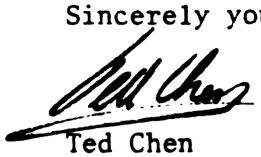
To The Desktop SE:

I am pleased to present you with the "Hewlett-Packard Series 80 to Series 200 BASIC Translation Guide". With it you will be better able to assist customers who wish to transport their BASIC application software from the Series 80 to the Series 200.

The guide was made possible through the diligence of an engineer in our Software Lab, who responded to your inputs for such a tool. There are no plans to make the guide into a product, and it is for internal distribution only.

We in FSD Technical Support are anxious to hear your reactions to this exclusive distribution.

Sincerely yours,



Ted Chen
FSD Technical Support Manager-
Language Systems

TC/mg

Table of Contents

Appendix D
Supported Disc Drives

A Note to the Systems Engineer

In order to give the best help to the Series 80 owner who needs to move his program to the Series 200, this document attempts to give very detailed instructions wherever they are needed. By its very nature, this document emphasizes the things which are hard to convert. It does not mention the many features of the Series 200 which are not available on the Series 80 (e.g., availability of very large memory, a friendlier human interface, data communications capabilities, shared resource management, unified I/O, the ability to call compiled subprograms, and the color graphics options), nor does it discuss the significant execution speed improvements which should result from transporting a program. It is important not to let the differences between the two systems cast an unfavorable light on the Series 200.

It is also important to encourage your customers to become familiar with Series 200 BASIC in its own right. They should write all of their new software using the full functionality of the Series 200. They should not use this document as a crutch to avoid learning Series 200 BASIC. It should only be used to convert existing software.

Introduction

This document is meant to be a guide for anyone needing to move a Series 80 BASIC program to a Series 200 machine. It begins by discussing several methods of moving programs to Series 200 hardware. The next chapter contains an alphabetical listing of all Series 80 BASIC capabilities together with an explanation of how to achieve the same effect in Series 200 BASIC. The final chapter discusses how to move data. Several appendices cover special topics such as simulating flags and how to implement global matrix functions.

This document does not address the issues involved with moving either programs or data from the Series 200 to the Series 80.

Note that throughout this document, the term "Series 80 system" is used to refer to any Series 80 computer—HP83, HP85, HP86, or HP87. The term "HP85" generally refers to either the HP83 or the HP85, and the term "HP87" generally refers to either the HP86 or the HP87. The term "Series 200 system" refers to the HP9816, HP9826, HP9836A, HP9836C, or to the HP9920. The minimum Series 200 software configuration is BASIC 2.0. Depending upon the program to be translated, various extensions to BASIC 2.0 may be needed. The extensions are AP2.0, AP2.1, GRAPH2.0, GRAPH2.1, and XREF 2.1 or their ROM equivalents. Whenever they are needed, they are mentioned explicitly. Note that AP2.1 supercedes AP2.0, and GRAPH2.1 supercedes GRAPH2.0. Thus whenever this document says, for example, that AP2.0 is required, either AP2.0 or AP2.1 (or the ROM equivalent) will suffice. The converse is not true.

Limitations

In writing this document, every reasonable attempt has been made to help you in converting your Series 80 BASIC program so that it will run on the Series 200. However, there are a number of Series 80 BASIC capabilities (such as CRT IS) which are not supported by the Series 200. The I/O and graphics subsystems are radically different, so straightforward conversions in these areas are likely to be less than totally satisfactory. Furthermore, the two systems have entirely different CPUs, so that Series 80 binary programs cannot be used on the Series 200. If your Series 80 program makes use of binaries, you have a couple of choices. You can find out if a Series 200 binary or compiled subprogram available commercially or through the Users' Library provides the same capability, or you can replace the binary subprogram with a Series 200 BASIC subprogram or a compiled subprogram that you write yourself.

The information in this document is current as of August 1, 1983. There may be future language enhancements or bug fixes to either system which will make this document incomplete or render portions of it obsolete. In particular, future releases of Series 200 BASIC may contain a different partitioning of the keywords into binaries. The location of a keyword can always be determined by consulting the Series 200 BASIC Language Reference manual.

Moving Your Program

The first problem which confronts you in translating a Series 80 BASIC program to Series 200 BASIC is how to move your program source and data to a Series 200 machine. The last thing you want to do is to retype the entire program and data. Fortunately, there are a couple of relatively painless solutions to this problem. We will look at methods of moving programs in this chapter. A later chapter in this document will address the problem of data transportation.

There are a number of transformations you may be able to perform on your program using Series 80 editing capabilities which will make the job of porting it to a Series 200 machine much easier. Following the hints at the end of this chapter (See *Simplifying Your Program*) before you create an ASCII file containing your Series 80 program could save you a great deal of time and frustration. Most of the hints take advantage of a Series 80 Advanced Programming ROM. If you do not have this ROM, you can make the indicated changes on your Series 80 machine, or you can wait until you have moved your program to the Series 200. If you have AP2.0 (or AP2.1) for your Series 200, it will help immensely.

The ASCII File Approach

The easiest way to move your program to a Series 200 machine is via an ASCII file. This requires that you have a disc drive supported by your Series 80 machine and a disc drive for the same size floppy discs which is supported by Series 200 BASIC (these can be the same disc drive). If your Series 80 machine is an HP85, you will also need a Series 80 Mass Storage ROM. A list of suitable disc drives can be found in Appendix D at the end of this document. The process of creating an ASCII file involves the use of two Series 80 programs which can be found either among your demonstration programs or from the Series 80 Users' Library.

The three-step process for moving your program is described in the following. The GETSAVE binary for the HP87 is found on your demonstration disc. The equivalent (GETSAV) binary for the HP85 is available from the Series 80 Users' Library (900-0022). The LIF program is also available from the Users' Library (9-0049 for the HP85 and 9-0069B for the HP87).

Step 1

Use the GETSAVE or GETSAV binary to create a Series 80 DATA file containing your program. To do this, execute the following sequence of commands:

```
LOAD "yourprog:msus"       ! Use msus if needed

LOADBIN "GETSAVE:msus"     ! LOAD BIN "GETSAV:msus" for HP85

SAVE "yourdata:msus"
```

If you have an HP87 equipped with an Advanced Programming ROM, omit the LOADBIN.

Step 2

Use the LIF program to create an ASCII file containing your program, as follows:

```
LOAD "LIF87:msus"          ! LOAD "LIF:msus" for HP85

LOADBIN "LIFg:msus"        ! LOAD BIN "C:msus" for HP85
```

Press [RUN]. Give your msus, (unquoted) starting with a semicolon, and press the softkey labeled LIFSAVE. Continue answering the questions asked.

Step 3

If you will not be using an internal disc drive on your Series 200 machine, reconfigure your equipment so that an appropriate disc drive is now connected to your Series 200 machine. Power up your Series 200 machine in BASIC, and load in whatever BASIC extensions are necessary. (Appendix D will tell you whether AP2.0 is required for your particular disc drive.) Insert the disc containing the ASCII file representation of your Series 80 BASIC program, and execute

```
PRINTER IS device specifier  ! Specify external printer,
                             ! optional but helpful

GET "yourascii:msus"
```

This will cause your Series 200 machine to read the ASCII file containing your Series 80 program. It will check each line for syntactic correctness and will convert into comments any lines which are not correct. Lines containing syntax errors will be printed on the specified device. You are now ready to use the material in the next chapter of this manual to convert your program.

The Machine-to-Machine Transfer Approach

If you do not own a disc drive for your Series 80 machine or if your Series 200 machine will not talk to your disc drive, the method outlined above is of no help to you. If you have a Series 80 HP-IB interface and I/O ROM (see below for an important exception to the I/O ROM requirement), a tape or disc drive for your Series 80 system, and a disc drive for your Series 200 machine (a built-in drive is fine), you can use the following method. If you use external mass storage on your Series 80 machine, you will need an extra Series 80 HP-IB interface card and a Series 80 Mass Storage ROM. You can also adapt this method to non-HP-IB interfaces.

You can move your program source from your Series 80 to your Series 200 machine by connecting the two machines together via HP-IB interfaces and then sending your Series 80 program one line at a time to your Series 200 machine.

NOTE

Because of irregularities in the Series 80 implementation of HP-IB, it is not possible to use this method if the same HP-IB interface connects your Series 80 machine to an external mass storage device and to your Series 200 computer. You need HP-IB interfaces on separate select codes. Connect the interface with the lower select code to the disc and the other interface to the Series 200 computer.

NOTE

If you have an HP87 and none of your program lines are longer than 80 characters, you can get by without the I/O ROM. You will need to modify both programs below to eliminate the "#" from the image specifiers.

This approach requires that you first create a DATA file containing the program to be translated. Follow step 1 above to do this. Any mass storage medium supported by your Series 80 system will do. Depending on your computer and your choice of mass storage medium, you may need a Series 80 Mass Storage ROM. You will also need a disc drive supported by your Series 200 system. Depending on your disc drive, you may also need AP2.0 (see Appendix D).

Before the two computers can communicate over HP-IB, you must modify one end of their common HP-IB interface to make one of the systems non-controller with bus address 20 on that interface. Be sure that the other system has some other bus address. It is easiest to modify the built-in HP-IB interface of the Series 200. However, if you are using external mass storage on your Series 200 connected on the same HP-IB interface which joins the Series 200 to the Series 80 computer, you will need to make the Series 80 the non-controller. Consult your Series 200 Installation Manual if you are using the built-in interface, or the documentation supplied with your card if you are using an add-on card. If you need to modify a Series 80 card, consult the Series 80 HP-IB Interface Owner's Manual.

Configure your equipment so that each computer has access to the appropriate mass storage, and connect the computers together via HP-IB. Power up both systems and enter the following programs.

NOTE

These programs assume that the Series 200 computer is the non-controller. If this is not the case, change the Series 80 program to contain PRINTER IS 7 instead of PRINTER IS 720. Also change the Series 200 program to contain ASSIGN @S80 TO 720 instead of ASSIGN @S80 TO 7.

```
 10 ! PROGRAM FOR YOUR SERIES 80 COMPUTER, REQUIRES I/O ROM
 20 DIM A$[160], F$[30]
 30 PRINTER IS 720 ! CHANGE AS NEEDED
 40 DISP "ENTER THE FILE SPECIFIER OF FILE TO BE SENT"
 50 INPUT F$
 60 ASSIGN# 1 TO F$
 70 ON ERROR GOTO 140
 80 READ# 1;A$
 90 DISP A$
100 IF A$="" THEN GOTO 80
110 A$=A$[1,LEN(A$)-1]
120 PRINT USING "#,A,K";CHR$(LEN(A$)),A$
130 GOTO 80
140 IF ERRN=72 THEN GOTO 170
150 DISP "UNEXPECTED ERROR ENCOUNTERED"
160 STOP
170 PRINT USING "#,A,K";CHR$(4),"!@#$"
180 BEEP
190 DISP "ALL DONE"
200 END
```

```
 10 ! PROGRAM FOR YOUR SERIES 200 COMPUTER, RUNS IN BASIC 2.0
 20 DIM A$[160],F$[30]
 30 INPUT "NO. OF SECTORS OF DESTINATION FILE",Sectors
 40 INPUT "FILE SPECIFIER OF DESTINATION FILE",F$
 50 CREATE ASCII F$,Sectors
 60 ASSIGN @F TO F$
 70 ASSIGN @S80 TO 7
 80 Read_line: ENTER @S80 USING "#,B";Length
 90 Image$ = "#,"&VAL$(Length)&"A"
100 ENTER @S80 USING Image$; A$
110 IF (Length=4 AND A$="!@#$") THEN
120   DISP "ALL DONE"
130   STOP
140 ELSE
150   PRINT A$
160   OUTPUT @F;A$
170   GOTO Read_line
180 END IF
190 END
```

Make sure that the necessary mass storage media are in their proper drives. You will need to tell your Series 200 computer how big a file to create to hold your program. You can get this

number by doing a CAT on your Series 80 system to determine the size in sectors of the DATA file which holds your program. Use the same size. Press the [RUN] keys on both of your systems (in either order) and respond to the requests for input to cause the program to be transferred. When you are done and disconnect the systems, remember to undo your HP-IB interface modification to restore its controller status. To proceed with the translation, do a GET of the ASCII file you have just created for your Series 200 system to bring the program into memory.

A simpler but less reliable way to ship a program over HP-IB is as follows. Configure your hardware as described above. Load your Series 80 program into memory directly from a PROG file or type it in. Execute PRINTER IS 720 on your Series 80 machine. Modify the Series 200 program given above so that it does not expect to receive the line length before the line. It should use free-field ENTER to read a program line. Run the Series 200 program and do a PLIST on your Series 80 machine. You will also need to arrange to terminate this process somehow. This method will fail if your Series 80 program contains embedded line-feeds, e.g., if a program line is greater than the CRT width.

The Keyboard Entry Method

If you cannot use any of the above methods of transferring a program, you will need to type your program in to your Series 200 computer. You will need an up-to-date listing of your Series 80 program. Probably the best thing to do in this case is to use the next chapter of this manual to rewrite your program for the Series 200 before typing it in. As an alternative, you may write a small Series 200 program which creates an ASCII file large enough to hold your program and which writes your program to this file one line at a time as you type it in. Be sure to start each line typed in with a line number. Once you have done this, you can bring the program into memory using a GET and then proceed with the translation.

Simplifying Your Program

As mentioned above, performing a few simple transformations on your Series 80 program before you attempt to move it to a Series 200 machine can greatly expedite the translation process. A number of suggestions are given here.

Line Length

The maximum line length on the 9826 computer running BASIC 1.0 is 100 characters. (For all other machines and all other versions of Series 200 BASIC, the maximum line length is 160 characters.) If this is your target machine, you will want to be sure that your program meets this restriction. Break up long lines before trying to move the program. Otherwise, the GET operation will terminate with an error.

There is no convenient way of remedying this situation on the Series 200 machine.

Multi-statement Lines

Series 200 BASIC does not support multiple statements per line. You can break up multi-statement lines either before or after you move your program to the Series 200. (Details of how to do this are given in the next chapter). However, if you decide to break up lines after porting, you should be sure that you have line numbers spaced widely enough to allow you to do this without renumbering your program on the Series 200. This is because moving your program to the Series 200 machine will cause all lines which are not syntactically legal on the Series 200 to be commented out. Renumbering a program in this state is disastrous because line numbers embedded in such lines will not be updated.

Maximum Line Number

The highest line number allowed in Series 200 BASIC programs is 32766. The HP87 allows line numbers up to 99999. If your program contains line numbers greater than 32766, you must renumber it before you attempt to port it, because the Series 200 GET operation will terminate at 32767. If necessary, break your program up into several smaller programs first.

Variable Names

If your program was written for an HP85, all of your variable names will be legal Series 200 variable names, and you may skip this section. However, if your program was written for the HP87, you may need to make some modifications. The problem which is most likely to occur is that your variable names will not observe the Series 200 requirements that the first character be an upper-case letter and that all succeeding alphabetic characters must be lower-case. Fortunately, this is handled automatically by the Series 200 syntaxer.

There are, however, several other problems with variable names which are not handled automatically. If you own a Series 80 Advanced Programming ROM, it will be easier to change your problem variable names before porting your program. This can be done as follows.

First, load in the program to be modified and execute an XREF|tV. This will tell you all the variable names used in your program. Use this cross-reference to determine which variable names are potential problems. There are three categories of problems to watch for:

1. Variable names longer than 15 characters (16 including the terminal $ of a string name). These must be shortened. Use the REPLACEVAR command in your Series 80 Advanced Programming ROM to substitute a shorter name for each such name. Be sure not to duplicate any existing names.

2. Variable names which will map to the same Series 200 variable name. The Series 200 syntaxer (invoked during keyboard entry of a program or during a GET) allows variable names to be entered in any combination of upper and lower case letters. It automatically transforms them so that the first character is in upper-case and the rest are in lower-case. For example, the distinct Series 80 variable names XXx and XXX would both map to Xxx in the Series 200. You will need to find such conflicts in your XREF V listing and use REPLACEVAR to remove them. Again, be sure not to duplicate any existing names.

3. Variable names which conflict with Series 200 keywords (e.g., CASE). This is a hard one because it requires some familiarity with the Series 200 vocabulary. You can use REPLACEVAR to change these names to anything you want. If you wish, you may change only a single character of each name, and the Series 200 will then be able to distinguish the name from the keyword. For example, if CASE is changed to CASe, the Series 200 syntaxer will map it to Case, and there will be no problem (mixing upper and lower case causes the system to interpret the name as a variable name).

If you do not take care of variable name problems before you move your program, lines which contain illegal variable names will be turned into comments by the Series 200 syntaxer. You will need to find the problems as you look over your program on a line-by-line basis. You may be able to use the FIND and CHANGE commands in AP2.0 to help you eliminate the problems once you have identified them. The Series 200 XREF will not be helpful since the problem variable names will occur in lines which have been commented out.

Scalar/Array Naming Conflicts

In the Series 80, scalar and array variables in the same environment may have the same name. On the Series 200, this is not legal and will cause an error at program prerun. If you have a Series 80 Advanced Programming ROM, you can easily eliminate these conflicts before porting your program. Do an XREF V. This will generate a cross-reference which distinguishes between scalar and array occurrences of a variable name. Do not use REPLACEVAR, since it will change all occurrences for you. Instead, do a manual replacement of either the scalar or the array name, whichever occurs less frequently. The XREF output will tell you which line numbers need to be changed.

If you do not have a Series 80 Advanced Programming ROM or if you choose not to do this step, you can eliminate the conflicts once you have moved your program to the Series 200. If you have AP2.0 or XREF2.1, you can do an XREF to determine where changes are needed. FIND and CHANGE can then be used to make the changes. If you do not have AP2.0, you can use the XREF utility in your Series 200 BASIC Utilities Library.

Keyword Information

This section lists each capability of Series 80 BASIC and gives an explanation of how that capability is supported in Series 200 BASIC. It begins with a discussion of some general topics and concludes with an alphabetical keyword listing.

