APPLESOFT PROGRAM MOVER

Move your Applesoft programs on the fly to safe locations without time-consuming reloading

Is your long Applesoft program moving in on Hi-Res territory? Jog your Apple's memory with Applesoft Program Mover (APM), which moves a running Applesoft BASIC program up or down to any available location in main RAM and leaves it running with its variables intact. Use it as a fast relocating loader, and almost halve loading time. Or use it as a dynamic relocator while programs are running to take advantage of unused memory.

Applesoft programs longer than 6K which use the mixed text and graphics available on Hi-Res page 1 often begin with a line similar to

```
IF PEEK(104) < > 64 THEN POKE
   103,1:POKE 104,64 : POKE
   16384,0 : PRINT CHR$(4)"RUN
   PROGRAM"
```

With APM you can replace the above line with

```
POKE 6,64 : PRINT CHR$(4)"BRUN
   APM.OBJ"
```

Using the old method, the program PEEKs the page number held by the start of program pointer (location 104) to see whether the program starts above the Hi-Res screen buffer on page 64. If not, it sets the pointer to 64 with a POKE (telling BASIC and DOS where the program starts) and then RUNs itself from disk a second time.

With APM as a relocating loader, the BASIC program LOADs just once before you can start using it. The time difference is substantial; a 45-sector file can be up and running in 13 seconds with APM, compared to 24 seconds without it.

Some programs, instead of reloading, use separate reloader programs to set the pointers and load them; however, these are usually written to load only a specific program. APM has the advantage of being generic. Only one copy of APM is needed on a disk containing several long BASIC programs.

APM is a short, relocatable machine language program you can BRUN directly from disk or CALL from BASIC. It will run on any Apple II with Applesoft BASIC, DOS 3.3 or ProDOS, and one disk drive.

USING THE PROGRAM

To use APM, simply POKE into location 6 the page number where you want your BASIC program moved, then CALL or BRUN APM.OBJ.

For example, suppose your long program makes only occasional use of Hi-Res graphics. You can use APM to its full capacity as a dynamic relocator to reclaim up to 14K of additional memory while the BASIC program uses the text display. First load APM to a safe location where it can be CALLed whenever you want it. Since page 3 and the area just below HIMEM: are often used to hold shape tables and the like, the safest location for APM is tucked neatly between the BASIC program and its variables. (Please see the demo program in **Listing 3** for program line references.)

Set LOMEM: exactly 314 bytes above the end of the BASIC program (**line 80**). Then BLOAD APM.OBJ into that safe area (**line 110**); note the use of a function variable to represent the load address of APM.OBJ.

When you want to use Hi-Res page 1, issue a POKE 6,64 (or POKE 6,96 to use Hi-Res page 2) and CALL to where APM has been loaded (**line 130**); when you don't need to protect the Hi-Res buffer anymore (when in text mode, for example) issue a POKE 6,8 and CALL (**line 170**). Page 8 is the normal starting location for Applesoft programs. A program starting on page 64 doesn't use the 14K available between page 8 and page 64.

This technique can also provide a clean exit from a relocated Applesoft program: Issuing a POKE 6,8 followed by a CALL or BRUN will neatly restore the start-of-program pointers to normal for the next program.

Error Trapping

APM will not let you move your code below page 8. It also prevents your program and variables from overlapping string storage in upper memory (see **lines 31-41** of **Listing 1**).

Additional Applications

Applesoft Program Mover has great possibilities. You might team it with AUX-MOVE in 128K IIe's and IIc's to move binary files as well as BASIC files to auxiliary memory. Or perhaps you'll move a program out of the way while a machine language program at $800 does its work — then back when the machine language program is finished.

ENTERING THE PROGRAM

If you have an assembler, type in the source code from **Listing 1** and save the object code as APM.OBJ. If you do not have an assembler, key in the hex code from **Listing 2**. Save it to disk with the command

TABLE 1: Zero Page Locations

Name	Address		Function
	dec.	hex.	
TXTTAB	103-104	$67-68	hold the location of the start of the program.
VARTAB	105-106	$69-6A	hold the location where storage of simple variables starts.
ARYTAB	107-108	$6B-6C	hold the location where storage of array variables begins.
STREND	109-110	$6D-6E	hold the location where storage of array variables ends.
PGREND	175-176	$AF-B0	hold the location of the end of the program.
TXTPTR	184-185	$B8-B9	the text pointer—the program location currently being interpreted.

```
BSAVE APM.OBJ,A$8E00,L$13A
```

Finally, key in the demo program shown in **Listing 3** and save it on disk with

```
SAVE APM.DEMO
```

For help with entering *Nibble* program listings, see the Typing Tips section.

