$1.20

Apple Assembly Line

Volume 1 -- Issue 10 August, 1981

In This Issue...

Renewing Subscriptions

The 4-digit number in the upper right corner of your mailing label is the expiration date of your subscription. The first two digits are the year, and the last two digits are the month of the last issue you have paid for. If your label says "8109" or "8110", now is the time to renew to be sure of uninterrupted service.

We now have about 500 subscribers, and are shooting for 1000 by the end of the year. (Look for my full page ad in the next NIBBLE.) I am printing 1000 copies of each issue so there will be plenty of back issues for latecomers.

Notice that I have a new address. The old one will still work for a while, but you should start using the new one: Bob Sander-Cederlof, S-C Software, P. O. Box 280300, Dallas, TX 75228.

Things For Sale

Here is an up-to-date list of some of the things which I have that you might need:

```
Quarterly Disk #1 (source code from Oct 80 - Dec 80)...$15.00
Quarterly Disk #2 (source code from Jan 81 - Mar 81)...$15.00
Quarterly Disk #3 (source code from Apr 81 - Jun 81)...$15.00
S-C ASSEMBLER II Version 4.0...........................$55.00
Beneath Apple DOS (book)...............................$18.00
Apple Machine Language (book)..........................$11.65
Blank Diskettes (Verbatim, with hub rings, no labels,
                 plain white jackets, in cellophane
                 wrapper)................20 disks for $50.00
Zip-Lock Bags (2-mil, 6"x9")...............100 bags for $8.50
```

If you are interested in getting a regular monthly shipment of 100 or more disks, we can work out an even lower price.

If you are in Texas, remember to send 5% sales tax on books, disks, or bags.

Finding Applesoft Line Numbers.......................Bob Potts

Sometimes I have needed to know where in memory a certain Applesoft line is located. Maybe I want to patch in a code which cannot be typed from the keyboard. Or maybe the program has been "compressed and optimized", so that the lines are too long to edit. Or maybe I am just curious.

It is simple enough, because the line number is stored in binary at the beginning of each line. I would looke at locations $67,68 to get the address of the first line. Then look at that location to get the address of the next line, and so on. Each line is stored in memory with the first two bytes telling where to find the next line. and the third and fourth bytes giving the line number. Of course, the line number is in binary, and the bytes are backward, and the whole screen is full of hex numbers making it very hard to keep everything straight....

There has to be an easier way! Working with Bob Sander-Cederlof last week, I came up with this simple little program which will print the address of any line in hex. It uses the ampersand (&) statement of Applesoft. You simply BRUN this program, which I call AMPERFIND, and then type an ampersand and the line number. BRUNning sets up the ampersand vector at $3F5-3F7 and returns.

Here is the program. Note that it takes more code to set up the ampersand vector than it takes to do the line number search! Lines 1210-1260 could be put anywhere in memory, just so $3F6 and $3F7 are made to point to that place.

[Bob Potts is an Assistant Vice President at the Bank of Louisville in Kentucky. this bank has 115 Apple IIs in use doing a variety of banking functions.]

```
                  1000 *-------------------------------
                  1010 *      FIND AN APPLESOFT LINE NUMBER
                  1020 *      AND PRINT ADDRESS IN HEX
                  1030 *-------------------------------
                  1040        .OR $300
                  1050        .TF AMPERFIND
                  1060 *-------------------------------
F941-             1070 MON.PRNTAX .EQ $F941    PRINT TWO BYTES IN HEX
DA0C-             1080 AS.LINGET  .EQ $DA0C    CONVERT LINE NUMBER TO BINARY
D61A-             1090 AS.FNDLIN  .EQ $D61A    FIND LINE IN APPLESOFT PROGRAM
                  1100 *-------------------------------
                  1110 *      SET UP AMPERSAND VECTOR
                  1120 *-------------------------------
0300- A9 4C       1130        LDA #$4C      "JMP" OPCODE
0302- 8D F5 03    1140        STA $3F5
0305- A9 10       1150        LDA #AMPERFIND
0307- 8D F6 03    1160        STA $3F6
030A- A9 03       1170        LDA /AMPERFIND
030C- 8D F7 03    1180        STA $3F7
030F- 60          1190        RTS
                  1200 *-------------------------------
                  1210 AMPERFIND
0310- 20 0C DA    1220        JSR AS.LINGET  CONVERT LINE NUMBER TO BINARY
0313- 20 1A D6    1230        JSR AS.FNDLIN  FIND THE LINE
0316- A6 9B       1240        LDX $9B
0318- A5 9C       1250        LDA $9C        GET THE LINE'S ADDRESS
031A- 4C 41 F9    1260        JMP MON.PRNTAX PRINT THE ADDRESS IN HEX
```

Binary Keyboard Input

David Holladay, from Madison, Wisconsin, wrote a recent article for the Adam & Eve Apple II Users Group about a technique he uses for turning the Apple keyboard into a Braille input device. He chose 6 keys which can be "simultaneously" depressed to give a composite code. The keys form a 2-by-3 rectangle, like the dots of Braille characters.

Because the Apple keyboard has N-key rollover, simultaneous depression of several keys results in each keycode being sent to the program one at a time. The order that the codes are produced appears random to the program. Some quirks in the way the Apple keyboard is wired up prevent the N-key rollover from working with every combination of keys. Some of them OR together to create a ghost code, different from the actual depressed keys. Apple has used many different keyboards, so the keys which can be used for David's program vary considerably from one Apple to another.

After playing around with his program for a while, I got interested in making a Binary Input Keyboard, rather than a Braille one. My keyboard, which is almost 4 years old (Apple serial # 2191), allows me to press any combination of the keys J, K, L, 1, 2, 3, and 4. I set up these keys with binary weights of hex 40, 20, 10, 08, 04, 02, and 01 respectively.

When you type a combination of these seven keys all at once, the time interval between keys is much shorter than the normal spacing between keystrokes. The program waits for one keyboard strobe, and then initiates a timeout loop. All keycodes received within the timeout window will be considered to have been struck "simultaneously". Each keycode is compared with the list of seven keys (JKL1234), and the appropriate binary weight ORed into the character. If a keycode is received which is not in the legal character list, the bell rings.

I made a test loop which calls the input routine, and displays the hex code on the screen.