If the support for a Series 80 capability requires Series 200 BASIC extensions, the binary which contains the needed feature is named. Note that future releases of Series 200 BASIC may contain a different partition of keywords into binaries, so that the information given here would no longer be valid. Consult the Series 200 BASIC Language Reference manual for your current release of BASIC whenever any doubts arise.

Operators

Arithmetic Operators

Note that because the Series 80 and the Series 200 use different internal representations for floating point numbers and different algorithms for floating point operations, these operations are likely to give slightly different results.

+ (Addition)

No change.

- (Subtraction)

For binary -, no change is required. For unary -, parenthesize the - and its argument.

***** (Multiplication)

No change.

/ (Division)

No change.

^ (Exponentiation)

No change under normal circumstances. Note that 0^0 returns 1 and issues a warning on the Series 80. Similarly, 0 raised to a negative power returns the largest machine real and issues a warning. On the Series 200, 0^0 or 0 raised to a negative power gives Error 26.

MOD

If B is not a positive constant, replace `A MOD B` by `A-B*INT(A/B)`. If B is guaranteed to be positive, no change is required.

\ or DIV

Parenthesize the second argument if it begins with a negative sign.

Relational Operators

=

If AP2.0 (or AP2.1) is loaded, string comparisons are done using the collating sequence associated with the current LEXICAL ORDER IS. If the current collating sequence is not ASCII, execute LEXICAL ORDER IS before doing the string comparison, and reset the LEXICAL ORDER IS after the comparison. Otherwise, no change is required.

>

See =.

<

See =.

>=

See =.

<=

See =.

<> or #

Replace # by <>. See =.

AND

No change.

OR

No change.

EXOR

No change.

NOT

No change.

String Operators

&

No change.

[first subscript] (delimits a substring)

No change is required if the substring is used in right context. Note however, that on the Series 200 as well as on the HP85, the value of first subscript may be one greater than the current length of the string. On the HP87, this condition causes an error. Hence, if you are trapping this error with ON ERROR, you will need to restructure your program.

In left context, the Series 80 permits addressing a substring which starts beyond the current length of the string. The designated part of the string is given the value specified, and the portion of the string between its current length and the newly assigned part is filled with blanks. The Series 200 allows (current length + 1) as the maximum first subscript of a substring. It may be impossible to detect these errors until run-time. If they occur, you will need to do the padding with blanks explicitly to extend the string's current length before attempting to assign characters to the "far reaches" of the string.

[first subscript, last subscript] (delimits a substring)

See the immediately preceding entry.

Arithmetic Expressions

There are two differences in the mathematical hierarchies between Series 80 and Series 200. These differences mean that you need to examine all numeric expressions to see whether they contain either NOT or a negative constant.

A negative constant can be made to assume its Series 80 behavior on the Series 200 by enclosing it in parentheses. For example, the expression A/-3 should become A/(-3).

If the operator to the right of NOT is in the following list, the NOT and the operand immediately to its right should be parenthesized to force the early evaluation of the NOT:

```
  *, /, MOD, DIV
+, -
=, >, <, >=, <=, <>, #
```

For example, NOT A < B should become (NOT A) < B so that the NOT will be applied before the <.

Data Precision

The Series 80 uses a BCD numeric representation, and the Series 200 uses binary representations. Hence the ranges for the various data types are different; in all cases, Series 80 allows greater range. Consult the following table for details.

Data type	Series 80 Range	Series 200 Range
REAL	-9.99999999999E+499 thru -1.0E-499 0, and +1.0E-499 thru +9.99999999999E+499	-1.797693134862315E+308 thru -2.225073858507202E-308, 0, and +2.225073858507202E-308 thru +1.797693134862315E+308
SHORT	+-9.9999E+-99	*Not supported, use REAL*
INTEGER	+-99999	-32768 thru 32767

Special Characters

@ (Statement separator)

The Series 200 does not permit multi-statement lines.

You can break up multi-statement lines either before or after you move your program to the Series 200. If you choose to break them up on your Series 80 machine and you have an Advanced Programming ROM, you can use the SCAN command to help you find all of the multi-statement lines.

If you choose to break up multi-statement lines after you have moved your program to the Series 200, remember to renumber your program if necessary *while it is still on the Series 80 machine*. You will need an increment of at least n+2 between line numbers, where n is the maximum number of statements per line. (Use n+2 to allow for the insertion of DISABLE and ENABLE where needed. See below.) You must provide sufficient line numbers to do the breaking apart without needing to renumber you program on the Series 200. (You don't want to renumber on the Series 200 before your program is completely translated, because Series 80 line numbers which are in commented out lines will not be updated by the renumbering, causing a real mess).

After you have moved to the Series 200, all multi-statement lines will have been commented out by the GET operation. You can use the FIND command in AP2_0 to help you find all occurrences of "@".

You can use Series 200 EDIT mode to help you break lines apart. Consider the following example line:

```
50 !I=I+1 @ PRINT "HELLO" @ GOTO 100
```

This line consists of three statements, so the first thing you want to do is to create three "copies" of it. To do this, first position the cursor in line 50, and then press [ENTER]. Your original line is

unchanged, but you have placed it in the recall buffer. Now, press [RECALL], press [SHIFT]-[LEFT] to move the cursor to the left end of the line, change the line number to 51, and press [ENTER]. You now have two copies of the line. Repeat the process of pressing [RECALL], editing the line number, and pressing [ENTER]. You now have three copies of the line.

Now, edit the three copies of statement 50 so that the first one (still line 50) contains only the leftmost statement, etc. Be sure to remove the exclamation point after the line number from each of these lines, too.

A couple of complications can occur when you break up lines. First, the line in question may be a "critical section." That is, you may want to be sure that program interrupts do not get serviced while the line is being executed. In this case, you will want to insert a DISABLE statement before the first statement on the line and an ENABLE statement after the final statement. (This is why you need n+2 line numbers between lines, instead of just n).

Another complication arises with line labels. If a multi-statement line contains a line label, you will normally want to associate that label with the first statement on the line as you break it up. However, if you need to insert a DISABLE before the first statement, the line label should be associated with the DISABLE instead.

One special case which deserves mention is the Series 80 IF...THEN...ELSE construct. In order to break this up for the Series 200, the IF expression THEN must occupy a line, each statement following the THEN must occupy its own line (precede statement numbers or labels by GOTO), the ELSE must occupy a line, and each statement following the ELSE must occupy its own line (again, insert GOTOs before statement numbers or labels). If you break this construct apart on your Series 80 machine, you will need to insert an exclamation point (!) to comment out the resulting IF and ELSE statements because they will not syntax properly on the Series 80. Remove the exclamation points once you have moved your program to the Series 200.

! (Remark)

No change.

" " (String delimiters)

No change.

Variables

Variable Names

If your program was written for the HP85, all of your variable names will be legal Series 80 variable names. However, if your program as written for the HP87, you may need to make some modifications. Detailed instructions are given in the previous chapter of this document, under the heading *Simplifying Your Program.*

The Series 80 allows scalars and arrays to have the same variable name, but the Series 200 does not. The previous chapter gives some instructions on easy ways to find and eliminate these conflicts. See *Simplifying Your Program.*

Simple String Variables

The Series 80 allows strings of up to 65530 characters, depending on the memory available. The Series 200 limit is 32767 characters. Your program will need significant restructuring if it uses very long strings.

Uninitialized Variables

When the Series 80 encounters an uninitialized variable, one of two things can happen. If no ON ERROR is in effect, the system issues a warning and supplies 0 or the null string as the value of the variable, as appropriate. The Series 200 does not supply the warning, but makes the same default assignments. If an ON ERROR is in effect, however, the Series 80 will branch to the user-supplied error handling routine when it encounters an uninitialized variable. The Series 200 does not trap this condition. It is best, therefore, to be sure that all variables are initialized explicitly.

IMAGE Specifiers

The Series 200 does not support the C or P specifiers, and there is no easy way to replace them.

The e specifier may be replaced by E, and E by ESZZZ.

The / specifier must be separated from the preceding or following specifier by a comma.

The R and * specifiers are supported in AP2_0.

You will probably need to adjust termination conditions to get the results you expect.

File Specifiers

Most Series 80 files are not usable directly by Series 200 BASIC. ASCII files are the exception. The previous chapter of this document discusses methods of moving stored Series 80 programs to the Series 200. A last chapter discusses ways of transporting data.

You will probably try to give your Series 200 files the same names as their Series 80 counterparts. You will need to observe the following restrictions. The Series 200 allows a more limited character set in file names (upper and lower case alphabetic letters, digits, the underscore, and the blank—which is ignored) and restricts the length of a file name to 10 characters. It will not truncate longer names.

It is important to use your Series 80 computer to rename files to legal Series 200 file names before you try to access these files on the Series 200. The Series 200 will be able to CAT files with illegal files names, but will not be able to perform any other operations on them because all other operations require checking the correctness of the file specifier.

Mass storage unit specifiers are quite different on the two systems. Consult your Series 200 BASIC Language Reference Manual for the form of a Series 200 BASIC mass storage unit specifier. One word of caution is necessary. The numeric part of the Series 80 msus needs to be changed to be interpreted properly by the Series 200. For example, if you have an HP82901 on select code 7 at primary address 3, the Series 80 msus for the right-hand drive is ":D731". The Series 200 msus for the same configuration is ":HP82901,703,1".

HP-IB Operations

In a number of HP-IB operations, the Series 80 and the Series 200 send the same bus commands, but in a slightly different order. The MTA and UNL are often send in the reverse order. This should have no effect on the operation of your program.

Alphabetical Keyword Listing

ABORTIO

The Series 80 ABORTIO statement serves two purposes—it halts ongoing TRANSFERs, and it sends an interface-dependent sequence of signals along the specified bus.

If there is any possibility that your Series 80 ABORTIO statement is used to terminate a TRANSFER, you will need to replace it by a Series 200 ABORTIO statement plus whatever is needed to send the bus sequences appropriate for the interface in use (see next paragraph). For the Series 200 ABORTIO statement (found in AP2.0), replace the Series 80 select code by the I/O path name associated with that select code (It is easiest if you use @Sc7 for select code 7, etc.). If you have used a variable to specify the select code in your Series 80 program, see the discussion of ASSIGN for a method of constructing the ABORTIO statement you need using OUTPUT 2. If your Series 80 ABORTIO is not used to terminate a TRANSFER, you may omit the Series 200 ABORTIO statement.

In any case, you will need to send the appropriate bus signals. If you are using the HP-IB interface, you may be able to use the Series 200 ABORT statement, updating the select code to specify your Series 200 HP-IB interface. The sequence of signals is slightly different from that sent by the Series 80 ABORTIO. If you are not happy with the Series 200 ABORT, you can create your own bus sequence with SEND. For non-HP-IB interfaces, consult your Series 200 BASIC Interfacing Techniques manual to see if you can use CONTROL and WRITEIO to create the desired effect.

ABS

No change.

ABSUM

Write a function which sums the absolute values of your array elements and replace ABSUM by an invocation of this function. The Series 200 cannot handle empty arrays.

ACS

No change if your Series 80 trig mode is RAD or DEG. If your Series 80 trig mode is GRAD, apply FNRtg to the result of your Series 200 ACS (see GRAD).

ALPHA (parameterless)

Replace by the two statement sequence

```
GRAPHICS OFF
ALPHA ON
```

If your program had been in ALPHALL or GRAPHALL mode, add GINIT after ALPHA ON to reset the graphics default conditions.

ALPHA (with parameters)

To set the cursor positon on a Series 200 system, use CONTROL to CRT registers 0 and 1. Register 0 controls the column position, and register 1, the row. The value to send to register 1 to position the cursor in a particular row may not be obvious because in the Series 80, row 1 is always the first row of CRT memory, while in the Series 200, row 1 is always the first visible (on-screen) row.

To home the cursor, use CONTROL 1;1,1.

ALPHALL

Replace by the three statement sequence

```
GRAPHICS OFF
ALPHA ON
GINIT
```

AMAX

If your program uses AMAX but not AMAXCOL or AMAXROW, you may simply use the MAX function found in AP2.0. If you need to use AMAXCOL or AMAXROW as well, then be sure that the program segment which contains the AMAX has a reference to COM /Matrix/ (see the appendix on Matrices). Write a function which also references COM /Matrix/ and replace AMAX by an invocation of this function. Your function should scan an array to find its maximum element (the value returned by the function) and the row (Amaxrow) and column (Amaxcol) location of this element. You will find the BASE, RANK, and SIZE keywords in AP2.0 very helpful.

AMAXCOL

Provide the COM reference and function discussed in AMAX. Note that the Series 200 syntaxer will have changed AMAXCOL to Amaxcol, which is just what you want. This will reference the appropriate COM variable.

AMAXROW

Provide the COM reference and function discussed in AMAX. Note that the Series 200 syntaxer will have changed AMAXROW to Amaxrow, which is just what you want. This will reference the appropriate COM variable.

AMIN

If your program uses AMIN but not AMINCOL or AMINROW, you may simply use the MIN function found in AP2.0. If you need to use AMINCOL or AMINROW as well, then be sure that the program segment which contains the AMIN has a reference to COM /Matrix/ (see the appendix on Matrices). Write a function which also references COM /Matrix/ and replace AMIN by an invocation of this function. Your function should scan an array to find its minimum element (the value returned by the function) and the row (Aminrow) and column (Amincol) location of this element. You will find the BASE, RANK, and SIZE keywords in AP2.0 very helpful.

AMINCOL

Provide the COM reference and function discussed in AMIN. Note that the Series 200 syntaxer will have changed AMINCOL to Amincol, which is just what you want. This will reference the appropriate COM variable.

AMINROW

Provide the COM reference and function discussed in AMIN. Note that the Series 200 syntaxer will have changed AMINROW to Aminrow, which is just what you want. This will reference the appropriate COM variable.

AND

No change.

AREAD

Use ENTER 1;*<string var>*, keeping in mind the following differences.

AREAD does not move the current print position, but ENTER 1 does. If this is important to you, do a STATUS to CRT registers 0 and 1 to determine the cursor position, then do the ENTER and restore the cursor position with CONTROL to CRT registers 0 and 1.

AREAD reads sufficient characters to fill the specified string. It does not terminate on CR/LF, and it does not enter CR/LF into the string. ENTER terminates on CR/LF. Hence, if your AREAD attempts to read more than one line of the CRT. ENTER is not a complete solution. Furthermore, the Series 200 puts CR/LF immediately following the last character output to the screen, but the Series 80 puts the CR/LF in "columns" 81 and 82 and fills the rest of the line with blanks. To read more than one line of the Series 200 CRT, therefore, use ENTER to read a line, pad it with blanks as necessary, use ENTER to read the next line, concatenate the lines, pad with blanks, etc.

ASN

No change if your Series 80 trig mode is RAD or DEG. If your Series 80 trig mode is GRAD, apply FNRtg to the result of your Series 200 ASN (see GRAD).

ASSERT

There is no direct parallel to the Series 80 ASSERT statement. Check the documentation for the interface you are using and your Series 200 BASIC Interfacing Techniques manual to see if you can create the desired effect using CONTROL and WRITEIO statements. For example, you can use CONTROL register 5 on an ordinary serial interface and CONTROL register 8 on a data communications interface. Be sure to use your Series 200 select code.

ASSIGN

You cannot open Series 80 DATA files from a Series 200 BASIC program. You must first convert them to Series 200 ASCII or BDAT files. See the last chapter of this document for methods of doing this. The remainder of this discussion assumes that you are trying to open an ASCII or a BDAT file.

If a constant (n) is used to specify the file number, replace

```
ASSIGN# n TO <file specifier>
```

by

```
ASSIGN @Fn TO <new file specifier>
```

If a variable (or numeric expression) is used to specify the file number, replace

```
ASSIGN# I to <file specifier>
```

by

```
OUTPUT 2;"ASSIGN @F"&VAL$(I)&" TO "&<new file specifier>    CHR$(255)&"X";
```

If you use a literal for the <new file specifier>, it must begin and end with three double quote (") marks. If you use a string variable or a string expression instead, you have two choices. If the value of the expression has double quotes as its first and last characters, no action is required. If not, you must concatenate the needed quotes into the output stream immediately before and immediately after the string expression. For example, if the value of A$ is MYFILE:INTERNAL, in order to get the effect of

```
... TO "MYFILE:INTERNAL"
```

you will need to use

```
...&" TO """&A$&""""
```

Make the analogous replacements for ASSIGN# ... TO *.

NOTE

There is one possible complication which can arise when using OUTPUT 2 to construct an ASSIGN statement. The I/O path name must appear explicitly somewhere else in the program segment. When OUTPUT 2 is used, the I/O path name is built as part of the output list. It does not appear as a whole in program source where it can be tokenized by the syntaxer. Hence, if all references to the I/O path name in that environment are so constructed, the I/O path name will not appear in the program segment's symbol table, and when the OUTPUT 2 statement is executed, the system will generate Error 910. To get around this, you can insert anywhere in the program segment a STATUS statement which looks at register 0 of the I/O path name and sets the value of a dummy variable. This will cause the I/O path name to appear in the symbol table, and the error will be avoided.

ATN

No change if your Series 80 trig mode is RAD or DEG. If your Series 80 trig mode is GRAD, apply FNRtg to the result of your Series 200 ATN (see GRAD).

ATN2

Replace ATN2 by FNAtn2 (see below). If your Series 80 trig mode is GRAD, apply FNRtg to the result of FNAtn2 (see GRAD).

Append the following code to your program:

```
DEF FNAtn2(Y,X)
IF X>=0 THEN RETURN ATN(Y/X)
Z=PI
IF SIN(90)=1 THEN Z=180
RETURN SGN(Y)*(Z-ATN(ABS(Y/X)))
FNEND
```

AUTOSTART

There is no direct analog to the AUTOSTART statement. Replace it with LOAD "AUTOST". You may wish to reset some system conditions to their power-on values before doing the LOAD. Consult the Master Reset Table in your Series 200 BASIC Lanugage Reference manual to determine what conditions will be set automatically by the LOAD.

