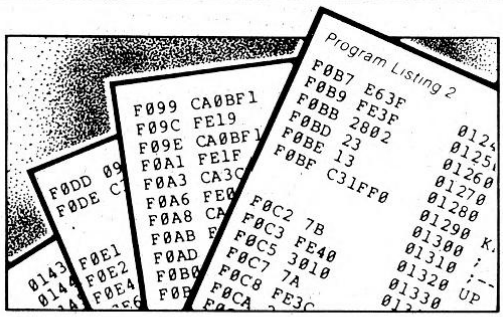
by Hardin Brothers



Understanding relocatable code

In my two previous columns, I discussed fixed-location routines. This month's Assembly-language topic is relocatable code.

Programs written in relocatable code can be executed from any place in memory without any changes. All Basic programs are relocatable because they are entirely independent of their position in memory. Neither Basic nor the programmer cares if the line 10 CLS: CLEAR 500 is stored at 5200H or 0F000H. Both positions interpret and execute the line in the same manner.

Internal CALLs, JUMPs (JPs), and tables are not relocatable because they use absolute instead of relative addresses. The former two mnemonics assemble as 3-bytes. The first byte is the instruction itself; the second two are the address to branch to. If you relocated the object code, this address would be incorrect. Likewise, internal tables are assigned labels that are assembled to absolute addresses.

But if you can avoid internal CALLs, JPs, and tables, writing relocatable machine-language routines isn't difficult. Any program that avoids these three items is relocatable, including programs that CALL or JP to ROM routines and those that access fixed-position tables. In general, short routines are often relocatable; complex commercial programs rarely are.

Relocatable code is not magical or pure, but it does lend itself to at least three common interfacing techniques when combined with Basic programs. Relocatable routines, unlike fixed-position routines, can be stored in string variables, literal strings, and integer arrays.

The Demonstration Routines

Program Listings 1 and 3 show left and right screen rotations; Program Listing 2 interfaces the rotation routine with Basic.

You can alter the routines in Listings 1 and 3 to clear the screen by scrolling everything off to the left or right. Simply change lines 280 and 600 in Listing 1 and lines 330 and 690 in Listing 3 to read: LD A,20H. Scrolling new information onto the screen requires a 1K buffer and some minor rewriting. If you modify the routines, be sure to adjust the data statements in the Basic programs.

Listings 1 and 3 each contain a time delay to help synchronize the screen rotation to the video refresh circuitry and the video scan rate. I used the delay value (0C8H or 200) that gave the best-looking screen on my Model I; you may have to adjust the delay for your computer. I found my delay value by trial-and-error, so if any reader can explain how to calculate the correct number of T states to synchronize the software with the computer, please write.

Packing String Variables

The simplest method for interfacing a relocatable routine to Basic is to use a string variable. First, translate the routine into 1-byte decimal values (use the Convert program from last month), and place those values in a data statement. After clearing sufficient string space, load the routine into a string using the instructions:

```
FOR I = 1 TO (routine length)
READ D
A$ = A$ + CHR$(D)
NEXT I
```

(See lines 80, 90, 150, and 160 in Listing 3.)

Your second task is to access the routine from Basic, for which you'll need the VARPTR function. To find the USR