The choice of keys (JKL1234) works fine on my Apple, but it may not work on yours. Experiment with various choices until you find seven keys which will work together on your keyboard. Then modify line 1420 with your list of keys, and it will be ready to go.

Possible applications? Maybe fast input of hexadecimal machine language programs. You would have to add one more key so that all eight bits could be specified. And you would have to train your mind and fingers to instantaneously translate from hex to binary finger-patterns. Or, maybe some sort of a game. The basic idea of reading simultaneous keystrokes could effectively create new keys. Or, maybe the basic idea of simultaneous keystrokes could be used for entering secret passwords.

```
                  *--------------------------------
                  *      BINARY KEYBOARD
                  *--------------------------------
0024-             MON.CH   .EQ $24
0025-             MON.CV   .EQ $25
C000-             KEYBOARD .EQ $C000
C010-             STROBE   .EQ $C010
FC24-             MON.VTAB .EQ $FC24
FC58-             MON.HOME .EQ $FC58
FBE2-             MON.BELL .EQ $FBE2
FDDA-             MON.PRBYTE .EQ $FDDA
                  *--------------------------------
0800- A9 00       GETCHR LDA #0
0802- 8D 51 08    .1     STA CHARCODE
0805- A9 F0              LDA #-16
0807- 8D 52 08           STA CNTR
080A- 8D 53 08           STA CNTR+1
080D- AD 00 C0    .2     LDA KEYBOARD
0810- 30 10              BMI .4       SOMETHING TYPED
0812- EE 52 08           INC CNTR
0815- D0 F6              BNE .2
0817- EE 53 08           INC CNTR+1
081A- D0 F1              BNE .2
081C- AD 51 08           LDA CHARCODE GET COMPOSITE CODE
081F- F0 DF              BEQ GETCHR   NO KEYS HIT YET
0821- 60          .3     RTS
                  *--------------------------------
0822- 8D 10 C0    .4     STA STROBE   CLEAR KEYBOARD STROBE
0825- 29 7F              AND #$7F
0827- C9 20              CMP #$20     HANDLE BLANK SEPARATELY
0829- F0 F6              BEQ .3
082B- A0 06              LDY #6       SEARCH LIST OF LEGAL KEYS
082D- D9 43 08    .5     CMP LEGAL.KEYS,Y
0830- F0 09              BEQ .6
0832- 88                 DEY
0833- 10 F8              BPL .5
0835- 20 E2 FB           JSR MON.BELL
0838- 4C 00 08           JMP GETCHR
083B- B9 4A 08    .6     LDA KEY.BITS,Y
083E- 0D 51 08           ORA CHARCODE
0841- D0 BF              BNE .1       ...ALWAYS
                  *--------------------------------
0843- 4A 4B 4C
0846- 31 32 33
0849- 34          LEGAL.KEYS .AS /JKL1234/
084A- 40 20 10
084D- 08 04 02
0850- 01          KEY.BITS   .HS 40201008040201
                  *--------------------------------
0851-             CHARCODE   .BS 1
0852-             CNTR       .BS 2
                  *--------------------------------
                  *      TEST BINARY KEYBOARD
                  *--------------------------------
0854- 20 58 FC    TEST   JSR MON.HOME
0857- 20 00 08    .1     JSR GETCHR
085A- 8D 03 04           STA $403     LINE 1, COLUMN 4 OF SCREEN
085D- A9 00              LDA #0
085F- 85 24              STA MON.CH
0861- 85 25              STA MON.CV
0863- 20 24 FC           JSR MON.VTAB
0866- AD 03 04           LDA $403
0869- 20 DA FD           JSR MON.PRBYTE
086C- 4C 57 08           JMP .1
```

Apple Machine Language -- A Review

Many of you have asked me, "What book will help me, an absolute beginner, learn 6502 machine language? I don't know what these other books are talking about!"

If these are your words, then the book "Apple Machine Language", by Don and Kurt Inman, is for you. It is published by Reston Publishing Company, in both hardback ($17.95) and paperback ($12.95). The book has 296 pages, is set in clear, easy-to-read type, and has lots of good diagrams and illustrations.

The authors assume that you are at least familiar with Applesoft Basic. Chapter 1 gives a brief review of Applesoft, with special emphasis on the PEEK, POKE, and CALL statements. (These are the statements you will be using to communicate between Basic and machine language programs.) The authors also assume that you have your own Apple, and that you will not just READ the book. They expect you to follow along every example with your own Apple, so you can EXPERIENCE the material. You will not only learn a lot faster, but it will stick with you and you will UNDERSTAND what is going on.

Chapter 2 takes you across the bridge from Basic to machine language, very gently. You develop, with the authors, a little Applesoft program which helps you enter and test machine language programs.

Chapter 3 finally introduces the ideas of binary numbers, hexadecimal, the A-register in the 6502, and a few instruction codes. You will learn how to load a value into the A-register, modify that value, and store the result back into memory.

There are exercises at the end of each chapter which review the material covered. Don't let that worry you, though...they also printed the answers!

Chapter 4 starts to get interesting and useful. You learn how to use machine language to put some simple color graphics on the Apple screen. You can plot individual points, draw rectangles, and color them in. All the while, you are learning more machine instructions, more registers, more about memory addressing, and so forth.

Chapter 5 introduces you to writing text on the screen. You learn how to call some of the monitor subroutines for text output, how to print characters at particular screen locations, and how to write messages of your choice. Some new instructions are covered, and you learn some new address modes. In particular, you learn all about relative branching.

Chapter 6 is one of my favorites. I have always enjoyed twiddling Apple's little built-in speaker, and this chapter shows you how. You build and play with a tone generator program, even to the point of tuning it up to make a simulated piano keyboard.

Chapter 7 takes you deeper into sound and graphics, helping you code a routine to display the notes as you play them from the keyboard. By the time you finish this chapter you will understand how to use 28 of the 6502's 56 instructions, and 8 of its 13 addressing modes. You will also have used 9 of the subroutines found inside the Apple Monitor ROM.

Chapter 8 takes you inside Apple's Monitor...just a little. Until now, you have been using the Applesoft program developed in chapter 2 to enter and test all your machine language programs. In chapter 8 you learn how to do it from the monitor. You will also learn how to do addition and subtraction.

