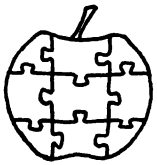

Apple Assembly Line

Volume 1 -- Issue 1 October, 1980

Welcome to the premier issue of the Apple Assembly Line!

This new monthly newsletter is dedicated to the many Apple owners using assembly language, or who would like to learn how. Articles will include commented disassemblies of Apple ROM routines, DOS, and other commercial software; how to augment and modify existing products; beginner's lessons in assembly language; handy subroutines every programmer needs in his tool kit; and many more.

In this issue you will find a tutorial on efficient ways to increment and decrement multiple-byte values, a very powerful subroutine for formatting messages on the screen, and patch code for the S-C ASSEMBLER II Version 4.0 to "adapt" it to the Paymar Lower-Case Adapter. There is also an article describing a recently reported error found in all 6502 chips, and a brief announcement of some new products from S-C SOFTWARE.

Since there will be a lot of source code printed in this and forthcoming issues of the Apple Assembly Line, I plan to offer quarterly diskettes containing all published source code (in the format of the S-C ASSEMBLER II Version 4.0) at a nominal price. How does $15 per quarter sound? Of course, you can always type it in.... The articles should be considered copyrighted, but feel free to use the code in any way you can. It is printed here for your enlightenment, entertainment, and for your USE. I hope you find it all helpful.

I do not know all there is to know about the 6502, or the Apple, or about anything! Nor do I have an infinite amount of time. Therefore, I will be happy to accept articles and programs from you. I may print them exactly as you write them, or I may modify them first. In any case, you will get credit, and the satisfaction of knowing you are helping many others in their conquest of the computer.

If you know others who should be receiving this newsletter, spread the word! If you are not subscribing yet, then send your $12 today! If you have any comments about the content, format, or whatever, write now! Or, you can call me during reasonable hours at (214) 324-2050.

Sincerely,

Bob Sander-Cederlof

Bob Sander-Cederlof

How to Add and Subtract One

I suppose there are as many ways to do it as there are programmers. Some are short and fast, some long and slow, some neat, some not so neat.

Adding one to a number is called "incrementing", and subtracting one is called "decrementing". The 6502 has two instructions for these two functions: INC and DEC. (For the moment I will overlook the four instructions for doing the same to the X and Y registers: INX, INY, DEX, and DEY.) It is easy to see how to use INC and DEC on single-byte values; with a little more trouble they can also be used for values of two or more bytes.

Adding one using ADC

```
   CLC
   LDA VALL   Add 1 to low
   ADC #1       byte
   STA VALL
   LDA VALH   Add carry to
   ADC #0       high byte
   STA VALH
```

Adding one using INC

```
   INC VALL   Add 1 to low byte
   BNE .1     Skip if no carry
   INC VALH   Add carry
.1 ....
```

Of course, there are many other variations. Not indicated here are the various address modes possible in addressing VALL and VALH. They may be in page zero, or elsewhere. They may be directly addressed, as shown above; or, you may use the indirect indexed, indexed indirect, and plain indexed modes.

It is easy to see how to extend both methods above to triple precision, or more. Here is a three-byte version using INC:

```
   INC VALL   Increment low byte
   BNE .1     Unless zero, no carry
   INC VALM   Increment middle byte
   BNE .1     Unless zero, no further carry
   INC VALH   Increment high byte
.1 ....
```

Believe it or not, there is one disadvantage to using INC code like this. Sometimes code is required to have a constant running time, regardless of the data values. Then you either must use straight line code with ADC instructions, or add complicated padding code to the INC form.

How about subtracting one? Here are two ways to do it to two-byte values:

Decrementing with SBC

```
   SEC
   LDA VALL   Low byte - 1
   SBC #1
   STA VALL
   LDA VALH   High byte - 1
   SBC #0
   STA VALH
```

Decrementing with DEC

```
   LDA VALL   Need to borrow?
   BNE .1     No
   DEC VALH   Yes, hi-byte - 1
.1 DEC VALL   low byte - 1
```

Which one do you like better? It is still a matter of taste, unless the amount of memory used or time consumed is very important. There also different side effects, such as the final state of the Carry status. INC and DEC do not change

the Carry status, while of course ADC and SBC do. You may wish to preserve Carry through the process, making the INC/DEC code preferable. Or, you may wish to know the resulting Carry status after incrementing or decrementing for some reason; then you should use the ADC/SBC code.