Note that the Series 200 syntaxer will automatically change AUTOSTART to Autostart.

AXES

No change if your AXES statement has parameters. Note, however, that AXES with no parameters has a special definition on the Series 80. There is no corresponding definition on the Series 200.

Note also that the Series 80 automatically changes into GRAPH mode if it has been executing in ALPHA mode. The Series 200 does not automatically turn GRAPHICS ON and ALPHA OFF. You will need to do this explicitly.

AWRIT

Use OUTPUT 1;*<string exp>*.

See AREAD for a discussion of how to handle the fact that AWRIT does not move the current print position.

BEEP

The default values and allowable ranges of parameters are different on the Series 200. If you want to create a tone of a particular frequency or duration, consult the Series 200 BASIC Language Reference for details. If you are satisfied with the Series 200 defaults, no change is required.

BINAND

No change.

BINCMP

No change.

BINEOR

No change.

BINIOR

No change.

BIT

No change.

BLINK

Delete this statement. The Series 200 does not have this capability. Note that the Series 200 syntaxer will not have commented it out because it will have parsed it as an invocation of the user-supplied subprogram `Blink`.

BPLOT

You should be able to use the `Bload` compiled subprogram in the Series 200 BASIC Graphics Utilities package to produce the same result. This requires AP2.0, GRAPH2.1, and the compiled subprogram utility.

In order to apply `Bload`, you will need to put the data to be plotted in an INTEGER array rather than in a string. Form the string expression argument for BPLOT just as you would for your Series 80 program. Pass this expression and the INTEGER array to a routine which packs the first two characters of the string into the first element of the INTEGER array, the next two characters into the next element, etc. This routine can use the NUM function to convert the two string characters to integers, the SHIFT function to shift the integer resulting from the odd-indexed string character 8 bits to the left, and the BINAND function to pack the shifted integer and the other integer into a single integer.

You may need to insert a GRAPHICS ON statement.

BREAD

You should be able to use the `Bstore` compiled subprogram in the Series 200 BASIC Graphics Utilities package to produce the same result. This requires AP2.0, GRAPH2.1, and the compiled subprogram utility.

You will need to move the data you have read from the CRT from the INTEGER array used by `Bstore` into the string used by BREAD. Reverse the process described in BPLOT. Use SHIFT and ROTATE operations to break the an integer apart into two 8-bit integers, and use CHR$ to convert each of these to a character.

You may need to insert a GRAPHICS ON statement.

BTD

Use IVAL(*<string expr>*,2) or DVAL(*<string expr>*,2). This requires AP2.0.

CALL

If your CALL statement specifies the subprogram name as a string constant, remove the delimiting quote marks. If your CALL statement uses a string variable or expression, you will need to use OUTPUT 2 (OUTPUT to live keyboard) to create an acceptable Series 200 CALL statement. For example, if your Series 80 program contains

```
CALL Name$(A,B$)
```

replace it with

```
OUTPUT 2 ; "CALL "&Name$&"(A,B$)"&CHR$(255)&"X";
```

In either case, you must also be sure that the name in the CALL statement exactly matches the name in the SUB statement. The Series 200 does not truncate subprogram names to a fixed length.

The Series 80 automatically searches the current default mass storage device when it executes a CALL statement which specifies a subprogram not currently in memory. The Series 200 gives an error instead. It is your responsibility to be sure that all subprograms are present before they are invoked. You can use LOADSUB to bring in the needed subprograms. If you have AP2.0, you can use the LOADSUB FROM *<file specifier>* command before you begin program execution or you can use the OUTPUT 2 trick, as above, to embed LOADSUB FROM in a program statement. Alternately, you can use the LOADSUB *<subprogram name>* FROM *<file specifier>* statement before the CALL or in an ON ERROR routine. If you do not have AP2.0, you can use the LOADSUB ALL FROM *<file specifier>* statement. This may bring in more subprograms than you actually need. You can use DELSUB to selectively delete the extras.

Check your pass parameter list for arrays. You must change all array references of the form A() or A(,) to references of the form A(*).

Check the SUB statement to see if you have made any changes to the formal parameter list. If you have, then make the corresponding changes to the pass parameter list.

CAT

Replace the Series 80 mass storage unit specifier or volume label by the appropriate Series 200 mass storage unit specifier. The format of the output is somewhat different on the two systems.

If you execute CAT of a disc which contains Series 80 files, their file types will be encoded as follows:

Series 80 type	Series 200 type
BPGM	-8184
DATA	-8176
GRAF	-8180
PKEY	-8164
PROG	-8160

NULL files do not appear in the Series 200 CAT output.

ASCII files are the only Series 80 files which can be read directly by Series 200 BASIC. ASCII files have "****" in the type field of HP87 and "asci" in the type field of HP85 CAT output.

CCHR$

Replace CCHR$ by an invocation of a user-defined function which does a CONTROL to CRT (select code 1) registers 0 and 1 to reset the cursor position to the desired starting location and then reads characters from the CRT using ENTER 1.

CCLEAR

Replace CCLEAR by an invocation of a user-defined subprogram which uses CRT STATUS register 3 to determine the number of lines of CRT memory which are above screen. Your subprogram should then do CONTROL to CRT registers 0 and 1 to move the cursor to the beginning of CRT memory. Next it should output blanks to the entire CRT memory. Finally it should do CONTROL 1,0;1,1 to home the cursor.

Note that the Series 200 syntaxer will automatically change CCLEAR to Cclear.

CCPOS

Replace CCPOS with an invocation of a user-defined function which determines the current cursor position using CRT STATUS registers 0 and 1 and then computes a single-number address using these values.

Note that the Series 200 syntaxer will automatically change CCPOS to Ccpos.

CCURSOR

Replace CCURSOR with CONTROL to CRT (select code 1) registers 0 and 1.

CDISP

Replace CDISP by OUTPUT 1.

CEIL

Append the following code to your program and replace CEIL by FNCeil.

```
DEF FNCeil(X)
IF X=INT(X) THEN RETURN X
RETURN INT(X)+1
FNEND
```

Page missing from original document

CLEAR interface

If your CLEAR statement acts on an HP-IB interface, you can probably use the Series 200 CLEAR. Update the device specifier to reflect your Series 200 configuration. If your Series 80 CLEAR specifies multiple listeners, you will need to replace the multiple listener specification by the I/O path name associated with that multiple listener or issue a separate Series 200 CLEAR for each listener.

If your CLEAR statement acts on a GPIO interface, note that the Series 200 GPIO does not support primary addressing. You may use WRITEIO to register 1 to reset the GPIO interface.

CLEAR screen

Use OUTPUT 2; CHR$(255)&"K"; to clear the entire screen. This is equivalent to pressing the CLR SCR key. It does not clear the key label area. You must use CONTROL 1,12;1 to do this. This CONTROL operation requires AP2.0.

If you want to clear CRT memory below the current cursor position, write a subprogram which uses OUTPUT 1 to send blanks to the CRT. This subprogram can move the cursor to a given spot before clearing the screen by sending a CONTROL to registers 0 and 1 of select code 1.

CLINE

Replace CLINE with a CONTROL to CRT (select code) register 1.

CLIP

If your CLIP statement has parameters, no change is required unless you have specified the same value for the lower limit as for the corresponding upper limit. The Series 200 requires distinct lower and upper limits.

If your CLIP is parameterless, it is untranslatable.

CLOSE_KEY_FILE

See the appendix on MIKSAM.

CLPOS

Replace CLPOS with an invocation of a user-defined function which uses CRT STATUS register 1 to determine the current print line.

Note that the Series 200 syntaxer will automatically change CLPOS to Clpos.

CNORM

Be sure that the program segment which contains the CNORM has a reference to COM /Matrix/ (see the appendix on Matrices). Write a function which also references COM /Matrix/ and replace CNORM by an invocation of this function. Your function should determine the column norm (the value returned by the function) and the column number (Cnormcol) associated with this norm.

CNORMCOL

Provide the COM reference and function discussed in CNORM. Note that the Series 200 syntaxer will have changed CNORMCOL to Cnormcol, which is just what you want. This will reference the appropriate COM variable.

COM

Change all occurrences of SHORT to REAL in your COM declarations.

Be sure that all of your COM declarations observe the Series 200's limits on array and string sizes. If OPTION BASE is 0, the maximum upper bound for any dimension of an array is 32766. If OPTION BASE is 1, the maximum upper bound is 32767. The maximum string length is 32767. If these restrictions are not met, your program will need to be restructured.

Be sure that all of your COM lists are complete. The Series 80 allows programs which use a preserved COM area to specify only the needed portion of that COM. The Series 200 requires that the entire COM list be respecified. Otherwise, it will delete the entire old COM area and reallocate only the portion of COM which is specified by the new program. This will cause all of the old COM values to be lost.

Check carefully for the following subtle difference. In the Series 80, the type of a scalar or an array is the last type which was specified (or REAL if no type has been specified). In the Series 200, the appearance of a string or string array declaration in a COM list will cause the next numeric declaration to be REAL unless a type specification is explicitly given. As an example, consider

```
COM INTEGER A, B$[20], C
```

In the Series 80, C is an INTEGER, but in the Series 200, it is a REAL. In order to port your program properly, you will need to insert the keyword INTEGER before the variable C in the COM list.

CON

See MAT.

CONFIG

You can use INITIALIZE with a mass storage unit specifier of ":MEMORY" to create a memory volume. This requires AP2.1. Consult your Series 200 BASIC Language Reference manual for details. No volume label may be specified.

COMPEQ

Replace COMPEQ(*string1*,*string2*) with *string1*=*string2*. If AP2.0 is loaded, the string comparison will be done according to the collating sequence associated with the current LEXICAL ORDER IS character set.

COMPGE

Replace COMPGE(*string1*,*string2*) with *string1*>=*string2*. If AP2.0 is loaded, the string comparison will be done according to the collating sequence associated with the current LEXICAL ORDER IS character set.

COMPGT

Replace COMPGT(*string1*,*string2*) with *string1*>*string2*. If AP2.0 is loaded, the string comparison will be done according to the collating sequence associated with the current LEXICAL ORDER IS character set.

COMPLE

Replace COMPLE(*string1*,*string2*) with *string1*<=*string2*. If AP2.0 is loaded, the string comparison will be done according to the collating sequence associated with the current LEXICAL ORDER IS character set.

COMPLT

Replace COMPLT(*string1*,*string2*) with *string1*<*string2*. If AP2.0 is loaded, the string comparison will be done according to the collating sequence associated with the current LEXICAL ORDER IS character set.

COMPNE

Replace COMPNE(*string1*,*string2*) with *string1*<>*string2*. If AP2.0 is loaded, the string comparison will be done according to the collating sequence associated with the current LEXICAL ORDER IS character set.

CONTROL

The Series 200 CONTROL statement is very similar syntactically to the Series 80 CONTROL, but there are many differences in semantics. You will need to translate this statement very carefully. If your CONTROL statement uses variables rather than constants, translation may be impossible.

If your Series 80 CONTROL statement specifies an interface select code, replace it with the appropriate Series 200 select code. If it specifies a buffer, replace the string variable by the corresponding I/O path name.

You will need to check the documentation for both systems to determine which Series 200 register corresponds to your Series 80 register and how the Series 80 control byte needs to be modified for the Series 200. (For example, in the Series 80, a newly created buffer has a fill pointer value of 0. In the Series 200, the value is 1). If you do not find a CONTROL register correspondence, also check the Series 200 WRITEIO registers for the interface to see if they provide the needed functionality.

If your CONTROL statement contains a list of control bytes, you will probably need to replace it with a list of CONTROL statements each specifying one control byte. This is because the order of registers is often different.

CONVERT

There is no CONVERT statement in Series 200 BASIC. To specify character conversions, use the CONVERT attribute of the ASSIGN statement.

If your Series 80 CONVERT statement specifies an interface select code, you will need a Series 200 ASSIGN statement which assigns an I/O path name to that select code and specifies the character conversions (e.g., ASSIGN @Sc7 TO 7; CONVERT IN BY PAIRS A$). Note that the Series 200 will not apply conversions on ENTER and OUTPUT statements which specify interface select codes, only on those which specify I/O path names.

If your Series 80 CONVERT statement specifies a buffer, you can add a CONVERT attribute to the ASSIGN statement which replaced your IOBUFFER statement for that buffer, or you can replace the CONVERT statement with an ASSIGN statement which adds the CONVERT attribute without resetting the buffer pointers (this form of the ASSIGN does not have TO after the I/O path name).

You will need to beware of the restrictions which the Series 200 places on the string variable which holds the convert string. The lifetime of the string variable must be at least as great as the lifetime of the I/O path name for which it specifies the CONVERT.

If your Series 80 CONVERT statement specifies INDEX, you will need to rework your convert string. The Series 80 places the convert character for CHR$(*i*) in byte *i*+1 of the string, for 0<=*i*<=255. The Series 200 places the convert character for CHR$(*i*) in byte *i* of the string, for 1<=*i*<=255, and it places the convert character for CHR$(0) in byte 256.

COPY parameterless

On the HP85, COPY causes the alpha or graphics memory to be dumped to the internal printer, depending upon which mode is active. On the Series 200, use DUMP ALPHA or DUMP GRAPHICS to cause the dump to go to the current DUMP DEVICE IS printer or to the specified location.

COPY with parameters

To copy a single file, simply update the mass storage unit specifier part of each file specifier.

If you want to copy an entire disc without destroying the current contents of the target disc, do not use the Series 200 COPY *<msus>* TO *<msus>*. It first destroys the directory on the target disc. Instead, write a subprogram which uses CAT *<msus>* TO *<string array>* to help determine what files are on the source disc. Your subprogram can then COPY each file individually. This requires AP2.0.

If you don't mind destroying the contents of the target disc when copying an entire disc, you may use the Series 200 volume-to-volume COPY if you first change the two file specifiers appropriately.

COS

No change is required if your Series 80 trig mode is RAD or DEG. If your Series 80 trig mode is GRAD, apply FNGtr to the argument of COS before applying COS.

COT

Use 1/TAN(*<num exp>*) if your Series 80 trig mode is either RAD or DEG. This will be equivalent to the Series 80 COT(*<num exp>*) if your Series 80 program has DEFAULT OFF. It will give an error whenever the value of *<num exp>* is a multiple of $\pi/4$ radians (90 degrees).

If your Series 80 program has DEFAULT ON and you want to avoid these errors, write a function using TAN which returns machine infinity rather than giving an error. Note, however, that the value of machine infinity is different on the two systems.

If your Series 80 trig mode is GRAD, write a function to convert grads to radians and use it to convert *<num exp>* to radians before invoking TAN.

CPRINT

Delete this statement. It applies only to the internal printer of the HP-85.

CREATE

Use CREATE BDAT and update the file specifier appropriately. Note that the Series 200 cannot create logical records having an odd number of bytes. Also note that the Series 80 may create more logical records than were requested because it creates as many records as it can in the number of sectors required to create the requested records. For example, CREATE "MYFILE" 10,10 actually causes 25 records of ten bytes each to be allocated. The Series 200, on the other hand, creates only the requested number of records.

You may wish to adjust logical record sizes. Consult your Series 200 Programming Techniques manual for an explanation of how data is stored in BDAT files.

CREATE_KEY

See the appendix on MIKSAM.

CROSS

See MAT.

CRT IS

Untranslatable. Delete this statement. There is no way to redirect DISP output on the Series 200. LIST output may be redirected by specifying a destination in the LIST statement itself.

CRT OFF

Untranslatable. Delete this statement.

CRT ON

Untranslatable. Delete this statement.

CSC

Use 1/SIN(*<num exp>*) if your Series 80 trig mode is either RAD or DEG. This will be equivalent to the Series 80 CSC(*<num exp>*) if your Series 80 program has DEFAULT OFF. It will give an error whenever the value of <num exp> is a multiple of π radians (180 degrees).

If your Series 80 program has DEFAULT ON and you want to avoid these errors, write a function using SIN which returns machine infinity rather than giving an error. Note, however, that the value of machine infinity is different on the two systems.

If your Series 80 trig mode is GRAD, write a function to convert grads to radians and use it to convert *<num exp>* to radians before invoking SIN.

CSIZE

The Series 200 does not support the third parameter (slant) of the Series 80 CSIZE statement, so delete this. There is no way to achieve this effect on the Series 200.

The Series 80 uses different default values for CSIZE parameters depending upon whether the PLOTTER IS device is the CRT or an external plotter. The Series 200 uses one set of defaults across the board. You may need to insert a CSIZE statement after a PLOTTER IS which names an external plotter in order to get the proportions you expect in your labels.

Because the character height parameter of CSIZE is expressed in graphics display units and the CRT sizes vary considerably across models of the Series 80 and Series 200, you may need to experiment with the value in order to get the desired effect.

CSUM

See MAT.

CTAPE

Delete this statement. The Series 200 does not support tape operations. Note that the Series 200 syntaxer will have changed CTAPE to Ctape.

CURSCOL

Replace CURSCOL by a function which uses STATUS on register 0 of select code 1.

CURSOR

If your CURSOR statement is interpreted by your HP87 mainframe as opposed to by the Plotter ROM, use the WHERE statement in GRAPH2.1. Note that in the Series 200, the status is returned in a string variable, not in a numeric variable. If you need the pen status, you can obtain it from the first character of the status string.

If your CURSOR statement is interpreted by your HP85 or HP87 Plotter ROM, it returns the physical rather than the logical pen position. Series 200 BASIC does not have the ability to provide the physical pen position when it differs from the logical position. Your program will need to be restructured.

CURSROW

Replace CURSROW by a function which uses STATUS on register 1 of select code 1. Note that on the Series 200, a value of 1 represents the first visible (on-screen) line, not the first line of CRT memory.

CWRITE

Replace CWRITE by OUTPUT 1. Note that control characters (ASCII codes 0-31) will be handled differently by the Series 200.