Chapter 9 show you how to add numbers too big to fit in one byte. Since one byte will only hold numbers between 0 and 255, or between -128 and +127, you can see that most numbers ARE too big to fit in one byte. You will also learn all about the way negative numbers are handled in the 6502.

Chapter 10 delves deeper into the Apple Monitor, and explores 6502 decimal mode arithmetic.

Chapter 11 is only for those fortunate readers who have Integer BASIC in their Apples. It doesn't matter whether Integer BASIC is on the Apple Monitor board, on a firmware card in ROM, or in a 16K RAM card...just so you have it. Why? Because there is another program in there you might not even be aware of: the Apple Mini-Assembler. If you are lucky enough to have it, chapter 11 will tell you how to use it. If not, skipover this chapter and use your S-C ASSEMBLER II instead! On second thought, don't skip chapter 11 entirely. It is here that indirect addressing is covered, and you need to know this material.

Chapter 12, "Putting It All Together", puts it all together. The programming experience you work through is a multiplication subroutine.

There are four appendices which summarize the information about the Apple hardware found throughout the book. Several of the charts in Appendix-A list page number references. (Early editions of the book had blank columns where the page numbers were supposed to be, but that has been corrected.) And finally, there is a regular alphabetic index.

By the time you finish this book, you have a solid foundation for learning to use an assembler like the S-C ASSEMBLER II. I would like to think that my assembler is easy enough to learn that books like this one would not be needed, but there are a lot of concepts that are completely foreign to new computer owners.

I want to do all I can to help every one of you become proficient in assembly language, so I am making "Apple Machine Language" available to you at a discount. You can buy the $12.95 paperback edition from me for $11.65 (plus 58 cents tax if you are in Texas). Include a dollar for shipping, so I don't go broke.

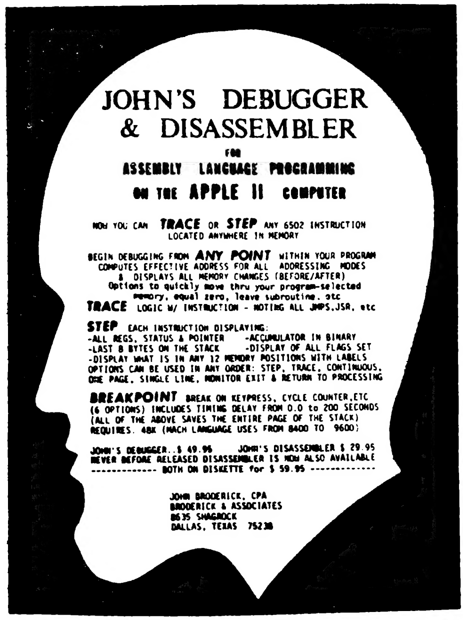
```
* ------------------------------- *
           JOHN'S BOOT
   FOR THE S-C ASSEMBLER 4.0
* ------------------------------- *
```

BY
JOHN BRODERICK, CPA

I am working of an assembly language account system having more than 20 SOURCE CODE PROGRAMS (each one 500-700 lines of code).

The trick is to be able to boot any one of the 20 disks without disturbing hex memory from 2500 to 9600. I do this with my boot program which front-ends the S-C Assembler 4.0. It loads the SOURCE CODE into the LANGUAGE CARD.

This method also allows me to overide the memory protect error when modifing DOS, since HIMEM is set to F800, instead of 9600. It also sets up 1000 to 1FFF as a workarea since I move the assembler into bank #2 of the language card. Here is how it works.



YOU TURN THE COMPUTER POWER ON & IT AUTOMATICALLY:

1. Moves ROM from F800-FFFF into lang card F800-FFFF.
   and sets it so you can now press reset with the card open.
2. loads S-C Assembler into memory (a normal load)
3. Sets HIMEM: F800.
4. Loads your SOURCE PROGRAM into lang card (D000-F800).
5. Lists the first 10 lines of your SOURCE PROGRAM.

THAT'S JUST FOR STARTERS -- NOW LOOK:

You press ASM to assemble , and then USR to execute your code:

1. Sets up a CTRL-Y return.
2. Moves S-C Assembler from 1000-1FFF into lang card - bank #2.

3. Zeros 1000-1FFF so you can use as a workarea.
4. Locks lang card to protect SOURCE CODE during execution.
5. Does a jmp ($FFE) jumps to first instruction to execute.

Everything is now up in the lang card with the exception of S-C Assembler code 2000-24FF and John's Boot at F00. All other memory from 800 to 9600 is usable by your programs.

That's 34,816 bytes of code you can use. Booting another disk leaves hex memory $2500-9600 completely undisturbed because boot will load the SOURCE CODE into the LANGUAGE CARD.

JOHN'S BOOT FOR THE S-C ASSEMBLER 4.0 is FREE when you purchase JOHN'S DEBUGGER AND DISASSEMBLER, otherwise it is $24.95. My address is shown inside the head above. That is not my head--I still have some hair left.

Two Ways to Compare a Byte.........................Lee Meador

I have noticed two ways to compare a byte used inside DOS and other Apple software. In the cases I am thinking of, the following code required the Y-register to be zero. The first way I have seen is straightforward:

```
     LDA ...        BYTE TO BE TESTED
     CMP #$19       VALUE WE WANT TO TEST FOR
     BNE .1         ALSO AFFECTS CARRY STATUS
     LDY #0         IF =, CARRY SET
     ...
```

The other way is a little trickier, but it saves one byte:

```
     LDA ...        BYTE TO BE TESTED
     EOR #$19       VALUE WE WANT TO TEST FOR
     BNE .1         DOESN'T AFFECT CARRY STATUS
     TAY            A AND Y BOTH ZERO
     ...
```

This may help you understand some of those disassemblies you are making, or help you save a byte here and there.

A Selective Catalog from FID........................Lee Meador

If you have DOS 3.3, you have no doubt enjoyed using the FID program to copy files from one disk to another. The wildcard feature in filenames is especially nice, because it lets you set up a semi-automatic copy of a whole set of files, or even the whole disk.

Sometimes I am reluctant to let the wildcard name go through without prompting, because there might be a file or two I don't want copied which matches the specified name. However, there are so many files involved that I really don't want to sit there and type "Y" for every one of them. What we need is a "selective catalog" command -- a FID command to list all files names which match the wildcarded-name.