Back to subtracting one...how about doing it to a three-byte value? We just add three more lines:

```
   LDA VALL  See if need to borrow
   BNE .2    No
   LDA VALM  See if need to borrow again
   BNE .1    No
   DEC VALH  Borrow from high byte
.1 DEC VALM  Borrow from middle byte
.2 DEC VALL  Decrement low byte
```

Easier than you thought, right? You would not believe the many strange ways I have seen this operation coded in commercial software (even some released by Apple themselves!). Yet it seems to me that this method is the same way we would do it with pencil and paper in decimal arithmetic. Think how you would do this:

```
    123040
     -1
    ------
    xxxxxx
```

If you think of each digit being a byte...isn't the algorithm the same?

Now it is time for all of us to go back over the programs we wrote during the past three years for the Apple, and replace a lot of old code!

New Products from S-C SOFTWARE

As many of you know, because you have already bought it, version 4.0 of the S-C ASSEMBLER II is now on the market. With this new version, the price has gone up from $35 to $55. An upgrade kit for owners of previous versions is only $22.50.

Now another new version is available, for those of you without disks! Tape Version 4.0 requires only 16K RAM and a cassette drive. The price is $45 for the complete package, or $22.50 for an upgrade kit from the previous tape version. All of the new features of Disk Versions 3.2 and 4.0 are included, except those which require a disk drive. For the time being, the manual consists of a copy of the disk version 4.0 manuals, with a single sheet describing the differences in the tape version. Purchasers of tape version 4.0 will be able to upgrade to the disk version when they get a disk drive, for only $12.50.

And still another version of the assembler! This one is a cross assembler for the Motorola 6800, 6801, and 6802 mocroprocessors. It has all the features of the S-C ASSEMBLER II Disk Version 4.0, but the source language accepted is that of the 6800 family rather than the 6502. The price for this package is only $300, which is less than a month of time-sharing services for an equivalent capability would cost! An Apple, a ROM blower from Mountain Hardware, and the S-C ASSEMBLER II-6800 are all you need for a full blown development system.

General Message Printing Subroutine

Formatting a series of nice messages or screens-full of messages is hard enough to do in Applesoft...but in assembly language it can really be a difficult job. And it seems to take so much memory to do the equivalent of VTAB, HTAB, HOME, and PRINT. I was recently motivated to do something about this for a large verbose program. I designed a general subroutine for printing text, which can print all 128 character of ASCII, plus do some fancy footwork on the way.

Embedded control codes in the text to be printed perform such handy functions as HTAB, VTAB, HOME, NORMAL, INVERSE, Clear to End of Line, Clear to End of Page, Two-Second Delay, and Repeat. All characters to be printed directly are entered with the high-order bit set to one; bytes with the high order bit zero are control codes. Comments in lines 1250-1350 of the listing show what the codes are.

To simplify the calling sequence, a table of message addresses is built along with the messages themselves. To print a specific message, merely load the message index number into the A-register (LDA #0 for the first message, LDA #1 for the second, etc.), and JSR MESSAGE.PRINTER. Some sample messages are given in the listing, starting at line 2240.

There are a lot of unused control codes, which you can use to augment the subroutine. I am planning to add a code to switch to a HI-RES TEXT driver, for writing text on either of the two Hi-Res screens. You can probably think of a lot of useful ones yourself. The point is that this type of subroutine can simplify programming of an interactive program, and save memory too.

Using the Paymar Lower-Case Adapter with S-C ASSEMBLER II Version 4.0

Bob Matzinger
817-275-2910

Since purchasing the Paymar adapter, I have spent a lot of time adapting software to effectively use it! The program given here will adapt the version 4.0 of Bob Sander-Cederlof's assembler to allow lower-case comments.

