

Porting S80 to RMB

v3.0 / 09 jul 95 / gvg

* The old HP Series-80 (S80) BASIC controllers were popular in their time, but they are now well out of support life and those few that remain in operation will not stay there for long; customers who wish to continue use of their old S80 programs will need to port them to other HP Rocky Mountain BASIC (RMB) platforms, such as RMB for Windows that runs on a PC.

This document describes how to port data and programs between the two platforms and explains some hints on how to convert the S80 program syntax to RMB syntax. It is not specific to any particular RMB platform and will not cover the precise idiosyncracies unique to each platform.

Note that this document also in general applies to programs written in Integral PC Tech BASIC, which is compatible with S80 BASIC.

[%%]

[1.0] DATA TRANSFERS

* It is simple to transfer data from an S80 computer to an RMB platform over HPIB. As an example, I transferred data on tape from an HP-85A with I/O ROM and HPIB card to an RMB machine.

First, I created a data file on tape so I would have something to transfer:

```
10 CLEAR
20 DISP "Creating data!"
30 CREATE "DSET",100,32           ! Create data file DSET.
40 ASSIGN# 1 TO "DSET"
50 FOR N=1 TO 100                ! Store 100 sets of data ...
60   PRINT# 1 ; N,1/N,N*N,SQR(N) ! ... for N = 1 to 100 ...
70 NEXT N                       ! ... store N, inverse, square, root.
80 ASSIGN# 1 TO *                ! Close the data file.
90 DISP "Finished!"
100 END
```

To transfer the data, I configured the RMB unit as a non-system controller, and wrote a program to allow it to read data sent by the HP-85 as follows:

```
10 REAL W,X,Y,Z
20 DIM Buf$(256)                ! Stores data (including CR/LF terminator).
30 CLEAR SCREEN                 ! Clear display.
40 !
50 CONTROL 7,3;1                ! Set RMB address to "1".
60 !
70 FOR N=1 TO 100
80   ENTER 7;Buf$               ! Enter S80 printed output string.
90   PRINT Buf$                 ! Print it.
100  W=VAL(Buf$(1,3))           ! Convert numeric strings to values.
```



```

110   X=VAL(Buf$[5,12])
120   Y=VAL(Buf$[14,18])
130   Z=VAL(Buf$[20,25])
140   PRINT USING 150;W,X,Y,Z ! Print numeric values as a check.
150   IMAGE 3D,X,D.6D,X,5D,X,2D.3D
160   PRINT
170   NEXT N
180   !
190   END

```

I ran this program and then wrote and ran another program for the HP-85 to dump the data:

```

10   CLEAR ! Clear display.
20   PRINTER IS 701 ! Designate "printer" on HPIB.
30   DISP "Reading data file!"
40   ASSIGN# 1 TO "DSET" ! Open data file.
50   FOR N=1 TO 100 ! Read 100 records:
60     READ# 1 ; W,X,Y,Z ! Read N, 1/N, N^2, SQR(N).
70     PRINT USING 80 ; W,X,Y,Z ! Print them over HPIB.
80     IMAGE 3D,X,D.6D,X,5D,X,2D.3D ! NNN N.NNNNNN NNNNN NN.NNN
90   NEXT N
100  ASSIGN# 1 TO * ! Close data file.
110  DISP "Finished!"
120  END

```

Running this program dumps the data over the HPIB; while the RMB program only prints the data, it could just as easily write it to a file.

Note, however, that there is one catch which is not clearly discussed here: an HP-85 with no Plotter/Printer ROM cannot print any more than 32 characters on a line; the excess characters will "wrap around" to the next line. (This is not the case on an HP-86 or HP-87, or an HP-85 with a Plotter/Printer ROM; the print output with can be programmed by adding a "line length" parameter to the PRINTER IS statement, to allow print widths of up to 220 characters -- for the HP-86 and HP-87 -- or 140 characters -- for the HP-85.)

Fortunately (well, sort of), since the user must know the contents of a S80 data file in order to access it, the user has complete control over the output over the HPIB; and -- for data files, at least -- the restricted print width should not cause any problems.