Here are some easy patches which you can apply to FID which will convert the VERIFY command to just what we want.

```
]BLOAD FID                       load FID
]CALL -151                       get to Apple's monitor
*DBE:60                          return before verifying
*C10:EA EA EA                    no double spacing
*3D0G                            return to BASIC
]BSAVE FID/CATALOG,A$803,L$124E  save the new version
```

Now if you BRUN FID/CATALOG you will see the normal FID menu. Select option 8 (VERIFY), specify a slot and drive, and type a file name (preferably with the "=" wildcard in it). Specify NO prompting. When you "PRESS ANY OTHER KEY TO BEGIN" you will see a list of all files whose names match the filename you typed.

Random Number Generator from Integer BASIC

When you are writing games or other simulation exercises, you frequently need a source of random numbers. In Basic it's easy, but how about assembly language?

The WozPak from Call A.P.P.L.E. has directions for calling the RND(X) function in the Integer BASIC ROMs. Remember that this function returns a random integer between 0 and X-1 for an argument X. Linda Egan, from Maywood, California, wrote that she had trouble making the WozPak method work. I don't know what that method was, but I looked up the code in the ROM and came up with some working code.

```
                    1000 *---------------------------------
                    1010 *       RANDOM FUNCTION
                    1020 *       ---------------
                    1030 *       CALLS SUBROUTINE IN INTEGER BASIC ROM TO GET
                    1040 *       A RANDOM NUMBER BETWEEN 0 ANT X-1
                    1050 *
                    1060 *       CALL:   VALUE X IN Y- AND A-REGISTERS
                    1070 *               JSR RANDOM
                    1080 *     RETURN:   RANDOM NUMBER IN Y- AND A-REGISTERS
                    1090 *               LO-BYTE IN Y, HI-BYTE IN A
                    1100 *---------------------------------
00CE-               1110 IB.ARG      .EQ $CE,CF
0050-               1120 IB.LOSTACK  .EQ $50 THRU $6F
00A0-               1130 IB.HISTACK  .EQ $A0 THRU $BF
                    1140 *---------------------------------
EF51-               1150 IB.RANDOM   .EQ $EF51
FDDA-               1160 MON.PRBYTE  .EQ $FDDA
FDED-               1170 MON.COUT    .EQ $FDED
                    1180 *---------------------------------
0800- A2 20         1190 RANDOM LDX #$20      I/B NOUN-STACK POINTER
0802- 85 CF         1200        STA IB.ARG+1
0804- 84 CE         1210        STY IB.ARG
0806- A0 00         1220        LDY #0        FLAG VALUE ON STACK
0808- 20 51 EF      1230        JSR IB.RANDOM
080B- B5 A0         1240        LDA IB.HISTACK,X
080D- B4 50         1250        LDY IB.LOSTACK,X
080F- 60            1260        RTS
                    1270 *---------------------------------
                    1280 TEST.RANDOM
0810- A9 A0         1290        LDA #160
0812- 8D 2E 08      1300        STA COUNT
0815- A0 E8         1310 .1     LDY #1000
0817- A9 03         1320        LDA /1000
0819- 20 00 08      1330        JSR RANDOM    RND(1000)
081C- 20 DA FD      1340        JSR MON.PRBYTE
081F- 98            1350        TYA
0820- 20 DA FD      1360        JSR MON.PRBYTE
0823- A9 A0         1370        LDA #$A0      PRINT BLANK
0825- 20 ED FD      1380        JSR MON.COUT
0828- CE 2E 08      1390        DEC COUNT
082B- D0 E8         1400        BNE .1
082D- 60            1410        RTS
082E-               1420 COUNT  .BS 1
```

Lines 1190-1260 are all you need. They set up a call to the ROM code, and pick up the returned value.

Line 1190 sets the X-register to $20. The ROM code uses X for a stack index, and $20 means an empty stack. This is not the hardware stack ($100-1FF), but a software-implemented stack. The stack is in three parts. The part I call IB.LOSTACK runs from $50 thru $6F. IB.HISTACK runs from $A0 thru $BF. A third part runs from $78 thru $97. The ROM code pushes our argument on these stacks like this: the low byte goes on LOSTACK, the high byte on HISTACK, and a zero (from the Y-register) on the FLAGSTACK. (If the value pushed on FLAGSTACK was not zero, it would be used as the high-byte of an address along with the low-byte from LOSTACK to indirectly address the data value.)

Lines 1200 and 1210 store our argument where the ROM code expects it to be, in $CE and $CF. Lines 1240 and 1250 retrieve the resulting random number from the stack.

Lines 1280 through 1420 are a test loop to demonstrate the random function. Twenty lines of eight random numbers each are printed on the screen in hexadecimal. I used an argument of 1000, so all the numbers are between 0 and 999.