The two patches at lines 1340 and 1390 have to be entered, and the body of the patch loaded at $300. Once installed, typing a control-A will toggle the shift-lock; control-S will perform a single-character upper-case shift; control-K, -L, and -O give access to the characters normally missing from the Apple keyboard.

Only comments can be entered in lower-case. Further modification to the assembler would be required to allow commands, labels, and opcodes to be entered in lower- or mixed-case.

```
                1000 *--------------------------------
0024-           1010 MON.CH      .EQ $24
0025-           1020 MON.CV      .EQ $25
FC22-           1030 MON.VTAB    .EQ $FC22
FC42-           1040 MON.CLREOP  .EQ $FC42
FC58-           1050 MON.HOME    .EQ $FC58
FC9C-           1060 MON.CLREOL  .EQ $FC9C
FCA8-           1070 MON.WAIT    .EQ $FCA8
FDED-           1080 MON.COUT    .EQ $FDED
FE84-           1090 MON.NORMAL  .EQ $FE84
FE80-           1100 MON.INVERSE .EQ $FE80
                1110 *--------------------------------
0018-           1120 MSG.PNTR    .EQ $18,19
001A-           1130 MSG.SCANNER .EQ $1A
                1140 *--------------------------------
                1150 *       MESSAGE PRINTER
                1160 *
                1170 *    CALL:
                1180 *       (A) = MESSAGE #   (0-N)
                1190 *       JSR MESSAGE.PRINTER
                1200 *
                1210 *    ACTION:
                1220 *       1.  FINDS SPECIFIED MESSAGE
                1230 *       2.  PRINTS ON THE SCREEN
                1240 *       3.  INTERPRETS CHARACTERS AS FOLLOWS:
                1250 *           $00      END OF MESSAGE
                1260 *           $01-28   HTAB 1-40
                1270 *           $40-57   VTAB 1-24
                1280 *           $60      CLEAR SCREEN, HOME CURSOR
                1290 *           $61XXYY REPEAT CHARACTER YY, XX TIMES
                1300 *           $62      DELAY ABOUT TWO SECONDS
                1310 *           $63      NORMAL MODE
                1320 *           $64      INVERSE MODE
                1330 *           $65      CLEAR TO END OF LINE
                1340 *           $66      CLEAR TO END OF SCREEN
                1350 *           $80-FF   PRINT AS IS
                1360 *
                1370 *--------------------------------
                1380 MESSAGE.PRINTER
0800- 0A        1390         ASL            DOUBLE MSG NUMBER TO GET INDEX
0801- A8        1400         TAY
0802- B9 83 08  1410         LDA MESSAGE.ADDRESS.TABLE,Y
0805- 85 18     1420         STA MSG.PNTR
0807- B9 84 08  1430         LDA MESSAGE.ADDRESS.TABLE+1,Y
080A- 85 19     1440         STA MSG.PNTR+1
080C- A9 00     1450         LDA #0
080E- 85 1A     1460         STA MSG.SCANNER
0810- 20 76 08  1470 .1      JSR GET.NEXT.CHAR.FROM.MESSAGE
0813- D0 01     1480         BNE .3
0815- 60        1490         RTS            $00:  EOM
0816- 10 06     1500 .3      BPL .5         SPECIAL ACTION
0818- 20 ED FD  1510         JSR MON.COUT PRINT THE CHARACTER