HOW THE PROGRAM WORKS

I wondered whether you could move an Applesoft program around without harming it. In disovering that you can, I had to address four areas: program pointers, variables, string arrays, and link field addresses.

Applesoft Program Mover follows three steps to move a program. First, it adjusts Applesoft program and variable pointers, then it adjusts link field addresses that point to program lines, and finally it moves the program byte by byte to its new location.

TABLE 2: Variable Names and Types

Name	Type	ASCII type	ASCII code			
			decimal		hex	
A	Real	pos-pos	65	00	$41	$00
A%	Integer	neg-neg	193	128	$C1	$80
A$	String	pos-neg	65	128	$41	$80
FN A	Function	neg-pos	193	00	$C1	$00

Program Pointers

Applesoft uses several pair of zero page locations as pointers, including those shown in **Table 1**. Addresses are always given low-order byte first with the page number held in the high-order byte. VARTAB+1 ($6A) holds the page number on which variables begin; VARTAB ($69) holds the location on that page where the variables begin. For simplicity, APM ignores the low-order byte whenever possible.

The values held in these locations must be changed as follows whenever a program is moved.

1. The page number held in VARTAB+1 is subtracted from the destination page number previously POKEd into PAGE from BASIC.
2. The difference is saved in TEMP. Since APM must use these pointers in its other routines, this difference is added to the value held in the high-order byte in other pointer locations used by APM.

The zero page location (**Table 1**) shows the pointers to the locations where an Applesoft program and its variables and strings reside in memory.

Variables

Function variables (defined with DEF FN) and string constants use pointers that hold addresses within the program text/variable space (these must be adjusted appropriately).

APM first examines simple variables (between VARTAB and ARYTAB) for string literals and functions, then checks arrays (between ARYTAB and STREND) for string literals.

APM examines each variable name to identify variable type. Applesoft uses two-byte variable names with positive (0-127) or negative (128-255) ASCII numbers to differentiate type, and considers all variables to have two-byte names, regardless of their length in the program text. See **Table 2** for a listing of variable types and storage characteristics. Simple variables are each allotted seven-byte areas.

The first function pointers hold the address of the function formula following the equals sign (=) in the DEF FN statement in the program text; the second points to the argument data within the variable space.

APM resets bytes 4 and 6 in function variables and compares byte 5 with FRETOP+1 before attempting to reset a string pointer. If the content of byte 5 is less than FRETOP+1, the string is a literal and must be reset.

The program steps through the variable space seven bytes at a time and checks whether ARYTAB has been reached.

String Arrays

Strings in arrays sometimes point to text in the program. Here, for example, strings point to DATA statements within the program text:

```
22Ø DIM Z$(73) : FOR X = 1 TO 73 :
    READ Z$(X) : NEXT X
```

Information about strings in arrays consists of an array header, followed by three-byte pointers that hold the length and address of each string.

An array header for a string variable contains the variable name in bytes 1-2, the offset to the next array in bytes 3-4, the number of indices in byte 5, and the number of elements in each pair of bytes following.

APM adds the offsets in bytes 3 and 4 to the present location and saves the sum. Then it examines the variable name. If it's a string, APM locates the zeroth element by adding to VARPNT twice the number of indices plus the five overhead bytes in the array header. Then it examines each element, adding the contents of TEMP to all pointers which contain less than FRETOP+1. This alters the pointers of null elements but—no harm, no foul.

One element sits atop the last. After examining each one, APM checks to see whether the next array has been reached. If it has, the program checks the next header; if not, it checks the next element.

Link Field Addresses

Each Applesoft program line contains the absolute address of the next line (the link field address). The line

```
1Ø HGR : REM
```

is tokenized as follows: 09, 08, 0A, 00, 91, 3A, B2, 00. The first two bytes, 09 and 08, are the link field address. The next two bytes are the line number. The next three bytes are the tokens for HGR, :, and REM. The zero byte marks the end of the line.

APM adjusts the page number held in the high-order byte using the addresses held in LINK as pointers.

Finally, the program and its variables are moved to its new location, using these zero-page locations:

$3C-$3D	Point to the start of the source block to be moved
$3E-$3F	Point to the end of the source block to be moved
$42-$43	Point to the destination address of the move

The memory location preceding an Applesoft program must contain a zero (hence the POKE 16384,0 in the loader), so APM sets the move parameters to point to the address immediately preceding the program. APM then moves the BASIC program byte by byte to its new location.

