$1

Washington Apple Pi

Volume 2 June 1980 Number 6

Highlights

In This Issue

ComputerLand® and apple II

For the best in personal computing

SOFTAPE™

Personal Software™

D. C. Hayes Associates, Inc.
MICROCOMPUTER PRODUCTS

CENTRONICS®

Mountain Hardware, Inc.

Integral Data Systems, Inc.

MUSE

THE SOURCE
Information Utility

PDI

houston
instrument

Heuristics
INC.

Automated Simulations

SANYO

ComputerLand®

VISA

We Know Small Computers.
ComputerLand/Tysons Corner
8411 Old Courthouse Road at Rt. 123 — 893-0424

master charge

Officers & Staff

President	-Bernard Urban	(301) 229-3458
Vice President	-Rich Wasserstrom	(703) 893-9147
Treasurer	-Bob Peck	(301) 468-2305
Secretary	-Dana Schwartz	(301) 725-6281
Members-at-Large	-Scooter Conrad	(301) 725-6251
	-Mark Crosby	(202) 488-1980
	-Sandy Greenfarb	(301) 674-5982
Editor	-Bernard Urban	(above)
Associate Editor	-Rich Wasserstrom	(above)
	-Genevie Urban	(301) 229-3458
Librarian	-David Morganstein	(301) 972-4263

Washington Apple Pi
P. O. Box 34511
Washington, D.C. 20034
(301) 468-2305

Membership dues for Washington Apple Pi are $12.00 per calendar year. If you are interested in joining our club, call our number and leave your name and address. An application form will be mailed to you. Or if you prefer, write us at the above PO Box.

EVENT QUEUE

Washington Apple Pi meets on the 4th Saturday of each month at 9:30AM at George Washington University School of Engineering, 23rd and H Streets, N.W. The July meeting will be held on July 26.

NOVAPPLE meets on the 2nd Thursday at 7:30PM at Computers Plus in Franconia, and on the 4th Thursday at 7:30 PM at Computerland of Tysons Corner.

Classifieds

FOR SALE: Applesoft Firmware with Autostart ROM and Applesoft Reference Manual. $135. Tom Jones (H) 460-8773, (O) 881-5310.

WANTED TO RENT: I would like to rent an Apple II or II+ for one to three months while waiting delivery of Apple III. John Robb, Box 61, Rockville, Md. 340-2652 (day or evening).

Classified ads accepted from members 50 words or less at no charge provided the material is obviously non-commercial. Submit your classified at least 30 days in advance attention CLASSIFIED ADS, PO Box 34511, Washington, DC 20034.

EDITORIAL

PRESIDENT'S MESSAGE

The votes are in and counted. As of this month, we have a new administration. Your new officers are listed elsewhere on this page.

Let me express my personal thanks and that of the entire membership to our outgoing officers, John, Sue and Gena for jobs well done. But please don't stray too far (or at all) - we need your help and support, and hopefully you will be available to serve in similar capacities again. John will stay on as honorary member of the Executive Board. Sue is still our Program Chairperson, and Gena continues as an Associate Editor.

At a Board meeting held subsequently to the elections (see Dana Schwartz's minutes of the meeting elsewhere in this issue), we agreed to the following:

. Board meetings will be held "regularly" on the second Wednesday of each month. These meetings are open to our membership and we encourage attendance by any of you who wish to recommend new directions or services, criticize, or whatever. Exact time and place will be announced on our official phone 468-2305.

. We appointed Tom Jones as our Membership Chairperson. Tom will look into such things as a membership directory (disk or otherwise), our makeup and interests, and the issuance of membership cards.

. Dave Efron is our new Ad Manager. He will be our contact with the local computer stores and the software and hardware firms in general. Dave is also interested in organizing any and all printed materials on the APPLE II. Hopefully, we can establish a library or libraries of such materials for the benefit of our membership. Dave can help immediately by focusing on the APNotes that have come in from the IAC which have been piling up and unfortunately gathering dust at my home. Also, requiring attention are the newsletters I've been getting from our associate user groups in the Eastern IAC region who reciprocate the mailing of our newsletters to them.

Voiced again and again was the feeling that we must do something about our Saturday meetings. We have, it appears, at least three groups of attendees. We have many members who are new to computers and who need all the assistance that the club can muster. We also have members who are highly expert in selected or several areas. They become bored by discussions at the elementary or intermediate levels. Then there are the youngsters who like to come to hear about the latest in games, to play them and to swap them. Hersch Pilloff's efforts to run a question and answer session and Al Gass's SIG on games are attempts to rectify some of these problems. This needs further attention. All suggestions are most welcome.

minutes

The Washington Apple Pi meeting of May 31, 1980 was called to order at 9:45 AM by the Vice President. The results of the annual election (by mail ballot) were announced by Chuck and Nancy Philipp, who had tallied the ballots. The new officers, effective June 1, are:

President	- Bernie Urban
Vice President	- Rich Wasserstrom
Secretary	- Dana Schwartz
Treasurer	- Bob Peck
Members At Large	- Scooter Conrad
	- Mark Crosby
	- Sandy Greenfarb

The following items of interest were discussed. Sue Zakar announced that the program for the June meeting would be given by Paul Sand on the subject of APPLE III. She asked for volunteers for future programs. The question of a membership directory for the club was discussed. Ideas were presented pro and con regarding listing members with their names and addresses. It was announced that the club is in the process of setting up a modem bulletin board, with John Moon in charge. Dave Morganstein reported that we now have 19 library disks. There was a discussion of why the purchased disks sometimes do not work and suggestions for possible correction of this were presented, particularly that of adjusting the speed of the disk drive. Bob Peck presented some information on group purchases.

The meeting was then turned over to the speaker of the morning, Theron Fuller. He presented a program on Pilot, a higher level language used for computer assisted instruction. His discussion described compilers, translators and their structure. It was a most informative presentation.

MINUTES OF JUNE EXECUTIVE BOARD MEETING

The June Executive Board meeting was held on June 4 at the home of the President. The meeting was called to order at 7:45PM. The following items were covered:

1. The Treasurer reported that the club was having difficulties collecting on some bills for newsletters and ads. A motion was made that delinquent commercial accounts be given newsletters on a cash-only basis. The motion was tabled until the next meeting to allow further collection efforts.

2. It was decided that the Board would hold regular meetings on the second Wednesday of each month and the membership should be reminded that all are invited.

3. A revised format for the regular club meetings was decided upon, and will be announced by the President. The aim is to provide more useful information to members at all levels of experience.

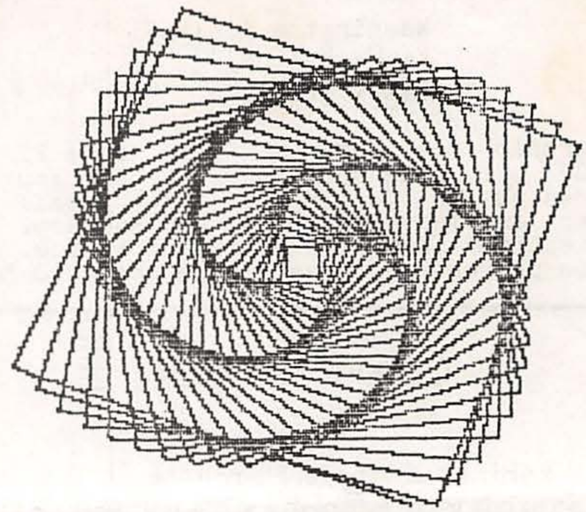
4. The membership directory was discussed and member Tom Jones volunteered, and was directed by the board, to come up with a suggested format. A motion was made and passed that whatever form the directory takes, the membership will be given the option to withhold any or all personal data as each wishes.