What if you don't have the Integer BASIC ROMs in your Apple? Since the code is not very long, you could make your own copy of Woz's routines. I did that, and came up with the following program. I used the same test loop, but this time it is in lines 1760 thru 1900.

```
                    1000 *--------------------------------
                    1010 *      STAND-ALONE RANDOM FUNCTION
                    1020 *      ---------------------------
                    1030 *
                    1040 *      GET A RANDOM NUMBER BETWEEN 0 AND X-1
                    1050 *
                    1060 *      CALL:   VALUE X IN Y- AND A-REGISTERS
                    1070 *              JSR RANDOM
                    1080 *    RETURN:   RANDOM NUMBER IN Y- AND A-REGISTERS
                    1090 *              LO-BYTE IN Y, HI-BYTE IN A
                    1100 *--------------------------------
004E-               1110 MON.RNDL   .EQ $4E
004F-               1120 MON.RNDH   .EQ $4F
FDDA-               1130 MON.PRBYTE .EQ $FDDA
FDED-               1140 MON.COUT   .EQ $FDED
                    1150 *--------------------------------
0800- 8C 6F 08      1160 RANDOM STY LIMIT     SAVE LIMIT VALUE
0803- 8D 70 08      1170        STA LIMIT+1
0806- A5 4F         1180        LDA MON.RNDH GET SEED HI-BYTE
0808- D0 04         1190        BNE .1       BE SURE SEED BTWN 1 AND 7FFF
080A- C5 4E         1200        CMP MON.RNDL SET CARRY IF BOTH BYTES ZERO
080C- 69 00         1210        ADC #0       CHANGE 0000 TO 0100
080E- 29 7F         1220 .1     AND #$7F     MAKE SURE NOT LARGER THAN 7FFF
0810- 85 4F         1230        STA MON.RNDH
0812- 8D 72 08      1240        STA VALUE+1
0815- A5 4E         1250        LDA MON.RNDL
0817- 8D 71 08      1260        STA VALUE
081A- A9 00         1270        LDA #0
081C- 8D 73 08      1280        STA VALUE+2
081F- 8D 74 08      1290        STA VALUE+3
                    1300 *--------------------------------
0822- A0 11         1310        LDY #17      LOOP TO MAKE NEXT RANDOM VALUE
0824- A5 4F         1320 .2     LDA MON.RNDH (WOZNIAK'S ALGORITHM)
0826- 0A            1330        ASL
0827- 18            1340        CLC
0828- 69 40         1350        ADC #$40
082A- 0A            1360        ASL
082B- 26 4E         1370        ROL MON.RNDL
082D- 26 4F         1380        ROL MON.RNDH
082F- 88            1390        DEY
0830- D0 F2         1400        BNE .2
                    1410 *--------------------------------
0832- AD 6F 08      1420        LDA LIMIT
0835- 0D 70 08      1430        ORA LIMIT+1
0838- F0 2E         1440        BEQ .5       RETURN ZERO
                    1450 *--------------------------------
                    1460 *      DIVIDE RANDOM VALUE (1-7FFF) BY LIMIT
                    1470 *      AND USE REMAINDER (0<=REMAINDER<LIMIT)
                    1480 *--------------------------------
083A- A0 10         1490        LDY #16      LOOP FOR 16-BITS
083C- 0E 71 08      1500 .3     ASL VALUE    DOUBLE DIVIDEND
083F- 2E 72 08      1510        ROL VALUE+1
0842- 2E 73 08      1520        ROL VALUE+2
0845- 2E 74 08      1530        ROL VALUE+3
0848- AD 73 08      1540        LDA VALUE+2
084B- CD 6F 08      1550        CMP LIMIT
084E- AD 74 08      1560        LDA VALUE+3
0851- ED 70 08      1570        SBC LIMIT+1
0854- 90 0F         1580        BCC .4       PARTIAL DIVIDEND < LIMIT
0856- 8D 74 08      1590        STA VALUE+3
0859- AD 73 08      1600        LDA VALUE+2  CARRY IS SET, SUBTRACT
085C- ED 6F 08      1610        SBC LIMIT    LO-BYTE OF LIMIT
085F- 8D 73 08      1620        STA VALUE+2
0862- EE 71 08      1630        INC VALUE    SET BIT IN QUOTIENT
0865- 88            1640 .4     DEY
0866- D0 D4         1650        BNE .3
                    1660 *--------------------------------
                    1670 *      RETURN RANDOM VALUE MOD LIMIT
                    1680 *--------------------------------
0868- AD 74 08      1690 .5     LDA VALUE+3  PICK UP REMAINDER FROM DIVISION
086B- AC 73 08      1700        LDY VALUE+2
086E- 60            1710        RTS
                    1720 *--------------------------------
086F-               1730 LIMIT  .BS 2
0871-               1740 VALUE  .BS 4
                    1750 *--------------------------------
```

```
                  1760 TEST.RANDOM
0875- A9 A0       1770          LDA #160
0877- 8D 93 08    1780          STA COUNT
087A- A0 E8       1790 .1       LDY #1000
087C- A9 03       1800          LDA /1000
087E- 20 00 08    1810          JSR RANDOM    RND(1000)
0881- 20 DA FD    1820          JSR MON.PRBYTE
0884- 98          1830          TYA
0885- 20 DA FD    1840          JSR MON.PRBYTE
0888- A9 A0       1850          LDA #$A0      PRINT BLANK
088A- 20 ED FD    1860          JSR MON.COUT
088D- CE 93 08    1870          DEC COUNT
0890- D0 E8       1880          BNE .1
0892- 60          1890          RTS
0893-             1900 COUNT    .BS 1
```

Lines 1160 and 1170 save the argument for later use. Lines 1180-1260 get the current random seed from the Apple Monitor and store it in VALUE. However, if the seed was 0000 it is converted to 0100. This is because a seed of 0000 replicates itself forever. Furthermore, the sign bit is stripped off; in other words, VALUE is set to the seed value modulo 32768. This is supposed to force the VALUE to be between 1 and 7FFF.

The random seed is also modified by the monitor whenever you are in KEYIN waiting for an input from the keyboard. This code is at $FD1B thru $FD24 in the monitor ROM. This means the seed might have any (truly random) value between 0000 and FFFF. If by chance it is $8000 when the RND function is called, VALUE will be set to 0000.

Lines 1270-1290 clear two more bytes of VALUE, which will be used later, in the division loop.

Lines 1300-1400 are Woz's algorithm for generating a sequence of random integers. It is a binary polynomial technique, but there seems to be a bug in it. If you run it 32768 times, you should generate each and every value between 0 and $7FFF exactly one time, but in random order. I tested it, and it really generates the values between $6000 and $60FF twice, and never generates $2000-20FF at all! You can play with it and see if there are some seed values which will produce numbers between $2000 and $20FF.

Lines 1420-1440 check the argument. If it is zero, I return the value zero for the function. Integer BASIC would give you "*** >32767 ERR" with a zero argument.

Lines 1490-1650 are a division program, to divide the random VALUE by the LIMIT. After it is finished, the quotient is in VALUE and VALUE+1, and the remainder is in VALUE+2 and VALUE+3. We don't need the quotient; the remainder is the random value we want.

Lines 1690-1710 pick up the result in registers A and Y, and return to the calling program.

What Does This Code Do?.........................John Broderick

What does it do? Why would you want to use it? Those who send in correct answers will get their names published here in a few months with the solution.

```
SUBROUTINE:  BRK
             PLA
             PLA
             PLA
             RTS
```

OK, I'll give you a little hint. One of the five instructions is not used by the 6502 processor. Can you tell which one?