If you have a serial (RS-232) port with your S80, you can similarly send data to a terminal-emulation program on a remote computer (on a PC you can use Windows Terminal and download the data to a file for use later).

[%%]



[2.0] PROGRAM TRANSFERS

* Sending programs over HPIB from an S80 to an RMB machine is just as easy as sending data.

All the user has to do on the S80 is load the program, set PRINTER IS to the RMB machine's HPIB, PLIST the program, and send some sort of terminator over the HPIB when the list is done. The RMB machine reads the lines and stores them in a text file, which can be used loaded as a program using the GET statement.

The following RMB reads an S80 program listing and dumps it to a text file:

```
10 DIM Buf$(256)
20 CLEAR SCREEN
30 !
40 CONTROL 7,3;1 ! Set HPIB address to 1
50 !
60 ON ERROR GOTO Nofile ! If file exists, wipe it.
70 PURGE "S80PGM"
75 Nofile: !
80 OFF ERROR
90 CREATE "S80PGM",1 ! Create file.
100 PRINTER IS "S80PGM" ! Set it up as printer output.
110 !
120 LOOP
130 ENTER 7;Buf$
140 DISP Buf$ ! Display text as a test.
150 IF Buf$(1,5)=">END<" THEN Finis ! Last line, bail out.
160 PRINT Buf$ ! Dump line to file.
170 END LOOP
180 !
190 Finis: !
200 PRINTER IS CRT ! Restore print output to CRT.
210 END
```

Note that the RMB program terminates when it receives the ">END<" string over the HPIB. This means that the user must execute the following statement from the S80 after PLISTing the S80 program:

```
PRINT ">END<"
```

The program will be stored in a file named S80PGM; it may need to be edited to rejoin lines that were broken by wraparound and will definitely need to be modified to get it to conform to RMB syntax.

Again, if the S80 has a serial port the program may be sent to a terminal program and stored to a file.

[%%]



[3.0] AN EXAMPLE (1)

* For an example of porting a program, I found an old HP-85 contributed program that created the "dragon" fractal curve:

```
10 DIM S(13,7),D(13),L(13),F(13)
20 DEG
30 CLEAR
40 DISP "ORDER TO PLOT";
50 INPUT N
60 CLEAR
65 GCLEAR
70 SCALE -1,14.74,-11,1
80 H0=10
90 RESTORE
100 FOR I=1 TO 13 @ READ D(I),L(I),F(I)@ NEXT I
110 L0=H0
120 X0=H0*SQR(3)/2
130 Y0=0 @ X=X0 @ Y=Y0 @ K=1
140 MOVE X,Y
150 S(1,1)=N
160 S(1,2)=90
170 S(1,3)=L0
180 S(1,4)=1
190 GOSUB 220
200 BEEP @ DISP "DONE !"
210 STOP
220 IF S(K,1)=0 THEN 360
230 J=1
240 IF J > 13 THEN 340
250 S(K,5)=J
260 S(K+1,1)=S(K,1)-1
270 S(K+1,2)=S(K,2)+D(J)*S(K,4)
280 S(K+1,3)=S(K,3)*L(J)
290 S(K+1,4)=S(K,4)*F(J)
300 K=K+1
310 GOSUB 220
320 J=S(K,5)+1
330 GOTO 240
340 K=K-1
350 RETURN
360 D1=S(K,2)+120
370 X=X+S(K,3)*COS(D1)
380 Y=Y+S(K,3)*SIN(D1)
390 PLOT X,Y
400 K=K-1
410 RETURN
420 DATA 60,.3333,-1
430 DATA 60,.3333,1
440 DATA 0,.3333,1
450 DATA 300,.3333,1
460 DATA 150,.19245,1
470 DATA 150,.19245,-1
480 DATA 210,.19245,-1
490 DATA 270,.19245,-1
500 DATA 0,.3333,1
510 DATA 210,.19245,1
```



```
520 DATA 210,.19245,-1
530 DATA 0,.3333,-1
540 DATA 0,.3333,1
550 END
```

I used the configuration steps described in the previous section to set up the transfer, then ran the RMB program; it sat there and waited for input. I loaded this program from tape on the HP-85, then executed the following S80 statements:

```
PRINTER IS 701
PLIST
```

The program flowed over the HPIB and appeared on the DISPLAY line; when the listing was done, I went back to the HP-85 and executed the following statement:

```
PRINT ">END<"
```

-- to terminate the transfer.