Program Listing 1

```
                00100 ;*********************************
                00110 ;*                               *
                00120 ;*  LEFT SCREEN SCROLL           *
                00130 ;*     ROUTINE                   *
                00140 ;*   USE ONLY WITH               *
                00150 ;*     STRING VARIABLES          *
                00160 ;*                               *
                00170 ;*********************************
                00180 ;
                00190 ;
7F00            00200           ORG     7F00H
7F00 0640       00210           LD      B,40H           ;64 TIMES THROUGH ROUTINE
7F02 C5         00220 LOOP1     PUSH    BC              ;SAVE COUNTER
7F03 11003C     00230           LD      DE,3C00H        ;DE==> TOP OF SCREEN
7F06 21013C     00240           LD      HL,3C01H        ;HL==> TOP OF SCREEN +1
7F09 0610       00250           LD      B,10H           ;16 LINES PER SCREEN
7F0B C5         00260 LOOP2     PUSH    BC              ;SAVE LINES COUNTER
7F0C 013F00     00270           LD      BC,003FH        ;63 CHAR.S IN LINE
7F0F 1A         00280           LD      A,(DE)          ;GET 1ST CHAR. IN LINE
7F10 EDB0       00290           LDIR                    ;MOVE THEM ALL LEFT
7F12 12         00300           LD      (DE),A          ;THE FIRST BECOMES LAST
7F13 13         00310           INC     DE              ;BUMP
7F14 23         00320           INC     HL              ;  POINTERS
7F15 C1         00330           POP     BC              ;GET LINE COUNT
7F16 10F3       00340           DJNZ    LOOP2           ;DO 16 LINES
7F18 01C800     00350           LD      BC,00C8H        ;VALUE FOR TIME DELAY
7F1B CD6000     00360           CALL    0060H           ;ROM DELAY ROUTINE
7F1E C1         00370           POP     BC              ;GET CHARACTER COUNTER
7F1F 10E1       00380           DJNZ    LOOP1           ;GO UNTIL FINISHED
```

Listing 1 continues

The Key Box

Model I and III
16K RAM, Cassette Basic
32K RAM, Disk Basic
Editor/Assembler Optional

address, include in your Disk Basic program:

```
A=PEEK(VARPTR(A$)+2)*256
      + PEEK(VARPTR(A$)+1)
IF A>32767 THEN A=A-65536
DEFUSR0=A
```

The command VARPTR(A$) returns the address of the length of A$. The 2 bytes in memory following the string length contain the address of the beginning of the string. VARPTR(A$)+2 contains the most-significant byte; multiply its contents by 256 before adding them to the contents of VARPTR(A$)+1. Put the resulting value in integer range before the DEFUSR statement.

Finding the string address is simpler in tape-based systems. Once again the VARPTR function is used, but no extra variables or conversions are needed. Simply transfer the string address to the USR address 1 byte at a time as follows:

```
POKE 16526,PEEK(VARPTR(A$)+1)
POKE 16527,PEEK(VARPTR(A$)+2)
```

You need to observe one important rule when packing string variables. Because any string command can invoke Basic's garbage collection routine and move all strings around in high memory, reset the USR address before every new access of the routine.

Variable string packing has three disadvantages. First, the Data statements must be included in the Basic program, which uses up memory. Second, the packed strings reside in cleared high memory and cause more frequent program pauses for garbage collection. And third, the data must be read and the strings packed each time the program is run; if the program contains more than one or two short routines, string packing can cause unnecessary delays.

Packing Literal Strings

Instead of storing machine-language routines in string variables that move around in high memory, store them in literal strings that are fixed within the Basic program itself. Once the Basic program is loaded into memory, literal strings don't move, because they are never transferred to the cleared string storage area. To keep the strings from being transferred to high memory, make sure your program never modifies them after they are defined.

