WASHINGTON APPLE PI

Vol. 1 No. 5
June, 1979

Dear APPLE Enthusiast,

I would like to express our thanks to Bernie Urban for all the work that he put in as our "first president." He very unselfishly put in much time and effort into getting the club up and running. In a less serious frame, Bernie - don't rest on your laurels, we still need you!

We've had our first official elections now. I'm sure I can speak for all the newly elected officers when I extend our appreciation for the trust that you've given us. However, I hope you still feel that this is your club. We can only be of service in so far as each of you participate and are satisfied with what you get from the club.

I feel that this is only just a beginning. Good things start small and grow. I see a number of areas for us to grow in:

(1) Putting Washington Apple Pi on a sounder financial footing. (This means some form of dues and possible ads in the newsletter, cake sales? whatever....).

(2) Formation of special interest groups, either formally or informally depending on how many people are interested. These groups could address areas such as: hardware modifications, programming languages, educational applications, business applications, telecommunications (Applenet?), graphics and whatever else that members are interested in doing. Our diversity is our strength here.

(3) Establishing a club library of programs and documentation. This should include some form of standards so that for disk files, programming techniques, etc., each of us does not have to re-invent the wheel each time we do some application.

(4) Continuing to help the newsletter mature into an interesting, current, relevant and professional (?) record of the club's activities, of members' articles, and of shared information.

Looking forward, I see a year filled with potentials, challenges, and opportunities. I see us changing to meet our interests in ways we don't yet know. -- See you at the next meeting.

John L. Moon

DISK OPERATING SYSTEM (DOS) NOTES, By Sandy Greenfarb

INTRODUCTION: One thing this is not, is the ultimate reference on DOS. All I've done was condense my own research into a short article with hopes that it might provide some assistance to others about to go the same route and possibly spur someone else to present his or her efforts in writing.

REFERENCES: WOZPAK, "Using RWTS Routine", by Steve Wozniak.
CALL-A.P.P.L.E., Mar 79, "Disk Access Utility", by Dan Paymar.
CALL-A.P.P.L.E., Apr-May 79, "Apple Disk Operating System", by Richard F. Suitor.

GENERAL: All number values are decimal unless otherwise stated. Following the Apple standard, hexadecimal numbers are preceded with a dollar sign ($). For new computer journeymen (Since you own an APPLE II, you have been raised above the category of novice.), I'll explain a little about addresses. In most computer situations, the first item is numbered 0 and succeeding items are numbered relative to the first. Thus, the first byte of an Apple is at address 0 and the first track on a diskette is number 0. In fact, if you've done any simple integer basic programming with a DIM, you've used this addressing without knowing it. The statement DIM A(3) does not say to reserve an array of size 3, but says starting with A(0), reserve an array whose last address is A(3). Try equating and then printing. You'll find that A(0), A(1), A(2), and A(3) are all legitimate members of the array. Remember this type of addressing. The deeper you get into computers, the more you'll find it.

Also, be aware of the standard internal representation of addresses in low byte/high byte format. In a two-byte machine address, the first byte plus 256 times the second byte value equals the actual machine address being referenced. The deeper you get, the more you'll see this also.

DISKETTE: Each diskette may be thought of, as a phonograph record except that the "grooves" of these records are unconnected. In computer terms, the grooves are called tracks. Logical portions of tracks are called sectors. As the disk is spinning, the Apple can locate and position itself and read or write one specific sector of one specific track at a time, similar but not identical to a magnetic tape read/write operation.

One diskette contains 35 tracks numbered 00 thru 34.
One track contains 13 sectors numbered 00 thru 12.
One sector contains 256 bytes.
403 sectors per diskette available for user files.

DOS reserves tracks 00, 01, and 02 for itself and reserves track 17 for maintaining the DOS catalog. This explains why, even though there are 455 sectors on a diskette, only 403 are available.

FILES: Regardless of type, any user file established under DOS consists of three distinct parts; catalog entry, pointer sector, and data sectors. These are described later.

CATALOG: Track 17 of a DOS diskette is reserved for the system catalog. Seven catalog entries, file names with associated data, can be stored in each of 12 sectors starting with sector 12 and working down thru sector 1 as needed. NOTE: 12x7=84, the maximum number of files per diskette.

SECTOR MAP: In track 17, sector 0 is an essential part of the catalog system I call the sector map. Starting with byte addresses 56/57 (the 57th/58th bytes of the sector) and ending with bytes 192/193, every other pair of bytes describes which sectors in each track are still available to DOS for writing. The first pair (56/57) represent track 0. Incrementing by 4, the next pair (60/61) represent track 1, and so on thru byte pair 192/193 which represents track 34. From left to right, the first 13 bits of each of these byte pairs indicate the status of a sector. If the bit is 0, the sector is already in use and, if the bit is 1, the sector is available for writing. This is best explained by example: Let bytes 76/77 = $07F8. Track number represented = ADDRESS of first byte of pair divided by 4 minus 14. (76/4-14=5). This represents track 5. Break the hexadecimal value down to binary and label the first 13 bits from 12 to 0. Hexadecimal $07F8 becomes binary 0000 0111 1111 1000.

```
 0  0  0  0  0  1  1  1  1  1  1  1  0  0  0
12 11 10 09 08 07 06 05 04 03 02 01 00 N/A....
```

This shows that in track 5 that sectors 12, 11, 10, 9, and 8 are already being used by DOS. Sectors 7 thru 0 are still available for writing.