081B- 4C 10 08  1520 .4      JMP .1
                1530 *--------------------------------
081E- C9 40     1540 .5      CMP #$40       CHECK FOR VTAB
0820- B0 0A     1550         BCS .6         YES
0822- C9 29     1560         CMP #$29       IN RANGE FOR HTAB?
```

```
0824- B0 F5     1570         BCS .4        NO, IGNORE
0826- 85 24     1580         STA MON.CH
0828- C6 24     1590         DEC MON.CH
082A- 90 EF     1600         BCC .4        ...ALWAYS
                1610 *-------------------------------
082C- C9 58     1620 .6      CMP #$58      IN RANGE FOR VTAB?
082E- B0 0A     1630         BCS .7        NO
0830- 29 1F     1640         AND #$1F      MASK VALUE
0832- 85 25     1650         STA MON.CV    YES
0834- 20 22 FC  1660         JSR MON.VTAB
0837- 4C 1B 08  1670         JMP .4
                1680 *-------------------------------
083A- 49 60     1690 .7      EOR #$60      CHECK FOR TOKENS
083C- C9 07     1700         CMP #7        $60 THROUGH $66
083E- B0 DB     1710         BCS .4        NOT TOKEN, SO IGNORE
0840- 0A        1720         ASL           MAKE DUBLE INDEX
0841- AA        1730         TAX
0842- A9 08     1740         LDA /.4-1     PUT RETURN ON STACK
0844- 48        1750         PHA           TO SIMULATE A JSR ADDR,X
0845- A9 1A     1760         LDA #.4-1
0847- 48        1770         PHA
0848- BD 52 08  1780         LDA MSGTKNTBL+1,X
084B- 48        1790         PHA
084C- BD 51 08  1800         LDA MSGTKNTBL,X
084F- 48        1810         PHA
0850- 60        1820         RTS
                1830 *-------------------------------
                1840 MSGTKNTBL
0851- 57 FC     1850         .DA MON.HOME-1
0853- 5E 08     1860         .DA MSG.REPEAT-1
0855- 6C 08     1870         .DA LONG.DELAY-1
0857- 83 FE     1880         .DA MON.NORMAL-1
0859- 7F FE     1890         .DA MON.INVERSE-1
085B- 9B FC     1900         .DA MON.CLREOL-1
085D- 41 FC     1910         .DA MON.CLREOP-1
                1920 *-------------------------------
                1930 MSG.REPEAT
085F- 20 76 08  1940         JSR GET.NEXT.CHAR.FROM.MESSAGE
0862- AA        1950         TAX           NUMBER OF MULTIPLES
0863- 20 76 08  1960         JSR GET.NEXT.CHAR.FROM.MESSAGE
0866- 20 ED FD  1970 .1      JSR MON.COUT
0869- CA        1980         DEX
086A- D0 FA     1990         BNE .1
086C- 60        2000         RTS
                2010 *-------------------------------
                2020 LONG.DELAY
086D- A0 0C     2030         LDY #12
086F- 20 A8 FC  2040 .1      JSR MON.WAIT DELAY 167309 CYCLES
0872- 88        2050         DEY
0873- D0 FA     2060         BNE .1
0875- 60        2070         RTS
                2080 *-------------------------------
                2090 GET.NEXT.CHAR.FROM.MESSAGE
0876- A4 1A     2100         LDY MSG.SCANNER
0878- B1 18     2110         LDA (MSG.PNTR),Y
087A- E6 1A     2120         INC MSG.SCANNER
087C- D0 02     2130         BNE .1
087E- E6 19     2140         INC MSG.PNTR+1
0880- C9 00     2150 .1      CMP #0
0882- 60        2160         RTS
```

```
                        2170 *--------------------------------