Listing 1 continued

```
7F21 C9        00390          RET                    ;RETURN TO BASIC
               00400 ;
               00410 ;
               00420 ;*********************************
               00430 ;*                               *
               00440 ;*  RIGHT SCREEN SCROLL           *
               00450 ;*    ROUTINE                    *
               00460 ;*   USE ONLY WITH               *
               00470 ;*    STRING VARIABLES           *
               00480 ;*                               *
               00490 ;*********************************
               00500 ;
               00510 ;
7F80           00520          ORG     7F80H
7F80 0640      00530          LD      B,40H          ;64 TIMES THROUGH ROUTINE
7F82 C5        00540 LOOP3    PUSH    BC             ;SAVE COUNTER
7F83 11FF3F    00550          LD      DE,3FFFH       ;DE==> BOTTOM OF SCREEN
7F86 21FE3F    00560          LD      HL,3FFEH       ;HL==> BOT. OF SCREEN -1
7F89 0610      00570          LD      B,10H          ;16 LINES PER SCREEN
7F8B C5        00580 LOOP4    PUSH    BC             ;SAVE COUNTER
7F8C 013F00    00590          LD      BC,003FH       ;63 CHAR.S PER LINE
7F8F 1A        00600          LD      A,(DE)         ;GET LAST CHAR. OF LINE
7F90 EDB8      00610          LDDR                   ;MOVE THEM ALL RIGHT
7F92 12        00620          LD      (DE),A         ;THE LAST BECOMES 1ST
7F93 1B        00630          DEC     DE             ;BUMP
7F94 2B        00640          DEC     HL             ;  POINTERS
7F95 C1        00650          POP     BC             ;GET LINE COUNTER
 F96 10F3      00660          DJNZ    LOOP4          ;FINISH SCREEN
7F98 01C800    00670          LD      BC,00C8H       ;VALUE FOR TIME DELAY
7F9B CD6000    00680          CALL    0060H          ;ROM DELAY ROUTINE
7F9E C1        00690          POP     BC             ;GET COUNT
7F9F 10E1      00700          DJNZ    LOOP3          ;GO UNTIL DONE
7FA1 C9        00710          RET                    ;RETURN TO BASIC
0000           00720          END
00000 TOTAL ERRORS
```

```
10 CLEAR 500
20 CLS
30 '*** CODE FOR ROTATE LEFT ***
40 DATA 6, 64, 197, 17, 0, 60, 33, 1, 60, 6
50 DATA 16, 197, 1, 63, 0, 26, 237, 176, 18
60 DATA 19, 35, 193, 16, 243, 1, 200, 0, 205
70 DATA 96, 0, 193, 16, 225, 201
80 FOR I=1 TO 34: READ C
90 A$=A$+CHR$(C): NEXT I
100 '*** CODE FOR ROTATE RIGHT ***
110 DATA 6, 64, 197, 17, 255, 63, 33, 254, 63
120 DATA 6, 16, 197, 1, 63, 0, 26, 237, 184, 18
130 DATA 27, 43, 193, 16, 243, 1, 200, 0, 205, 96
140 DATA 0, 193, 16, 225, 201
150 FOR I=1 TO 34: READ C
160 B$=B$+CHR$(C): NEXT I
170 M$=STRING$(23,133)+" Screen Rotation "+STRING$(23,138)
180 FOR I=1 TO 6: PRINT M$: NEXT
190 PRINT: PRINT STRING$(15,32)
     + "Press any key to change directions"
     + STRING$(15,32)
200 FOR I=1 TO 6: PRINT M$: NEXT
210 F=0
220 IF F=0 THEN C$=A$ ELSE C$=B$
230 A=VARPTR(C$): B=PEEK(A+2)*256 + PEEK(A+1)
240 IF B>32767 THEN B=B-65536
250 DEFUSR=B
260 '******* Lines 220 - 250 above are for DISK BASIC
          For Tape Systems use:
       230 A=VARPTR(C$)
       240 POKE 16526,PEEK(A+1)
       250 POKE 16527,PEEK(A+2)
270 K=0
280 A=USR(0)
290 FOR I= 1 TO 100: NEXT
300 K=K+1
310 IF K<5 AND INKEY$="" THEN 280
320 F=NOT F: GOTO 220
```

Program Listing 2

However, there is one important difference in the way you must write a routine that will be packed into a literal string. The Basic interpreter reads any byte of 0 in your routine as an end-of-line marker and any byte of 22H as an end-of-string marker; your routine cannot contain either of these values. It will run fine the first time, but after you press break and rerun the program, it is likely to bomb, leaving you with a syntax error in a line that doesn't even ex-

```
Program Main Line:

        .
        .
        CALL    000BH     ;Shift address of next instruction
                          ; to HL pair
        JR      SUB1      ;Relative jump to subroutine
        .                 ;Return here after subroutine
        .
        .
        .
SUB1    INC     HL        ;This is the CALLed subroutine
        INC     HL        ;HL+2 = => return address
        PUSH    HL        ;Put return address on stack
        .                 ; Now perform subroutine processing
        .
        RET               ; end subroutine with RETurn
```