5. The Treasurer reported that it would be in the club's best interest to have an Advertising Manager for the Newsletter. A motion was made and passed that the club should search for and obtain an Advertising Manager, expenses and commission negotiable, for a trial period of six months.

The meeting was adjourned at 10:15 PM.

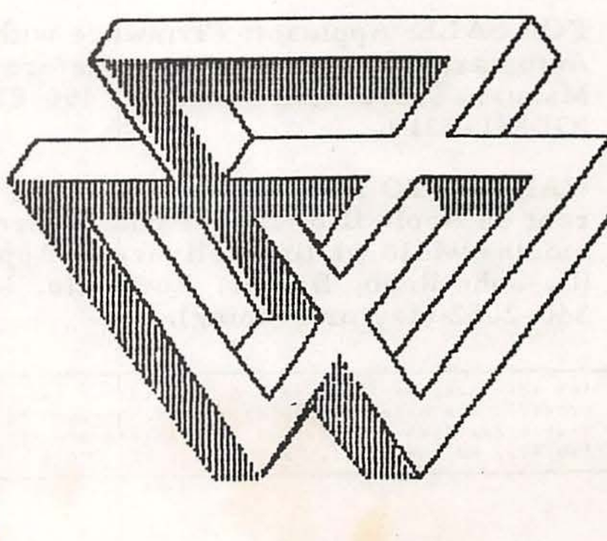
Dana J. Schwartz, Secretary

(Ed. Note: The Executive Board has decided to publish the minutes of their meetings in order to keep the general membership better informed on club business.)



SIGNEWS

In a previous Newsletter, Michael Thomas offered to set up a SIG group for students. His phone number was listed incorrectly. The correct number is (703) 978-8411



SOFTWARE FOR THE EXPERIMENTER'S REAL-TIME CLOCK

by Bruce F. Field

This article is a continuation of the description of a real-time clock for the APPLE. The hardware details were described in last month's Washington Apple Pi newsletter. This month we will look at the software that is required for the clock to keep time. When completed we will have the ability to continuously display the time in the upper right-hand corner of the screen even when running other BASIC or machine language programs.

To review the hardware described last month, the frequency of the power line is divided by 60 to produce an interrupt to the APPLE once a second. If for any reason the interrupt input on the APPLE is disabled (i.e. for I/O transfers) the hardware counts the number of seconds and when the interrupt input is re-enabled the clock will continuously generate interrupts until the software count is properly updated.

There are two programs necessary to make the system behave like a real-time clock.

1. We must have an interrupt service routine to update the time stored in memory, and display the new time on the screen each time an interrupt occurs.

2. We also need a program to load in the interrupt service routine, set the interrupt vector to the routine, and set the initial time in the clock.

The interrupt service routine must be written in machine code and should be able to reside in memory with other application programs. The assembly language program to do this is shown in listing 1. The program is fairly flush with comments to make it easy to understand, however for those of you not familiar with assembly language I will attempt to explain what is happening.

The first thing the program does is to perform a load at an address that will pulse the device select line of the slot that contains the clock. This will decrement by one the count stored in the hardware. After that, the time stored as hours, minutes, and seconds is increased by one second. In order to make the arithmetic easy, the time is stored in three bytes with each byte representing hours, minutes, or seconds. The actual values are in BCD (binary coded decimal). One of the nice features of the 6502 is the ability to do decimal arithmetic simply by setting the decimal flag in the processor status register. This is done by executing an SED (set decimal mode) instruction. After this is done all machine language arithmetic instructions will do arithmetic as if the numbers are BCD. If a memory location contains $69 ($ denotes a hexadecimal number) and $3 is added to it, the result is $72, not $6C as it would be in binary. Once the processor is set to operate in decimal mode, 1 is added to the value for seconds and the new value is tested to see if it equals $60. If it does, then seconds are set equal to 0 and 1 is added to the minutes value. The same thing is done for minutes and hours, only hours are checked for equality to $25. If you want twelve hour time rather than twenty-four set this test equal to $13.

Now we need to put the new time on the screen. You should all be aware by now that the APPLE takes an area of memory and displays this on the screen for text. Text page 1 is located at memory locations $400 to $7FF. We are going to bypass all the usual APPLE output routines and store the proper ASCII characters directly in memory so they appear on the screen exactly where we want them.

Internally the APPLE uses characters expressed in standard ASCII code but with the eighth bit set. If you try to put ASCII characters on the screen without the eighth bit set the characters appear in inverse video. This is just fine for our clock and will serve to differentiate the time from other printing on the screen. The program thus takes the bytes for hours, minutes, and seconds, breaks them up into two parts (tens and units digits), converts each digit to ASCII, and stores them in the screen area of memory. Finally colons are inserted between the hours-minutes and minutes-seconds to separate them.

Now we can turn our attention to the second part of the problem, initalizing the clock. A program called 'HCLOCK' which handles this is shown in listing 2. This program is compatible with either integer BASIC or Applesoft. The first thing the program does is load in the machine language routine using a series of pokes. When typing this in I recommend that you use the Monitor to enter the machine code into memory, verify that it is correct, and then use the 'COKE-POKER' program described on page 77 of the DOS 3.2 manual to convert it to a BASIC program. Line 230 modifies one byte of the machine code so that the clock may be put in any slot by changing the value for 'SLOT' in line 210.

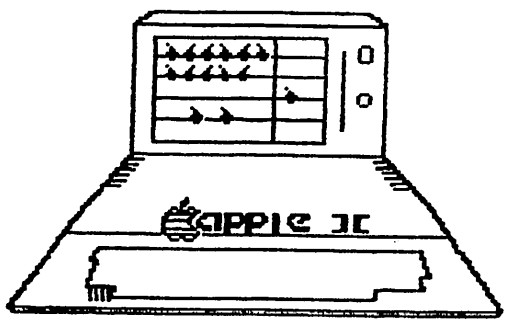
After loading the machine program HCLOCK sets the interrupt vector at $3FE and $3FF to $300. Also the screen scrolling area is modified so that the top line is left undisturbed. Since the time is going to be put in the top line we don't want it being scrolled off the screen everytime we print something. The pokes in lines 310 to 330 set the clock time to zero. When this program is first started the memory locations for the time will contain garbage, to avoid printing this on the screen they are zeroed first. The CALL 900 runs the two byte program attached to the end of the machine language program which clears the interrupt disable and allows the processor to respond to future interrupts.

At this point the 60 Hz should be connected to the clock which will then start interrupting the processor and the time will appear on the screen. Of course we don't have the correct time yet, but that is taken care of by the next few instructions. The user is prompted for hours, minutes, and seconds and in each case the number is converted to BCD and poked into the appropriate memory location.