                        2180 MESSAGE.ADDRESS.TABLE
0883- 8B 08             2190         .DA MSG0
0885- FF 08             2200         .DA MSG1
0887- 10 09             2210         .DA MSG2
0889- 3B 09             2220         .DA MSG3
                        2230 *--------------------------------
088B- 60                2240 MSG0    .HS 60        HOME SCREEN
                        2250 * CELL 1 -- VOCABULARY CHECK
088C- 64                2260         .HS 64        INVERSE MODE
088D- 61 29 AD          2270         .HS 6129AD    4A DASHES
0890- 28 AD AD          2280         .HS 28ADAD    2 DASHES
0893- 28 AD AD          2290         .HS 28ADAD
0896- 28 AD AD          2300         .HS 28ADAD    2 DASHES
0899- 28 AD AD          2310         .HS 28ADAD    2 DASHES
089C- 28 AD AD          2320         .HS 28ADAD    2 DASHES
089F- 28 AD AD          2330         .HS 28ADAD    2 DASHES
08A2- 28 61 29
08A5- AD                2340         .HS 286129AD 41 DASHES
08A6- 63                2350         .HS 63        NORMAL MODE
08A7- 42 05             2360         .HS 4205      VTAB 3, HTAB 5
08A9- C4 C5 CD
08AC- CF CE D3
08AF- D4 D2 C1
08B2- D4 C9 CF
08B5- CE A0 CF
08B8- C6 A0 CD
08BB- C5 D3 D3
08BE- C1 C7 C5
08C1- A0 D0 D2
08C4- C9 CE D4
08C7- C5 D2             2370         .AS -/DEMONSTRATION OF MESSAGE PRINTER/
08C9- 44 0F             2380         .HS 440F      VTAB 5, HTAB 15
08CB- D3 AD C3
08CE- A0 D3 CF
08D1- C6 D4 D7
08D4- C1 D2 C5          2390         .AS -/S-C SOFTWARE/
08D7- 45 0E             2400         .HS 450E      VTAB 6, HTAB 14
08D9- D0 AE A0
08DC- CF AE A0
08DF- C2 CF D8
08E2- A0 B5 B5
08E5- B3 B7             2410         .AS -/P. O. BOX 5537/
08E7- 46 0B             2420         .HS 460B      VTAB 7, HTAB 11
08E9- D2 C9 C3
08EC- C8 C1 D2
08EF- C4 D3 CF
08F2- CE AC A0
08F5- D4 D8 A0
08F8- B7 B5 B0
08FB- B8 B0             2430         .AS -/RICHARDSON, TX 75080/
08FD- 4A                2440         .HS 4A        VTAB 11
08FE- 00                2450         .HS 00
                        2460 *--------------------------------
08FF- 49 01 66          2470 MSG1    .HS 490166    VTAB 10, HTAB 1, CLR EOP
0902- D3 C5 CC
0905- C5 C3 D4
0908- A0 CF CE
090B- C5 BA A0
090E- A0                2480         .AS -/SELECT ONE: /
```

```
090F- 00        2490         .HS 00
                2500 *-------------------------------
0910- 57 01 65  2510 MSG2    .HS 570165    VTAB 24, HTAB 1, CLR EOL
0913- 64        2520         .HS 64        INVERSE MODE
0914- A0 BC D3
0917- D0 C1 C3
091A- C5 BE A0
091D- C6 CF D2
0920- A0 CD C5
0923- CE D5 AC
0926- A0 BC D2
0929- C5 D4 D5
092C- D2 CE BE
092F- A0 C6 CF
0932- D2 A0 CD
0935- CF D2 C5
0938- A0        2530         .AS -/ <SPACE> FOR MENU, <RETURN> FOR MORE
0939- 63 00     2540         .HS 6300      NORMAL MODE, EOM
                2550 *-------------------------------
093B- 87 87 8D  2560 MSG3    .HS 87878D
093E- AA AA AA
0941- D3 D9 CE
0944- D4 C1 D8
0947- A0 C5 D2
094A- D2 CF D2  2570         .AS -/***SYNTAX ERROR/
094D- 8D 00     2580         .HS 8D00
```