LISTING 1: APM Source Code

```
**************************
*           APM          *
*     BY  MIKE MIYAKE    *
*   COPYRIGHT(C) 1988    *
*    MICROSPARC, INC.    *
*   CONCORD, MA 01742    *
*                        *
*    MERLIN ASSEMBLER    *
**************************
*
*
PAGE     EQU 6                  ;holds dest. page #
TEMP     EQU 7                  ;holds source page - dest.page
LINK     EQU 8                  ;holds link field addresses
A1       EQU $3C                ;source block start ptr.for move
A2       EQU $3E                ;source end ptr.for move
A4       EQU $42                ;dest.ptr for move
TXTTAB   EQU $67                ;start of BASIC program ptr.
VARTAB   EQU $69                ;start of variables ptr.
ARYTAB   EQU $6B                ;start of array space ptr.
STREND   EQU $6D                ;end of array storage ptr.
FRETOP   EQU $6F                ;bottom of strings ptr.
VARPNT   EQU $83                ;general storage
PGREND   EQU $AF                ;end of program ptr.
TXTPTR   EQU $B8                ;address being interpreted
PTR      EQU $CE
MOVE     EQU $FE2C              ;monitor move routine
*-----------------------------
         ORG $8E00
*-----------------------------
         LDA PAGE               ;get page #
         STA A4+1               ;insert in dest.
         CMP #8                 ;is dest.page too low?
         BCC BADPAGE            ;yes, then exit
         SBC TXTTAB+1           ;find difference
         STA TEMP               ;save it
         CLC
         ADC STREND+1
         CMP FRETOP+1           ;is dest.page too high?
         BCC GOOD               ;no, then branch & begin
BADPAGE  RTS
GOOD     LDY #0                 ;set MOVE parms
         STY A1                 ;
         STY A4
         INY
         STY LINK               ;set link field ptrs
         LDA TXTTAB+1
         STA A1+1
         STA LINK+1
         LDA STREND             ;copy STREND for MOVE
         STA A2
         LDA STREND+1
         STA A2+1
         LDA VARTAB
         STA PTR                ;copy VARTAB
         LDY VARTAB+1
         STY PTR+1
L1       CMP ARYTAB             ;end of simple variables?
         LDA PTR+1
         SBC ARYTAB+1
         BCS L5                 ;yes, branch to do arrays
         LDY #0
         LDA (PTR),Y            ;fetch 1st byte of var.name
         BMI L2                 ;if neg, it may be a FN
         INY                    ;if pos, fetch next byte--
         LDA (PTR),Y            ;must be neg.to be string
         BPL L4                 ;if pos, branch to get next var
         LDY #4
         BPL L3
L2       INY                    ;(FN's)
         LDA (PTR),Y            ;2nd byte of FN must be pos
         BMI L4                 ;else branch to get next var.
         LDY #3
         CLC
         LDA (PTR),Y            ;adjust FN argument ptr.
         ADC TEMP
         STA (PTR),Y
         LDY #5
L3       LDA (PTR),Y            ;get FN formula/string ptr
         CMP FRETOP+1           ;is it a string literal or FN?
         BCS L4                 ;no, then get next variable
         ADC TEMP               ;adjust ptr.to FN/string
         STA (PTR),Y
L4       CLC                    ;get ptr.to next variable
         LDA PTR
         ADC #7                 ;simple vars.stored in 7 bytes
         STA PTR
         BCC L1
         INC PTR+1
         BCS L1                 ;always branch
L5       LDA PTR                ;begin to process arrays
         CMP STREND             ;has end of arrays been reached
         LDA PTR+1
         SBC STREND+1
         BCS L12                ;yes, branch to do BASIC ptrs
         LDY #2                 ;no, get offset to next array
         CLC
         LDA PTR
         STA VARPNT             ;save ptr.to current array
         ADC (PTR),Y            ;add offset to next array
         PHA
         INY
         LDA PTR+1
         STA VARPNT+1
         ADC (PTR),Y
         STA PTR+1              ;and save ptr.to next array
         PLA
         STA PTR
         LDY #0
         LDA (VARPNT),Y         ;fetch 1st byte in var.name
         BMI L5                 ;if neg, it's not a string, branch
         INY
         LDA (VARPNT),Y         ;fetch 2nd byte
         BPL L5                 ;if pos, it's not a string, branch
         LDY #4                 ;(check string arrays only)
         LDA (VARPnT),Y         ;fetch # of indices
         ASL                    ;2 bytes/index
         BCC L6
         INC WARPNT+1           ;find zeroth element:
         CLC
L6       ADC WARPNT             ;add header location to
         BCC L7                 ;2 - the # of indices
         INC WARPNT+1
         CLC
L7       ADC #5                 ;plus 5 bytes overhead
         STA WARPNT             ;save in VARPNT
         BCC L8
         INC WARPNT+1
L8       LDY #2                 ;get MSB of ptr
L9       LDA (VARPNT),Y
         CMP FRETOP+1           ;is string a literal?
         BCS L10                ;no, get next element
         ADC TEMP               ;yes, adjust ptrs
         STA (VARPNT),Y
L10      CLC                    ;fetch next element
         LDA WARPNT
         ADC #3                 ;3 bytes each
         STA WARPNT
         BCC L11
         INC WARPNT+1
L11      CMP PTR                ;all elements done?
         LDA WARPNT+1
         SBC PTR+1
         BCC L9                 ;no, get another
         BCS L5                 ;yes, get next array
L12      LDX #8
L13      DEX
         CLC
         LDA TXTTAB,X           ;reset BASIC pointers:
         ADC TEMP               ;TXTTAB, VARTAB,ARYTAB,
         STA TXTTAB,X           ;STREND
         DEX
         BNE L13
         CLC
         LDA PGREND+1           ;adjust end of program ptr
         ADC TEMP
         STA PGREND+1
         LDA TXTPTR+1           ;adjust text pointer
         CMP #2                 ;but only if program is running
         BEQ L14
         CLC
         ADC TEMP
         STA TXTPTR+1
L14      LDY #0
         LDA (LINK),Y           ;reset link field addresses
         TAX                    ;fetch & save ptr to next addr
         INY
         LDA (LINK),Y           ;get MSB of addr
         BEQ L15                ;if it's a zero, branch--we're done
         PHA                    ;save it
         CLC
         ADC TEMP               ;adjust it
         STA (LINK),Y           ;put it back in the program
         STX LINK               ;install ptrs. saved earlier
         PLA
         STA LINK+1
         BNE L14                ;branch always
L15      DEY
         LDA PAGE
         CMP A1+1               ;is move up or down?
         BCS L16                ;if up, branch
         JMP MOVE               ;else use monitor MOVE & exit
L16      LDA STREND+1           ;set dest. ptrs
         STA A4+1
         LDA STREND
         STA A4
L17      LDA (A2),Y             ;fetch a byte off the top
         STA (A4),Y             ;move it
         SEC
         LDA A2                 ;adjust pointers
         SBC #1
         STA A4
         STA A2
         BCS L18
         DEC A4+1
         DEC A2+1
L18      CMP A1                 ;are we done?
         LDA A2+1
         SBC A1+1
         BCS L17                ;if carry set, go back & repeat
         RTS                    ;if carry clear, we're done
* END
```