This gave me the listing in the text file S80PGM. The next trick was to modify the program for running under RMB, as discussed in the next section.

[%%]



[4.0] AN EXAMPLE (2)

* Since the example program was (by S80 standards) simple and cleanly written, modifying it for operation under RMB wasn't too difficult; I merged program lines that had been cut by line wraparound and checked the program for traps.

* One of the biggest traps to check for was the fact that S80 BASIC allows several statements to be concatenated on one line, using the "@" character, as follows:

```
100 FOR I=1 TO 13 @ READ D(I),L(I),F(I)@ NEXT I
```

RMB doesn't like this; each statement has to be on a separate line, as follows:

```
100 FOR I=1 TO 13
101   READ D(I),L(I),F(I)
102 NEXT I
```

This expansion should be performed before loading the program into RMB. RMB will comment out any lines that it doesn't understand when it performs a GET; concatenated lines will be commented out, which causes a major problem if they contain GOTO or GOSUB statements.

The problem occurs when the user tries to RENumber a program: the REN will change all the line numbers, but it *won't* change the line numbers following the GOTO or GOSUB statements in the commented-out lines -- which means that GOTOS or GOSUBS are left pointing at the wrong lines, which can confuse the program a little. It's best to expand these concatenated statements before loading them into RMB.

Note that:

```
IF Z=10 THEN A=5 @ B=0 ELSE A=0 @ B=5
```

-- must be expanded as follows:

```
IF Z=10 THEN
  A=5
  B=0
ELSE
  A=0
  B=5
END IF
```

* Another trap to look out for is multiple variable assignments. S80 allows several variables to be initialized at once:

```
A,B,C = 0
```

RMB doesn't like this; each assignment must be made separately:

```
A = 0
B = 0
C = 0
```

* S80 BASIC allows variables and arrays to have the same name: "A" and "A()" can coexist in the same program. RMB does not tolerate this.

* The syntax of S80 and RMB timing statements is very different. one thing must be remembered above all: S80 timing constants are generally specified in **milliseconds**;



RMB timing constants are generally specified in *seconds*. A WAIT 500 statement innocently ported from an S80 program will cause RMB to mysteriously "hang" for a little over six minutes. (More on this in the next section.)

* S80 is not particular about where the END statement is placed; subroutines, DATA statements, and the like can be placed after it in the program listing. RMB is very fussy about the placement of the END statement; it must be at the actual end of the program listing proper, and is followed only by subprogram and function declarations (which cannot appear *before* the END statement).

* Functions have different syntax in S80 and RMB; they have to be rewritten. (More on this in the next section.)

* S80 and RMB have major differences in numeric types. S80 is based on a BCD-oriented microprocessor, derived from HP's handheld calculators, and so uses BCD floating-point numbers -- while RMB uses binary floating-point numbers.

The problem is that binary floating-point numbers are prone to small round-off errors, while BCD numbers are not; that means that comparisons between floating-point numbers -- or FOR-NEXT loops that use floating-point numbers as counters -- that work on S80 may *not* work properly under HP BASIC.

The size of INTEGER values is different on the two systems: S80 has a maximum value of 99999, while RMB INTEGERS can be no larger than 32767.