DATA

The Series 80 allows double quotes (") to appear in unquoted strings in a DATA list, but the Series 200 does not. You will need to replace each embedded " by "" and also enclose the string in double quote marks. For example,

```
DATA 10,20,HE SAID "NO."
```

should become

```
DATA 10,20,"HE SAID ""NO."""
```

Also, the Series 80 allows any number of "+" and "-" signs to precede the digits of a number, but the Series 200 number builder allows only one.

DATE

Write a function using TIMEDATE to compute the date and express it in your desired format. Replace DATE by an invocation of your function.

Note that the origin of the clock is different on the two systems. Hence, if you have not explicitly set the clock time, different results will be obtained. Also, if your Series 200 machine has powerfail protection, the clock will continue to run even when the machine is off.

DATE$

Write a function which uses DATE$(TIMEDATE) to compute the date. This requires AP2.0. This function can also reformat the result any way you wish. See DATE.

DEF FN single-line

The Series 200 does not support single-line functions. You will need to convert your single-line functions to multi-line functions. First, create two copies of your function definition at the end of your program, and then delete the original function definition. See the *Simplifying Your Program* section of the previous chapter for an easy way to do this. From the first copy of the function, remove the equal sign and everything to its right. Also remove the exclamation mark which had been added by the Series 200 syntaxer and press the [ENTER] key. If you get a syntax error, see DEF FN multi-line below for more changes that are necessary. From the second copy, remove everything from the exclamation point through the equal sign, and insert the keyword RETURN before the expression which constitutes the function definition. Again press the [ENTER] key. Finally, add an FNEND statement immediately following the RETURN statement.

See below for more details.

DEF FN multi-line

There is a major difference in the way the two systems handle functions. In the Series 80, a function shares the main program's environment, but in the Series 200, a function has its own environment. This means that you will need to do some program restructuring to get your functions to work.

First, you must rearrange the order of the lines in your program so that each function definition occurs after the main program's END statement. You can use the MOVELINES command in AP2.0 to do this. Note that since MOVELINES automatically renumbers program lines as needed, you want to perform this operation only after you have already modified as much of the rest of your program as possible. This is because line numbers which appear in program lines which have been commented out by the syntaxer will not be updated by the renumbering operation.

Consider the example Series 80 program, which appears on the Series 200 with a number of lines commented out:

```
 10 I=1+FNX
 20 DEF FNX
 30 ! FNX=5
 40 ! FN END
 50 J=2+FNY
 60 DEF FNY
 70 ! FNY=3
 80 ! FN END
 90 PRINT "DONE"
100 END
```

First, execute MOVELINES 50 TO 15 to give

```
 10 I=1+FNX
 15 J=2+FNY
 20 DEF FNX
 30 ! FNX=5
 40 ! FN END
 60 DEF FNY
 70 ! FNY=3
 80 ! FN END
 90 PRINT "DONE"
100 END
```

Next, execute MOVELINES 90,100 TO 16 to give

```
 10 I=1+FNX
 15 J=2+FNY
 16 PRINT "DONE"
 17 END
 20 DEF FNX
 30 ! FNX=5
 40 ! FN END
 60 DEF FNY
 70 ! FNY=3
 80 ! FN END
```

The program lines are now in an order acceptable to your Series 200 machine. See FN and FN

END later in this chapter for an explanation of what to do with the lines which are commented out.

In addition, you must determine which variables used by the main program are also used by the function. Those which are passed as parameters will pose no problem, but those which are not in the parameter list will need to be added to it. The only limit on the number of parameters which can be passed to a Series 200 function is imposed by the statement length limit of 100 or 160 characters. If you exceed this limit, you may be able to shorten variable names to avoid the problem. An alternate solution is to construct a labeled COM block which contains the variables which are needed by the function. Put the COM statements for this COM in both the main program and your function. Remember to delete any existing declarations of these variables from your main program. You may encounter difficulties with the COM approach if you need to pass the same variables to more than one function.

The easiest way to determine which variables need to be added to the function's parameter list or put into a labeled COM block is to first move the function to the end of your program and then to use the XREF command in AP2.0 or the XREF utility in the Series 200 BASIC Utilities Library to generate a cross-reference. This cross-reference will give a separate variable lists for the main program and the function.

You will also need to remove string length declarations from the parameter list in your DEF FN statement. In the Series 200, strings are always passed by reference, so string formal parameters "inherit" the length of the corresponding pass parameters.

If your function does any buffer operations, you will need to modify your parameter list. If the function does not do any string operations on the buffer, you can replace the string name by an I/O path name. If the function does both string operations and buffer operations, you will need to add the I/O path name to the parameter list.

There are a number of additional complications which arise in converting Series 80 functions for use by Series 200 computers. One is that OPTION BASE is automatically reset to 0 for the declarations within the function. If your main program used OPTION BASE 1, be sure to insert this statement in the function also. Another is that GOSUBs and GOTOs in the main program must reference only lines which remain in the main program, and similarly, GOSUBs and GOTOs in the function must reference only lines which are in the function. Any violations of this rule will be reported as errors when the program is prerun. A related potential problem is that end-of-line branching which is defined in the main program (e.g., ON INTR) and which uses GOSUB or GOTO is disabled while the function is executing. The occurrence of the event is logged, but the interrupt routine cannot be serviced until execution returns to the main program. Yet another problem area is that if your function contains any READ statements, you will need to be sure that the appropriate data is moved along with the function statements. This may involve breaking apart DATA statements which occur in the main program.

DEFAULT OFF

Delete this statement. The Series 200 always has default off.

DEFAULT ON

Delete this statement. The Series 200 does not have the ability to automatically check for overflows—it always uses DEFAULT OFF. If your program uses this option, it may need to be reworked.

DEG

No change. Note, however, that changes to the trig mode are global on the Series 80 and local on the Series 200. There is no problem with invoking subprograms because on the Series 200, the subprogram inherits the trig mode of the calling program. However, there is a problem with returning from a subprogram. For example, if a subprogram executes a DEG statement, the Series 80 will remain in degree mode when execution returns to the calling program, but the Series 200 will reset the trig mode to the mode which was in effect when the subprogram was invoked. You may need to insert some additional trig mode changes in your program to make it work properly.

DELETE_KEY

See the appendix on MIKSAM.

DET

No change. This requires AP2.0. Note that the Series 200 cannot handle empty arrays.

DETL

Change DETL to DET. This requires AP2.0. Note that the Series 200 syntaxer will have changed DETL to Detl.

Since the Series 80 DETL function is affected by the last use of MAT . . . SYS, as well as by DET, it is your responsibility to implement MAT . . . SYS so as to make this happen. See MAT for details.

DIGITIZE

DIGITIZE requires GRAPH2.0. Delete the third parameter, pen status. The Series 200 DIGITIZE statement does not return this information.

Note that the Series 80 DIGITIZE reads a point from the PLOTTER IS device, while the Series 200 DIGITIZE reads from the GRAPHICS INPUT IS device. You may need to precede your DIGITIZE statement by a GRAPHICS INPUT IS statement which specifies the proper device. (If you have AP2.0, you can determine the current PLOTTER IS device using SYSTEM$.)

DIM

You must make sure that the upper bound for any array dimension does not exceed 32766 if OPTION BASE 0 is in effect, or 32767 if OPTION BASE 1 has been specified. You must also make sure that no string length greater than 32767 is specified.

DIRECTORY

There is no direct Series 200 analog to the Series 80 DIRECTORY statement. If you are interested in the names of the programs but don't need size information, you may use the XREF command found in AP2.0. Since XREF is not programmable, this must appear in your program as

```
OUTPUT 2;"XREF"&CHR$(255)&"X";
```

This will cause the XREF output to be directed to the PRINTER IS device. XREF may be replaced by XREF #device selector if some other destination is desired. It is not practical to direct XREF output to the CRT because it will fly by too quickly to be read.

If you want both names and size information, do a STORE to some temporary file, CAT that file (this requires AP2.0), and finally, PURGE the file. Here you run the risk of not having sufficient disc space to create the temporary file.

DISC FREE

Use CAT TO a string array, extract the number of unused records from its footer line, and use VAL to convert it to a number. This requires AP2.0. The information can also be obtained from the DIRECTORY utility in your Series 200 BASIC Utilities Library, but this cannot be incorporated in a program.

DISP

The Series 200 DISP statement causes the output to be directed to the display line of the CRT. If you want your output directed to the main portion of the CRT, insert PRINTER IS 1 and change DISP to PRINT. Add another PRINTER IS after the DISP if you need to redirect your printed output to another destination. You may use OUTPUT 1 instead (without needing to worry about PRINTER IS), but realize that OUTPUT treats comma and semicolon separators identically and that TAB is not allowed in an OUTPUT statement.

In the Series 200, the tabbed fields are ten columns apart, not 21 columns apart as in the Series 80. If your Series 80 machine has an 80-column screen and you want to preserve the appearance of your Series 80 DISP, you may insert TAB(21), TAB(42), and TAB(63) in your PRINT list. This will work as long as none of the quantities displayed overflows one field.

Trailing punctuation does not cause the DISP output to be deferred on the Series 200. Instead, it causes suppression of the EOL sequence.

Initial punctuation is not allowed on the Series 200. You can insert a pair of double quote marks ("") before the leading punctuation to get around this.

DISP USING

See DISP and IMAGE.

DIV

Parenthesize the second argument if it begins with a negative sign.

DOT

No change. This requires AP2.0. Note that the Series 200 cannot handle empty matrices.

DRAW

No change. You may need to insert a GRAPHICS ON statement.

DTB$

Use IVAL$(*<num exp>*,2) or DVAL$(*<num exp>*,2). This requires AP2.0.

DTH$

Use IVAL$(*<num exp>*,16) or DVAL$(*<num exp>*,16). This requires AP2.0.

DTO$

Use IVAL$(*<num exp>*,8) or DVAL$(*<num exp>*,8). This requires AP2.0.

DTR

Append the following code to your program and replace DTR by FNDtr.

```
DEF FNDtr(X)
RETURN PI*X/180.
FNEND
```

DUMP ALPHA

Delete both parameters.

The Series 80 DUMP ALPHA output goes to the PRINTER IS device. The Series 200 output goes to the DUMP DEVICE IS device unless a destination has been explicitly specified. You may need to precede your DUMP ALPHA statement with a DUMP DEVICE IS statement.

DUMP GRAPHICS

See DUMP ALPHA.

The Series 200 does not rotate the dump output unless EXPANDED has been specified in the DUMP DEVICE IS statement.

ENABLE AS-ST

Untranslatable. Delete this statement.

ENABLE INTR

Update the select code to reflect your Series 200 configuration. Also, check the documentation for your interface card to determine the proper value for the mask byte.

ENABLE KBD

The Series 200 does not have the ability to mask keys while the system is waiting for INPUT (bits 0-3 of the mask byte).

To mask keys while the program is running, you can use an ON KBD statement to trap the keys and store them in the keyboard buffer. You can use the KBD$ function to retrieve the contents of the buffer and write code to determine which keys were trapped and what to do about it.

END

Every Series 200 main program must have an END statement as its last statement (other than comments). The END statement must occur before any SUB or DEF FN statements. No other END statements are permitted. If necessary, insert an END as the last statement of your main program. Change all other END statements to STOP.

ENTER

If your Series 80 ENTER statement uses a device specifier, replace the device specifier with the appropriate Series 200 device specifier or with the I/O path name associated with it. Remember that if you want a CONVERT IN to be applied, you must use the I/O path name.

If your Series 80 ENTER statement specifies a buffer, replace the string variable with the I/O path name associated with that buffer. Note that Series 200 buffers are circular. In order to prevent the system from wrapping around to the beginning of the buffer once it has read the rightmost byte, precede the ENTER statement with a STATUS statement which determines the current fill pointer location and a CONTROL statement which sets the fill pointer to 1. After the ENTER, use another CONTROL statement to restore the fill pointer to the location read by the STATUS statement.

EPS

The Series 200 syntaxer will convert EPS to Eps, which the system will interpret to be a REAL scalar. You may insert the statement Eps=2.225073858507202E-308 (the smallest REAL representable on the Series 200) prior to any of your references to EPS. Note that this is not the value of EPS used by the Series 80. This may cause some of your computations to produce different results. Note also that if you fail to initialize Eps, the system will set it to 0.

ERASETAPE

Delete this statement. The Series 200 does not support tape mass storage. Note that the Series 200 syntaxer will have converted ERASETAPE to Erasetape. A comparable disc operation is INITIALIZE.

ERRL

On the Series 200, the ERRL function requires a line number or line label as a parameter. It returns 1 if the most recent error was in that line, and 0 otherwise.

You will need to rework all references to ERRL. Those which are of the form IF ERRL=*xxx* THEN ... can be changed to IF ERRL(*xxx*) THEN ... .

As an alternative, if you have AP2.0, you can write a function to extract the line number from ERRM$.

ERRM

Replace this statement by one which does a DISP, PRINT, or OUTPUT of ERRM$. This requires AP2.0. Note that the Series 200 syntaxer will have changed ERRM to Errm.

ERRN

The error numbering schemes are totally different on the two systems. All of your tests for particular error numbers will need to be updated. Consult your BASIC Language Reference for a list of Series 200 error numbers.

ERROM

Untranslatable. It would be best to delete all references to ERROM. Note that the Series 200 syntaxer will convert ERROM to Errom, a REAL scalar. At prerun time, Errom will get the value 0.

ERRSC

Replace ERRSC by an invocation of a user-defined function which uses LEN(VAL$(ERRDS)) (this requires AP2.0). If the length is odd, extract the leftmost digit. If it is even, extract the leftmost two digits. Then apply VAL to the extracted digit(s).

Remember to change all tests on the value of ERRSC to reflect your Series 200 configuation.

Note that the Series 200 syntaxer will have changed ERRSC to Errsc.

EXOR

No change.

EXP

No change. Note, however, that the range of allowable arguments is smaller on the Series 200 than on the Series 80.

FIND

Replace FIND with an invocation of a user-defined function which search the argument string for the first character whose ASCII code is greater than 127.

FLAG

See the appendix on Flags.

FLAG$

See the appendix on Flags.

FLIP

Use OUTPUT 2;CHR$(255)&"U";. Note that the Series 200 syntaxer will have changed FLIP to Flip.

FLOOR

Change FLOOR to INT. Note that the Series 200 syntaxer will have changed FLOOR to Floor.

FNORM

Write a function to do this. Note that the Series 200 cannot handle empty arrays.

FNfunction name[$] (function invocation)

Check the DEF FN statement to see if you have made any changes in the formal parameter list. If you have, make the corresponding changes in the pass parameter list.

FNfunction name[$] (returning a value)

Replace

```
FNfunction name[$] = <exp>
```

by

```
RETURN <exp>
```

See RAD for a discussion of the effect of subprograms (including functions) on trigonometric computations.

FN END

Change FN END to FNEND.

FOR

There are a number of differences in the implementations of FOR loops. You will need to examine your FOR statements and FOR loops carefully to see if any changes are required. Some differences may not become apparent until the program is run.

One very fundamental difference is that, in the Series 200, there must be exactly one NEXT statement for each FOR statement. FORs and NEXTs are statically matched at program prerun. If they do not match, a prerun error is reported. If you have more than one NEXT for any FOR, you will need to re-write the FOR loop to eliminate the extra NEXTs. (If you have no NEXT for some loop, you will need to supply one.)

There are several conditions which can cause FOR loops to execute a different number of times on the two systems. The known conditions are:

1. The loop counter variable is used to determine the final or step value for the loop (e.g. `FOR I=1 TO I+2`).

2. The expression which is evaluated to determine the final or step value for the loop contains a side effect which modifies the loop counter variable (e.g. `FOR I=1 TO FNX(I)` where FNX modifies I).

3. The loop counter variable and step values are not INTEGERs, and rounding errors occur (e.g., `FOR I=1 TO 2 STEP .1` where I is REAL).

4. The initial value given to the loop counter is beyond the final value specified (e.g., FOR I=20 TO 10 STEP 1 or FOR I=10 TO 20 STEP -1). In such cases, the Series 80 will execute the loop one time, and the Series 200 will not execute it at all.

FP

If the argument of FP is guaranteed to be positive, you can replace FP by FRACT, found in AP2.0. If not, append the following code to your program and replace FP by FNFp.

```
DEF FNFp(X)
RETURN SGN(X)*(ABS(X) MOD 1)
FNEND
```

FRAME

No change.

Note that the Series 80 automatically switches into GRAPH mode if it had been executing in ALPHA mode when FRAME is encountered. The Series 200 does not automatically turn GRAPHICS ON, so the frame will be drawn, but not displayed unless you explicitly execute GRAPHICS ON. You may also execute ALPHA OFF if you wish.

FRE

Replace FRE by VAL(SYSTEM$("AVAILABLE MEMORY")).

FXD

Untranslatable. Delete this statement.

GCLEAR

Note: The pen numbers referred to in this discussion are those used by the HP87. If you have an HP85, all negative pen numbers map to PEN -1, and all positive pen numbers and 0 map to PEN 1.

If your GCLEAR statement has no parameter and you are using PEN 1, no change is required. If it has a parameter, you may delete it and clear the entire screen, or you may write and invoke a subprogram using the Series 200 GSTORE, zeroing out the proper portion of the array holding the CRT image, and then doing a GLOAD.

If your Series 80 program uses PEN -1 or PEN -2, use the GSTORE, GLOAD technique described in the previous paragraph, but set the array elements to −1. If your GCLEAR statement has no parameter, you may omit the GSTORE operation and set every element of the array to −1. Otherwise, set only the required portion.

If your Series 80 program uses PEN 0, GCLEAR fills the graphics screen with the current background color. Use either the Series 200 GCLEAR or the subprogram described above, as needed.

You may need to add a GRAPHICS ON statement either before or after your GCLEAR.