END OF LISTING 1

LISTING 2: APM.OBJ

```
Start: 8E00          Length: 13A

A9 |8E00:A5 06 85 43 C9 08 90 0B
84 |8E08:E5 68 85 07 18 65 6E C5
B5 |8E10:70 90 01 60 A0 00 84 3C
88 |8E18:84 42 C8 84 08 A5 68 85
DE |8E20:3D 85 09 A5 6D 85 3E A5
DC |8E28:6E 85 3F A5 69 85 CE A4
DC |8E30:6A 84 CF C5 6B A5 CF E5
)2 |8E38:6C B0 36 A0 00 B1 CE 30
AB |8E40:09 C8 B1 CE 10 1E A0 04
C1 |8E48:10 10 C8 B1 CE 30 15 A0
0D |8E50:03 18 B1 CE 65 07 91 CE
A9 |8E58:A0 05 B1 CE C5 70 B0 04
FF |8E60:65 07 91 CE 18 A5 CE 69
AE |8E68:07 85 CE 90 C6 E6 CF B0
43 |8E70:C2 A5 CE C5 6D A5 CF E5
7F |8E78:6E B0 5B A0 02 18 A5 CE
31 |8E80:85 83 71 CE 48 C8 A5 CF
54 |8E88:85 84 71 CE 85 CF 68 85
8E |8E90:CE A0 00 B1 83 30 DA C8
FF |8E98:B1 83 10 D5 A0 04 B1 83
CF |8EA0:0A 90 03 E6 84 18 65 83
F4 |8EA8:90 03 E6 84 18 69 05 85
67 |8EB0:83 90 02 E6 84 A0 02 B1
C1 |8EB8:83 C5 70 B0 04 65 07 91
29 |8EC0:83 18 A5 83 69 03 85 83
FF |8EC8:90 02 E6 84 C5 CE A5 84
BB |8ED0:E5 CF 90 E3 B0 9B A2 08
05 |8ED8:CA 18 B5 67 65 07 95 67
32 |8EE0:CA D0 F5 18 A5 B0 65 07
47 |8EE8:85 B0 A5 B9 C9 02 F0 05
09 |8EF0:18 65 07 85 B9 A0 00 B1
F2 |8EF8:08 AA C8 B1 08 F0 0D 48
69 |8F00:18 65 07 91 08 86 08 68
80 |8F08:85 09 D0 E9 88 A5 06 C5
57 |8F10:3D B0 03 4C 2C FE A5 6E
BA |8F18:85 43 A5 6D 85 42 B1 3E
84 |8F20:91 42 38 A5 3E E9 01 85
86 |8F28:42 85 3E B0 04 C6 43 C6
9B |8F30:3F C5 3C A5 3F E5 3D B0
5A |8F38:E5 60

TOTAL: 8CBC
```