CATALOG ENTRY: A catalog entry, reference to a file stored under DOS, is exactly 35 bytes long. The catalòg begins in sector 12 of track 17 and continues downward thru sector 1, as more entry space is needed. Each sector can hold up to 7 entries. The first entry begins in byte 11 of each sector being used. Succeeding entries occur in 35 byte increments until the sector is filled. Following is the catalog entry format:

Byte 00 - Address of track for pointer record.
Byte 01 - Address of sector for pointer record.
Byte 02 - File type according to the following table:

TYPE FILE	If file is unlocked	If file is locked
(T) Text..........	$00	$80
(I) Integer Basic.	$01	$81
(A) Applesoft.....	$02	$82
(B) Binary........	$04	$84

Bytes 03/32 - File Name
Byte 33 - Number of sectors in file (includes 1 for pointer record.)
Byte 34 - zeros (possibly not used).

The file name is a maximum of 30 characters, internally left justified. Any unused characters to the right of your assigned file name will be filled with spaces, $A0.

POINTER RECORD: The catalog entry points to the pointer record. The pointer record, in turn, points to every data record of the file. Starting in byte pair 12/13 and continuing in sequence thru 1 less than the number of sectors referenced in the catalog, is a track/sector pointer to each data sector of the file.

DATA RECORDS: Data records or sectors contain the actual data or program of the file. Depending on the type of file, the first several bytes may contain additional control information.

For Integer Basic and Applesoft files, the first two bytes contain the actual program length to be loaded. This byte count is in the standard low/high format. Anyone taking the trouble to examine an Applesoft file will observe that the count appears too high by 2. They should also observe two extra zero bytes at the end of the file. The Applesoft structure does require these bytes and they do belong as part of the file.

For a binary file, the first pair of bytes of the first record provide the default BLOAD address. The second pair represents the file length.

In text files, the first byte of the file is the first character of the file, and so on.

EXPERIMENTS: In producing these notes, I ran several experiments. Some reached worthy conclusion; others did not. I will relate some which might have significance.

1. The first pointer record is assigned to sector 12 of track 18. The first data record will begin in sector 11 of track 18 and continue downward, continuing in sector 12 of track 19 and so on. Once it reaches track 34, DOS will pick up in track 16 and work down thru track 3 (tracks 0-2 reserved). It will then pick up its search for more space beginning again in track 18. Initially, pointer records are assigned to sector 12 of the next unused track, with a starting point of the last sector assigned by DOS according to the rotating algorithm. If a previous file had been deleted, its space would still be ignored until the search had circled back to the empty space. Not conclusively, but it appears that DOS will always search out the higher numbered sectors as first consideration for pointer records.

2. Examining the diskette after deleting files brought some significant facts to light. Once DOS declares a sector as written in the sector map, only a DELETE command (or INIT of course) reverses the status. Thus, if one should modify a stored program and the new program would be smaller, the preferred action would be to DELETE the old and then save the new. I do mean, of course, this would only be done if the same name were retained.

3. Have you ever wondered why your last-saved DOS file is not last in the list. When a DELETE command is issued, DOS will take the catalog entry and alter it. Byte 32 will become the value of byte 0 and then byte 0 will become $FF. Hypothetically byte 32 is altered to prevent file name mismatch and byte 0 is altered to tell DOS that the entry space is again available. Also, as all catalog entry space is initialized to zero, it appears DOS knows it has reached the end of the catalog at the end of the assigned space or a string of zeros, whichever comes first.

4. The final experiment deals with writing a sector independent of the DOS file system. Some background first. Build within DOS is a utility entitled "READ/WRITE TRACK/SECTOR" or RWTS, for short. This is an article in itself. To continue, I set up the program and parameters necessary to write a sector, specifically an unused one which still showed as a "1" on the available sector map. I examined the track and sector map after my write operation and noted that the data was there and that the sector map remained unchanged. The impact of this I leave as a warning to RWTS users.

LETTING IT ALL HANG OUT (THE GAME I/O,THAT IS)

by Susan Eickmeyer

Have you ever been the victim of the APPLE's game I/O? Like maybe you've been happily doodling with the light-pen or game paddles, and then decided to run a program that needed your joystick--you opened up the lid (after taking off any number of pieces of junk or peripherals which were lying on top of the lid) and carefully removed the 16-pin plug, then straightened out the three pins you bent in the process, grabbed your joystick, or whatever, and with inadequate light summoned your last bit of patience trying to al ign the new plug to the invisible black holes in the socket, and maybe even succeeding the first try. after all of this you replaced the lid, the junk and whatever, and finally were able to run your next program. You probably thought that APPLE could have come up with something better. They didn't, but you can, and to prove that it can be done by just about any clumsy kid, I even did it myself. Fortunately, I didn't need to do all the thinking myself, since my knowledge of hardware is extremely limited. I got a lot of help from Joe Zakar (my boyfriend), who explained a lot of the workings of the various pins etc. to me. So, what I learned, I'll try to pass on.