GET$

Remove the keyword GET$ and the set of parentheses associated with it. For example, GET$(A$(3)[5,10]) becomes A$(3)[5,10].

GLOAD

Both systems have GLOAD statements, but their semantics are quite different. On the Series 80, GLOAD reads the graphics screen image directly from a GRAF file. On the Series 200, GLOAD reads the graphics screen image from an array in main memory. If you wish to emulate the Series 80 GLOAD on the Series 200, you can do so by reading your data from a BDAT or ASCII file, since the Series 200 does not support the Series 80 file type GRAF. The BDAT representation (with FORMAT OFF) is more compact, so it is probably your method of choice. Guidelines for how to transport a GRAF file to the Series 200 are given in the last chapter of this document.

The following approach will allow you to emulate the Series 80 GLOAD if you have already transported your GRAF file to the Series 200 or emulated GSTORE and thereby created a file which contains a graphics CRT image. First ALLOCATE an INTEGER array large enough to hold the screen image. This requires 7500 elements on the HP9816 and HP9826 and on the HP9920 with a monochromatic monitor, 12480 elements on the HP9836A, 49152 elements if using an HP98627A interface, and 49920 elements on the HP9836C. Use an ASSIGN statement to open the file which contains your graphics data, then use ENTER to write the file to your array. Next, do a Series 200 GLOAD to write the contents of this array to the graphics CRT. Finally, unless you are going to be doing frequent GLOAD and GSTORE operations, DEALLOCATE the large array to recover your RAM.

GOSUB

No change. Note, however, that the line specified by the GOSUB must be in the same program segment as the GOSUB statement. For example, a GOSUB in the main program cannot reference a line in a function.

GOTO

No change. Note, however, that the line specified by the GOTO must be in the same program segment as the GOTO statement. For example, a GOSUB in the main program cannot reference a line in a function.

GRAD

GRAD mode is not supported by the Series 200.

Replace the GRAD statement with a RAD statement. Note that the Series 200 syntaxer will have converted GRAD to Grad. If you want your program to continue to behave as though your trigonometric computations are being done in grad mode so that you can avoid major program restructuring, append the following functions to your program.

```
DEF FNGtr(X)
RETURN PI*X/200
FNEND
```

```
DEF FNRtg(X)
RETURN 200*X/PI
FNEND
```

Insert calls to these functions as appropriate in your code. For example, replace all occurrences of SIN(*<num exp>*) by SIN(FNGtr(*<num exp>*)). Replace all occurrences of ASN(*<num exp>*) by FNRtg(ASN(*<num exp>*)).

See RAD for an explanation of problems which may occur if you change your trig mode in a subprogram.

GRAPH

Replace by the two-statement sequence

```
ALPHA OFF
GRAPHICS ON
```

If your Series 80 program had been in ALPHALL or GRAPHALL mode, add GINIT.

GRAPHALL

Replace by the three-statement sequence

```
ALPHA OFF
GRAPHICS ON
GINIT
```

If your Series 80 program is already in GRAPHALL mode, omit GINIT.

GRAPHICS

Replace by the two-statement sequence

```
ALPHA OFF
GRAPHICS ON
```

If your Series 80 program had been in ALPHALL or GRAPHALL mode, add GINIT.

GRID

No change. Note, however, that GRID with no parameters has a special definition of the Series 80. There is no corresponding definition on the Series 200.

You may need to add a GRAPHICS ON statement immediately before or after your GRID statement.

GSTORE

Both systems have GSTORE statements, but their semantics are quite different. On the Series 80, GSTORE writes the graphics screen image directly to a file. On the Series 200, GSTORE writes the graphics screen image to an array in main memory. If you wish to emulate the Series 80 GSTORE on the Series 200, you can do so by writing your data to a BDAT or ASCII file, since the Series 200 does not support the Series 80 file type GRAF. The BDAT representation (with FORMAT OFF) is more compact, so it is probably your method of choice.

The following approach will work. First ALLOCATE an INTEGER array large enough to hold the screen image. This requires 7500 elements on the HP9816 and HP9826 and on the HP9920 with a black and white monitor, 12,480 elements on the HP9836A, 49,152 elements if using the HP98627A interface card, and 49,920 elements on the HP9836C. Do a Series 200 GSTORE to write the CRT image to this array. Create a file large enough to hold the array, and open this file via an ASSIGN statement. Use OUTPUT to write the array to the file. Finally, unless you are going to be doing frequent GLOAD and GSTORE operations, DEALLOCATE the large array to recover your RAM.

HALT

Replace HALT by ABORTIO and replace the interface select code by the I/O path name of the device or select code involved in the TRANSFER you wish to terminate. This requires AP2.0.

HGL$

There is no direct analog to the Series 80 HGL$ function.

The Series 80 HGL$ function produces a string in which each character is different from the corresponding character in the function's argument. This string is the same length as the argument string.

On the Series 200, you may underline an output string by preceding the string in the output stream with the special character CHR$(132). Be sure to follow the string with CHR$(128) to turn off underlining. You may also concatenate CHR$(132) and CHR$(128) to the front and back of the string respectively, and store this new string in memory. However, you will need to be very careful if you do any length checks or character comparisons on this new string since it will be considerably different in composition from the result of HGL$.

Not all Series 200 computers have the ability to display underlined characters.

HMS

Use the Series 200 TIME function. This requires AP2.0. Note, however, that TIME operates over a one-day range and HMS over a four-day range.

HMS$

Use the Series 200 TIME$ function. This requires AP2.0. Note, however, that TIME$ operates over a one-day range and HMS$ over a four-day range.

HTD

Use IVAL(*<string expression>*,16) or DVAL(*<string expression>*,16). This requires AP2.0.

IDN

See MAT.

IDRAW

No change.

You may need to add a GRAPHICS ON statement immediately before or after your IDRAW.

IF

If your IF statement contains no ELSE clause and has only a statement number or label following the THEN, no change is required. If a statement follows the THEN, check the entry for that statement to see whether it requires any changes.

If your IF statement has more than one statement after the THEN but has no ELSE, you will need to replace it by a multi-line construct, as follows:

```
IF <numeric expression> THEN
  then part, one statement per line
END IF
```

If your IF statement contains an ELSE, rework it as follows:

```
IF <numeric expression> THEN
  then part, one statement per line
ELSE
  else part, one statement per line
END IF
```

IMAGE

The Series 200 does not support the C or P image specifiers in any way. Support for the R and * specifiers is found in AP2.0.

Change the e specifier to E and change E to ESZZZ.

Note that the / may not be used as a delimiter on the Series 200. It must be separated from the adjacent specifiers by commas.

You may need to adjust your termination conditions if you use # or %.

IMOVE

No change. See PDIR.

INF

The Series 200 syntaxer will convert INF to Inf, which the system will interpret to be a REAL scalar. You may insert the statement Inf=1.79769313486231E+308 (the largest REAL representable on the Series 200) prior to any of your references to INF. Otherwise, the system will initialize Inf to 0. Note that the value above is not the value the Series 80 uses for INF. This may affect some of your computations.

INITIALIZE

The Series 200 INITIALIZE statement cannot be used to assign a volume label or a directory size to a mass storage medium. It can only be used to specify the interleave factor. (The other functions of the Series 80 INITIALIZE are performed by the INITIALIZE utility found in your Series 200 BASIC Utilities Library).

To convert your INITIALIZE statement, delete the new volume label and directory size parameters and any commas immediately following them. Change your mass storage unit specifier to be appropriate for the Series 200. You must include a mass storage unit specifier for the Series 200. For INITIALIZE, it will not assume the current default device. In addition, you may wish to specify a new interleave factor. The default interleave factor used by the Series 200 is device dependent. A table of the defaults can be found under INITIALIZE in your Series 200 BASIC Language Reference manual.

INPUT

No change is required in the INPUT statement itself. However, the method of responding to INPUT prompts is different in the two systems. Be sure to read INPUT in your Series 200 BASIC Language Reference manual for details.

INT

No change.

INTEGER

The Series 200 allows a maximum upper bound of 32766 if OPTION BASE is 0 and 32767 if OPTION BASE is 1. If your arrays are larger than this, you will need to restructure your program.

INV

See MAT.

IOBUFFER

Replace the IOBUFFER statement with an ASSIGN statement which associates an I/O path name with the string variable. Remember that the lifetime of the string variable must be at least as long as the lifetime of the I/O path name.

If your program was written for an HP85, you will also want to adjust the declaration of the string. You will need to reduce its size by eight bytes (these bytes were used for buffer management, not for data). This may cause problems if you ever use the string as a string rather than as a buffer.

Note that buffers are managed somewhat differently by the two systems. Consult your Series 200 BASIC Interfacing Techniques manual for details.

IP

Replace IP(*<num exp>*) by (*<num exp>* DIV 1).

IPLOT

No change. See PDIR.

You may need to insert a GRAPHICS ON statement.

ISLANG

Untranslatable. Delete references to this function.

KBD$

No change. Note, however, that in the Series 200, it is not possible to apply KBD CONVERT. You will need to write your own conversion routine and apply it to the results of KBD$.

KDB CONVERT

Untranslatable. Delete this statement. See KBD$ above.

KEY DOWN

Untranslatable. Delete all references to this function.

KEY LABEL

Use CONTROL 1,12;2 to turn on the display of the soft key labels. (This is the default condition on the Series 200). This requires AP2.0.

The Series 80 KEY LABEL statement also sets the print position to the upper left corner of the CRT. If you wish to do this, use CONTROL 1,0;1,1.

KEYLAG

Replace the KEYLAG statement with CONTROL 2,3;*repeat speed*,*wait parameter*. Note that the parameters occur in the opposite order in the KEYLAG and CONTROL statements. You may need to change the values somewhat to achieve the conditions you want.

KILL_KEY_FILE

See the appendix on MIKSAM.

LABEL

The Series 200 does not allow the TAB function to appear in LABEL lists.

The Series 200 cannot slant labels. See CSIZE.

The Series 200 clips labels at the current clip boundary, and the Series 80 does not, so you may need to precede your LABEL statement with CLIP OFF and follow it with CLIP ON in order to get the output you expect.

You may need to add a GRAPHICS ON statement.

LABEL USING

See LABEL and IMAGE.

LANG

Write a function which reads STATUS register 8 of select code 2. This function must then map the result to the corresponding Series 80 keyboard number. Replace LANG by an invocation of this function.

LANG OFF

Untranslatable. Delete this statement.

LANG ON

Untranslatable. Delete this statement.

LANG$

Replace LANG$ by SYSTEM$("KEYBOARD LANGUAGE"). This requires AP2.0.

LAXES

The Series 200 does not support LAXES. You will need to use AXES and label them explicitly.

You may need to add a GRAPHICS ON statement.

LBND

Change LBND to BASE. This requires AP2.0.

LDIR

If your LDIR statement has one parameter, no change is required unless your Series 80 trig mode is grads. If your trig mode is grads, apply FNGtr to the parameter of LDIR (see GRAD).

If your LDIR has two parameters, convert the rise and run to an angle. The function which replaces ATN2 can be used.

LEN

No change.

LESCAPE IS

Untranslatable. Delete this statement.

LESCAPE$

Untranslatable. Delete all references to this function.

LET

You will need to create *n* separate LET statements wherever you have a LET statement with *n* variables to the left of the equal sign. You will want to be sure that the leftmost variable name occurs in the first of these LET statements, etc., to eliminate the possibility of introducing side effects from breaking up statements like

```
LET A(I),I = 5
```

You will also need to change expressions of the form A=B=C to A=(B=C).

LEX$

Untranslatable. Delete all references to this function.

LGRID

The Series 200 does not support LGRID. You will need to label grids explicitly.

LGT

No change.

LIMIT

If your PLOTTER IS device is the CRT, LIMIT is untranslatable. You may be able to use CLIP or VIEWPORT to restrict the plotting area, but the effect will be different.

If your PLOTTER IS device is an external plotter, consult the Graphics chapter of your Series 200 BASIC Programming Techniques manual and the documentation for your plotter. You may be able to restrict the plotting area by setting the plotter's P1 and P2 either via your program or manually.

LINETYPE

See LINE TYPE.

LINE TYPE

The syntax of LINE TYPE is the same on the two systems, but there are many subtle differences in semantics. Check the documentation for both systems and for your external plotter for details. You will probably need to experiment a little to get just the effect you want on the Series 200.

LINPUT

The Series 80 allows the prompt to be specified by a string expression, while the Series 200 requires a string constant for the prompt. To achieve the effect of a string expression prompt, replace the string expression with "". Immediately before the LINPUT, insert a DISP statement which displays the value of the prompt expression. Be sure to end the DISP statement with a semicolon.

LIST

The Series 200 directs the output of a LIST which specifies no destination to the PRINTER IS device, rather than to the CRT. Use LIST #1 to direct the output to the CRT.

Note also that the Series 200 does not have the ability to list a single CRT's worth of program.

LLWC$

Replace LLWC$ by LWC$. When AP2.0 is present, LWC$ always uses the collating sequence of the current LEXICAL ORDER IS.

LOADBIN

Delete this statement. Series 80 binary programs are not usable by the Series 200.

To load a Series 200 binary program, you must use the LOAD BIN command (which must be executed from the keyboard). Some Series 200 binaries are non-scratchable (e.g. AP2.0) and must be loaded when the PHYREC binary is not in the machine.

LOAD BIN

See LOADBIN.

LOCAL

Update the device specifier(s) to reflect your Series 200 configuration. Note that the bus sequences are somewhat different in order on the two systems.

LOCAL LOCKOUT

Update the device specifier to reflect your Series 200 configuration.

LOCATE

The Series 200 syntaxer will automatically convert LOCATE *xmin*,*xmax*,*ymin*,*ymax* to VIEWPORT *xmin*,*xmax*,*ymin*,*ymax*. In addition, you need to add

```
WINDOW xmin,xmax,ymin,ymax
```

immediately following the VIEWPORT. If you have specified a maximum value less than the corresponding minimum value in order to set up a reflected plot, you must change the VIEWPORT statement so that the minimum is less than the maximum. Leave the parameters in the WINDOW statement in the same order as in the LOCATE.

Note that since the CRT sizes vary greatly on the Series 80 and Series 200 machines, you may need to experiment with the bounds you set in order to get the effect you want.

The parameterless version of LOCATE is untranslatable.

LOG

No change.

LORG

No change is required if the label position evaluates to an integer in the range 1 through 9. Note however, that the Series 200 gives an error for numbers outside of this range, while the Series 80 uses LORG 1.

LPRINT

Untranslatable. Delete this statement. To achieve the effect of a user-defined mapping scheme, define an I/O path name which corresponds to your printer. Give it the CONVERT OUT BY PAIRS attribute, and define the mapping by the convert string. This works only for one-to-one mappings. If you need to do one-to-many or many-to-one mappings, you will need to write your own "filter" routine and pass all printer output through this filter before sending it to the printer.

LUPC$

Replace LUPC$ by UPC$. When AP2.0 is present, UPC$ always uses the collating sequence of the current LEXICAL ORDER IS.

LWC$

No change. This requires AP2.0. Note, however, that the Series 80 LWC$ function does not change upper case letters which also have character highlights, but the Series 200 does. This is because the two machines use different representations for highlighted characters. See HGL$ for details. Note also that in the Series 200, LWC$ always uses the collating sequence of the current LEXICAL ORDER IS, while in the Series 80, it uses the ASCII collating sequence. If an international keyboard is in use, it may be necessary to precede the LWC$ by LEXICAL ORDER IS ASCII and follow it by another LEXICAL ORDER IS.

MAKE_KEY_FILE

See the appendix on MIKSAM.

MASS STORAGE IS

Replace your Series 80 volume label or mass storage unit specifier by the appropriate Series 200 mass storage unit specifier.

Check your Series 200 BASIC manuals for information on determining the default mass storage unit specifier.

MAT

There is a large number of statements which begin with the single keyword MAT. Each type is identified and explained below. Consult other entries for statements such as MAT DISP which have a secondary keyword.

The Series 200 cannot handle empty arrays.

MAT result array = (numeric expression)

No change.

MAT result array = operand array

No change.

MAT result subarray = operand subarray

Write a subprogram to do this.

MAT result array = -operand array

Use MAT *result array* = (-1)***operand array*.

MAT result array = oparray 1 op oparray 2 where op can be +,-,.,/,or *.

No change.

MAT result array = (scalar) op oparray where op can be +,-,*, or /.

No change.

MAT result array = (scalar)*oparray1+(scalar)*oparray2

Break into several statements, as follows. Be sure, also, to include declarations for your temporary arrays.

```
MAT temparray1 = (scalar)*oparray1
MAT temparray2 = (scalar)*oparray2
MAT result array = temparray1+temparray2
```

MAT result array = CON[(redim subscripts)]

Replace this statement by the two-statement sequence

```
REDIM result array (redim subscripts)
MAT result array = (1)
```

MAT result vector = CROSS(opvector1,opvector2)

Write a subprogram to compute the cross product and replace this statement by an invocation of your subprogram.

MAT result matrix = IDN[(redim subscripts)]

Replace by the two-line sequence

```
REDIM result matrix(redim subscripts)
MAT result matrix = IDN
```

MAT result matrix = INV (operand matrix)

No change.

MAT result array = INV (oparray1)*oparray2

Break into two statements, as follows. Be sure to declare your temporary array.

```
MAT temp = INV (oparray1)
MAT result array = temp*oparray2
```

MAT result array = RSUM (oparray)

No change.

MAT result array = SYS (coefficient matrix, constant array)

Write a subprogram to do this and replace this statement by an invocation of your subprogram. If your program makes use of DETL, you will want your subprogram to save away the coefficient matrix so that it can invert it (again) as the last operation in the subprogram.

MAT result array = TRN (oparray)

No change.