SYMBOL TABLE

```
0876- GET.NEXT.CHAR.FROM.MESSAGE
.01=0880
086D- LONG.DELAY
.01=086F
0883- MESSAGE.ADDRESS.TABLE
0800- MESSAGE.PRINTER
.01=0810, .03=0816, .04=081B, .05=081E
.06=082C, .07=083A
0024- MON.CH
FC9C- MON.CLREOL
FC42- MON.CLREOP
FDED- MON.COUT
0025- MON.CV
FC58- MON.HOME
FE80- MON.INVERSE
FE84- MON.NORMAL
FC22- MON.VTAB
FCA8- MON.WAIT
0018- MSG.PNTR
085F- MSG.REPEAT
.01=0866
001A- MSG.SCANNER
088B- MSG0
08FF- MSG1
0910- MSG2
093B- MSG3
0851- MSGTKNTBL
```

```
                    1000 * Lower case conversion for
                    1010 * ASMDISK 4.0 (<c> S-C SOFTWARE)
                    1020 * Complete with 126 ASCII Characters
                    1030 *-------------------------------
                    1040 * The CTRL-A and CTRL-S keys are used similar to
                    1050 * shift and lock keys on a standard typewriter.
                    1060 * CTRL-S will enter one UPPER CASE character the
                    1070 * return to lower case mode.
                    1080 * CTRL-A is a "permanent" shift lock.  Each time
                    1090 * CTRL-A is pressed the case mode will change to
                    1100 * either upper or lower case until it or CTRL-S
                    1110 * is pressed.  CTRL-S will give you one more
                    1120 * upper case letter.
                    1130 *-------------------------------
                    1140 * REMEMBER!!!
                    1150 * ALL COMMANDS AND MNEMONIC ENTRIES MUST BE
                    1160 * IN UPPER CASE.  Use lower case only for
                    1170 *------------------------------- comments!!!
0081-               1180 CTRLA   .EQ $81       SHIFT LOCK
008B-               1190 CTRLK   .EQ $8B       [ or (
008C-               1200 CTRLL   .EQ $8C       \ or |
008F-               1210 CTRLO   .EQ $8F       _ or RUB
0093-               1220 CTRLS   .EQ $93       SHIFT
FD0C-               1230 RDKEY   .EQ $FD0C
                    1240 *-------------------------------
                    1250 * Remember:
                    1260 * Shift M yields ] or }
                    1270 * Shift N yields ↑ or ^
                    1280 * Shift P yields @ or `
                    1290 *-------------------------------
                    1300 *-------------------------------
                    1310         .OR $1380
                    1320         .TA $0300
                    1330 *-------------------------------
1380- 20 00 03      1340 PATCH1 JSR LC
                    1350 *-------------------------------
                    1360         .OR $139A
                    1370         .TA $0300
                    1380 *-------------------------------
139A- 29 FF         1390 PATCH2 AND #$FF
                    1400 *-------------------------------
                    1410         .OR $0300
                    1420 * CAUTION: Do not assemble your programs into
                    1430 * $0300 up.  You will destroy this routine!!!
                    1440 *-------------------------------
0300- 20 0C FD      1450 LC      JSR RDKEY
0303- C9 81         1460         CMP #CTRLA
0305- F0 10         1470         BEQ LOCK
0307- C9 93         1480         CMP #CTRLS
0309- D0 1D         1490         BNE CHECK
030B- A9 00         1500 SHIFT   LDA #$00
030D- 8D 4A 03      1510         STA LCKFLG
0310- A9 00         1520 SHIFT1 LDA #$00
0312- 8D 4B 03      1530         STA CASE
0315- F0 E9         1540         BEQ LC
0317- AD 4A 03      1550 LOCK    LDA LCKFLG
```

```
031A- 49 01     1560         EOR #01
031C- 8D 4A 03  1570         STA LCKFLG
031F- D0 EF     1580         BNE SHIFT1
0321- A9 20     1590         LDA #$20
0323- 8D 4B 03  1600         STA CASE
0326- D0 D8     1610         BNE LC
0328- C9 8B     1620 CHECK   CMP #CTRLK
032A- F0 08     1630         BEQ SPEC
032C- C9 8C     1640         CMP #CTRLL
032E- F0 04     1650         BEQ SPEC
0330- C9 8F     1660         CMP #CTRLO
0332- D0 02     1670         BNE CONV
0334- 09 50     1680 SPEC    ORA #$50
0336- C9 C0     1690 CONV    CMP #$C0
0338- 90 03     1700         BCC RETURN
033A- 0D 4B 03  1710         ORA CASE
033D- 48        1720 RETURN  PHA
033E- AD 4A 03  1730         LDA LCKFLG
0341- D0 05     1740         BNE OUT
0343- A9 20     1750         LDA #$20
0345- 8D 4B 03  1760         STA CASE
0348- 68        1770 OUT     PLA
0349- 60        1780         RTS
034A- 00        1790 LCKFLG  BRK
034B- 20        1800 CASE    .DA #$20
                1810 *--------------------------------
                1820 * Written by Bob Matzinger
                1830 * September 6, 1980
                1840 *--------------------------------
```