First, if you don't want to read the rest of this article, just turn in the Red Reference Book to the page where it explains the game I/O pin. If you can understand it all, then this article won't help you much, anyway. If you can't understand any of the info, except where the socket is located, then you're on my level.

The first thing to realize, is that the paddles, and just about any other peripheral, don't use all the pins on the plug. Most of the pins aren't even wired. That brings up the question of why it is necessary to tie up all your pins, if you only need three? If you have a connector-box out, it isn't.

First we'll look at the pins individually, or in related groups. We'll examine what each of these types of pins or groups does, and then explain what you need to bring the socket out.

The first two pins we'll look at are pins 9 and 16. They are easy to explain, because they aren't hooked up to anything--no-connection. Hence, in case you haven't figured it out, they don't do anything, either, maybe APPLE planned them for extending to a write-only memory. If you're following me so far, we'll rush headlong into two other pins, pin 1 and pin 8. Pin 1 is your connection to 5 volts positive DC. This is one of several voltages produced by your APPLE's power supply. This voltage is what will be utilized by the game paddles to tell the APPLE what's going on. Pin 8 is ground. It is utilized, among other things, by the switches and annunciator connections.

Now that we're down to 12 pins, I'll probably start getting tough on you. The next set of pins are 6,7,10 and 11. These are the PADDLE INPUTS. Finally we get to something useful. As soon as I can get Joe to explain it to me, I'll tell you how the computer handles the paddle inputs, but for now, let's just say that if you have a variable resistance, otherwise known as a potentiometer, or "pot" which is hooked up to the pin 1 (5V+) and to one of the game paddle pins, the computer will read the resulting current flow as a number. That gives you your paddle reading when you ask for it, say in BASIC. This forms our first real connection for the box, the paddles, each of which needs a 5v connection, and a pin connection.

All this is to say, that to run paddle one, not counting the button, all the APPLE needs is pin 10 and 5v+ , or pin 1. If you wanted to connect 4 paddles, you need just 5 pins, the four pins which connect to the paddle locations in the APPLE, and again, 5v+.

The switches, or buttons on your APPLE-supplied paddles, are handled by three pins on the game I/O connector. These are pins 2,3,and 4. To run a button/switch, you need to use one of these pins, and ground. These pins are connected, roughly speaking, to addresses in memory,$C061, $C062, $C063. when your button is pushed, it causes the high bit to be set at this location, and results in a number larger than 127. (That's because the binary form of the number at the address has eight digits, numbered from 0 to 7, going from right to left. the highest number that can be expressed without using the leftmost digit is 127, or 0111111 binary. 10000000 is 128 decimal. That leftmost digit is the 'high bit').

For you people who like blinking lights, you can control them by using the annunciator pins 12, 13, 14, 15. I personally have never used them for anything, but as soon as something good comes up, I'll let you know. Or maybe you'll let me know. In any case, I will assume that you may want to control little lights or some such thing in the future, and we will bring the annunciators out into the box, too. They require connections to the pin, and, to be usable, to ground.

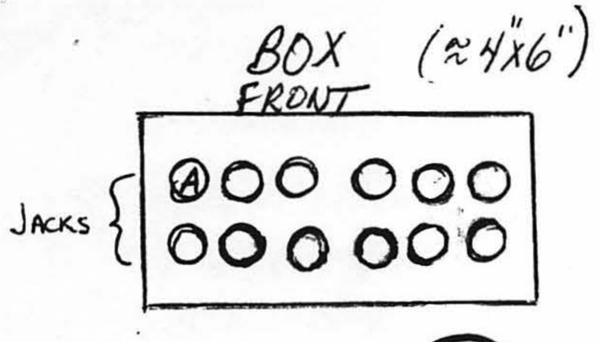
That leaves us just the pin 5. I am not going to bring it out into the box as a separate jack, but it will still be accessible from the box in one of the 2 16 pin sockets. A strobe, put simply, just causes a signal, normally at a set level, to dip for just a clock cycle whenever the associated memory location (C04X) is accessed. This may be used to trigger other hardware logic, or act like an interupt.

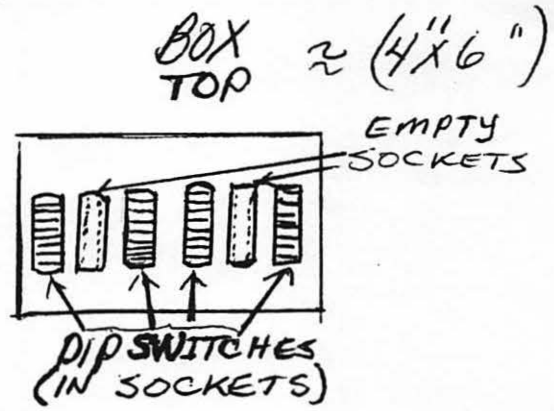
NOW, the box! To build the box you need a box of some kind. I used a 3X5 card file box. You will also need the following, most of which you can get at Radio Shack.