MAT result array = TRN (oparray1)*oparray2

Break this statement into two statements, as follows. Be sure to declare your temporary array.

```
MAT temp array = TRN (oparray1)
MAT result array = temp array*oparray2
```

MAT result array = oparray1*TRN(oparray2)

Break this statement into two statements, as follows. Be sure to declare your temporary array.

```
MAT temp array = TRN (oparray2)
MAT result array = oparray1*temp array
```

MAT result matrix = ZER[(redim subscripts)]

Replace this statement by the two-statement sequence

```
REDIM result array (redim subscripts)
MAT result array = (0)
```

MAT DISP

Replace MAT DISP by PRINT or OUTPUT (see DISP). Also, delete each occurrence of ROW or COL, and append (*) to each array name. The Series 200 always prints out matrices by row. If you want a matrix printed out by column, either compute the transpose (TRN) of the matrix and print that or write code to print by columns. TRN requires AP2.0. If your Series 80 program uses a slash separator, replace the slash with a comma and add USING "K,/"; to your PRINT or OUTPUT statement.

MAT DISP USING

See MAT DISP and USING.

MAT INPUT

Change MAT INPUT to INPUT. Append (*) to each array name. Note that this will not prompt you for individual array elements.

MAT PRINT

Replace MAT PRINT by PRINT. Also, delete each occurrence of ROW or COL, and append (*) to each array name. The Series 200 always prints out matrices by row. If you want a matrix printed out by column, either compute the transpose (TRN) of the matrix and print that or write code to print by columns. TRN requires AP2.0. If your Series 80 program uses a slash separator, replace the slash with a comma and add USING "K,/"; to your PRINT statement.

MAT PRINT USING

See MAT PRINT and USING.

MAT READ

Change MAT READ to READ and append (*) to the name of each array.

MAX

No change. This requires AP2.0.

MAXAB

If your program uses MAXAB but not MAXABCOL or MAXABROW, replace MAXAB *array* by MAX(ABS(MAX(*array*)),ABS(MIN(*array*))). This requires AP2.0. If you need to use MAXABCOL or MAXABROW as well, then be sure that the program segment which contains the MAXAB has a reference to COM /Matrix/ (see appendix on Matrices). Write a function which also references COM /Matrix/ and replace MAXAB by an invocation of this function. Your function should scan an array to find the element which is largest in absolute value (the value returned by the function) and the row (Maxabrow) and column (Maxabcol) location of this element. You will find the BASE, RANK, and SIZE keywords in AP2.0 very helpful.

Note that you do not want to use the functions you may have written to replace AMAX and AMIN to help you compute MAXAB because doing so will alter the values of some of the variables in COM /Matrix/ (Amaxcol, etc.).

MAXABCOL

Provide the COM reference and function discussed in MAXAB. Note that the Series 200 syntaxer will have changed MAXABCOL to Maxabcol which is just what you want. This will reference the appropriate COM variable.

MAXABROW

Provide the COM reference and function discussed in MAXAB. Note that the Series 200 syntaxer will have changed MAXABROW to Maxabrow which is just what you want. This will reference the appropriate COM variable.

MDY

Write a function to do this. This function should use the Series 200 DATE function found in AP2.0.

MDY$

Write a function to do this. This function should use the Series 200 DATE$ function found in AP2.0.

MIN

No change. This requires AP2.0.

MOD

If B is not a positive constant, replace `A MOD B` by `A-B*INT(A/B)`.

MOVE

No change.

MSCALE

Replace

```
MSCALE x,y
```

by

```
SHOW -x,x_max-x,-y,y_max-y
```

where x_max and y_max give the size of the plotting area in millimetres and are dependent upon which device is being used. Values of x_max and y_max for the various Series 200 CRTs are given below.

Series 200 Model	x_max	y_max
16	160	120
20		
82913A	210	158
82912A	152	114
26	120	88
36	210	160
36C	217	163

MSI

Replace your Series 80 volume label or mass storage unit specifier by the appropriate Series 200 mass storage unit specifier.

Check your Series 200 BASIC manuals for information on determining the default mass storage unit specifier.

MSUS$

Replace MSUS$ with SYSTEM$("MSI"). This requires AP2.0.

M_STATUS

See the appendix on MIKSAM.

NEXT

No change. See FOR for a detailed description of the changes which need to be made in FOR loops.

NOBLINK

Delete this statement. Note that the Series 200 syntaxer will have converted it to Noblink, and it will not be commented out.

NORMAL

In a Series 80 program, NORMAL immediately halts all tracing activity and cancels print-all operations. In the Series 200, use TRACE OFF to cancel tracing. Use OUTPUT 2; CHR$(255)&"A"; to toggle (not necessarily turn off) the PRT ALL switch from a program.

NOT

The precedence of NOT in the mathematical hierarchy is different in the two systems. Numeric expressions containing NOT may need to be reparenthesized to give the expected result. See the section on Arithmetic Expressions for further details.

NPAR

No change.

NUM

No change is required if the argument of NUM is never the null string. The Series 80 returns 0 for NUM(""), but the Series 200 gives an error. You can write a function to imitate Series 80 behavior if you wish.

OFF CCODE

Untranslatable. Delete this statement.

OFF CURSOR

Delete this statement. The Series 200 does not have this capability.

OFF EOT

Replace the interface select code with the I/O path name associated with that select code. Note that the Series 80 OFF EOT only disables branching caused by TRANSFER termination, while the Series 200 OFF EOT cancels the ON EOT definition. That is, the Series 80 will continue to log transfer termination after OFF EOT has been executed, but the Series 200 will not.

The Series 200 does not have the ability to disable branching selectively. However, if you wish to disable ON EOT and don't mind disabling your other ONs as well, you can use DISABLE. In this case, you want to use ENABLE, rather than ON EOT, to re-enable branching.

OFF ERROR

No change.

OFF INTR

Replace the interface select code with the appropriate Series 200 select code. Note that the Series 80 OFF INTR only disables branching caused by an interface-generated interrupt, while the Series 200 OFF INTR cancels the ON INTR definition. That is, the Series 80 will continue to log interrupts after OFF INTR has been executed, but the Series 200 will not.

The Series 200 does not have the ability to disable branching selectively. However, if you wish tc disable ON INTR and don't mind disabling your other ONs as well, you can use DISABLE. In this case, you want to use ENABLE, rather than ON INTR, to re-enable branching. In any case, do not use DISABLE INTR, since it will prevent the interface from sending the interrupt message.

OFF KBD

No change.

OFF KYBD

Replace OFF KYBD by OFF KBD and delete any string expression which appears in this statement. See ON KYBD for more details. Note that the Series 80 OFF KYBD only disables branching caused by keyboard activity, while the Series 200 OFF KBD cancels the ON KBD definition. That is the Series 80 will continue to log keyboard interrupts after OFF KYBD has been executed, but the Series 200 will not log them after executing OFF KBD.

OFF KEY#

Delete the #.

OFF TIMEOUT

Replace the interface select code with the appropriate Series 200 select code. Note that the Series 80 OFF TIMEOUT only disables branching caused by an interface timeout, while the Series 200 OFF TIMEOUT cancels the ON TIMEOUT definition. That is, the Series 80 will continue to log timeouts after OFF TIMEOUT has been executed, but the Series 200 will not.

The Series 200 does not have the ability to disable branching selectively. However, if you wish to disable ON TIMEOUT and don't mind disabling your other ONs as well, you can use DISABLE. In this case, you want to use ENABLE, rather than ON TIMEOUT, to re-enable branching.

OFF TIMER#

See ON TIMER#, and make the corresponding change.

ON num exp GOSUB

See GOSUB.

ON num exp GOTO

See GOTO.

ON CCODE

Untranslatable. Delete this statement.

ON CURSOR

Delete this statement. The Series 200 does not have this capability.

ON EOT

Replace the interface select code with the I/O path name associated with it. If you have used a variable or numeric expression to specify the select code, this statement is not translatable. The "OUTPUT 2" trick often used to get around such problems will not work here since ON EOT cannot be executed from the keyboard.

ON ERROR

No change is required for ON ERROR GOTO if the line specified is in the proper environment. See GOTO.

ON ERROR GOSUB works quite differently on the two systems. On the Series 80, if an ON ERROR GOSUB branch is taken, the system branches to the line (not the statement) following the line containing the error when the RETURN is executed. On the Series 200, the system attempts to re-execute the line containing the error once it executes the RETURN. Hence, on the Series 200, ON ERROR GOSUB should be used only when a particular error is anticipated and can be corrected in the subroutine. Failure to correct the error will cause the program to fall into an infinite loop.

If your Series 80 ON ERROR GOSUB subroutine is designed to detect and repair a particular error in a particular program line, you can imitate its behavior on the Series 200 as follows. Change the ON ERROR GOSUB to an ON ERROR GOTO, and replace the RETURN statement with a GOTO which returns control to the line following the suspect line.

ON INTR

No change.

ON KBD

No change. Note however, that the Series 80 ON KBD traps the RESET key, but the Series 200 ON KBD does not.

ON KEY#

If your ON KEY# statement defines a typing aid, you will need to delete it from your program. You can define typing aids on the Series 200 by using the EDIT KEY command.

The following applies to ON KEY# statements which contain a GOSUB or GOTO. Delete the #. If your ON KEY# statement contains a key label definition, replace the comma preceding the label by the keyword LABEL. You may wish to add a priority specifier, too, because the algorithm for determining which end-of-line branch to service when several are pending is different on the two systems.

Note that in the Series 200, only keys 0 through 9 may display key labels. (Key labels which are specified for the remaining keys are simply ignored by the system. No error is reported.) Furthermore, key labels are truncated to 8 characters on the 9826. In the Series 80, keys 1 through 14 may have labels, and labels may be up to 10 characters long.

ON KYBD

You may be able to rework your program using the ON KBD statement together with the KBD$ function to achieve the effect of ON KYBD. ON KBD is not selective—it applies to all keyboard keys with very few exceptions. However, the KBD$ function allows you to determine exactly which key was pressed and thereby select the action to be performed. Consult your Series 200 BASIC Language Reference or Interfacing Techniques Manual for more details.

ON TIMEOUT

The Series 200 ON TIMEOUT statement contains both the interface select code and the timeout time in seconds. To convert your Series 80 ON TIMEOUT statement, update the interface select code and follow it with a comma and the timeout time. You can obtain the timeout time in milliseconds from the SET TIMEOUT statement in your Series 80 program and convert this to seconds. Note that a timeout time of 0 is not legal on the Series 200. Use OFF TIMEOUT instead.

The Series 80 ON TIMEOUT applies to all I/O to external devices. The Series 200 version applies to a more restricted set of statements.

ON TIMER#

The Series 80 has three identical timers all dedicated to generating interrupts once the specified time has elapsed. The Series 200 also has three timers which generate interrupts, but each one has a different function. ON TIMER# is closest in meaning to the Series 200's ON CYCLE, found in AP2.0. You will need to convert your time specification from milliseconds to seconds and apply ABS to it to guarantee that the argument is positive.

The Series 80 timer is reset only after the interrupt branch has been serviced, but in the Series 200, the timer is reset immediately. To imitate Series 80 behavior, insert an OFF CYCLE statement at the beginning of the interrupt service routine and a new ON CYCLE definition at its end.

See ON CYCLE, ON DELAY, and ON TIME in your Series 200 BASIC Language Reference manual. It may be possible that you have been manipulating ON TIMER# to make it perform one of the other time interrupt functions.

OPEN_KEY_FILE

See the appendix on MIKSAM.

OPTION BASE

No change.

OR

No change.

OTD

Use IVAL(*<string expression>*,8) or DVAL(*<string expression>*,8). This requires AP2.0.

OUTPUT

If your Series 80 OUTPUT statement uses a device specifier, replace the device specifier with the appropriate Series 200 device specifier or with the I/O path name associated with it. Remember that if you want a CONVERT OUT to be applied, you must use the I/O path name.

If your Series 80 OUTPUT statement specifies a buffer, replace the string variable with the I/O path name associated with that buffer. Note that Series 200 buffers are circular. In order to prevent the system from wrapping around to the beginning of the buffer once it has written to the rightmost byte, precede the OUTPUT statement with a STATUS statement which determines the current empty pointer location and a CONTROL statement which sets the empty pointer to 1. After the OUTPUT, use another CONTROL statement to restore the empty pointer to the location read by the STATUS statement.

PACK

Delete this statement. If you need to pack a disc, you can use the REPACK utility in your Series 200 BASIC Utilities Library.

PAGE

Delete this statement. It applies only to the internal printer of the HP85.

PAGESIZE

Delete this statement. There is no way to achieve this effect on the Series 200.

PASS CONTROL

Replace the Series 80 device selector with the appropriate Series 200 device selector. Note that the Series 80 can specify only a select code, but the Series 200 requires a primary address.

PAUSE

No change.

PCHAIN

Replace PCHAIN with LOAD, and replace the file specifier with the correct Series 200 file specifier. If the file is in EPROM, AP2.1 is required.

PDIR

The Series 80 PDIR statement and the Series 200 PIVOT statement both cause rotations to be applied to lines. However, they apply to different sets of line drawing statements. There are two possible ways to convert your Series 80 program's handling of PDIR. Which method to use depends on the character of your program.

If your program contains more RPLOT, IPLOT, IMOVE, and IDRAW statements than other line drawing statements, use the following method.

Replace PDIR by PIVOT. If your PDIR statement specifies a rise and run, convert this to the proper angle. Use the function which replaces ATN2 to do this. Insert PIVOT 0 statements in your program prior to the other line drawing statements and later reset PIVOT to the specified value to get the desired effect.

If your program contains comparatively few statements affected by PDIR, use the method below.

Create a named COM area which contains a REAL scalar called Pdir. Insert the required COM statement in every program context which does line drawing operations. Replace your PDIR statements with LET statements which give the value of the desired rotation to the COM variable Pdir. Immediately before each RPLOT/IPLOT/IMOVE/IDRAW statement, insert the statement PIVOT Pdir. Immediately after each RPLOT/IPLOT/IMOVE/IDRAW, insert the statement PIVOT 0.

Regardless of which way you simulate PDIR, if your PDIR statement has a single parameter and your trig mode is grads, you need to apply FNGtr to the PDIR parameter (see GRAD).

PEN

The syntax of the PEN statement is the same, but the semantics is quite different on the two systems, especially when the active plotter is the CRT. The Series 200 does not have some of the capabilities of the Series 80. Consult your Series 200 BASIC Language Reference for details.

Here is a table of correspondences of pen numbers for CRT graphics. Check your system documentation to see how values other than these are mapped to legal pen values.

HP85	HP87	Series 200
	-2	0
-1	-1	-1
	0	none
1	1	1

PENUP

No change.

PI

No change.

PLIST

Replace PLIST by LIST.

PLOADBIN

Untranslatable. Delete this statement. See LOAD BIN.

PLOADGO

Replace PLOADGO with LOAD, and replace the file specifier with the appropriate Series 200 file specifier. If the file is in EPROM, AP2.1 is required.

PLOT

No change.

PLOTTER IS

Change the device selector to the appropriate Series 200 device selector. Add ,"INTERNAL" if the CRT (select code 3) is the specified plotter, and ,"HPGL" otherwise. In addition, insert a GINIT statement before the PLOTTER IS to reset the graphics default conditions.

POS

No change.

PPOLL

Replace the Series 80 device selector by the appropriate Series 200 device selector.

PRINT

No change. Note, however, that on the Series 200, the tabbed fields are only 10 characters wide and there will be either 5 or 8 fields on the CRT, depending on the machine used. If you want to create a particular effect, you may need to use PRINT USING.

PRINT ALL

Use OUTPUT 2;CHR$(255)&"A"; to toggle the PRT ALL key.

Since the default printall printer on the Series 200 is the CRT, you will probably want to include a PRINTALL IS statement to define an external printer to be the printall device.

PRINT USING

See PRINT and IMAGE.

PRINT#

If you have used a constant to specify the buffer (file) number, replace PRINT# by OUTPUT and replace the buffer number by the I/O path name associated with the file. (See ASSIGN. It directs you to replace buffer 1 by @F1, etc.)

If you have used a variable or a numeric expression to specify the buffer number, replace

```
PRINT# exp; print# list
```

by

```
OUTPUT 2; "OUTPUT @F"&VAL$(exp)&"; print# list"
  &CHR$(255)&"X";
```

Similarly, replace

```
PRINT# exp,record; print# list
```

by

```
OUTPUT 2; "OUTPUT @F"&VAL$(exp)&"," &VAL$(record)       "; print# list"&CHR$(
```

Note that if the PRINT# list contains any literals, the quote marks (") which delimit the literal will need to be doubled (""). Also, if the PRINT# list contains any array references, they need to be modified. For example, A() or A(,) needs to be replaced by A(*).

PRINTER IS

Replace the device selector with the appropriate Series 200 device selector. If you have specified a line length, replace the comma by ;WIDTH. Note that a line length of 0 is illegal on the Series 200. Use 80 instead. AP2.0 is required if WIDTH is to be specified.

PRINTER TYPE

Untranslatable. Delete all references to this function.

PRINTER TYPE IS

Untranslatable. Delete this statement. If you need to interface to a printer which does not support the standard 8-bit Extended Roman character codes, you will need to write a filter routine as described in LPRINT and pass all of your output through this filter before sending it to the printer.

PROM IS

Untranslatable. Delete this statement. Statements such as PCHAIN which refer to the PROM IS device need to contain a full file specifier.

PURGE

If your PURGE statement does not specify a purge code, simply replace the file specifier by the appropriate Series 200 file specifier.

The Series 200 does not use purge codes and does not have the ability to purge multiple files with a single statement. If you want to do this, you will need to write a subprogram which uses CAT to a string array, finds the name of the first file to be purged in this array, purges it, then purges the next one, etc. This requires AP2.0. Replace your PURGE statement by a call to this subprogram.

Before you write a subprogram to do PURGE, note that the method of allocating space for files on a disc is different on the two systems, so that files are likely to be stored in a different order. If you apply this algorithm carelessly, you run the risk of purging needed files.

Note also that the Series 200 does not allow open files to be purged, but the Series 80 does. If PURGE gives you an Error 77, you will need to ASSIGN *<file>* TO * before re-attempting the PURGE.