As a final little goodie, the screen is erased and the program is wiped out by the 'NEW' command. If you know how to get the 'NEW' command into an integer BASIC program fine, if not use Applesoft.

Now that we have the clock running you will find that sometimes you don't want it on. The easiest way to turn it off is to hit RESET. This sets the interrupt disable bit so the processor will not respond to the interrupts, but the clock is still producing them. This brings us to a little problem. If after hitting RESET you want to get back to BASIC without disconnecting the DOS you do a 3D0G; unfortunately this also seems to clear the interrupt disable bit and you're back to having the clock run. Thus the only sure-fire way to stop the clock is disconnect the 60 Hz.

Well it's been fun, we found out how to build a simple clock and hopefully learned something about the interrupt structure of the APPLE. If I may inject a personal opinion here; one of the most important functions of a computer is to increase one's knowledge. To this end I prefer to do things myself rather than buy canned software or hardware. I also view a magazine or newsletter not so much as a source of knowledge but as a source of IDEAS that encourages the reader to extend his own horizons.



```
                1000 *********************************
                1010 *                               *
                1020 *    SOFTWARE CLOCK ROUTINE     *
                1030 *USING 1 SEC HARDWARE INTERRUPTS*
                1040 *                               *
                1050 *********************************
                1060 *
                1070 *
                1080 *    TIME IS STORED IN DECIMAL MODE
                1090 *    TWO DIGITS PER BYTE
                1100 *
                1110 *    THIS PROGRAM INCREMENTS THE TIME
                1120 *    AND DISPLAYS IT IN THE UPPER RIGHT
                1130 *    HAND CORNER OF THE SCREEN
                1140 *
                1150 *
                1160 DEVS .EQ $C0A0      SLOT 2
                1170 ACC  .EQ $45        MONITOR SAVES ACC HERE
                1180 *                   WHEN INTERRUPTED
                1190 SCRN .EQ $420       POSITION ON SCREEN
                1200 *                   FOR TIME TO APPEAR
                1210 *
                1220 SECS .EQ $390
                1230 MINS .EQ $391
                1240 HRS  .EQ $392
                1250 *
                1260 *
                1270 *    INTERRUPT COMES HERE
                1280 *
                1290      .OR $300
                1300 *
                1310 *    PULSE DEVICE SELECT LINE
0300- AD A0 C0  1320 TICK LDA DEVS
                1330 *
0303- F8        1340      SED            SET DECIMAL MODE
0304- 18        1350      CLC            CLEAR CARRY
0305- AD 90 03  1360      LDA SECS
0308- 69 01     1370      ADC #01        ADD 1 TO SECS
030A- 8D 90 03  1380      STA SECS       PUT BACK IN SECS
030D- C9 60     1390      CMP #$60       TEST = 60
030F- D0 2A     1400      BNE DISP       IF NOT, GOTO DISP
0311- A9 00     1410      LDA #0         SECS=60, STORE ZERO
0313- 8D 90 03  1420      STA SECS       IN SECS
0316- 18        1430      CLC            CLEAR CARRY
0317- AD 91 03  1440      LDA MINS
031A- 69 01     1450      ADC #01        ADD 1 TO MINS
031C- 8D 91 03  1460      STA MINS       PUT BACK IN MINS
031F- C9 60     1470      CMP #$60       TEST = 60
0321- D0 18     1480      BNE DISP       IF NOT, GOTO DISP
0323- A9 00     1490      LDA #0         MINS=60, STORE ZERO
0325- 8D 91 03  1500      STA MINS       IN MINS
0328- 18        1510      CLC            CLEAR CARRY
0329- AD 92 03  1520      LDA HRS        GET HOURS
032C- 69 01     1530      ADC #01        ADD 1 TO HOURS
032E- 8D 92 03  1540      STA HRS        PUT BACK IN HOURS
0331- C9 25     1550      CMP #$25       EQUAL TO 25
0333- D0 06     1560      BNE DISP       IF NOT, GOTO DISP
0335- A9 01     1570      LDA #1         HRS=25, STORE ONE
0337- 8D 92 03  1580      STA HRS        IN HOURS
                1590 *
033A- D8        1600      CLD            CLEAR DECIMAL MODE
                1610 *
                1620 *    THIS PART OF THE PROGRAM PUTS
                1630 *    THE TIME ON THE SCREEN
                1640 *
033B- AD 90 03  1650 DISP LDA SECS       GET SECONDS
033E- 48        1660      PHA            SAVE TEMPORARILY
033F- 20 7D 03  1670      JSR ASCI       MAKE IT ASCII
0342- 8D 27 04  1680      STA SCRN+7     PUT ON SCREEN
0345- 68        1690      PLA            RECOVER SECS
0346- 20 79 03  1700      JSR ASC        MAKE UPPER DIGIT ASCII
0349- 8D 26 04  1710      STA SCRN+6     PUT ON SCREEN
                1720 *
034C- AD 91 03  1730      LDA MINS       GET MINS
034F- 48        1740      PHA            SAVE IT
0350- 20 7D 03  1750      JSR ASCI       MAKE LOWER DIGIT ASCII
0353- 8D 24 04  1760      STA SCRN+4     PUT ON SCREEN
0356- 68        1770      PLA            RECOVER MINS
0357- 20 79 03  1780      JSR ASC        MAKE UPPER DIGIT ASCII
035A- 8D 23 04  1790      STA SCRN+3     PUT ON SCREEN
                1800 *
035D- AD 92 03  1810      LDA HRS        GET HOURS
0360- 48        1820      PHA            SAVE IT
0361- 20 7D 03  1830      JSR ASCI       MAKE LOWER DIGIT ASCII
0364- 8D 21 04  1840      STA SCRN+1     PUT ON SCREEN
0367- 68        1850      PLA            RECOVER HOURS
0368- 20 79 03  1860      JSR ASC        MAKE UPPER DIGIT ASCII
036B- 8D 20 04  1870      STA SCRN       PUT ON SCRN