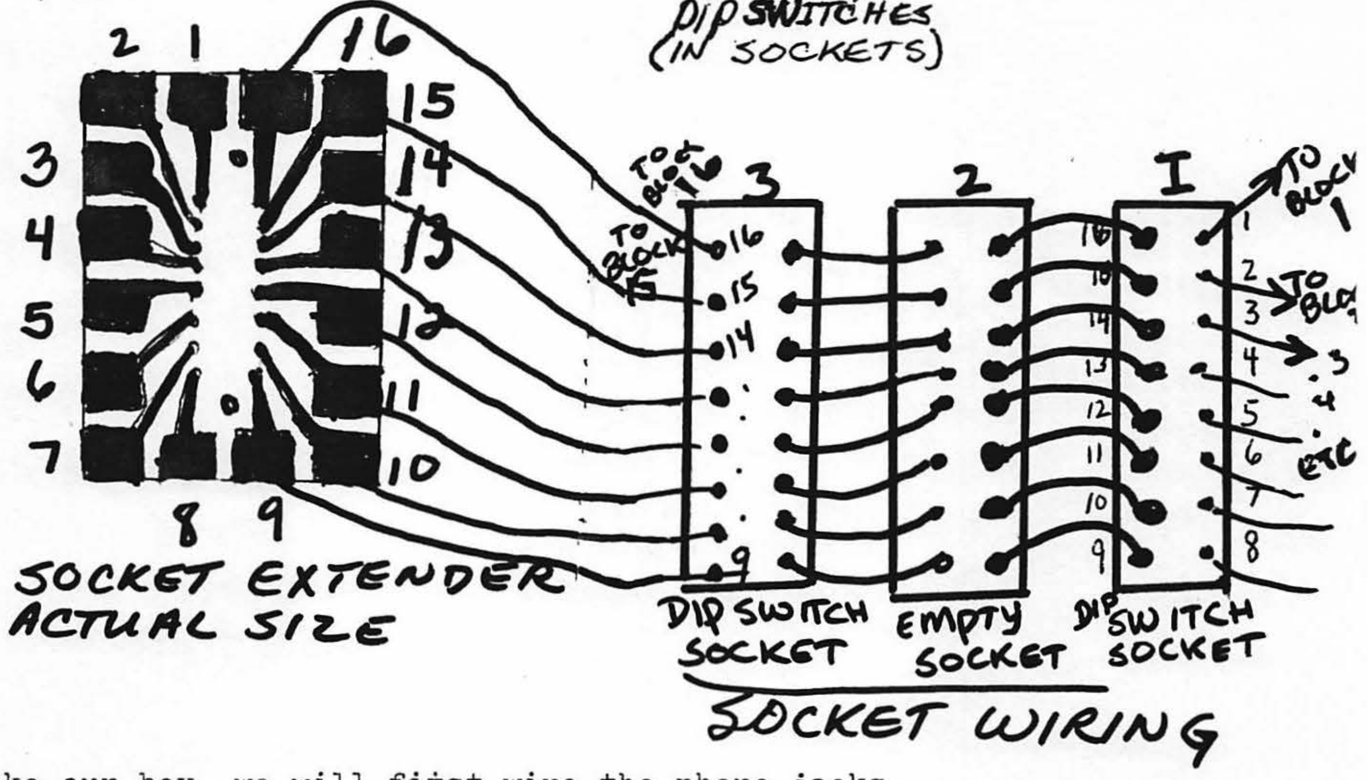
--- 12 phono jacks (I used 3 sets of 4 mounted jacks-RCA phono type)
--- 7 16-pin dip sockets
--- 1 extender board for the dip socket (to make soldering easier)
--- 4 rocker-type dip switches, 8 rockers each (also 16 pin)
--- a piece of perf board with .1 holes
--- wire, preferably 26-28 guage
--- solder
--- 1 18-24 inch extender ribbon with 16 pin plugs on both ends
--- patience (not available at Radio Shack)

Each of the 12 jacks will have 2 connections as follows (the 'name' of the jack is followed by the two connectons given as pin numbers)

5V: 1/8	PDL0: 6/1	AN0 15/8
SW0: 2/8	PDL1: 10/1	AN1 14/8
SW1: 3/8	PDL2: 7/1	AN2 13/8
SW2: 4/8	PDL3: 11/1	AN3 12/8







To make our box, we will first wire the phono jacks.
-Wire together all jack terminals which will go to ground
-Wire together all jack terminals which will go to 5v
-Bring one wire out from the grounded terminals to extender 'block' #8
-Bring one wire out from the 5v terminals to extender 'block'#1
-wire the remainder of the terminals to the blocks as indicated above
  (the first number indicates the inner terminal of the jack)
-you will now wire the six sockets for 16 pin extensions as follows:
  -place the six sockets into the perfboard. tape them in place if necessary, since you will now turn the board over so the pins are on top. Number the sockets RIGHT TO LEFT FROM 1 to 6.
  -on each socket, the pins are numbered as follows:on the right side from top to bottom 1-2-3-4-5-6-7-8, and on the left side from bottom to top 9-10-11-12-13-14-15-16. My convention from here on out is to refer to socket 6, pin 8 as 6/8. Blocks on the extender will be *8, and a wire connection noted as __. Thus to say that block 16 is wired to socket six,pin 16, I will simply type *16__6/16

THE CONNECTIONS ARE AS FOLLOWS:

*1__1/1	1/16__2/1	2/16__3/1	*16__3/16
*2__1/2	1/15__2/2	2/15__3/2	*15__3/15
*3__1/3	1/14__2/3	2/14__3/3	*14__3/14
*4__1/4	1/13__2/4	2/13__3/4	*13__3/13
*5__1/5	1/12__2/5	2/12__3/5	*12__3/12
*6__1/6	1/11__2/6	2/11__3/6	*11__3/11
*7__1/7	1/10__2/7	2/10__3/7	*10__3/10
*8__1/8	1/9 __2/8	2/9 __3/8	* 9__3/9

*1__ 4/1	4/16__5/1	5/16__6/1	*16__6/16
*2__ 4/2	4/15__5/2	5/15__6/2	*15__6/15
*3__ 4/3	4/14__5/3	5/14__6/3	*14__6/14
*4__ 4/4	4/13__5/4	5/13__6/4	*13__6/13
*5__ 4/5	4/12__5/5	5/12__6/5	*12__6/12
*6__ 4/6	4/11__5/6	5/11__6/6	*11__6/11
*7__ 4/7	4/10__5/7	5/10__6/7	*10__6/10
*8__ 4/8	4/9 __5/8	5/9 __6/8	*9 __6/9

This is the essential wiring. The ribbon extender plugs into your apple and into the socket in the socket extender. The dip switches can be used to turn the empty sockets on or off, so if you have your game paddles plugged into one, and your lightpen into the other, the APPLE wont get confused. You will have to pull the plugs on the sockets if you aren't using them, no big deal, since that's exactly what jacks are meant for.

For all of you who haven't the least idea how to build this thing after reading this mess, just come to the next meeting, and I'll have my box there for you to looka at it.

CONVERTING PIMS FOR THE APPLE II DISC by Nicholas B. Cirillo, M.D. (NOVAPPLE)

As an absolute computer novice, the six months that I have owned my APPLE II have been full of learning experiences. Since my interests are in business applications, I found myself looking for a good way to learn data base management concepts. I purchased some Osbourne books but they had little that I could learn at my level of background. Then I came upon PIMS (Personal Information Management System) by Madan L. Gupta, Scelbi Publications, Milford, Conn., at the recommendation of Tom Bowen at the Computer Hardware Store.

Although written for the TRS-80 with application to the PET 2001, it looked as though I could make the conversion to the APPLE. I was well on the way to converting the program to tape storage on the APPLE, using the string store routine in CONTACT 3, when my disk arrived. (Perhaps I will complete that conversion and make it the subject of a future article.) It initially seemed as though the conversion would be simple, but in retrospect the "fixes are obscure enough to share them.

It is worthwhile to discuss the way this program arranges data for storage. The data are concatenated into long strings that are stored as units of a sequential file. Spacers are appropriately placed in the compiling process and then used to break the strings apart in a parsing routine after reloading. The first problem that arose on my system was that the spacer, CHR$(126) - the "less than" symbol in the second ASCII series, would return the upward arrow on the READ routine. This was corrected by changing all the CHR$(126) to CHR$(94), which is the upward arrow in the first ASCII series. These changes will be noted in order below. Some additional changes were noted in a recent Kilobaud article and will be referenced.

Others were in an errata printed in my copy of PIMS and are also referenced for completeness. There will be a complete LOAD and SAVE listing, since that is where most of the conversion occurs.

CHANGES

70 PRINT CHR$(4); "NOMON I, C, O" (Note: Leave this line out at first and watch how the routine works)
80 HOME
90 D$=" ": REM CONTROL D IN QUOTES (Note: Deletes TRS-80 ONERRGOTO routine)
250 PRINT I
370 T$ = T$ + ";" + T1$ (Note: This takes care of DOS recognizing the comma as a separator. This was a major bug. Found it, thanks to Bill Kennedy.)
960 change CHR$(126) to CHR$(94)
990 T$ = N$ + T$ + CHR$(94) (Note: This is in the errata)
1210 GOTO 1330 (Note: The original line reference did not appear that it would work)
1260 GOTO 540 (Note: from Kilobaud article)
1440 change CHR$ (126) to CHR$(94)
Add 1675 T1$ = " " (Note: from Kilobaud article)
1790 change CHR$ (126) to CHR$(94)
2260 it is the letter O (Note: from the errata)
2370 change GOSUB to GOTO (Note: from the errata)
2450 GOTO 540 (Note: from Kilobaud article)
2710 change CHR$(126) to CHR$(94)
Add 3115 PRINT D$; "MON I, C, O" (See note to line 70)
Delete lines 3430 and 3440 unless you want to write an APPLESOFT II ONERRGOTO routine.