END OF LISTING 2

LISTING 3: APM.DEMO

```
37 | 10  REM  ***********************
C0 | 20  REM  * APM.DEMO            *
B9 | 30  REM  * BY MIKE MIYAKE      *
AE | 40  REM  * COPYRIGHT(C) 1988 *
CB | 50  REM  * MICROSPARC, INC.    *
24 | 60  REM  * CONCORD, MA 01742 *
45 | 70  REM  ***********************
AB | 80  LOMEM:  PEEK (175) + 256 *  PEEK (176) + 31
   |     4
83 | 90  TEXT : HOME : PRINT "PROGRAM MOVER DEMO":
   |     VTAB 5
E3 | 100 D$ =  CHR$ (4): DEF  FN X(X) =  PEEK (X) +
   |     256 *  PEEK (X + 1)
EF | 110  PRINT D$"BLOAD APM.OBJ,A" FN X(175)
47 | 120  PRINT "A 'POKE 6,64' AND A CALL OR BRUN WI
   |     LL": PRINT "MOVE A LONG BASIC PROGRAM TO PA
   |     GE 64:": LIST 130: PRINT "...SO IT CAN USE
   |     HI-RES GRAPHICS": PRINT : PRINT : PRINT "PR
   |     ESS <RETURN> TO DO IT ";: GET ANS: PRINT
46 | 130  POKE 6,64: CALL  FN X(175): REM MOVE TO PG
   |     .64
A0 | 140  HGR : HCOLOR= 3: HPLOT 0,80: FOR I = 19 TO
   |      279 STEP 20: HPLOT  TO I, INT ( RND (1) *
   |     140) + 10: NEXT
DF | 150  VTAB 21: PRINT "APM MOVED THE DEMO TO PAGE
   |      " PEEK (104)", JUST": PRINT "BEFORE HI-RES
   |      GRAPHICS WERE INVOKED": PRINT : PRINT "PRE
   |     SS <RETURN> TO CONTINUE ";: GET ANS: PRINT

A9 | 160  TEXT : HOME
FD | 170  POKE 6,8: CALL  FN X(175): REM  BACK TO NO
   |     RMAL
00 | 180  PRINT "AND APM USED THE SAME TECHNIQUE TO"
   |     : PRINT "RESTORE THE DEMO TO NORMAL--PAGE "
   |     PEEK (104): LIST 170: PRINT "THIS PROGRAM
   |     NOW HAS 14K MORE MEMORY": PRINT "AVAILABLE
   |     IN TEXT MODE THAN IT HAD": PRINT "JUST A MO
   |     MENT AGO!"
2F | 190 : PRINT : PRINT : PRINT "PRESS <RETURN> TO
   |     CONTINUE ";: GET ANS: PRINT
B4 | 200  HOME : PRINT "VARIABLES AND FUNCTIONS DEFI
   |     NED BEFORE": PRINT "THE MOVES, SUCH AS D$ A
   |     ND FN X:": LIST 100: PRINT : PRINT "...REMA
   |     IN INTACT": LIST 210
F2 | 210  PRINT : PRINT "ASC(D$) = " ASC (D$): PRINT
   |      "FN X(103) = " FN X(103)

TOTAL: 6627
```

END OF LISTING 3