Fig. 1. Using 000BH for Relocatable Calls. Uses HL register pair.

```
Program Main Line:

        .
        .
        PUSH    HL        ;Save HL value on stack
        CALL    000BH     ;Shift address of next instruction
                          ; to HL pair
        JR      SUB1      ;Relative jump to subroutine
        .                 ;Return here after subroutine
        .
        .
SUB1    INC     HL        ;This is the CALLed subroutine
        INC     HL        ;HL+2 = => return address
        EX      (SP),HL   ;Put return address on stack
        .                 ;and recover HL value
        .                 ; Now perform subroutine processing
        .
        RET               ;end subroutine with RETurn
```

Fig. 2. Using 000BH for Relocatable Calls. Retains value in HL register pair.

Program Listing 3

```
                00100 ;*********************************
                00110 ;*                               *
                00120 ;*  LEFT SCREEN SCROLL           *
                00130 ;*     ROUTINE                   *
                00140 ;*   (NO 0 BYTES USED)           *
                00150 ;*    USE WITH VARIABLE OR       *
                00160 ;*    LITERAL STRINGS            *
                00170 ;*                               *
                00180 ;*********************************
                00190 ;
                00200 ;
7F00            00210          ORG     7F00H
7F00 0640       00220          LD      B,40H          ;64 TIMES THROUGH ROUTINE
7F02 C5         00230 LOOP1    PUSH    BC             ;SAVE COUNTER
7F03 AF         00240          XOR     A              ;A=0
7F04 163C       00250          LD      D,3CH
7F06 5F         00260          LD      E,A            ;DE==> TOP OF SCREEN
7F07 21013C     00270          LD      HL,3C01H       ;HL==> TOP OF SCREEN +1
7F0A 0610       00280          LD      B,10H          ;16 LINES PER SCREEN
7F0C C5         00290 LOOP2    PUSH    BC             ;SAVE LINE COUNTER
7F0D AF         00300          XOR     A              ;A=0
7F0E 47         00310          LD      B,A
7F0F 0E3F       00320          LD      C,3FH          ;BC=63 CHAR.S PER LINE
7F11 1A         00330          LD      A,(DE)         ;GET 1ST CHAR. OF LINE
7F12 EDB0       00340          LDIR                   ;MOVE THEM ALL LEFT
7F14 12         00350          LD      (DE),A         ;THE 1ST BECOMES LAST
7F15 13         00360          INC     DE             ;BUMP
7F16 23         00370          INC     HL             ;  POINTERS
7F17 C1         00380          POP     BC             ;GET LINE COUNTER
```

Listing 3 continues

"Variable string packing has three disadvantages."

ist. Therefore, Listing 3 shows the horizontal scroll routines rewritten without any zero bytes.

Program Listing 4 shows the literal string method in operation. The strings are defined in lines 110 and 120 with enough dummy characters to accommodate the entire routine. Next, the address of each string is found using VARPTR. Then the data list is read and each value is POKEd into the string. The string is now packed with the routine.

Be sure to save or CSAVE the program before trying to run it. If you stop the Basic program after line 160 and list it, you will see unintelligible garbage as the strings scroll by; this is perfectly normal. Don't edit lines containing packed strings, because by doing so you'll lose part of the machine-language routine.

After the literal strings are packed, you can conserve memory by deleting the data statements and the Read and POKE commands from your Basic program (lines 10–100, 140, and 160 in Listing 4). You can save and CSAVE a program with packed strings just like any other program. If your Basic program uses literal strings to store machine-language routines, it only has to find the routine addresses once because the strings never move.