LOAD ROUTINE

```
1890 REM LOAD FROM DISC
1910 INPUT "ENTER  FILE NAME ";F$
1914 PRINT D$; "OPEN ";F$
1918 PRINT D$; "READ ";F$
1930 J = -1
1940 J = J + 1
1950 INPUT T$
1960 IF T$ = "EOF" THEN 1990
1970 R$(J) = T$
1980 GOTO 1940
1990 T$ = R$(0)
2000 T1$ = CHR$(94)
2010 GOSUB 3040
2020 FOR I = 0 TO 10
2030 N$(I) = B$(I)
2040 IF LEFT$(N$(I), 4) = "STOP" THEN 2060
2050 NEXT I
2055 I = I-1
```

```
2060   N = VAL(B$(I+1))
2065   FOR  Z = 1 TO 5: B$(Z) = " ": NEXT  Z (Note: from Kilobaud)
2070   PRINT D$; "CLOSE "; F$
2075   GOTO 540
```

SAVE TO DISC ROUTINE

```
3180   REM  SAVE TO DISC
3190   INPUT "ENTER LABEL FOR FILE BEING SAVED "; F$
3200   PRINT D$; "OPEN "; F$
3205   PRINT D$; "WRITE "; F$
3210   T$ = "00000":  REM 5 ZEROES
3220   FOR I = 1 TO 10
3230   T$ = T$ + CHR$(94) + N$(I)
3240   T1$ = LEFT$(N$(I), 4)
3250   IF T1$= "STOP" THEN 3270
3260   NEXT I
3270   T$ = T$ + CHR$(94) + STR$(N) + CHR$(94)
3280   PRINT T$
3290   REM
3300   FOR J = 1 TO N
3310   PRINT R$(J)
3320   NEXT J
3325   PRINT "EOF"
3330   PRINT D$;" CLOSE ";F$
3335   RETURN
  *****
```

Finally, it is obvious that this program can be condensed by combining lines and eliminating some of the REM lines. Be careful in the latter case since these lines are frequently used as reference lines in GOTO and GOSUB statements. If the reader plans to key in the code as a learning experience the compaction might be counterproductive.

Since I think my version has been debugged and is running (I have not tried the SUM routine) a few simple files, I would be willing to make a cassette copy for anyone who has purchased the book. I feel strongly about the rights of program authors, so I will insist on proof of purchase. In any case, without the book and its instructions and demonstrations, the program will be of very limited value since it is not really self-prompting or internally documented. (4616 Ravensworth Road, Annandale, Va. 22203; Telephone (703) 941-6366, Home (703) 323-6276)

SWEET 16, THE COMPUTER WITHIN THE COMPUTER. by John L. Moon

On page 96 of the Red Reference Manual, begins part of the Monitor listing of a program called "APPLE-II PSEUDO MACHINE INTERPRETER" written by Steve Wozniak and included in the ROM along with Integer Basic, the Monitor, and the Floating Point Routines (see the newsletter Vol. 1, No. 4). Unfortunately, there is very little (as in no) documentation as to what this program does. One of the early issues of BYTE magazine (I've mislaid the date) had an article "System Description: The APPLE-II" by S. Wozniak that described the rationale for how he designed the APPLE-II. In five paragraphs and two tables at the end of this article he put the only documentation for Sweet 16 that I have found. (Supposedly, the WOZPAK has some information on Sweet 16.) The article in BYTE is only approximately correct. It must have been written before he actually finished writing the program because there are a number of differences between the article and the way that Sweet 16 actually works.

What is Sweet 16? As the title of this article suggests, it is a kind of computer within a computer. Wozniak wrote a program that simulates a nonexistent 16-bit computer. The name of this fictional (albeit usable) computer is Sweet 16. I have written an instant assembler/disassembler for Sweet 16 which I can give to anybody that is interested at the next meeting, along with its documentation. In this article, I'll just describe Sweet 16 itself.

Sweet 16 is just a computer program as implied by its implementation in ROM. You turn on Sweet 16 by a JSR from a machine language program on the 6502, to ROM address F689. Sweet 16 then assumes that the bytes stored immediately after the JSR instruction are the beginning of the Sweet 16 instructions. There is a special Sweet 16 instruction that turns off Sweet 16 and returns to 6502 mode. The original 6502 registers are saved and restored by Sweet 16 upon entry and exit into some of the monitor low memory areas.

The first 32 bytes of the APPLE's memory are used as sixteen 16-bit registers. The registers are numbered 0 to F in hexadecimal. Register 0 (locations 0, 1) is used as the accumulator for many of the Sweet 16 instructions. Register C (locations $18, $19) is used as a stack pointer by the Sweet 16 subroutine call and return instructions. Register D (locations $1A, $1B) is used as the destination of the subtraction operation in the interpretation of a compare instruction. Register E ($1C, $1D) is used as a status register and Register F ($1E, $1F) is the program counter for Sweet 16. Registers 1 through B are available for general programming usage. To interface to a Sweet 16 program, you can store data or pointers into the Sweet 16 registers prior to turning Sweet 16 on, except for the program counter register.

Sweet 16 has a very austere instruction set. The instructions are mostly oriented to doing memory moves, and pointer style addressing. It does support 16-bit addition and subtraction. Most opcodes are just one byte long. However, since the instructions are interpreted, their speed is about 10 times slower than 6502 machine language. On the other hand, it is about 10-100 times faster than Integer Basic.

The following table lists the opcodes of Sweet 16. I have also included with it the mnemonics that are recognized by the assembler I wrote. A description of the terminology of the table follows after the table.