As far as I know, this routine has never before been published; however, I use it in almost every program I write. It's a jewel of a routine, worth many times its weight in gold!

Send your answers to John Broderick, 8635 Shagrock, Dallas, TX 75238. If you have any similar neat code segments, send them with explanation. I'll try to make this a regular column in the AAL.

Correction to "Assembly Source on Text Files"

Volume 1, Issue 2 of Apple Assembly Line contained a program for writing assembly source programs for the S-C Assembler II Version 4.0 on DOS text files. Peter Bartlett of Chicago was trying to use it with a Corvus Hard Disk, and found a problem with the program.

The Corvus system will not accept a CLOSE command unless there is a file name on it (unlike regular DOS). One solution is to delete the two calls to CLOSE.FILE at lines 1410 and 1570.

While talking with Peter I discovered a bug in my program, in the subroutine named ISSUE.DOS.COMMAND. It is supposed to allow slot and drive parameters on the file name. This was described in the write-up on page 11. Two errors made it not work.

First, line 1910 says:

```
     1910        CMP #',      COMMA?
```

but the character in the A-register has the high bit set to one. Cvhange line 1910 to:

```
     1910        CMP #$AC     COMMA?
```

Second, line 1940 says:

```
     1940        STA DOS.BUFFER,Y
```

Change it to:

```
     1940        STA DOS.BUFFER-1,Y
```

The line numbers above correspond to the printed listing in the AAL article. They may not be exactly the same as the source code on Quarterly Disk #1. If you have Quarterly Disk #1 with a serial number of 45 or higher, your copy is already fixed.

About Advertising

Do you have a new product you want to test market, which would appeal to the Apple Assembly Line readers? You ought to try an ad in these pages. The current price is $20 for a full page, $10 for a half page. Send it to me just as you want it printed (I can do the reduction to make it fit on the page).

Commented Listing of DOS 3.3 Boot ROM

The P5A ROM on your Apple Disk II Controller has a 256-byte program in it which reads track 0 sector 0 into memory and starts executing it.

The data in track 0 sector 0 is read into memory from $0800-08FF. Location $0800 contains a value indicating how many sectors to boot in. This is usually zero, meaning to read only sector zero. However, it could be as high as $0F, meaning to read all 16 sectors of track 0 into memory from $0800-17FF. (The BASICS diskette uses this feature.) Once the selected number of sectors has been read, the boot ROM jumps to $0801 to start execution. At this point (in a normal DOS boot) the rest of DOS is loaded.

My listing starts at $C600, which is where it will be if your controller is in slot 6. The code is all independent of position, so that it can be plugged into any slot. In fact, you can move the code into RAM if you like, just so the second digit of the address is the same as the controller card slot number. I do this some times when I am trying to crack locked disks. I go to the monitor, type 8600<C600.C6FFM, and then patch a BRK opcode on top of the JMP $0801 at $86F8. Then 8600G will read in track 0 sector 0 and BRK back to the monitor, and I can analyze the code to see how the rest is read in.

Enough of that, let's get into the code! Lines 1510-1690 are an esoteric loop which generate the nybble conversion table. The table is built in page 3, from $36C through $3D5. I tried out the loop after storing FF bytes throughout page 3, and got this:

```
0368- FF FF FF FF 00 01 FF FF   03A0- FF 1B FF 1C 1D 1E FF FF
0370- 02 03 FF 04 05 06 FF FF   03A8- FF 1F FF FF 20 21 FF 22
0378- FF FF FF FF 07 08 FF FF   03B0- 23 24 25 26 27 28 FF FF
0380- FF 09 0A 0B 0C 0D FF FF   03B8- FF FF FF 29 2A 2B FF 2C
0388- 0E 0F 10 11 12 13 FF 14   03C0- 2D 2E 2F 30 31 32 FF FF
0390- 15 16 17 18 19 1A FF FF   03C8- 33 34 35 36 37 38 FF 39
0398- FF FF FF FF FF FF FF FF   03D0- 3A 3B 3C 3D 3E 3F FF FF
```

These bytes are referred to at lines 2670 and 2740, indexed from a base of $02D6. This makes a disk code of $96 give a $00 value, and a code of $FF give a value of $3F.

Lines 1710-1790 determine the slot number and multiply it by 16. The JSR MON.RTS is to an RTS instruction in the monitor ROM. The only purpose of this JSR is to put its own address on the stack. Then lines 1720 and 1730 lift up the high byte of the address from the stack. The second digit of this address is the slot number, and 4 ASL's will isolate it and multiply it by 16. Lines 1800-1830 select drive 0 and turn on the motor. (If you want to boot from drive 2, you can copy this code into RAM at $8600 and change the byte at $8636 from $8A to $8B.)

Lines 1880-1990 move the head to track 0 from wherever it was. If you were already at track 0, it just sits there making a racket as it bangs against the stop. Lines 2030-2070 initialize the track and sector numbers and the memory address to read into.

Lines 2090-2480 read a sector into the input area. Lines 2110-2290 are used two different ways, depending on the CARRY status upon entry. The first time CARRY is clear, and we look for an address header (D5 AA 96). After finding an address header the sector and track are check in lines 2300-2480; if they are the ones we want, CARRY is set and we do lines 2110-2290 over again. This time they look for a data header. If one is found, it's time to read the data.

Lines 2530-2880 read in the sector. First 86 bytes are read into a little buffer at the bottom of page 3 ($0300-0355). Then 256 bytes are read into the target memory area (normally $0800-08FF). A checksum is computed and checked; if it doesn't match, we start all over. Lines 2770-2880 put the bits from $0300-0355 together with those in the main buffer, in the same way discussed two months ago in the listing of DOS 3.3 B800-BCFF.