Note that PURGE on the Series 200 does not create NULL files.

RAD

No change. Note, however, that changes to the trig mode are global on the Series 80 and local on the Series 200. There is no problem with invoking subprograms because on the Series 200, the subprogram inherits the trig mode of the calling program. However, there is a problem with returning from a subprogram. For example, if a subprogram executes a RAD statement, the Series 80 will remain in radians mode when it returns to the calling program, but the Series 200 will reset the trig mode to the mode which was in effect when the subprogram was invoked. You may need to insert some additional trig mode changes in your program to make it work properly.

RANDOMIZE

No change. Note, however, that the algorithms for generating random number seeds are different. The recommended range for seeds on the Series 200 is 1 to $2^{31}-2$.

RATIO

No change. Note, however, that the default value returned by RATIO when the CRT is the plotter will be different.

READ

No change.

If a READ statement occurs in a function, make sure that the required DATA statements are also in that function.

READTIM

The Series 200 does not have the ability to read the system timer. Remove this function from your program.

READ#

See PRINT#. Make the analogous changes, substituting ENTER for OUTPUT.

Recall that Series 200 machines cannot read Series 80 DATA files directly. See the data translation chapter of this document for help in transforming your files.

REAL

No change provided you do not exceed the maximum array size for the Series 200. The Series 200 allows an array subscript to have a maximum upper bound of 32766 if OPTION BASE is 0 and 32767 if OPTION BASE is 1. If your arrays are larger than this, you will need to restructure your program.

REDIM

No change provided you do not exceed the maximum array size for the Series 200. See REAL, above. Note, also, that if a Series 80 program redimensions an array and then reruns the program, the array does not reassume its original dimensions. On the Series 200, it does.

REM

No change.

REMOTE

If your REMOTE statement is directed to an HP-IB interface, update the device selector to the appropriate Series 200 device selector. If your REMOTE statement specifies multiple devices, you will need to replace it with a series of REMOTE statements, one for each device in the list or replace the multiple listener specification with the I/O path associated with that set of devices. Note that the bus sequences are somewhat different in order on the two machines.

There is no Series 200 analog to REMOTE to a BCD interface.

RENAME

Be sure that both the old and new file names are legal Series 200 file names. Also, change the old mass storage unit specifier to the appropriate Series 200 mass storage unit specifier.

Note that you cannot use a Series 200 computer to rename a file whose name is illegal on the Series 200. You will need to do this on your Series 80 machine.

REQUEST

If your REQUEST statement is directed to an HP-IB interface, simply update the interface select code. This requires AP2.0. If the REQUEST is directed to an ordinary serial interface, use BREAK (if you have AP2.0) or CONTROL to register 6 (if you do not have AP2.0). If it is directed to a datacomm interface, use BREAK or CONTROL to register 1.

RESET

Update the interface select code appropriately. RESET requires AP2.0. Note that the Series 200 RESET waits for an ongoing TRANSFER to complete, but the Series 80 RESET does not. If this is a concern, insert an ABORTIO statement before the RESET.

RESTORE

If the RESTORE statement occurs in a function or subprogram, the Series 200 can move the DATA pointer only to DATA statements within that function.

RESUME

If the RESUME is directed to an HP-IB interface, replace it with a SEND of DATA (with no data list) to the appropriate select code.

There is no Series 200 analog to RESUME on a serial interface.

RETURN

No change.

REV$

No change. This requires AP2.0.

REWIND

Delete this statement. Note that the Series 200 syntaxer will change REWIND to Rewind.

RMD

Append the following code to your program and replace RMD by FNRmd.

```
DEF FNRmd(X,Y)
RETURN X-Y*SGN(X/Y)*INT(ABS(X/Y))
FNEND
```

RND

No change. Note, however, that the sequence of numbers generated will be different.

RNORM

Be sure that the program segment which contains the RNORM has a reference to COM /Matrix/ (see the appendix on Matrices). Write a function which also references COM /Matrix/ and replace RNORM by an invocation of this function. Your function should determine the row norm (the value returned by the function) and the row number (Rnormrow) associated with this norm.

RNORMROW

Provide the COM reference and function discussed in RNORM. Note that the Series 200 syntaxer will have changed RNORMROW to Rnormrow, which is just what you want. This will reference the appropriate COM variable.

ROTATE$

Write a function using substring specifiers and concatenation (&) to do this. Replace ROTATE$ by a call to this function.

RPLOT

No change. See PDIR.

RPT$

No change. Note, however, that the Series 80 returns a null string if the number of repetitions specified is negative. The Series 200 gives an error instead. If this is important to you, write a function which tests for this special case and replace RPT$ with an invocation of this function.

RTD

Append the following code to your program and replace RTD by FNRtd.

```
DEF FNRtd(X)
RETURN 180*X/PI
FNEND
```

SARRAY

Delete the SARRAY statement. You will need to declare each string array in a COM or DIM statement or use the system default dimensions (subscript upper bound of 10 and 18 characters for each element). This means that you will need to decide on a subscript upper bound and a maximum length for the strings in the array. This also means that you cannot use the string variables which were in the SARRAY as both ordinary strings and string arrays.

SAVE

Make sure that the file specifier is a legal Series 200 file specifier. Note that a Series 80 SAVE creates a DATA file, while a Series 200 SAVE creates an ASCII file.

SCALE

SCALE is automatically changed to WINDOW by the Series 200 syntaxer. No change is required on your part.

SCRATCHBIN

Delete this statement. The Series 200 does not have this capability. Note that the Series 200 syntaxer will have changed SCRATCHBIN to Scratchbin.

SCRATCHSUB

Change SCRATCHSUB to DELSUB. See CALL for an explanation of how to handle the subprogram name.

SEC

Use 1/COS(*<num exp>*) if your Series 80 trig mode is either RAD or DEG. This will be equivalent to the Series 80 SEC(*<num exp>*) if your Series 80 program has DEFAULT OFF. It will give an error whenever the value of *<num exp>* is an odd multiple of $\pi/4$ radians (90 degrees).

If your Series 80 program has DEFAULT ON and you want to avoid these errors, write a function using COS which returns machine infinity rather than giving an error. Note, however, that the value of machine infinity is different on the two systems.

If your Series 80 trig mode is GRAD, write a function to convert grads to radians and use it to convert *<num exp>* to radians before invoking COS.

SECURE

Series 200 BASIC does not have a SECURE statement, but there is a Secure subprogram in your BASIC Utilities Library. To use this utility in your program, change SECURE to Secure. See the utility instructions for further details.

If your Series 80 SECURE statement uses security type 2, you may be able to use the Series 200 PROTECT statement.

SEEK_END

See the appendix on MIKSAM.

SEEK_FIRST

See the appendix on MIKSAM.

SEEK_KEY

See the appendix on MIKSAM.

SEEK_NEXT_KEY

See the appendix on MIKSAM.

SEEK_PRIOR_KEY

See the appendix on MIKSAM.

SEND

If your SEND statement is directed to an HP-IB interface, update the interface select code appropriately. Also, replace SCG by SEC and EOL by END.

There is no Series 200 analog of SEND directed to non-HP-IB interfaces. Check to see whether the CONTROL and WRITEIO can provide the desired capability.

SETGU

To establish graphics units as the current unit of measure, replace SETGU by

```
CLIP 0,100*RATIO,0,100
```

if the horizontal axis of the plotting area is longer than the vertical. Replace it by

```
CLIP 0,100,0,RATIO*100
```

otherwise. On the Series 200, this will cause all record of the units which were previously in effect to be lost. See SETUU for more details.

Note that the Series 200 syntaxer will have changed SETGU to Setgu.

SET I/O

The Series 80 SET I/O statement provides the ability to alter the contents of interface control registers. You may be able to perform the desired operation using the Series 200 CONTROL statement. In addition to changing the select code to the proper Series 200 select code, you will probably need to change the register number and the data value to be sent to it, as well. Consult your Series 200 BASIC Language Reference or BASIC Interfacing Techniques manual for details.

SET TIMEOUT

If your SET TIMEOUT statement is used with an ON TIMEOUT statement to define a timeout activity where none is currently defined, delete your SET TIMEOUT statement and use the time it specifies in the associated ON TIMEOUT statement. See ON TIMEOUT for more details.

If your SET TIMEOUT statement is used to change the timeout time for an existing ON TIMEOUT for a given interface, replace the SET TIMEOUT with an ON TIMEOUT which specifies the same action as the existing ON TIMEOUT and uses the time (converted from milliseconds to seconds) specified in the SET TIMEOUT.

SETTIME

Write a function which computes total elapsed seconds from the seconds and date parameters of SETTIME. Use the result of this function as the argument of SET TIMEDATE.

SETUU

In order to be able to switch between user and graphics units easily (i.e., to simulate SETGU and SETUU), you will need to save away user units when you set them up. The best way to do this is to set up a named COM area for this purpose and to set the appropriate COM variables whenever you do a SHOW or WINDOW. For example, let the COM declaration be

```
COM/Limits/Uu_xmin,Uu_xmax,Uu_ymin,Uu_ymax
```

Then whenever you execute a SHOW or a WINDOW, insert four assignment statements which set the COM variables to the limits just established.

Replace SETUU by a SHOW or a WINDOW statement which uses the COM variables as parameters. Choose SHOW if you want isotropic scaling and WINDOW if you want anisotropic.

Note the the Series 200 syntaxer will have changed SETUU to Setuu.

SET_UP

See the appendix on MIKSAM.

SFLAG

See the appendix on Flags.

SGN

No change.

SHORT

Change SHORT to REAL; the Series 200 does not support the SHORT data type. In addition, the maximum upper bound for any array subscript on the Series 200 is 32766 if OPTION BASE is 0 and 32767 if it is 1. If your arrays are larger than this, you will need to restructure your program.

SHOW

No change. See SETUU.

SIN

No change is required if your Series 80 trig mode is RAD or DEG. If your Series 80 trig mode is grads, apply FNGtr to the argument of SIN before applying it (see GRAD).

SLET

Delete the word SLET or change it to LET.

SLITE

Untranslatable. Delete all references to this function.

SMAX

Replace the call to SMAX with an expression which is 1 less than SIZE + BASE of the first dimension of the string array. This requires AP2.0. This will give you the dimensioned upper bound of the array, not the location of the last non-null element. If you want to find the last non-null element, write a function to do it and replace the call to SMAX by a call to your function.

SPOLL

Replace the device selector with the appropriate Series 200 device selector. Note that the Series 200 does not support SPOLL to an interface select code.

SQR

No change.

ST RESULT

The Series 200 has no analog to ST RESULT for returning the result of the AUTOSTART test. To determine which Series 200 machine is in use, use SYSTEM$("SYSTEM ID"). This requires AP2.0.

STATUS

The Series 200 STATUS statement is very similar syntactically to the Series 80 STATUS, but there are many differences in semantics. You will need to translate this statement very carefully. If your STATUS statement uses variables rather than constants, translation may be impossible.

If your Series 80 STATUS statement specifies an interface select code, replace it with the appropriate Series 200 select code. If it specifies a buffer, replace the string variable by the corresponding I/O path name.

You will need to check the documentation for both systems to determine which Series 200 register corresponds to your Series 80 register. If you do not find a STATUS register correspondence, also check the Series 200 READIO registers for the interface to see if they provide the needed functionality.

If your STATUS statement contains a list of variables to receive status information, you will probably need to rearrange the variable list or replace the entire statement with a list of STATUS statements each interrogating one status register. This is because the order of registers is often different.

In addition, you may need to make other modifications to your program if you take actions depending on the value returned by STATUS because the two systems do not always use the same values to mean the same thing. For example, in the Series 80, a newly created buffer has a fill pointer of 0. In the Series 200, the value is 1.

STOP

No change.

STOREBIN

Delete this statement. The Series 200 has a STORE BIN command, which is not programmable.

STORE BIN

See STOREBIN.

SUB

Remove the quotes that delimit the subprogram name, and be sure that the subprogram name is a legal Series 200 name. Insert the keyword OPTIONAL immediately after the opening parenthesis of the parameter list. Change any array references of the form A() or A(,) to references of the form A(*).

If your subprogram does any buffer operations, you will need to modify your parameter list. If the subprogram does not do any string operations on the buffer, you can replace the string name in the parameter list by an I/O path name. If the subprogram does both string operations and buffer operations, you will need to add the I/O path name to the list immediately after the string name. This will prevent any complications with optional parameters.

See RAD for a discussion of the effect of subprograms on trigonometric computations.

SUBEND

On the Series 200, every subprogram must have exactly one SUBEND statement, and this must be the highest-numbered statement in the subprogram (except for comments). Insert a SUBEND at the end of your subprogram if necessary, and change all other SUBENDs to SUBEXITs.

SUBEXIT

No change. See SUBEND.

See RAD for a discussion of the effect of subprograms on trigonometric computations.

SUM

No change. This requires AP2.0. Note that the Series 200 cannot handle empty arrays.

SWAP

Untranslatable. Delete this statement.

TAB

You must delete TAB from LABEL lists. No other changes are required.

Note that because the HP9826 has a 50-column screen, PRINT and DISP directed to this CRT and containing TABs may not produce the same effect as on an 80-column screen.

TAN

No change is required if your Series 80 trig mode is RAD or DEG. If your Series 80 trig mode is grads, apply FNGtr to the argument of TAN before applying it (see GRAD).

Note that if DEFAULT is ON and the argument of TAN is an odd multiple of $\pi/4$ radians (90 degrees), the Series 80 returns machine infinity. The Series 200 does not have DEFAULT ON and always reports an error.

TIME

Replace TIME by TIMEDATE MOD 86400.

TIME$

Replace TIME$ by TIME$(TIMEDATE). This requires AP2.0.

TRACE

Replace TRACE by TRACE ALL. Note that this will give trace information about variable assignments as well as about order of statement execution. In addition, the Series 80 TRACE statements direct their output to the PRINTER IS device. The Series 200 TRACE ALL directs its output to the system message line of the CRT and to the PRINTALL printer if printall mode has been turned on.

TRACE ALL

No change. See TRACE.

TRACE VAR

Replace the statement by TRACE ALL See TRACE.

TRANSFER

Change both your source and destination specifiers to the associated I/O path names. If you have not already associated an I/O path name with the device specifier which appears in the TRANSFER, you can insert an ASSIGN statement prior to the TRANSFER to make this association.

Remove the specification of INTR or FHS. The Series 200 system will pick a transfer mode for you. Consult the Series 200 BASIC Language Reference or Interfacing Techniques manual for information on how the mode is selected. This will show you how to force INTR or FHS if you really want to do this.

If you have specified DELIM, apply CHR$ to the delim byte. The Series 200 expects a string expression here. If you have specified EOI, change it to END.

Caution is needed to be sure that your TRANSFERs will terminate in the same way on the two systems. Both systems will terminate outbound TRANSFERs when the buffer is empty and inbound TRANSFERs when the buffer is full or when the first specified delimiter is received. However, the Series 80 and Series 200 check for full or empty buffers in different ways. The Series 80 checks to see if the fill or empty pointer has reached the end of the buffer. The Series 200 has circular buffers, so it checks to see if fill pointer has reached the empty pointer, or vice-versa. You can, however, make your TRANSFERs terminate as expected by using a CONTROL statement ahead of your TRANSFER to set your fill or empty pointer to 1, as needed. Be sure to save away the old value of the pointer so that you can restore it after the TRANSFER completes. This is an excellent use of ON EOT.

TRANSLATE

Delete this statement. Note that the Series 200 syntaxer will have changed TRANSLATE to Translate.

TRIGGER

If your TRIGGER statement contains a single device selector, update it appropriately. If it contains more than one, either update each one and replace the single TRIGGER statement by a series of TRIGGER statements, each specifying one of the devices, or replace the list of device specifiers by the I/O path name associated with that multiple listener configuration.

The bus sequences sent by the two systems are slightly different in order.

TRIM$

No change. This requires AP2.0. Note that the Series 80 TRIM$ does not trim leading or trailing blanks which also have character highlights. The Series 200 TRIM$ does not trim leading blanks with highlights, but does trim trailing ones. This is because the two machines use different representations for highlighted characters.

TYP

Delete all references to TYP and restructure the statements containing them. The Series 200 does not support typed data files. If your program relies heavily on TYP, you may want to consider encoding type information for your file in a string array and writing routines to read the type information from this array.

If you are using TYP to test for end-of-file and end-of-record, you can use the Series 200 STATUS statement to obtain the same information.

Consult the data transportation section of this document for more information about files.

UBND

Replace UBND(*array,dimension*) by BASE(*array,dimension*)+SIZE(*array,dim nsion*)-1. This requires AP2.0.

UNCLIP

Replace UNCLIP by CLIP OFF. Note that the Series 200 syntaxer will change UNCLIP to Unclip.

UNCONFIG

Untranslatable. Delete this statement. In the Series 200, only SCRATCH A or cycling power returns a memory volume to ordinary memory use.

UPC$

No change. This requires AP2.0. Note, however, that in the Series 200, UPC$ always uses the collating sequence associated with the current LEXICAL ORDER IS, while in the Series 80, it uses the ASCII collating sequence. If an international keyboard is in use, it may be necessary to precede the UPC$ by LEXICAL ORDER IS ASCII and follow it by another LEXICAL ORDER IS.

VAL

No change is required if the argument string contains no blanks and has at most a single sign character. The Series 80 will ignore imbedded blanks, but the Series 200 uses a blank to terminate the number it is trying to build. The Series 80 allows an arbitrary number of "+" and "-" signs in the argument, but the Series 200 allows only one.

VAL$

No change.

VOLUME ... IS

Delete this statement. Series 200 BASIC does not use volume labels. If for some reason you need to assign or change a volume label, use the INITIALIZE utility in your BASIC Utilities Library.

VOL$

Use CAT TO a string array and extract the volume label from the array. This requires AP2.0.

WAIT

Divide the argument by 1000 to convert milliseconds to seconds.