Any Program Is Relocatable

String packing is the most commonly used technique for interfacing relocatable machine-language routines with Basic programs, but it doesn't work with routines that aren't relocatable. Sometimes it seems impossible to write a routine that does not involve calls to internal subroutines, absolute JPs, or internal tables. There is a way, though, to make all your machine-language programs relocatable. (Jesse Bob Overholt's column in issue 16 of *The Alternate Source Journal* suggested the following technique.)

The folks at Microsoft have provided, in ROM, a key to making any routine relocatable. At 000BH (11 decimal) there are two bytes that, according

Continues on p. 45

Listing 3 continued

```
7F18 10F2     00390          DJNZ    LOOP2           ;DO 16 LINES
7F1A 0EC8     00400          LD      C,0C8H          ;B ALREADY = 0
7F1C 0B       00410 LOOP3    DEC     BC              ;DROP COUNT
7F1D 78       00420          LD      A,B             ;GET MSB
7F1E B1       00430          OR      C               ;MERGE LSB
7F1F 20FB     00440          JR      NZ,LOOP3        ;LOOP UNTIL DONE
7F21 C1       00450          POP     BC              ;GET SCREEN COUNTER
7F22 10DE     00460          DJNZ    LOOP1           ;SHIFT SOME MORE
7F24 C9       00470          RET                     ;RETURN TO BASIC
              00480 ;
              00490 ;
              00500 ;*********************************
              00510 ;*                               *
              00520 ;*  RIGHT SCREEN SCROLL          *
              00530 ;*      ROUTINE                  *
              00540 ;*    (NO 0 BYTES USED)          *
              00550 ;*                               *
              00560 ;*********************************
              00570 ;
              00580 ;
7F80          00590          ORG     7F80H
7F80 0640     00600          LD      B,40H           ;64 TIMES THROUGH ROUTINE
7F82 C5       00610 LOOP4    PUSH    BC              ;SAVE COUNTER
7F83 11FF3F   00620          LD      DE,3FFFH        ;DE==> BOTTOM OF SCREEN
7F86 21FE3F   00630          LD      HL,3FFEH        ;HL==>BOT. OF SCREEN -1
7F89 0610     00640          LD      B,10H           ;16 LINES PER SCREEN
7F8B C5       00650 LOOP5    PUSH    BC              ;SAVE COUNTER
7F8C AF       00660          XOR     A               ;A=0
7F8D 47       00670          LD      B,A
7F8E 0E3F     00680          LD      C,3FH           ;63 CHAR.S PER LINE
7F90 1A       00690          LD      A,(DE)          ;GET LAST CHAR. IN LINE
7F91 EDB8     00700          LDDR                    ;MOVE THEM ALL RIGHT
7F93 12       00710          LD      (DE),A          ;THE LAST BECOMES FIRST
7F94 1B       00720          DEC     DE              ;BUMP
7F95 2B       00730          DEC     HL              ;  POINTERS
7F96 C1       00740          POP     BC              ;GET COUNT
7F97 10F2     00750          DJNZ    LOOP5           ;FINISH SCREEN
7F99 0EC8     00760          LD      C,0C8H          ;BC HAS VALUE FOR DELAY
7F9B 0B       00770 LOOP6    DEC     BC              ;DROP COUNT
7F9C 78       00780          LD      A,B             ;GET MSB
7F9D B1       00790          OR      C               ;MERGE WITH LSB
7F9E 20FB     00800          JR      NZ,LOOP6        ;LOOP UNTIL DONE
7FA0 C1       00810          POP     BC              ;GET COUNT
7FA1 10DF     00820          DJNZ    LOOP4           ;GO UNTIL ALL LINES DONE
7FA3 C9       00830          RET                     ;BACK TO BASIC
0000          00840          END
00000 TOTAL ERRORS
```

```
5 CLS: CLEAR 500
10 '***** DATA FOR LEFT SCROLL ****
20 DATA 6, 64, 197, 175, 22, 60, 95, 33, 1, 60
30 DATA 6, 16, 197, 175, 71, 14, 63, 26, 237, 176
40 DATA 18, 19, 35, 193, 16, 242, 14, 200, 11, 120
50 DATA 177, 32, 251, 193, 16, 222, 201
60 '***** DATA FOR RIGHT SCROLL ****
70 DATA 6, 64, 197, 17, 255, 63, 33, 254, 63, 6
80 DATA 16, 197, 175, 71, 14, 63, 26, 237, 184, 18
90 DATA 27, 43, 193, 16, 242, 14, 200, 11, 120, 177
100 DATA 32, 251, 193, 16, 223, 201
110 A$="Save thirty-seven spaces here 1234567"
120 B$="Save thirty-six spaces here 00123456"
130 C=VARPTR(A$): A=PEEK(C+2)*256 + PEEK(C+1)
140 FOR I=A TO A+36: READ X: POKE I,X: NEXT I
150 D=VARPTR(B$): B=PEEK(D+2)*256 + PEEK(D+1)
160 FOR I=B TO B+35: READ X: POKE I,X: NEXT I
170 M$=STRING$(23,133)+" Screen Rotation "+STRING$(23,138)
180 FOR I=1 TO 6: PRINT M$: NEXT
190 PRINT: PRINT STRING$(15,32)
      + "Press any key to change directions"
      + STRING$(15,32)
200 FOR I=1 TO 6: PRINT M$: NEXT
210 DEFUSR0=A: DEFUSR1=B    'Use this line for Disk Basic ONLY!!
220 F=0
230 K=0
240 IF F=0 THEN X=USR0(0) ELSE X=USR1(0)
245 '******* Line 240 is only for Disk Basic
        For tape systems use
        235 IF F=0 THEN POKE 16526,PEEK(C+1):
            POKE 16527,PEEK(C+2) ELSE
            POKE 16526,PEEK(D+1): POKE 16527,PEEK(D+2)
        240 X=USR(0)
250 FOR I=1 TO 100: NEXT
260 K=K+1
270 IF K<5 AND INKEY$="" THEN 240
280 F=NOT F: GOTO 230
```