* S80 and RMB differ in several keywords; a short list follows:

----- S80 -----	RMB -----
ON KEY# N, "tag" GOSUB/GOTO M	ON KEY N LABEL "tag" GOSUB/GOTO M
OFF KEY# N	OFF KEY N
KEY LABEL	KEY LABELS ON
ALPHA	ALPHA ON
GRAPHICS	GRAPHICS ON
FP	FRACT
IP	INT
#	<>
-----	-----

* Anyway ... the example program actually needed little rewriting; after modifying it, I loaded it into RMB with:

```
GET "S80PGM"
```

The program worked perfectly well without further tweaking, though I added some improvements; the final listing looked like this:

```
10  INTEGER Order
20  DIM S(13,7),D(13),L(13),F(13)
30  DEG
40  !
50  CLEAR SCREEN
60  LOOP
70  DISP "ORDER TO PLOT (1-5)";
80  INPUT Order
90  EXIT IF Order>=1 AND Order<=5
```

```
100 BEEP
110 END LOOP
120 !
130 CLEAR SCREEN
140 GCLEAR
150 PEN 2
160 WINDOW -1,14.74,-11,1
170 H0=10
180 RESTORE
190 FOR I=1 TO 13
200 READ D(I),L(I),F(I)
210 NEXT I
220 L0=H0
230 X0=H0*SQR(3)/2
240 Y0=0
250 X=X0
260 Y=Y0
270 K=1
280 MOVE X,Y
290 S(1,1)=Order
300 S(1,2)=90
310 S(1,3)=L0
320 S(1,4)=1
330 GOSUB Compute
340 BEEP
350 DISP "DONE !"
360 STOP
370 !
380 Compute: !
390 IF S(K,1)=0 THEN Draw_curve
400 J=1
410 LOOP
420 EXIT IF J>13
430 S(K,5)=J
440 S(K+1,1)=S(K,1)-1
450 S(K+1,2)=S(K,2)+D(J)*S(K,4)
460 S(K+1,3)=S(K,3)*L(J)
470 S(K+1,4)=S(K,4)*F(J)
480 K=K+1
490 GOSUB Compute
500 J=S(K,5)+1
510 END LOOP
520 K=K-1
530 RETURN
540 !
550 Draw_curve: !
560 D1=S(K,2)+120
570 X=X+S(K,3)*COS(D1)
580 Y=Y+S(K,3)*SIN(D1)
590 DRAW X,Y
600 K=K-1
610 RETURN
620 !
630 DATA 60,.3333,-1
640 DATA 60,.3333,1
650 DATA 0,.3333,1
660 DATA 300,.3333,1
```



```
670 DATA 150,.19245,1
680 DATA 150,.19245,-1
690 DATA 210,.19245,-1
700 DATA 270,.19245,-1
710 DATA 0,.3333,1
720 DATA 210,.19245,1
730 DATA 210,.19245,-1
740 DATA 0,.3333,-1
750 DATA 0,.3333,1
760 !
770 END
```

The only changes I made were to:

```
% Specify a red pen.

% Deliberately constrain the number of levels of recursion the program
  would plot (the original program had no restrictions; I restricted it to
  levels 1 through 5, since it takes a long time to plot at level 5).

% "Pretty-print" the program listing a bit for later analysis. (I didn't
  *dare* try to modify it; recursive programs are notoriously tricky.)
```

This particular program was easy to port because it was simple and cleanly written; but please remember that the ease with which a program can be converted is proportional to the user's knowledge of the two BASICs, as well as the orderliness of the original S80 program. Some programs cannot be made to work without a complete rewrite.

However, S80 and RMB share a common ancestry in 9845 BASIC; and if an S80 program cannot be ported to RMB, it probably can't be ported to anything else, either.

[%%]



[5.0] FUNCTIONS, TIMERS, DISP

* This section covers a few specialized topics in S80 and RMB syntax differences in more detail.

* S80 and RMB differ in the ways that functions are declared. For example, a function declared on the S80 as:

```
DEF FNZ$(T0)
  IF T0<10 THEN FNZ$="0"&VAL$(T0) ELSE FNZ$=VAL$(T0)
FN END
```

-- must be changed as follows to run under RMB:

```
DEF FNDigits$(INTEGER T)
  IF T<10 THEN
    RETURN "0"&VAL$(T)
  ELSE
    RETURN VAL$(T)
  END IF
FNEND
```

That's the most obvious difference, but there are a few others. In an S80 function definition, variables used in the function are "global"; they are valid for both the function and the main program. In RMB, functions have their own "local" variables that are valid only within the "context" of the function (even if they have the same name as a variable in the main program; note that the function's local variables must also have their types declared so that they match the types of the parameters the function is sent).