WHERE

In an HP85 program, WHERE returns the coordinates of the last position to which the pen was moved under either manual or program control. This is not translatable.

If your HP87 program does not use the Plotter ROM, WHERE returns the physical pen position. This is untranslatable whenever the logical and physical pen positions differ.

If your HP87 program uses the Plotter ROM or if the logical and physical pen positions are identical, use the WHERE statement in GRAPH2.1 to return the logical pen position. Note that in the Series 200, the pen status is returned in a string variable, not in a numeric variable. If you need the pen status, you can obtain it from the first byte of the status string.

XAXIS

The Series 200 does not have the ability to draw a single axis. The closest you can come to this capability without writing a subprogram to draw a single axis is to use CLIP to make the plotting area only as high as you want your X-axis major tick marks, then use AXES to draw a pair of axes. The Y-axis will be very short. Be sure to execute CLIP OFF or reset CLIP to its former value after the AXES statement.

XREF

The Series 200 XREF is a (non-programmable) command in AP2.0 or XREF2.1 and lacks the L and V options. It always cross-references both lines and variables and gives much more information, too. In order to get a cross-reference from a program, use

```
OUTPUT 2; "XREF"&CHR$(255)&"X";
```

You can modify this output statement as necessary to direct the XREF output to the desired device or to specify a particular subprogram to cross-reference.

YAXIS

The Series 200 does not have the ability to draw a single axis. The closest you can come to this capability without writing a subprogram to draw a single axis is to use CLIP to make the plotting area only as wide as you want your Y-axis major tick marks, then use AXES to draw a pair of axes. The X-axis will be very short. Be sure to execute CLIP OFF or reset CLIP to its former value after the AXES statement.

Data Transportation

Most of this document deals with moving programs from a Series 80 computer to a Series 200 system. Moving program source, however, may be only part of your problem. You may have a considerable amount of data to transport, as well. This chapter will give you some guidelines on how to move your data. Some sections contain more techniques than exact directions because the details of what needs to be done depend heavily on the organization of your files.

Series 200 BASIC programs can access data in either ASCII or BDAT files. You probably want to get your data into BDAT files (with FORMAT OFF) because they require considerably less disc space. Also, BDAT files support random access, but ASCII files do not.

Series 80 BASIC uses DATA, GRAF, and PKEY files to hold data. Most of this section will be devoted to helping you convert DATA files to Series 200 BDAT files. GRAF files cannot be ported directly to the Series 200. However, you can write a Series 80 BASIC program which uses BREAD to write the contents of graphics memory to a string and then writes that string to a DATA file. PKEY files can be handled in a similar fashion. That is, you can write a Series 80 program to read the data from your PKEY file and write it to a DATA file. Then you can use one of the methods described below to convert your DATA file to a BDAT file.

Some fundamental differences between the Series 80 and Series 200 file systems may mean that you will need to make program modifications to be able to read your data correctly from a BDAT file. Series 80 DATA files contain type information before each data item. BDAT files contain no type information. Furthermore, all numbers require eight bytes of storage on a DATA file. On a BDAT file with FORMAT OFF, REAL numbers take eight bytes, but INTEGER numbers take only two. If when you convert your DATA file to a BDAT file you treat all numeric data as REALs, you need to be sure that the ENTER statements which read the data contain only REAL variables. Otherwise you will read the wrong amount of data from the disc and misalign yourself, creating an insufferable mess. To prevent the problem, either change the declarations of the necessary variables to REAL or read the data into temporary REAL variables and then use LET to assign the values to the INTEGER variables.

Via an Interface

The easiest way to transport your data is to read the data from your DATA file into your Series 80 machine, ship it via HP-IB to the Series 200 machine, and then write it to a BDAT file from there. Consult the *Moving Your Program* section of the first chapter of this document for details of how to configure your systems. Replace the programs given there with those below. The equipment requirements stated there pertain here with the following exceptions. If you are using an HP85, you need a Mass Storage ROM. If you are using an HP87 and your DATA file contains no strings longer than 80 characters, you can get by without the I/O ROM. Modify both programs below to eliminate the "#" from the image specifiers used in sending and receiving string data.

The following programs assume that your Series 200 computer is non-controller on the HP-IB interface which connects it to your Series 80 machine. If this is not the case, change the Series 80 program to contain `PRINTER IS 7` instead of `PRINTER IS 720`, and change the Series 200 program to do `ASSIGN @S80 TO 720` instead of `ASSIGN @S80 TO 7`. These programs also assume that your DATA file contains no strings longer than 400 characters. If necessary, change the dimensioned length of S$ in both programs.

These programs further assume that your file does not contain any data which would create numeric or string overflows on the Series 200. If this is not the case, modify the Series 80 program to test for overflows and take corrective action.

The Series 80 program requires you to specify an access mode for the translated file. You must choose R for random access or S for sequential access. If you choose random access, the programs will line up the data so that corresponding items always begin corresponding logical records. This may cause "holes" in the Series 200 file. In rare cases, it may also cause you to need to make your Series 200 logical records larger than your Series 80 records. If you choose sequential access, there will be no holes in the data, but the data will usually not line up on logical records just the way it did on the Series 80 DATA file. For these reasons, you should not attempt to do random access to a file which has been translated for sequential access, and vice-versa.

The Series 200 program requires you to specify size parameters for the BDAT file it creates to hold your translated data. In most cases, you can use the values for your DATA file. It may be necessary to experiment somewhat. REAL numbers require eight bytes of storage on a BDAT file. (INTEGERs require two bytes. If you modify these programs to write INTEGERs where possible, you can take advantage of this space savings.) Strings take four bytes plus one byte per character plus a pad byte if the string has odd length.

```
 10 ! PROGRAM FOR YOUR SERIES 80 MACHINE, REQUIRES
    PRINTER/PLOTTER ROM FOR HP-85
 20 DIM S$[400],F$[30]
 30 PRINTER IS 720       ! CHANGE AS NEEDED
 40 DISP "ENTER THE FILE SPECIFIER OF FILE TO BE
    TRANSLATED"
 50 INPUT F$
 60 ASSIGN# 1 TO F$
 70 DISP "LOGICAL RECORD LENGTH IN BYTES"
 80 INPUT L
 90 DISP "ENTER R FOR RANDOM OR S FOR SEQUENTIAL ACCESS"
100 INPUT A$
110 A=0 @ IF A$="R" THEN A=1 @ PRINT 4 @ PRINT 1
120 ON ERROR GOTO 230
130 R=1 @ B=1   ! CURRENT LOGICAL RECORD AND BYTE
140 T=TYP(1)    ! TYPE OF NEXT ITEM TO BE READ FROM FILE
150 IF T=1 THEN GOTO 240   ! NUMERIC DATA
160 IF T=2 THEN GOTO 260   ! WHOLE STRING
170 IF T=8 THEN GOTO 280   ! START OF STRING
180 IF T=3 THEN PRINT 3 @ DISP "DONE" @ BEEP @ STOP   ! END
    OF FILE
190 IF T<>4 THEN GOTO 230
200 R=R+1 @ B=1 @ READ# 1,R   ! END OF RECORD
210 IF A THEN PRINT 4 ! PRINT R   ! TELL SERIES 200 TO START
    NEW RECORD
220 GOTO 140
230 DISP "UNEXPECTED ERROR" @ BEEP @ STOP
240 PRINT 1 @ READ# 1;N @ PRINT N @ DISP N
250 B=B+8 @ GOTO 350
260 PRINT 2 @ READ# 1;S$ @ PRINT LEN(S$) @ PRINT USING
    "#,K";S$
270 B=B+3+LEN(S$) @ DISP S$ GOTO 350
280 PRINT 2 @ READ# 1;S$ @ PRINT LEN(S$) @ PRINT USING
    "#,K";S$
290 B=1 @ R=R+1 @ DISP S$
300 READ# 1,R
310 T=TYP(1)
320 IF T=9 THEN R=R+1 @ GOTO 300   ! PASS OVER ENTIRE RECORD
330 READ# 1;S$   ! T=10 -- RE-READ STRING BUT DON'T PRINT IT
340 B=3+LEN(S$)
350 IF B=L+1 THEN B=1 @ R=R+1
360 IF NOT A THEN GOTO 140
370 PRINT 4 @ PRINT R @ GOTO 140   ! TELL SERIES 200 TO
    START NEW RECORD
380 END
```

```
 10 ! PROGRAM FOR YOUR SERIES 200 MACHINE -- RUNS IN BASIC 2.0
 20 DIM S$[400],F$[30]
 30 INPUT "FILE SPECIFIER OF DESTINATION FILE",F$
 40 INPUT "LOGICAL RECORD LENGTH IN BYTES",L
 50 INPUT "NUMBER OF LOGICAL RECORDS",Rec
 60 CREATE BDAT F$,Rec,L
 70 ASSIGN @F TO F$
 80 CONTROL @F,7;Rec+1,1    ! WRITE END OF FILE MARKER
 90 ASSIGN @S80 TO 7
100 Next_item: ENTER @S80;T
110 SELECT T
120   CASE 1 ! NUMERIC DATA
130     ENTER @S80;N
140     OUTPUT @F;N
150     PRINT N
160   CASE 2 ! STRING DATA
170     ENTER @S80;Length
180     Image$="#,"&VAL$(Length)&"A"
190     ENTER @S80 USING Image$;S$
200     OUTPUT @F;S$
210     PRINT S$
220   CASE 3 ! END OF FILE
230     OUTPUT 1;"DONE"
240     BEEP
250     STOP
260   CASE 4 ! END OF RECORD
270     ENTER @S80;R
280     ENTER @F,R ! POSITION FILE POINTER TO BEGINNING OF
        RECORD R
290   CASE ELSE
300     DISP "UNEXPECTED ERROR"
310     BEEP
320     STOP
330 END SELECT
340 GOTO Next_item
350 END
```

Via an ASCII File

Another way to move your data to a BDAT file is to create an ASCII file using your Series 80 machine. If the program which accesses this data does only serial accesses to the file, your Series 200 machine can access this file directly. If you need to do random access or want more compact storage, then write a Series 200 BASIC program which reads your ASCII file and writes the data to a BDAT file.

To use this method, you must have a disc drive supported by your Series 80 machine and a disc drive for the same size floppy discs which is supported by Series 200 BASIC. (These can be the same disc drive). If your Series 80 machine is an HP85, you will also need a Series 80 Mass Storage ROM. A list of suitable disc drives can be found in an appendix at the end of this document. The process of creating an ASCII file involves the use of a Series 80 LIF program which is available from the Series 80 Users' Library (9-0049 for the HP85 and 9-0069B for the HP87.)

To create the ASCII file, execute the following sequence of steps:

```
LOAD "LIF87:msus"       ! LOAD "LIF:msus" for HP85

LOADBIN "LIFg:msus"     ! LOAD BIN "C:msus" for HP85
```

Press [RUN]. Give your msus (unquoted) starting with a semicolon, and press the softkey labeled LIFSAVE. Continue answering the questions asked.

NOTE

The LIF program contains a number of limitations and bugs which may trip you up. The program is not secured, however, so you can edit it as problems arise. As written, the program can handle strings up to 192 characters long. If your file contains longer strings, change the declaration and initialization of L$. If the program reads a null string from your file, the variable L is set to 0 (the length of the string) and then used to index the string. This produces an error. If your file may contain null strings, you need to add a test for them and avoid substring operations on them. The LIF program quits prematurely and prints "DONE" on the screen if it encounters an end of record mark. In the section which tests TYP(1), you need to include a test for the value 4. If TYP(1)=4, you need to advance to the beginning of the next record on the file and continue testing TYP(1). There may be other problems, too. Be sure to have the program print out your data so that you can verify that it is operating correctly. Also, don't believe the "DONE" message unless it is preceded by a message saying that the translation is 100% complete.

You will need to write your own program to read from the ASCII file and write to a BDAT file. As a first guess for size of the BDAT file, use the record length and number of records from your Series 80 DATA file. For maximum compactness, you may wish to try again with a shorter record length. This program will need to take into consideration the organization and use of the original DATA file. Such factors as whether the file is to be accessed randomly or sequentially, how the data is laid out, and the longest string on the file all enter in. Recall that Series 80

DATA files store all numeric data in 8 bytes. Series 200 BDAT files use 8 bytes for REALs but only 2 bytes for INTEGERs. Hence you can save considerable space if you can store your numeric data as INTEGERs. Recall also that the Series 80 has a greater range than the Series 200 for all data types. Your program must handle numeric and string overflows.

Using only the Series 200

If you no longer have access to a Series 80 computer, but you have a floppy disc containing the DATA file to be transported, you can still convert your data. Obtain the BASIC Flexible Disc Data Translator Utility (63.9520) from the Series 200 BASIC Users' Library. This utility will allow you to read the DATA file. You will need to write your own program to write the DATA to a BDAT file.

Appendix A
Flags

Series 200 BASIC does not have a system of built-in flags like that in Series 80 BASIC. There are, however, a number of ways to simulate Series 80 flags on the Series 200. Several approaches are outlined in this appendix. The method you use should depend on how you use the flags and which flag operations you perform most often.

All of the possible flag implementations discussed here involve setting up a named COM area dedicated to flag storage. Every program context which uses flags must reference this COM with a statement like

```
COM/Flags/ <flag declaration>
```

In addition, you will need to modify your program references to flags and write some subprograms to allow you to examine and manipulate your flags.

If your program does not use the flag operations which address the flags as an eight-character string, you can make *<flag declaration>* be `INTEGER Flag(1:64)`. Accessing individual flags is then quite simple. FLAG(*i*) becomes `Flag(`*i*`)` (the syntaxer will do this one automatically), CFLAG *i* becomes `Flag(`*i*`)=0`, and SFLAG *i* becomes `Flag(`*i*`)=1`. If however, you need to use FLAG$ and SFLAG *<string expression>*, you will need to write some fairly tedious subprograms to simulate these.

At the other extreme, if your flag operations are primarily FLAG$ and SFLAG *<string expression>*, you can let *<flag declaration>* be `Flag$[8]`. Then FLAG$ becomes `Flag$` (automatically) and SFLAG *<string expression>* becomes `Flag$=`*<string expression>*. You will need to write routines to simulate FLAG(*i*), CFLAG *i*, and SFLAG *i*.

As a compromise, you can let *<flag declaration>* be `Flag$[64]`. This allows you to performs all flag operations fairly easily, especially if you don't care that the data is no longer in an 8-character string. FLAG(*i*) becomes `Flag$[`*i*`;1]`, CFLAG *i* becomes `Flag$[`*i*`;1]="0"`, SFLAG *i* becomes `Flag$[`*i*`;1]="1"`, SFLAG *<string expression>* becomes `Flag$=`*<string expression>*, and FLAG$ becomes `Flag$`.

Appendix B
MIKSAM

The HP87 MIKSAM ROM contains a number of statements for manipulating PKEY files, which contain key-record number pairs for a data base. These statements automatically maintain the data in the PKEY file in a B-tree. The Series 200 does not have any similar mechanisms for supporting B-trees. Hence, there is no way to support most of the MIKSAM keywords.

If you wish to port data-base programs to the Series 200, you can create BDAT files to hold key-record number pairs. However, you will need to write your own routines to insert, delete, and find keys and to manage the file. B-trees are rather sophisticated, so you may decide that another data structure would be easier to use.

To move the data from a PKEY file to a BDAT file, first write a Series 80 BASIC program to write the contents of the PKEY file to a DATA file, and then use one of the methods discussed in the the *Data Transportation* chapter to convert the DATA file to a BDAT file.

Appendix C
Matrices

One significant difference between matrix manipulation on the Series 80 and on the Series 200 is that the Series 80 allows empty matrices while the Series 200 does not. You may need to restructure your programs because of this. Consult your Series 80 Matrix ROM manual for an explanation of empty matrices. The manual also shows how the special case of an empty matrix is handled for the built-in matrix functions.

In the Series 200, all of the matrix manipulation capabilities are found in AP2.0.

The Series 80 Matrix ROM contains a number of functions, such as AMAX, which are not supported by the Series 200. You can write your own versions of these functions fairly easily and replace the unsupported Series 80 keywords by invocations of these functions. You will find that the BASE, RANK, and SIZE functions will be very helpful to you in doing this.

Some of the Series 80 matrix functions, like AMAX, have side effects. For example, AMAX computes not only the maximum value in the array, but it also determines the location of this value in the array. This location can be retrieved later by using the AMAXROW and AMAXCOL functions. If your Series 80 program uses AMAXROW and AMAXCOL, you must provide a Series 200 function which determines the location of the array maximum at the same time that it computes AMAX. Since AMAXROW and AMAXCOL work on the Series 80 even if there has been a context switch since the AMAX was performed, the best way to simulate this behavior on the Series 200 is to use a named COM block exclusively to hold values like the location of the array maximum. Each Series 80 location-reporting function can be replaced by a scalar in this COM block. The COM block can be referenced by each program segment which uses a function like AMAX as well as by each program segment which uses a function like AMAXROW.

A sample COM declaration follows. This declaration assumes that you will use all of AMAX, AMIN, CNORM, MAXAB, and RNORM. If this is not true of your program, you may shorten the COM statement to suit your purposes.

```
COM /Matrix/ INTEGER Amaxcol,Amaxrow,Amincol,Aminrow,
Cnormcol,Maxabcol,Maxabrow,Rnormrow
```

Note that the Series 80 and Series 200 may use different algorithms for evaluating MAT operations. This, together with the different numeric representations used by the two systems, may cause results to be somewhat different in the least significant places.

Appendix D
Supported Disc Drives

The following is a list of disc drives supported by Series 200 computers. Software requirements beyond BASIC 2.0 are noted.

Device	Requires
HP7908	AP2.0
HP7911	AP2.0
HP9121S	AP2.0
HP9121D	AP2.0
HP9133	AP2.0
HP9134	AP2.0
HP9135	AP2.0
HP9885	AP2.0
HP9895	
HP82901	
HP82902	