036E- A9 3A     1880      LDA #$3A       PUT COLONS ON SCRN
0370- 8D 22 04  1890      STA SCRN+2
0373- 8D 25 04  1900      STA SCRN+5
                1910 *
                1920 *    RESTORE ACCUMULATOR
                1930 *    AND RETURN FROM INTERRUPT
                1940 *
0376- A5 45     1950      LDA ACC        ACCUM STORED BY MONITOR
0378- 40        1960      RTI
                1970 *
                1980 *    SHIFT ACC RIGHT 4 BITS TO GET
                1990 *    UPPER DIGIT INTO LOWER DIGIT
                2000 *    POSITION
                2010 *
0379- 4A        2020 ASC  LSR
037A- 4A        2030      LSR
037B- 4A        2040      LSR
037C- 4A        2050      LSR
                2060 *
                2070 *    NOW MASK OUT UPPER DIGIT AND
                2080 *    AND CONVERT TO ASCII
                2090 *
037D- 29 0F     2100 ASCI AND #$0F       KEEP 4 LSB
037F- 18        2110      CLC            CLEAR CARRY
0380- 69 30     2120      ADC #$30       ADD $30
0382- 60        2130      RTS
                2140 *
0383- 00        2150      BRK
                2160 *
                2162 *
                2170 *    CLEAR INTERRUPT DISABLE
```

```
                  2180 *
0384- 58          2190        CLI
0385- 60          2200        RTS
                  2210 *
                  2220 *
                  2230        .EN
```

```
SYMBOL TABLE

DEVS  C0A0   ACC   0045   SCRN  0420
SECS  0390   MINS  0391   HRS   0392
TICK  0300   DISP  033B   ASC   0379
ASCI  037D
```

```
:$300.385

0300- AD A0 C0 F8 18 AD 90 03
0308- 69 01 8D 90 03 C9 60 D0
0310- 2A A9 00 8D 90 03 18 AD
0318- 91 03 69 01 8D 91 03 C9
0320- 60 D0 18 A9 00 8D 91 03
0328- 18 AD 92 03 69 01 8D 92
0330- 03 C9 25 D0 06 A9 01 8D
0338- 92 03 D8 AD 90 03 48 20
0340- 7D 03 8D 27 04 68 20 79
0348- 03 8D 26 04 AD 91 03 48
0350- 20 7D 03 8D 24 04 68 20
0358- 79 03 8D 23 04 AD 92 03
0360- 48 20 7D 03 8D 21 04 68
0368- 20 79 03 8D 20 04 A9 3A
0370- 8D 22 04 8D 25 04 A5 45
0378- 40 4A 4A 4A 4A 29 0F 18
0380- 69 30 60 00 58 60
:
```

```
>LOAD HCLOCK
>LIST
  100 REM THIS PROGRAM LOADS IN THE
  110 REM MACHINE LANGUAGE CLOCK
  120 REM ROUTINE THAT COUNTS ONE
  130 REM SECOND INTERRUPTS FROM THE
  140 REM HARDWARE COUNTER.
  150 REM PRINTS THE TIME IN THE UPPER

  160 REM RIGHT HAND CORNER AND
  170 REM MODIFIES THE SCROLL WINDOW.
  180 REM
  190 REM SET SLOT FOR SLOT CLOCK IS I
      N
  200 REM CORRECTS MACH. CODE
  210 SLOT=2
  220 GOSUB 580
  230 POKE 769,128+SLOT*16
  240 TEXT : CALL -936
  250 REM SET INTERRUPT VECTOR
  260 POKE 1022,0: POKE 1023,3
  270 REM SET WINDOW TOP
  280 POKE 34,1
  290 CALL -936
  300 REM ZERO CLOCK
  310 POKE 912,0
  320 POKE 913,0
  330 POKE 914,0
  340 REM CLEAR INTERRUPT DISABLE
  350 CALL 900
  360 PRINT "TURN CLOCK ON NOW"
  370 PRINT
  380 REM SET CLOCK
  390 INPUT "HOURS   (1-24) ",T
  400 GOSUB 510
  410 POKE 914,T
  420 INPUT "MINUTES (0-59) ",T
  430 GOSUB 510
  440 POKE 913,T
  450 INPUT "SECONDS (0-59) ",T
  460 GOSUB 510
  470 POKE 912,T
  480 CALL -936
  490 NEW
  500 END
  510 REM CONVERT TIME TO BCD
  520 A=T/10
  530 T=T+A*6
  540 RETURN
  550 REM
  560 REM POKE MACHINE CODE TO
  570 REM COUNT INTERRUPTS
  580 POKE 768,173: POKE 769,160:
       POKE 770,192: POKE 771,248
      : POKE 772,24: POKE 773,173
      : POKE 774,144: POKE 775,3:
       POKE 776,105:
  590 POKE 777,1: POKE 778,141: POKE
      779,144: POKE 780,3: POKE 781
      ,201: POKE 782,96: POKE 783
      ,208: POKE 784,42: POKE 785
      ,169:
  600 POKE 786,0: POKE 787,141: POKE
      788,144: POKE 789,3: POKE 790
      ,24: POKE 791,173: POKE 792
      ,145: POKE 793,3: POKE 794,
      105:
  610 POKE 795,1: POKE 796,141: POKE
      797,145: POKE 798,3: POKE 799
      ,201: POKE 800,96: POKE 801
      ,208: POKE 802,24: POKE 803
      ,169:
  620 POKE 804,0: POKE 805,141: POKE
      806,145: POKE 807,3: POKE 808
      ,24: POKE 809,173: POKE 810
      ,146: POKE 811,3: POKE 812,
      105:
  630 POKE 813,1: POKE 814,141: POKE
      815,146: POKE 816,3: POKE 817
      ,201: POKE 818,37: POKE 819
      ,208: POKE 820,6: POKE 821,
      169:
  640 POKE 822,1: POKE 823,141: POKE
      824,146: POKE 825,3: POKE 826
      ,216: POKE 827,173: POKE 828
      ,144: POKE 829,3: POKE 830,
      72:
  650 POKE 831,32: POKE 832,125: POKE
      833,3: POKE 834,141: POKE 835
      ,39: POKE 836,4: POKE 837,104
      : POKE 838,32: POKE 839,121
      :
  660 POKE 840,3: POKE 841,141: POKE
      842,38: POKE 843,4: POKE 844
      ,173: POKE 845,145: POKE 846
      ,3: POKE 847,72: POKE 848,32
      :
  670 POKE 849,125: POKE 850,3: POKE
      851,141: POKE 852,36: POKE
      853,4: POKE 854,104: POKE 855
      ,32: POKE 856,121: POKE 857
      ,3:
  680 POKE 858,141: POKE 859,35: POKE
      860,4: POKE 861,173: POKE 862
      ,146: POKE 863,3: POKE 864,
      72: POKE 865,32: POKE 866,125
      :
  690 POKE 867,3: POKE 868,141: POKE
      869,33: POKE 870,4: POKE 871
      ,104: POKE 872,32: POKE 873
      ,121: POKE 874,3: POKE 875,
      141:
  700 POKE 876,32: POKE 877,4: POKE
      878,169: POKE 879,58: POKE
      880,141: POKE 881,34: POKE
      882,4: POKE 883,141: POKE 884
      ,37:
  710 POKE 885,4: POKE 886,165: POKE
      887,69: POKE 888,64: POKE 889
      ,74: POKE 890,74: POKE 891,
      74: POKE 892,74: POKE 893,41
      :
  720 POKE 894,15: POKE 895,24: POKE
      896,105: POKE 897,48: POKE
      898,96: POKE 899,0: POKE 900
      ,88: POKE 901,96:
  730 RETURN
  740 END
```