Hex Code	Assembler Mnemonic	Operands	Action
00	RTN		Return to 6502 mode of operation
01	BRA	Addr	Unconditional relative branch (±127)
02	BCC	Addr	Branch on carry clear (±127)
03	BCS	Addr	Branch on carry set (±127)
04	BPL	Addr	Brach on plus (positive) (±127)
05	BMI	Addr	Branch on minus (negative) (±127)
06	BEQ	Addr	Branch on equal (zero) (±127)
07	BNE	Addr	Branch on not equal (not zero) (±127)
08	BNO	Addr	Branch on negative one ($FFFF) (±127)
09	BNN	Addr	Branch on not negative one (±127)
0A	BRK		Break to Monitor
0B	RTS		Return from subroutine
0C	JSR	Addr	Jump to subroutine (within ±127)
0D	NOP		No operation
0E	NOP		
0F	NOP		
1r	SET	r, value	Rr = value Sets Register r to value
2r	TRA	r	R0 = Rr Transfer Register r to Accumulator
3r	TAR	r	Rr = R0 Transfer Accumulator to Register r
4r	LBI	r	R0 = @Rr, Rr = Rr+1 Load Byte Increment
5r	SBI	r	@Rr = R0, Rr = Rr+1 Store Byte Increment
6r	LDI	r	R0 = @Rr, Rr = Rr+2 Load Double Increment
7r	SDI	r	@Rr = R0, Rr = Rr+2 Store Double Increment
8r	LBD	r	Rr = Rr-1, R0 = @Rr Load Byte Decrement
9r	SBD	r	Rr = Rr-1, @Rr = R0 Store Byte Decrement
Ar	ADD	r	R0 = R0+Rr Add
Br	SUB	r	R0 = R0-Rr Sub
Cr	LDD	r	Rr = Rr-2, R0 = @Rr Load Double Decrement
Dr	CMP	r	RD = R0-Rr, set status Compare (result to RD)
Er	INC	r	Rr = Rr+1 Increment
Fr	DEC	r	Rr = Rr-1 Decrement

All of the above opcodes that have an operand of "r" or no operand at all are just one byte in length. All the Branch opcodes including the JSR are two bytes in length, with the second byte containing a relative offset just like a 6502 instruction. The SET instruction is a total of three bytes long with the last two bytes having the value. In most cases, I have selected as an opcode the 6502 mnemonic that performs a (roughly) equivalent function if one exists. The abbreviations are used as follows:

Addr	A relative address for a jump destination.
r	A register number from 0 to F for registers R0 to RF.
value	A 16-bit value
@	Used to signify indirect addressing through a register; the contents of the register are used as the address for loading or storing data.

Notes: LBI, LBD, SBI and SBD all use just the low order byte of the register; on loads the upper byte will be zeroed. LDI, SDI and LDD fetch and store 16-bit quantities.

As I said, the Sweet 16 interpreter is turned on with a JSR F689. The interpreter instructions follow this JSR just as if they were more instructions (which they are). To get back into 6502 mode, the Sweet 16 RTN instruction is executed. Instructions that follow that RTN must be 6502 instructions. Many times it is convenient to make the entire Sweet 16 routine a self-contained subroutine which is callable from 6502 or Integer Basic. In this case, the routine can begin with the Sweet 16 turnon call, and end with a 6502 return. An example is given below with comments.

Hex Machine Location	Hex value	Opcode	Comments
300	20 89 F6	JSR F689	Turns on Sweet 16
303	11 00 04	SET 1,0400	R1 = address of screen buffer
306	12 00 04	SET 2,0400	R2 = count of characters
309	13 E0 00	SET 3,00E0	R3 = ASCII blank
30C	23	TRA 3	R0 = ASCII blank
30D	51	SBI 1	Store blank on screen
30E	F2	DEC 2	R2 = R2-1
30F	07 FC	BNE 030C	Make all of screen blank
311	00	RTN	Return to 6502 mode
312	60	RTN	6502 return from subroutine

This example routine does the same thing as a CALL -936, but does it with a CALL 768. It clears the screen.

If you have the S-C Assembler, you can find a number of uses for Sweet 16 within it. Disassemble portions of it looking for the JSR F689. Anything following that is Sweet 16 until the 00 that means return to 6502. Next month I'll talk about writing a text processing program (including the text editor) for the APPLE.

HOW WOULD YOU LIKE TO "CLUSTER/ONE"? by Bernie Urban

At the last minute I decided to attend a portion of the Personal Computing Festival in New York, and as a result I am able to honor my promise of providing you with more information on the Nestar system. I have liberally borrowed from the promotional literature provided to me by Dr. Harry J. Saal, President, Nestar Systems Inc.

Cluster/One is a distributed computer system based on independent micros connected together via a high-speed parallel data bus. There can be up to 15 stations comprised of a mixture of PETs, TRS-80s and APPLEs. This can be doubled via their optional feature. Why such a system? It allows for the sharing of expensive peripherals, e.g. a disk drive and a printer. I asked whether tablets, plotters and modems could also be added, and apparently they can. "A Cluster/One system can be used in a drop-in computer center as a commercial venture. Computer dealers can very effectively demonstrate and compare the capabilities of the APPLE, PET and TRS-80..." Educational uses abound. "Very large software systems can be run... since one BASIC program can cause others to be loaded from the disk and executed. This facility can be used for tutorial applications... Whenever a change must be made to a program, only one copy need be changed. After that, all users have access to the updated version." The total cost of a system is "only a fraction of what a time sharing system costs."