Lines 2900-2950 check whether we have read all the sectors specified by the first byte of track 0 sector 0. If not, loop back to read the next sector one page higher in memory. When they have all been read, control branches to $0801. The normal DOS boot only reads one sector before branching to $0801.

```
                         1010 *      DOS 3.3 BOOT ROM $C600.C6FF
                         1020 *
                         1030 *      COMMENTS BY BOB SANDER-CEDERLOF
                         1040 *                JULY, 4, 1981
                         1050 *---------------------------------
                         1060 *      DISK CONTROLLER ADDRESSES
                         1070 *---------------------------------
C080-                    1080 PHOFF  .EQ $C080      PHASE-OFF
C081-                    1090 PHON   .EQ $C081      PHASE-ON
C088-                    1100 MTROFF .EQ $C088      MOTOR OFF
C089-                    1110 MTRON  .EQ $C089      MOTOR ON
C08A-                    1120 DRV0EN .EQ $C08A      DRIVE 0 ENABLE
C08B-                    1130 DRV1EN .EQ $C08B      DRIVE 1 ENABLE
C08C-                    1140 Q6L    .EQ $C08C      SET Q6 LOW
C08D-                    1150 Q6H    .EQ $C08D      SET Q6 HIGH
C08E-                    1160 Q7L    .EQ $C08E      SET Q7 LOW
C08F-                    1170 Q7H    .EQ $C08F      SET Q7 HIGH
                         1180 *
                         1190 *      Q6    Q7       USE OF Q6 AND Q7 LINES
                         1200 *     ----  ----      ----------------------
                         1210 *     LOW   LOW       READ (DISK TO SHIFT REGISTER)
                         1220 *     LOW   HIGH      WRITE (SHIFT REGISTER TO DISK)
                         1230 *     HIGH  LOW       SENSE WRITE PROTECT
                         1240 *     HIGH  HIGH      LOAD SHIFT REGISTER FROM DATA BUS
                         1250 *---------------------------------
0026-                    1260 BUFFER.PNTR .EQ $26,27
002B-                    1270 SLOT16 .EQ $2B        SLOT NUMBER TIMES 16
003D-                    1280 SECTOR .EQ $3D
0041-                    1290 TRACK  .EQ $41
0100-                    1300 STACK       .EQ $0100
02D6-                    1310 POST.NYBBLE.CODES .EQ $02D6
0300-                    1320 LITTLE.BUFFER     .EQ $0300
FF58-                    1330 MON.RTS     .EQ $FF58
FCA8-                    1340 MON.WAIT    .EQ $FCA8
                         1350 *---------------------------------
                         1360        .OR $C600
                         1370        .TA $0800
                         1380 *---------------------------------
                         1390 BOOT.3.3
C600- A2 20              1400        LDX #$20       REDUNDANT INSTRUCTION, USED
                         1410 *                     TO IDENTIFY CONTROLLER CARD
                         1420 *---------------------------------
                         1430 *      GENERATE POST-NYBBLE CONVERSION TABLE
                         1440 *          FILLS IN THOSE SLOTS WHOSE INDEX
                         1450 *          RELATIVE TO POST.NYBBLE.CODES IS
                         1460 *          A VALID NYBBLE CODE.  (VALID CODES
                         1470 *          HAVE AT MOST ONE PAIR OF ADJACENT
                         1480 *          0-BITS, AND AT LEAST ONE PAIR OF
                         1490 *          ADJACENT 1-BITS IN BITS 0-6.)
```

```
                    1500 *---------------------------------
C602- A0 00         1510        LDY #0
C604- A2 03         1520        LDX #3       COULD BE ANY VALUE FROM 0 TO $16
                    1530 *                   3 USED FOR CONTROLLER ID
C606- 86 3C         1540 .1     STX $3C      CHECK CODE FOR VALID NYBBLE
C608- 8A            1550        TXA
C609- 0A            1560        ASL
C60A- 24 3C         1570        BIT $3C      TEST (X .AND. 2*X)
C60C- F0 10         1580        BEQ .3       NO ADJACENT 1-BITS, NO GOOD
C60E- 05 3C         1590        ORA $3C      TEST ADJACENT 0-BITS
C610- 49 FF         1600        EOR #$FF     CHANGE TO 1'S FOR TEST
C612- 29 7E         1610        AND #$7E     DON'T CARE ABOUT BIT 7
C614- B0 08         1620 .2     BCS .3       NOT VALID NYBBLE CODE
C616- 4A            1630        LSR
C617- D0 FB         1640        BNE .2
C619- 98            1650        TYA
C61A- 9D 56 03      1660        STA POST.NYBBLE.CODES+$80,X
C61D- C8            1670        INY
C61E- E8            1680 .3     INX
C61F- 10 E5         1690        BPL .1
                    1700 *---------------------------------
C621- 20 58 FF      1710        JSR MON.RTS  GET THIS LOCATION ON STACK
C624- BA            1720        TSX          FIND THE PAGE BYTE ON STACK
C625- BD 00 01      1730        LDA STACK,X
C628- 0A            1740        ASL          ISOLATE SLOT NUMBER
C629- 0A            1750        ASL          AND MULTIPLY BY 16
C62A- 0A            1760        ASL
C62B- 0A            1770        ASL
C62C- 85 2B         1780        STA SLOT16   SLOT NUMBER TIMES 16
C62E- AA            1790        TAX
C62F- BD 8E C0      1800        LDA Q7L,X    SET UP TO READ DRIVE
C632- BD 8C C0      1810        LDA Q6L,X
C635- BD 8A C0      1820        LDA DRV0EN,X ENABLE DRIVE 0
C638- BD 89 C0      1830        LDA MTRON,X  TURN ON MOTOR
                    1840 *---------------------------------
                    1850 *      MOVE TO TRACK 0 (ASSUME WORST CASE
                    1860 *      INITIAL POSITION OF TRACK 40).
                    1870 *---------------------------------
C63B- A0 50         1880        LDY #80      80 HALF-TRACKS
C63D- BD 80 C0      1890 .4     LDA PHOFF,X  STEPPER MOTOR PHASE OFF
C640- 98            1900        TYA          COMPUTE NEXT PHASE
C641- 29 03         1910        AND #3       YIELDS 3,2,1,0
C643- 0A            1920        ASL          YIELDS 6,4,2,0