Hardware Error in ALL 6502 Chips!

INTERFACE, the newsletter of Rockwell International (P. O. Box 3669, RC 55, Anaheim, CA 92803), Issue No. 2, is the source for the following information. It should be noted by all Apple owners working in assembly language, because it could cause an almost unfindable bug!

There is an error in the JUMP INDIRECT instruction of ALL 6500 family CPU chips, no matter where they were made. This means the error is present in ALL APPLES. This fatal error occurs only when the low byte of the indirect pointer location happens to be $FF, as in JMP ($08FF). Normally, the processor should fetch the low-order address byte from location $08FF, increment the program counter to $0900, and then fetch the high-order address byte from $0900. Instead, the high-order byte of the program counter never gets incremented! The high-order address byte gets loaded from $0800 instead of $0900! For this reason, your program should NEVER include an instruction of the type JMP ($xxFF).

Try this example to satisfy yourself that you understand the problem: insert the following data from the monitor.

```
*800:09
*810:6C FF 08   (this is JMP ($08FF).)
*8FF:50 0A      (pointer)
*A50:00         (BRK instruction we SHOULD reach)
*950:00         (BRK instruction we DO reach!)
```

Execute the instruction at $0810 by typing 810G. If the JMP indirect worked correctly, it would branch to location $0A50 and execute the BRK instruction there. However, since the JMP indirect instruction has this serious flaw, it will actually branch to the BRK instruction at $0950!

Since it is very difficult to predict the final address of all pointers in a large assembly language program, unless they are all grouped in a block at the beginning of the program, I suggest that you take special measures to protect yourself against this hardware problem. (One measure, of course, was suggested in that sentence.) My favorite method is to avoid using the JMP indirect instruction. It takes too long to set it up in most cases anyway. I prefer to push the branch address (less one) onto the stack, and RTS to effect the branch. This allows me to create the effect of an indexed JMP. For example, suppose a command character is being decoded. I process it into a value in the A-register between 0 and N-1 (for N commands), and do the following:

```
ASL                 Double to create index
TAX                       for address table
LDA JUMP.TABLE+1,X  High order byte of branch address
PHA
LDA JUMP.TABLE,X    Low order byte
PHA
RTS
```

The jump table looks like this:

```
JUMP.TABLE
        .DA COMMANDA-1      The "-1" is
        .DA COMMANDB-1      on each line
        .DA COMMANDC-1      because the RTS
        .DA COMMANDD-1      adds one before
        et cetera           branching.
```

This trick was described by Steve Wozniak in an article in BYTE magazine back in 1977 or 1978. It is also used by him in the Apple monitor code, and in SWEET-16. In both of these cases, he has arranged all the command processors to be in the same page, so that the high order byte of the address can be loaded into the A-register with a load-A-immediate, and the jump table can be only one-byte-per-command. See your Apple ROMs at locations $FFBE-$FFCB (jump table at $FFE3-$FFF9) and in SWEET-16 at $F69E, F6A0, F684-F6B8 (jump table at $F6E3-$F702).

You can extend this idea of an indexed JMP instruction into a simulated indexed JSR instruction. All you have to do is first push onto the stack the return address (less one), and then the branch address (less one). I use this trick in the Message.Printer program described else where in this issue.

Apple Assembly Line
P. O. Box 5537
Richardson, TX 75080

FIRST CLASS MAIL

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 5537, Richardson, TX 75080. Subscription rate is $12/year, in the U.S.A., Canada, and Mexico. Other countries add $6/year for extra postage. All material herein is copyrighted by S-C SOFTWARE, all rights reserved. Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)