"Nestar Systems produces the central Cluster/One system which contains two disk drives, disk and bus controllers, buffer memory and power supplies. The unit costs $ 4,500 for the single disk version which holds 630K bytes on-line. A double sided version with 1.2M bytes costs an additional $500. The Cluster/One console is another personal computer (PET ed.) which is used for starting and monitoring the system and running utilities such as full or incremental disk back-ups." There is more information in my materials, or you can call (415) 327-0125 or write to Nestar Systems Inc., 430 Sherman Ave., Palo Alto, Ca 94306.

THE NEW YORK PERSONAL COMPUTER FAIRE by Bernie Urban

It was mind-boggling. Interestingly, APPLE did not exhibit with the other personal computers. Rather they chose to be with the big boys and pushed APPLE as a business computer. I picked up extensive literature -- you're welcome to browse through it at our next meeting. Here are some items of interest to APPLE fans:

1. PASCAL will be available in the "third quarter of 1979". It will cost $495 but will supply you with all sorts of goodies and do away with the need for APPLE-SOFT ROM (incompatible). You can run in any of the three modes but you must have 48K memory and one disk drive. PASCAL essentially gives you a 64 K RAM system - the language card has 16K of write-protectable RAM and 2K ROM for the Auto-Start. You get 5 diskettes including the Integer Basic, Applesoft Extended Basic and PASCAL.

2. Did you say that 110K bytes of disk storage on-line is not enough for your application? How would you like 10M bytes of storage? For $4990, CORVUS Systems, Inc., P.O. Box 1590, Cupertine, Ca 95014, will supply you with the IMI 7710 "Winchester" disk drive with 280-16K intelligent controller, power supply and APPLE interface card and associated software. Not enough? You can add three more drives for a total of 40 megabytes!

3. You can add a Digi-kit-izer to your APPLE (a computer graphics input device) for $499 plus $99 for the interface board. Check TALOS Systems, Inc., 7419 E Helm Drive, Scottsdale, Ariz 85260 for this one. APPLE was using their own version for business applications and for drawing color portraits at the show. It can be used, given some appropriate character/pattern recognition software, for teaching cursive writing.

4. I saw a plotter in operation but did not pick up any literature. It drew a great picture of the APPLE logo using multicolor pens. Should have tried to

draw our logo.

5. You can hook up your APPLE via a modem to Smart system, which for $2.75/hr for connect and CPU time (6 PM to 8 AM Eastern time, weekends and holidays) you gain access to an electronic mail function, downloaded programs for the APPLE, commercial data bases like UPI . You can program in FORTRAN, COBOL, SUPERBASIC, PL/C and RPG and more. Check with Smart System, 1301 West Estes Avenue, Chicago, Ill. 60626.

6. I saw the COMPRINT Model 912 in operation. It looked good, operated quietly and produced high quality print characters. This is due to their use of a 9 X 12 dot matrix to fill in greater detail. However, it was done on aluminized paper. It is supposed to "Xerox" very well. Your applications may call for white bond or high quality print paper. Price for the COMPRINT is $560.

All you CAI fans come see me about news I picked up from Roger Cutler re APPLE and education (at our next meeting).

MINUTES OF MEETING - 5/26/79

While waiting for latecomers we discussed several items. Superchip - Mark Crosby prefers software version like screen machine. Sue Eickmeyer discussed WOZPAK version - circuitry for lower case chip and software is available. John Moon mentioned a deal between APPLE and Bell and Howell for educational programs. Howard Richoux discussed the FORTH language put out by FORTH Co., and Bernie Urban mentioned the burgeoning nationwide interest in setting up an APPLE Network.

The group decided not to create a 4th member-at-large position for a "Green Apple" (our Constitution calls for only 3), but we would have such a designee with yet-to-be-decided responsibilties and authority.

The election took place and our new officers are:

President - John Moon
Vice-President - Bernie Urban
Treasurer - Robert Peck
Secretary - Genevie Urban
Members-at-Large - Mark Crosby, Sue Eickmeyer and Sandy Greenfarb

The meeting adjourned to John Moon's 6502 Machine Language Course and to the exchange of programs.

CALENDAR OF EVENTS

Date	Events/Meetings	For Further Info. Call
June 25	Chesapeake Microcomputer Club 7:30 PM, Kahler Hall, Harpers Choice Community Center, Columbia, Md.	Mani Alexander Off. 452-5232
June 28	NOVAPPLE	

CALENDAR OF EVENTS

Date	Events/Meetings	For Further Info. Call
June 25	Chesapeake Microcomputer Club 7:30 PM, Kahler Hall, Harpers Choice Community Center, Columbia, Md.	Mani Alexander Off. 452-5232
June 28	NOVAPPLE 7:30 PM, Computerland, Tysons Corner	Jim Nielsen Off. 693-7530
July 11	Assn. of Personal Computer Users 7:30 Pm, Chevy Chase Library	Daphne Schor Off. 544-8530

@@

NEXT MEETING OF WASHINGTON APPLE PI

Saturday, June 23, 9:30 am

GEORGE WASHINGTON UNIVERSITY
Tompkins Hall, School of Engineering, Rm 206
23rd & H Streets, NW

Parking roulette, or in students' parking lot, if the chains are down.
Convenient to Metro.

@@