Note that the RMB functions must be added to a program *after* the program's END statement.

* The two systems also differ in timer statements, the first and foremost difference being that S80 does not have a battery-backed clock, but RMB systems do. As a result, the clock must be set every time the S80 computer is turned on, using the SETTIME function, as follows:

```
SETTIME T,D
```

-- where "T" is the time in seconds since midnight, and "D" sets the initial value of a "date" count (which is simply a counter that is incremented every time the time count overflows through midnight).

RMB supports a similar statement -- SET TIMEDATE -- but, since the Language Processor has a battery-backed clock, there's no reason to set up the time every time the system is booted.

In S80, the current time can be obtained with:

```
TIME\b
```

-- which returns the number of seconds (down to a resolution of milliseconds) since midnight. The comparable RMB statement is:
\P

```
TIMEDATE
```

-- but TIMEDATE does not return the number of seconds since midnight, it returns the number of seconds since midnight on a certain date. TIMEDATE can be modified to return the number of seconds since midnight by performing modulo division by 86400 (the number of seconds in a day) on its output, as follows:

```
TIMEDATE MOD 86400
```

The only difference is that this statement returns the number of seconds as an integer, while the output of the S80 TIME statement also includes a fractional part that gives the



number of milliseconds.

S80 has three timers; the ON TIMER statement can be used to specify a repeating interrupt at a given interval. For example, to specify an interrupt every five seconds on timer number 2 that is handled by a service routine at line 2500, I specify the statement:

```
ON TIMER #2, 5000 GOSUB 2500
```

The interrupts occur every five seconds (5000 milliseconds) until I specifically disable them with:

```
OFF TIMER #2
```

RMB supports only one repeating interrupt, using the ON CYCLE statement. The RMB equivalent to the ON TIMER statement in the previous example is:

```
ON CYCLE 5 GOSUB 2500
```

This repeating interrupt is disabled with:

```
OFF CYCLE
```

Note that the interval in the S80 statement is specified in milliseconds while the interval in the RMB statement is specified in seconds. (This is true for other timer statements as well. The S80 and RMB WAIT statements look exactly the same, but as noted previously the time parameter in S80 is specified in milliseconds while it is specified in seconds under RMB. This can lead to such interesting observations as: "Gee, this program is sure taking a long time to respond!")

* In S80, DISP displays a line of text on the display; if a second DISP statement is executed, the second line of text appears on the display on the line below the first. Multiple DISP statements can be used to fill the display with text.

In RMB, the output of DISP appears only "prompt" on a "message line" at the bottom of the CRT. If a second DISP statement follows the first, the old prompt is over-written and lost.

The actual equivalent to the S80 DISP statement is the RMB PRINT statement (as long as the PRINTER IS CRT statement has been executed). A certain small amount of judgement is required as to whether an S80 DISP statement should be left as DISP statements or changed to PRINT statements. Simple messages or prompts can be safely left as DISP statements; text that forms part of a larger display (like a help screen) should be converted to PRINT statements.

[<>]

```
v1.0 / 05 dec 88 / gvg
v2.0 / 21 mar 90 / gvg / update for viper 2
v3.0 / 09 jul 95 / gvg / update for hpbw.
```

Note : hpbw = abréviation pour Hewlett-Packard Basic for Windows

```
*****
Cie : BOURBAKY
      BP 53
      13, Rue des Alpes
      07302 TOURNON Cedex - France
Tel (Nat.) : 04 75 07 81 21      Tel (Int.) : +33 4 75 07 81 21
Fax (Nat.) : 04 75 07 29 74      Fax (Int.) : +33 4 75 07 29 74
e-mail : <info@bourbaky.com>
web :    <http://www.bourbaky.com>
*****
```