Program Listing 4

Continued from p. 42

to James Farvour's *Microsoft Basic Decoded and Other Mysteries*, are never used by Level II. These two bytes are E1H and E9H, or, in Assembly language:

```
POP  HL
JP   (HL)
```

These bytes do not appear together anywhere else in ROM, and their chance of appearing together randomly is miniscule. Obviously they have a purpose: to assist all machine-language programmers. They can save you hours of work trying to make a machine-language routine relocatable.

Here's what happens when your routine performs a Call 000BH. First, the address of the next instruction is placed on the stack by the Call instruction. Then control is passed to 000BH, where the return address is POPed off the stack into the HL register and control is passed back to your routine—to the instruction after the call. You could obtain the same result with a call and a RET, except the HL register holds the same value as the program counter.

Perform a relative jump (JR) to your subroutine. HL still points to the JR instruction, so increment HL twice to point to the instruction after JR. Then push HL on the stack. Your program can now perform the subroutine and RET to the instruction after JR. (See Fig. 1.) This uses five more bytes than a regular call to a subroutine, but it makes your program truly relocatable without losing the power of the Call instruction.

There is one drawback: the original value in HL is obliterated. To pass the HL value to your subroutine, push HL *before* the call to 000BH and, in the subroutine, perform an EX (SP),HL instead of pushing HL. (See Fig. 2.) This takes an extra byte and 20 more T states, but the negligible delay is worth the extra convenience.

You can use the same idea to find tables in relocatable programs and for more intricate control shifting. Usually, the 000BH call saves you more in time and RAM space than your program expends avoiding a subroutine call. With Call 000BH you can make all your machine-language routines relocatable. You never again have to face the problem of two routines, written at different times, conflicting because they use the same memory space. ■