Washington Apple Digest

by Dave Efron

One of the useful services of a users group is the trading of information, to help members benefit from the experiences and knowledge of others. No single person can read every article written on a subject, nor can anyone be aware of everything written that may be of special interest. One of the attractions of a computer users' group is the opportunity to find short-cuts in the process of learning how to use computer equipment effectively, and this is an attraction shared by the experienced as well as the new users.

Oftentimes we pick up an old issue of a magazine and spot an article of interest which had earlier escaped notice. "if only I had seen this before!" often applies. Sometimes we pass up articles because as new uers we see no relevance in an item, until later when we realize the usefulness of the information to something we are now doing. Most often, however, we never subscribe to everything and we cannot find the time to review the contents of journals in the computer stores' racks.

A Call for A.I.D.

An Apple Information Digest could be a regular feature of the Washington Apple Pi Newsletter. The proposed concept would set as its objective the review of most (all, if possible) of the journals, magazines, newsletters, company-provided technical notes, books, and manuals that publish information on the Apple computer line and products designed for it.

A review would be a simple summary of the topics covered and the reviewer's evaluation of the article. It might also give a judgment of the article's appeal to different types of Apple users, for example, the novices, the pro's, the intermediates, the gamers, the scientists, etc. A few sentences would be enough, unless the reviwer desired to devote more attention to the article.

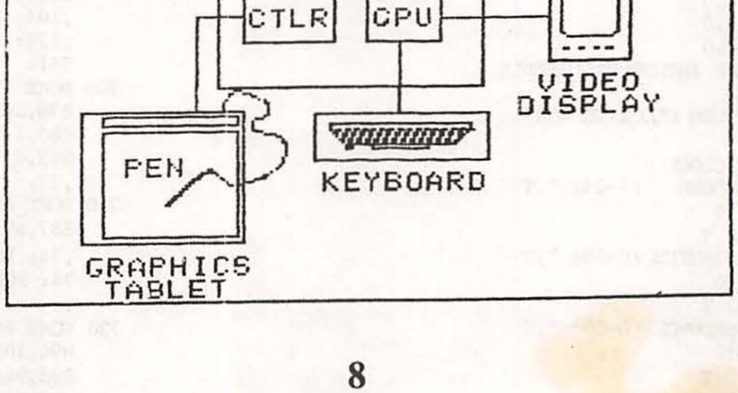
Readers would then know where to find technical or application information specific to individual interests. An extension of this service might be to re-print articles of wide interest in the Newsletter if we have the author's expressed permission to do so.

A Call for Volunteers

Feedback on this concept indicates that a project like this would be appreciated by many of our members, but it can work only if many of our members participate. A committee of reviewers is required to scan publications and abstract articles of interest to Apple users. One or more members would be assigned to review and prepare abstracts for each source every month. With enough interest in this project, there may be several volunteers per source who could share the effort by dividing the table of contents or alternating months.

Organization Meeting

A meeting to organize this project will be held at the regularly scheduled meeting of Washington Apple Pi, on June 28. Those who are interested in participating are encouraged to attend, but may indicate their interest by calling the club's telephone number instead (301 468-2305) to leave a recorded message. If the turn-out is good, this concept will turn into an on-going and useful service to our members.



Dealer's Corner

Apple Writer to Text File Conversion

Paul A. Sand
Computerland/Rockville

Have any of the following things ever happened to you?

- You want to create (or change or examine) a text file on disk that is to be read as data by a Basic program. But you don't necessarily want to write another program to do it.

- You want to create (or modify or examine) an EXEC file on disk, but (again) you don't want to write a program.

- You want to change the name of a variable or a line number reference everywhere it appears in a program, a laborious and error-prone task if done by hand.

- You want to put lower case letters into a program or data file.

- You want to put a program listing into the text of an article you are writing.

- You want to get a printed listing of a program (or data file) in other than "standard" format.

This article describes two small and simple Applesoft programs that work in conjunction with Apple Writer, the word processing software from the good folks at Apple Computer. One program (BTOT) will convert a file generated by Apple Writer into a normal sequential text file. The other (TTOB) performs the inverse operation, translating a text file into a file that can be read by Apple Writer. These two programs will allow you to do all the things mentioned above (and more) quickly and easily.

Text files are potentially powerful tools to solve programming problems. They are easily read and written by Basic programs. Basic program text can be saved and recovered from text file format as well. The main obstacle to wider use of text files is the limited availability of good utility software that will work with them. With the addition of these programs, Apple Writer becomes a powerful editor of program and data files in addition to its normal role of word processor. (Those interested in a more complete description of the things Apple Writer can do should refer to the excellent review by Phillip Wright in the February 1980 Apple Pi.)

In order to use Apple Writer to edit data files (or any sequential text file) the programs given here are used to translate the text files to and from Apple Writer format. The program TTOB will translate a text file named "XYZ" into the Apple Writer file "TEXT.XYZ". Similarly, BTOT will take the (possibly edited) Apple Writer file "TEXT.XYZ" and create the equivalent text file "XYZ". Of course, Apple Writer can also be used to create the text file initially.

The program offers the option of converting lower case in the Apple Writer file to upper case in the text file. This option should probably be accepted if one is editing a program text; it allows the program to be edited using lower case characters, which are easier to type into Apple Writer than upper case.

Here is the TTOB program:

```
rem ----------------------------------
rem Text to Apple Writer file converter
rem ----------------------------------

100 dim x%(127)
110 dc$ = chr$(4)
120 input "File to be converted?:"; fi$
130 input "Convert upper to lower case?(Y/N):"; an$
140 if an$ = "Y" then c5 = 128: go to 170
150 if an$ = "N" then c5 = -64: go to 170
160 go to 130
170 for i = 32 to 63
180   x%(i) = i + 192
190 next i
200 for i = 64 to 95
210   x%(i) = i + c5
220 next i
230 for i = 96 to 127
240   x%(i) = i + 96
250 next i
260 x%(13) = 141
270 poke 6400, 191: a = 6401
280 print dc$; "open "; fi$
290 print dc$; "read "; fi$
300 on err go to 340
310 get ic$
320 poke a, x%(asc(ic$))
330 a = a + 1: go to 310
340 if peek(222) <> 5 then print "Bad Error": end
350 print dc$; "close "; fi$
360 poke a, 96
370 l = a - 6399
380 print dc$; "bsave text."; fi$; ",a6400,l"; l
390 print l - 2; " characters"
400 end
```

This program has the option to convert upper case in the text file to lower case in the Apple Writer file in order to be compatible with the corresponding option in BTOT.

Note that neither program blows up if handed a character it doesn't know how to translate. An illegal character is transformed into an ASCII '0' (null) character by both programs.

Program editing with Apple Writer is only slightly more complex. A Basic program is first "captured" on disk as a text file (see below); then the text file is edited as above. The edited Apple Writer file is converted back to text, then EXECed back into memory as a program. (See the DOS 3.2 Manual, Chapter 7)

Apple Writer files are binary files loaded to and saved from memory starting at location 6400 (decimal). The first byte of the file is always a 191. The text follows, one byte per character. The last byte is 96, an end-of-file mark. "Legal" characters in Apple Writer are most of the printable ASCII characters plus carriage return. What makes the conversion non-trivial is that the text is not stored in ASCII format. But (fortunately) simple rules give the correct transformations - see the program listings.