C644- 05 2B         1930        ORA SLOT16   MERGE WITH SLOT*16
C646- AA            1940        TAX
C647- BD 81 C0      1950        LDA PHON,X   STEPPER MOTOR PHASE ON
C64A- A9 56         1960        LDA #86      WAIT 19.2 MILLISECONDS
C64C- 20 A8 FC      1970        JSR MON.WAIT NO CHANGE TO X OR Y, A=0
C64F- 88            1980        DEY          NEXT HALF-TRACK
C650- 10 EB         1990        BPL .4
                    2000 *---------------------------------
                    2010 *      A=0, X=SLOT*16
                    2020 *---------------------------------
C652- 85 26         2030        STA BUFFER.PNTR   ($00 --> LOW BYTE OF PNTR)
C654- 85 3D         2040        STA SECTOR 0
C656- 85 41         2050        STA TRACK 0
C658- A9 08         2060        LDA #8       BUFFER AT $0800
C65A- 85 27         2070        STA BUFFER.PNTR+1   ($08 --> HI-BYTE OF PNTR)
                    2080 *---------------------------------
                    2090 READ.SECTOR
C65C- 18            2100 .1     CLC          FLAG CLEAR, LOOK FOR $D5 AA 96
C65D- 08            2110 .2     PHP          SAVE FLAG ON STACK
C65E- BD 8C C0      2120 .3     LDA Q6L,X    READ DISK
C661- 10 FB         2130        BPL .3
C663- 49 D5         2140 .4     EOR #$D5
C665- D0 F7         2150        BNE .3       NO
C667- BD 8C C0      2160 .5     LDA Q6L,X    READ DISK
C66A- 10 FB         2170        BPL .5
C66C- C9 AA         2180        CMP #$AA
C66E- D0 F3         2190        BNE .4
C670- EA            2200        NOP
C671- BD 8C C0      2210 .6     LDA Q6L,X    READ DISK
C674- 10 FB         2220        BPL .6
C676- C9 96         2230        CMP #$96
C678- F0 09         2240        BEQ .7       FOUND ADDRESS MARK: $D5 AA 96
C67A- 28            2250        PLP          RETRIEVE FLAG
C67B- 90 DF         2260        BCC .1       LOOKING FOR ADDRESS HEADER
C67D- 49 AD         2270        EOR #$AD     LOOKING FOR DATA HEADER
C67F- F0 25         2280        BEQ FILL.BUFFER
C681- D0 D9         2290        BNE .1       START ALL OVER
```

```
                    *---------------------------------
C683- A0 03         .7       LDY #3        READ VOLUME, TRACK, SECTOR
C685- 85 40         .8       STA $40
C687- BD 8C C0      .9       LDA Q6L,X     READ DISK
C68A- 10 FB                  BPL .9
C68C- 2A                     ROL           SAVE UPPER SLICE
C68D- 85 3C                  STA $3C
C68F- BD 8C C0      .10      LDA Q6L,X     READ DISK
C692- 10 FB                  BPL .10
C694- 25 3C                  AND $3C       MERGE SLICES
C696- 88                     DEY           3RD BYTE YET?
C697- D0 EC                  BNE .8        NO, GET ANOTHER
C699- 28                     PLP           THROW AWAY FLAG
C69A- C5 3D                  CMP SECTOR    CORRECT SECTOR?
C69C- D0 BE                  BNE .1        NO
C69E- A5 40                  LDA $40       CORRECT TRACK?
C6A0- C5 41                  CMP TRACK
C6A2- D0 B8                  BNE .1        NO
C6A4- B0 B7                  BCS .2        YES, SET FLAG FOR DATA HEADER
                    *                      AND BRANCH BACK ALWAYS
                    *---------------------------------
                    *        A=0 ON ENTRY
                    *---------------------------------
                    FILL.BUFFER
C6A6- A0 56                  LDY #86       READ 86 BYTES
C6A8- 84 3C         .1       STY $3C
C6AA- BC 8C C0      .2       LDY Q6L,X     READ BYTE
C6AD- 10 FB                  BPL .2
C6AF- 59 D6 02               EOR POST.NYBBLE.CODES,Y  DECODE BYTE
C6B2- A4 3C                  LDY $3C
C6B4- 88                     DEY
C6B5- 99 00 03               STA LITTLE.BUFFER,Y
C6B8- D0 EE                  BNE .1
                    *---------------------------------
C6BA- 84 3C         .3       STY $3C       Y=0
C6BC- BC 8C C0      .4       LDY Q6L,X     READ BYTE
C6BF- 10 FB                  BPL .4
C6C1- 59 D6 02               EOR POST.NYBBLE.CODES,Y DECODE BYTE
C6C4- A4 3C                  LDY $3C
C6C6- 91 26                  STA (BUFFER.PNTR),Y
C6C8- C8                     INY
C6C9- D0 EF                  BNE .3
C6CB- BC 8C C0      .5       LDY Q6L,X     READ CHECKSUM BYTE
C6CE- 10 FB                  BPL .5
C6D0- 59 D6 02               EOR POST.NYBBLE.CODES,Y
C6D3- D0 87         .6       BNE READ.SECTOR  BAD CHECKSUM, START OVER
                    *---------------------------------
C6D5- A0 00                  LDY #0
C6D7- A2 56         .7       LDX #86       PATCH THE 6+2 BACK TOGETHER
C6D9- CA            .8       DEX
C6DA- 30 FB                  BMI .7        FINISHED A TRIP
C6DC- B1 26                  LDA (BUFFER.PNTR),Y
C6DE- 5E 00 03               LSR LITTLE.BUFFER,X
C6E1- 2A                     ROL
C6E2- 5E 00 03               LSR LITTLE.BUFFER,X
C6E5- 2A                     ROL
C6E6- 91 26                  STA (BUFFER.PNTR),Y
C6E8- C8                     INY
C6E9- D0 EE                  BNE .8
                    *---------------------------------
C6EB- E6 27                  INC BUFFER.PNTR+1  POINT AT NEXT PAGE
C6ED- E6 3D                  INC SECTOR      POINT AT NEXT SECOTR
C6EF- A5 3D                  LDA SECTOR
C6F1- CD 00 08               CMP $0800     SEE IF HAVE READ ENUF SECTORS
C6F4- A6 2B                  LDX SLOT16
C6F6- 90 DB                  BCC .6        NOT ENUF SECTORS YET
C6F8- 4C 01 08               JMP $0801     GO TO REST OF BOOT
                    *---------------------------------
C6FB- 00 00 00
C6FE- 00 00                  .HS 0000000000  UNUSED BYTES IN ROM
```

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $12 per year in the U.S.A., Canada, and Mexico. Other countries add $12/year for extra postage. Back issues are available for $1.20 each (other countries add $1 per back issue for postage). Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)