Here is the BTOT program:

```
rem ----------------------------------------
rem Apple Writer to text file converter
rem ----------------------------------------

100 dim x%(255)
110 dc$ = chr$(4)
120 input "File to be converted?:"; fi$
130 input "Convert lower to upper case?(Y/N):"; an$
140 if an$ = "Y" then c5 = -128: go to 170
150 if an$ = "N" then c5 =  -96: go to 170
160 go to 130
170 for i = 0 to 31
180   x%(i) = i + 64
190 next i
200 for i = 32 to 63
210   x%(i) = i
220 next i
230 for i = 192 to 223
240   x%(i) = i + c5
250 next i
260 for i = 224 to 255
270   x%(i) = i - 192
280 next i
290 x%(141) = 13
300 print dc$; "bload text."; fi$
310 print dc$; "open "; fi$
320 print dc$; "delete "; fi$
330 print dc$; "open "; fi$
340 print dc$; "write "; fi$
350 a = 6401
360 ch = peek(a): if ch = 96 then 390
370 print chr$(x%(ch));
380 a = a + 1: go to 360
390 print dc$; "close "; fi$
400 print a - 6401; " characters"
410 end
```

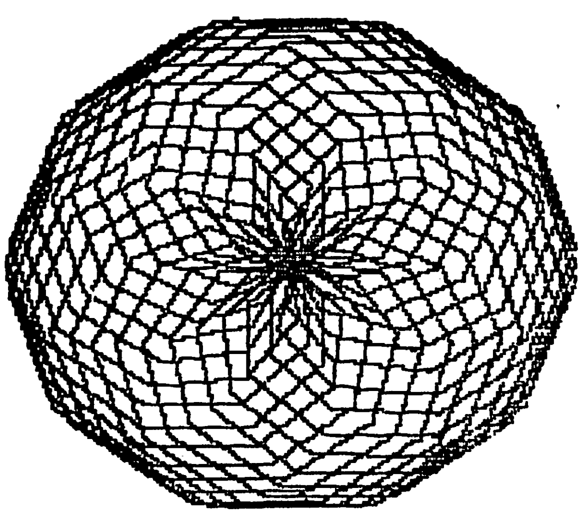
A third program might be useful to anyone attempting this system: the following can be stored as a text file under the name "CAPTURE":

```
rem ----------------------------
rem Capture program as text file
rem ----------------------------

0 d$ = chr$(4): input "Text file name:"; f$: print d$;
"open"; f$: print d$; "delete"; f$: print d$; "open"; f$:
print d$; "write"; f$: poke 33, 30: list 1, 63999: print d$;
"close"; f$: text: end
run
```

The purpose of this semi-program is to automatically save the Applesoft program currently in memory as a text file, the user only having to type "EXEC CAPTURE" and supply a name under which the file is to be saved. (This is essentially the process explained in the DOS 3.2 Manual pp. 76-77.) Despite appearances, everything from the "0" to the "end" statement is entered as one line. EXECing the CAPTURE file causes the program in memory to have a new line 0, which is then immediately executed by the "RUN" command. (The REM statements are ignored, of course.) The new line 0 asks the user for the file name and does all the work of actually storing the file. A version that would save both Integer and Applesoft programs as text files would be only slightly more difficult to write.

As another example of how these programs could be of benefit, consider the Apple owner who regularly uses his computer as a "dumb" terminal to access a timesharing system. A much more sensible (and economical) arrangement would be to let the Apple do some of the most time-consuming work. Software for transferring Apple text files to and from The Source is already widely available; a more general program for use with any timesharing system wouldn't be too difficult. For example, program source files could be prepared on the Apple using Apple Writer and later "uploaded" to the timesharing system for running. Or a section of a large data base on the big computer could be "downloaded" to the Apple for printing (again using Apple Writer) or processing by a Basic program.



BLAISE AWAY!

Dan Paymar meets M. Pascal:
Lower Case Input for Your Pascal Apple

by

Dr. Wo

Last month we talked about how to dump the hi-res screen to a Paper Tiger. We're not finished yet! However, the demand for an explanation of how to get lowercase input (!) using "shift-M" and the Paymar chip was more than could be ignored so I thought we would take that up this month. The discussion will serve as an introduction to the Apple BIOS.

* * *

Every implementation of UCSD Pascal requires an interpreter and a BIOS to support it. The interpreter translates P-code, the code emitted by the Pascal compiler, into the host machine's native code; and the BIOS, Basic I-O Subsystem, handles I-O to the devices connected to the system. The neat thing about the Apple's implementation is that both the interpreter and the BIOS can be rewritten by the programmer!

The Apple's interpreter and BIOS share a common 4K block of RAM addresses, \$D000 to \$DFFF, via memory swapping. The programmer can execute this swap by flipping soft switches at \$C083, "BIOSIN", and \$C08B, "BIOSOUT". One reference to BIOSIN swaps the BIOS in and two succesive references write enable it. Similarly, references to BIOSOUT sawp the interpreter into memory. Although you can modify both the interpreter and the BIOS, its likely you'll want to fool only with the BIOS. Documentation for it is available from Apple or from the club; the interpreter is proprietary and documentation is unavailable.

The interpreter and BIOS work together as follows: Normally, the interpreter is in memory. However, when a call for I-O is made from a Pascal program, the interpreter determines which device is being called, formats the outgoing or incoming data, and calls the BIOS. The BIOS is swapped in (and the interpreter out), a jump to the locations reserved for I-O through the selected device is made and the operation is performed. When all is done, the BIOS returns control to the interpreter.

When the BIOS takes over it knows that it is to read or write to a certain device. Before actually executing the operation, however, it first determines how the device is interfaced with the system. As currently configured, the interpreter-BIOS combination can recognize disks, printers, remote I-O devices and consoles, provided they are interfaced via Apple brand peripheral cards or "foreign" cards whose set-up coincides with an Apple card. (That's one reason your Pascal Apple doesn't recognize your D.C. Hayes Micromodem.) Fortunately, since you can rewrite the BIOS (within limits--space is at a premium) you can interface foreign cards.

Assuming the Apple recognizes the card you're using, it does one more thing before taking care of I-O: it polls the console keyboard via the routine "CONCK" located at \$D681. CONCK is the keyboard input routine and its here that our search for lower case input begins.

CONCK steps off by pushing the status of the machine and testing the keyboard for a character. If it finds none, the routine is over; if there is a character, the fun begins: If CONCK finds certain control characters (A,Z,F,S) it does its thing, such as flipping the pages of the screen, and exits. If it finds any other character, it adds it to the 78 character keyboard buffer. In the special case of control-K, it converts it to a left bracket. Whenever output to the console is requested, the console write routine, "CRWRITE", sends the appropriate character from the buffer to the screen, setting the upper case bit of the character on the way.

* * *

So how does any of this get us lower case input or output? There are three things we must do: install a Paymar chip; change the shiftmask so that characters are not automatically converted to upper case on their way to the screen; and patch into CONCK a program to allow case shifting on input via "shift-M". The secret to patching is to make use of a small block of free BIOS space located at $DABE to $DB7F inclusive (194 bytes).

(Randall Hyde referred to these addresses in his Apple Orchard article "Connecting with the UCSD BIOS" saying that "the free memory....is not really free." I've not found that to be true. Furthermore, his technique for lower case input requires a ROM+ board and rewiring the keyboard. I don't have such a board, and I'm messy with solder.)

"M" is one of only three alpha keys on the Apple that can actually be shifted. (Why?! Why? Why!) We use it here to set and reset bit 5 of location "SHIFTFLAG", which we have located at $DABE. Incoming characters are EORed with this byte to effect case shifting. You may recall that shift-M is used for "]" on a raw Apple. I simply gave that responsibility to control-J in my patch.

So..., the patch to CONCK, listed below as .PROC LOWINPUT, works as follows: CONCK is polled and entered just as above. If it finds no character the fun is over. However, if it does find a character, it immediately JMPs to LOWINPUT at $DABF. LOWINPUT checks to see if an alpha character, hex value at least 40, was found, shift-M included. If so, LOWINPUT jumps to SHIFTTEST and tests for the presence of shift-M. If present, bit 5 of SHIFTFLAG is set/reset and a JMP back into CONCK at "DONECK", $D71D, is made to finish off the console poll; if not, the program branches to SHIFIT where the input character is EORed with SHIFTFLAG, thus choosing between upper and lower case. After all this, LOWINPUT JMPs back into CONCK at the location "NOTFLUSH", $D706, to store the character in the console buffer. Note that only the alpha keys are affected by this routine; numeric and other non-alpha characters, except "@" and those mentioned below, are unaffected.

If a non-alpha character was found upon entering LOWINPUT, the program tests for 3 control characters used for special characters: control-K for left square bracket and its lower case counterpart, left curly bracket; control-J for right square bracket and curly bracket; and control-I for "¦" and "\". (You could change this set, add more characters if you like.) If one of these is found, the program branches to SHIFTIT; if not, it JMPs back into CONCK at "CONCK2", $D6AA, to continue testing for more control characters. Processing from this point continues as above.

* * *

My method for implementing LOWINPUT is to incorporate it into the procedure "SYSGEN" which is hosted by the Pascal program "startup" which is run atomatically at boot time by virtue of being stored in the SYSTEM.STARTUP file. SYSGEN is simple: It starts off by swapping in the BIOS and write enabling it. Then, it loads the program LOWOUT, which enables lower case output, followed by LOWINPUT. Finally, the program patches LOWINPUT and CONCK together, initializes the shift flag for uppercase input, swaps the interpreter into memory and returns to the Pascal calling program.

* * *

The effects of LOWINPUT are described in the following table:

	Bit 5 of SHIFTFLAG					
	Reset			Set		
Key	Alone	CTRL	Shift	Alone	CTRL	Shift
1!	1		!	1		!
2"	2		"	2		"
..	.		.	.		.
..	.		.	.		.
..	.		.	.		.
9)	9		)	9		)
0	0		0	0		0
:*	:		*	:		*
;+	;		+	;		+
..	.		.	.		.
..	.		.	.		.
..	.		.	.		.
/?	/		?	/		?
A	A		A	a		a
.	.		.	.		.
.	.		.	.		.
.	.		.	.		.
H	H		H	h		h
I	I	\	I	i	¦	i
J	J	]	J	j	}	j
K	K	[	K	k	{	k
L	L		L	l		l
M	M		<set>	m		<reset>
N↑	N		↑	n		~
O	O		O	o		o
P@	P		@	p		`
Q	Q		Q	q		q
.	.		.	.		.
.	.		.	.		.
.	.		.	.		.
Z	Z		Z	z		z

```
PAGE -    0
Current memory available:    10142
0000:                                  .ABSOLUTE
0000: D706                   NOTFLUSH  .EQU 0D706
0000: D71D                   DONECK    .EQU 0D71D
0000: DABE                   SHIFTFLAG .EQU 0DABE
0000: D681                   CONCK     .EQU 0D681
0000: D6AA                   CONCK2    .EQU 0D6AA
0000:
2 blocks for procedure code  9355 words left
```

```
PAGE -    1  LOWINPUT  FILE:SYSMODS


0000|                                      .PROC LOWINPUT
Current memory available:    9577
0000|                                      .ORG 0DABF
DABF|
DABF| C9 40                                CMP #40    ;ALFA CHARACTER?
DAC1| B0**                                 BCS SHIFTTEST ;YES. TEST FOR SHIFT M
DAC3|
DAC3| C9 0B                                CMP #0B       ;CONTROL-K?
DAC5| D0**                                 BNE NOTK
DAC7| A9 5B                                LDA #5B       ;L SQR BRACKET
DAC9| D0**                                 BNE SHIFTIT   ;ALWAYS TAKEN
DACB|
DAC5* 00
DACB| C9 0A                   NOTK         CMP #0A       ;CONTROL-J?
DACD| D0**                                 BNE NOTJ
DACF| A9 5D                                LDA #5D       ;R SQR BRACKET
DAD1| D0**                                 BNE SHIFTIT   ;ALWAYS TAKEN
DAD3|
DACD* 00
DAD3| C9 09                   NOTJ         CMP #09       ;CONTROL-I?
DAD5| D0**                                 BNE NOTI
DAD7| A9 5C                                LDA #5C       ;BACKSLASH
DAD9| D0**                                 BNE SHIFTIT   ;ALWAYS TAKEN
DADB|
DAD5* 00
DADB| 4C AAD6                 NOTI         JMP CONCK2
DADE|
DAC1* 00
DADE| C9 5D                   SHIFTTEST CMP #5D          ;SHIFT M?
DAE0| D0**                                 BNE SHIFTIT
DAE2| A9 20                                LDA #20
DAE4| 4D BEDA                              EOR SHIFTFLAG
DAE7| 8D BEDA                              STA SHIFTFLAG
DAEA| 4C 1DD7                              JMP DONECK    ;DONE CHECKING FOR
DAED|                                                    ;SPECIAL CHARACTERS
DAED|
DAE0* 00
DAD9* 00
DAD1* 00
DAC9* 00
DAED| 4D BEDA                 SHIFTIT      EOR SHIFTFLAG
DAF0| 4C 06D7                              JMP NOTFLUSH ;STORE CHARACTER
DAF3|                                                   ;IN CONSOLE BUFFER
DAF3|                                      .END
```

```
PAGE -    2  LOWINPUT  FILE:SYSMODS  SYMBOLTABLE DUMP


AB - Absolute      LB - Label      UD - Undefined      MC - Macro
RF - Ref           DF - Def        PR - Proc           FC - Func
PB - Public        PV - Private    CS - Consts


CONCK     AB D681|  CONCK2   AB D6AA|  DONECK   AB D71D|  LOWINPUT PR ----|
      NOTFLUSH AB D706| NOT   LB DADB|  NOTJ     LB DAD3
NOTK      LB DACB|  SHIFTFLA AB DABE|  SHIFTIT  LB DAED|  SHIFTTES LB DADE|
```

```
PAGE -    1  SYSGEN     FILE:SYSGEN

0000:                                     .PROC SYSGEN
Current memory available:    9617
0000:                          ;
0000:                          ;
0000:                          ;--------------------------------------
0000: C083                     BIOSIN    .EQU 0C083
0000: C08B                     BIOSOUT   .EQU 0C08B
0000: D681                     CONCK     .EQU 0D681
0000: D6AA                     CONCK2    .EQU 0D6AA
0000: D8E8                     LOWOUT    .EQU 0D8E8
0000: DABF                     LOWIN     .EQU 0DABF
0000: DABE                     SHIFTFLAG.EQU 0DABE
0000: D6A4                     PATCH     .EQU 0D6A4
0000:
0000: AD 83C0                            LDA BIOSIN
0003: AD 83C0                            LDA BIOSIN
0006:
0006: A0 00                              LDY #00
0008: B9 ****                  XLOWOUT   LDA PRG1,Y
000B: 99 E8D8                            STA LOWOUT,Y
000E: C8                                 INY
000F: C0 02                              CPY #02
0011: 90F5                               BCC XLOWOUT
0013:
0013: A0 00                              LDY #00
0015: B9 ****                  XLOWIN    LDA PRG8,Y
0018: 99 BFDA                            STA LOWIN,Y
001B: C8                                 INY
001C: C0 34                              CPY #34
001E: 90F5                               BCC XLOWIN
0020:
0020: A0 00                              LDY #00
0022: B9 ****                  XPATCH    LDA PRG9,Y
0025: 99 A4D6                            STA PATCH,Y
0028: C8                                 INY
0029: C0 03                              CPY #03
002B: 90F5                               BCC XPATCH
002D:
002D: A9 00                              LDA #00
002F: 8D BEDA                            STA SHIFTFLAG       ;INITIALIZE
                                                             ;SHIFTFLAG
0032:
0032:
0032: AD 8BC0                            LDA BIOSOUT
0035: 60                                 RTS
0036:
0009* 3600
0036: B0 02                    PRG1      .BYTE 0B0,02        ;BCC *+4
0038:
0016* 3800
0038: C9 40                    PRG8      .BYTE 0C9,40        ;CMP #40
003A: B0 1B                              .BYTE 0B0,1B        ;BCS SHIFTTEST
003C: C9 0B                              .BYTE 0C9,0B        ;CMP #0B
003E: D0 04                              .BYTE 0D0,04        ;BNE NOTK
0040: A9 5B                              .BYTE 0A9,5B        ;LDA #5B
0042: D0 22                              .BYTE 0D0,22        ;BNE SHIFTIT
```

```
PAGE -    2  SYSGEN     FILE:SYSGEN

0044: C9 0A                              .BYTE 0C9,0A        ;CMP #0A
0046: D0 04                              .BYTE 0D0,04        ;BNE NOTJ
0048: A9 5D                              .BYTE 0A9,5D        ;LDA #5D
004A: D0 1A                              .BYTE 0D0,1A        ;BNE SHIFTIT
004C: C9 09                              .BYTE 0C9,09        ;CMP #09
004E: D0 04                              .BYTE 0D0,04        ;BNE NOT I
0050: A9 5C                              .BYTE 0A9,5C        ;LDA #5C
0052: D0 12                              .BYTE 0D0,12        ;BNE SHIFTIT
0054: 4C AA D6                           .BYTE 4C,0AA,0D6 ;JMP CONCK2
0057: C9 5D                              .BYTE 0C9,5D        ;CMP #5D
0059: D0 0B                              .BYTE 0D0,0B        ;BNE SHIFTIT
005B: A9 20                              .BYTE 0A9,20        ;LDA #20
005D: 4D BE DA                           .BYTE 4D,0BE,0DA ;EOR SHIFTFLAG
0060: 8D BE DA                           .BYTE 8D,0BE,0DA ;STA SHIFTFLAG
0063: 4C 1D D7                           .BYTE 4C,1D,0D7  ;JMP DONECK
0066: 4D BE DA                           .BYTE 4D,0BE,0DA ;EOR SHIFTFLAG
0069: 4C 06 D7                           .BYTE 4C,06,0D7  ;JMP NOTFLUS
006C:
0023* 6C00
006C: 4C BF DA                 PRG9      .BYTE 4C,0BF,0DA ;JMP LOWIN
006F:
006F:                                    .END
```

```
PAGE -    3  SYSGEN     FILE:SYSGEN   SYMBOLTABLE DUMP

AB - Absolute      LB - Label       UD - Undefined      MC - Macro
RF - Ref           DF - Def         PR - Proc           FC - Func
PB - Public        PV - Private     CS - Consts

BIOSIN    AB C083:  BIOSOUT  AB C08B:  CONCK     AB D681:  CONCK2   AB D6AA:  LOWIN     AB DABF: LOW
UT    AB D8E8:  PATCH     AB D6A4
PRG1      LB 0036:  PRG8     LB 0038:  PRG9      LB 006C:  SHIFTFLA AB DABE:  SYSGEN    PR ----: XLO
IN    LB 0015:  XLOWOUT  LB 0008
XPATCH    LB 0022:
```

```
PROGRAM STARTUP;
PROCEDURE SYSGEN;EXTERNAL;
BEGIN
 SYSGEN;
 PAGE(OUTPUT);
 GOTOXY(0,5);
 WRITELN('WELCOME TO DR. WO''S CUSTOMIZED APPLE');
 WRITELN('FEATURING LOWER CASE INPUT AND OUTPUT');
 WRITELN;
 WRITELN('USE "SHIFT-M" TO SHIFT CASES');
 WRITELN;
 WRITELN('PLEASE USE THE FILER TO SET THE DATE!');
END.
```

YOUR AD HERE!

RATES

RATES	$30	full
	$15	half
	$10	quarter
	$6	eighth

(line copy only - no half-tones or colors)

WASHINGTON APPLE PI
MAIL ORDER FORM

Washington Apple Pi now has a program library, and disks are available for purchase by anyone. The price to members is $5.00 per disk, and $8.00 to non-members. These disks are chock full of exceptional programs - the utilities are especially useful. The games are some of the best - not just simple and uninteresting ones. You may pick them up at any meeting or have them mailed for $2.00 per disk additional. They will come in a protective foam diskette mailer.

Also available for purchase by members at a discount price is the new APPLE II REFERENCE MANUAL (replaces the Red Reference Manual). The price of this manual is $17.00. You may pick it up at a meeting or have it mailed to you at no extra charge.

Amount

1. New APPLE II REFERENCE MANUAL - $17.00 each ------

2. PROGRAM DISKETTES
   Members: $5.00 per disk picked up at meeting
   $7.00 mailed to you...
   Non-members: $8.00 per disk picked up a meeting
   $10.00 mailed to you...

Volume 1--Utilities I ()
Volume 2--Utilities II ()
Volume 3--Games I ()
Volume 4--Games II ()
Volume 5--Games III ()
Volume 6--Games IV ()
Volume 7--Games V ()
Volume 8--Utilities III ()
Volume 9--Educational I ()
Volume 10-Math/Science ()
Volume 11-Graphics I ()
Volume 12-Games VI ()
Volume 13-Games VII ()
Volume 14-IAC Utilities IV ()
Volume 15-Games VIII ()
Volume 16-Utilities V ()
Volume 17-Graphics II ()
Volume 18-Education II ()
Volume 19-Communications ()
Volume 20-Music ()
Volume 21-Apple Orchard ()

TOTAL ORDER = $ ------

Check here if you want these shipped---

NAME --------------------------------------

ADDRESS --------------------------------------

CITY, STATE, ZIP --------------------------------------

TELEPHONE --------------------------------------

Membership No.(1st three digits after WAP on mailing label) ---------

Make checks payable to "Washington Apple Pi"

Send order to: Washington Apple Pi- ATTN: Librarian
PO Box 34511
Washington, DC 20034